described in a subsequent section on the domain knowledge expert. Figure 4.5 shows an example of how a portion of the conceptual knowledge might be structured. The node labeled "document collection" indicates that the attached single word concepts are components of the document representatives.

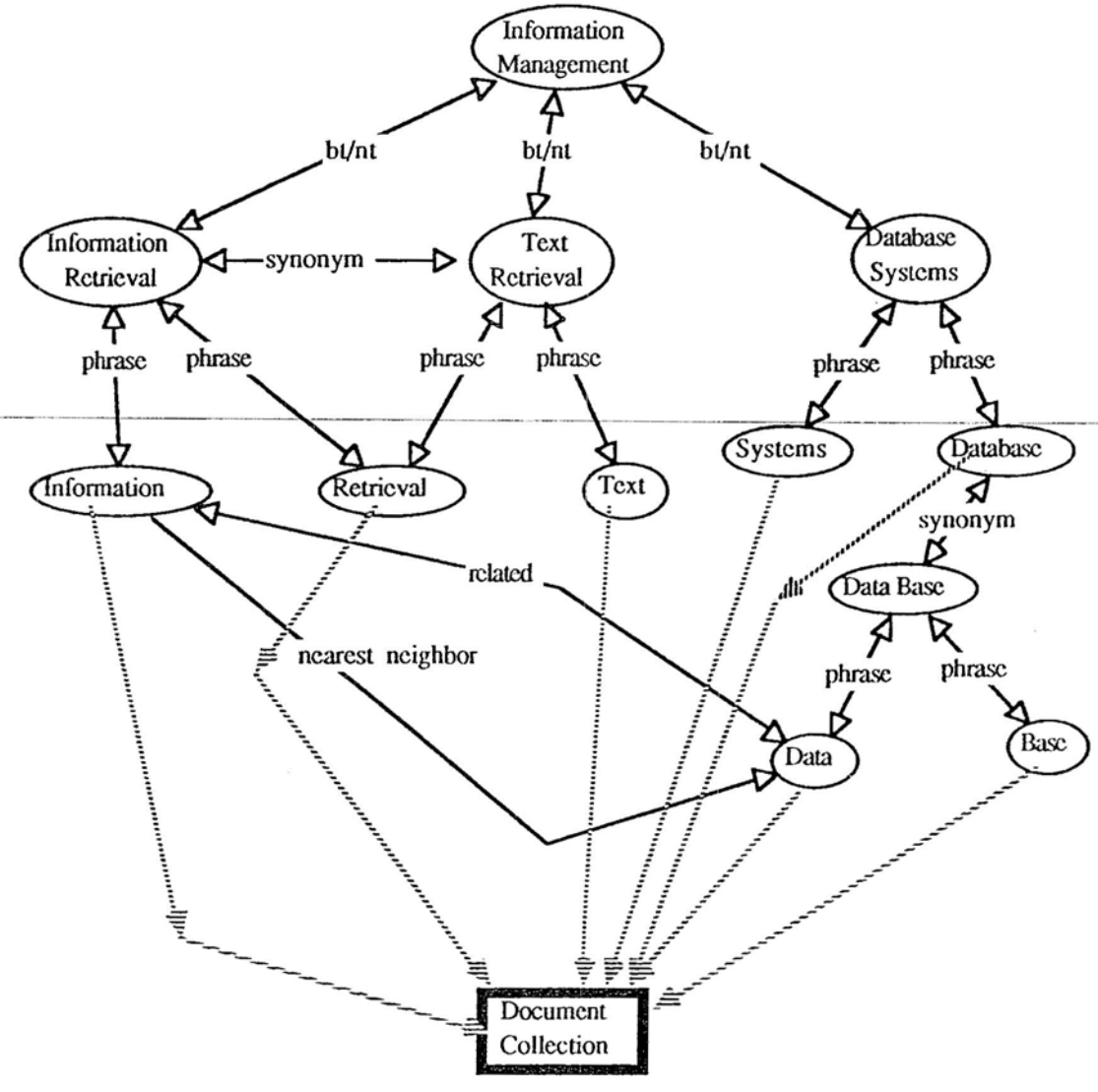

Figure 4.5: Sample Conceptual structure.

The concept level, during a session, is a combination of three knowledge bases, the global domain knowledge, the user's domain knowledge, and the text database. The global domain knowledge is derived from published thesauri or other classification information.

In the case of our test data, for example, it was derived from the old and new Computing Reviews classifications. The user's domain knowledge is gathered from him as he interacts with the system. When a user is developing his query initially or is examining retrieved text, he may make connections between words. For example, a user examining a document may indicate that the words "concurrent" and "processes" form the concept "concurrent processing." This phrase is entered in to his domain knowledge model by the Domain Knowledge Expert. He may then think of the concept "parallel processes," and decide that it is not close enough to be a synonym, but is certainly related. Figures 4.6 through 4.9 show the user selecting these phrases and connecting them with the "related" link.

The information from the text database simply gives the mapping from terms to the documents in which they occur. These links are represented by the dotted links in figure 4.5.

4.2.5.1.2 Implementation of the Concept Level

The semantic net representation of the concept level is implemented by means of a hash table of record structures. Hash tables are a convenient structure in Common Lisp. They are expanded in size automatically when they reach a specified percentage of their total capacity, the equality test used to determine if a hit has been made can be user specified, and there is a special function, `maphash`, for iterating through all the values in a table. The definition of record structure used for storing each entry is:

```
(Defstruct   (Concept
                (:Include Common-Content-Info)
                  ;; the above is used for DK entry
                (:Conc-Name nil)
                (:Predicate Concept?))
```

```
Id        ;either a term number or if a phrase a list
          ;of term numbers
Stem      ;either a stem or a list of stems
Text      ;full text of the concept
Synonym   ;list of synonyms
Related   ;list of related words
Broader   ;list of broader terms
Narrower  ;list of narrower terms
Phrase    ;text of single words that make up phrase or
          ;list of phrases that this word is in)
```

Each concept is stored using its full text as the key to the hash table. So, when the domain knowledge expert wants to find all of the phrases that a single word concept is a part of, it executes the function `Get-DK` which is defined as follows:

```
(Defun Get-DK (Connection Word Table)
          (Apply Connection (Gethash Word Hash-Table)))
```

where `Connection` is the particular link, `Word` is the concept of interest and `Table` is either the user's domain knowledge or the global domain knowledge. The `id` field provides the connection into the document representations, which are stored in VMS/RMS files.

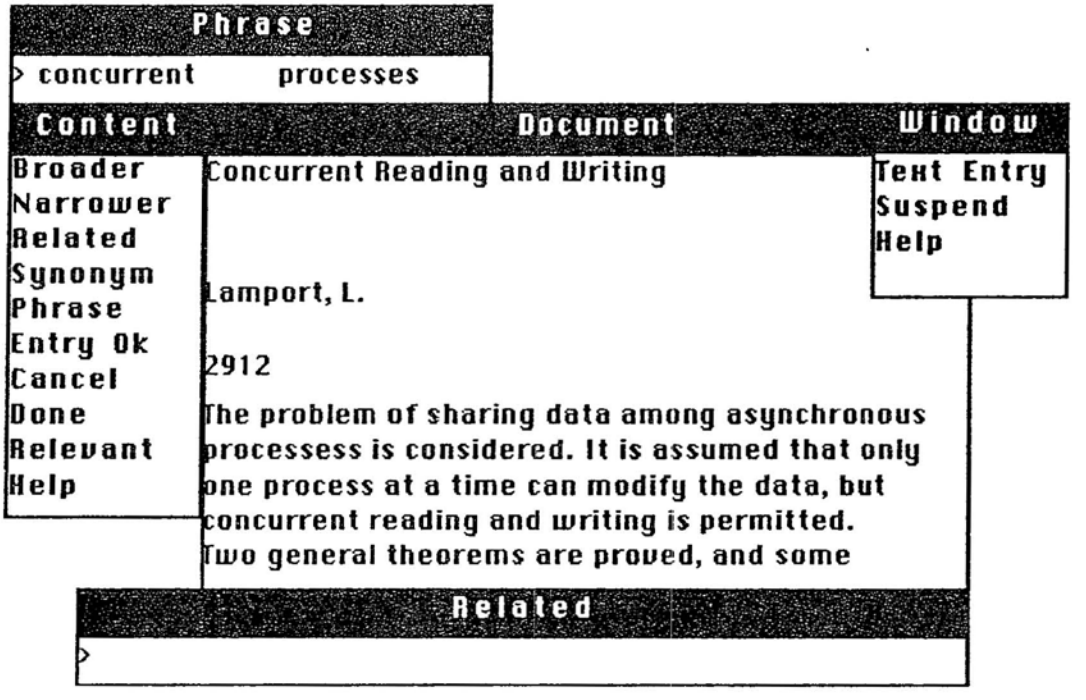

Figure 4.6: The user is making a connection between "concurrent processes" and "parallel processes." The first step is to select `Related` from the Content menu. This causes the related window to appear. The user then selects `Phrase` from the Content menu, causing the Phrase window to appear above the document. Selecting the words "concurrent" and "processes" using the mouse causes them to appear in the Phrase window.

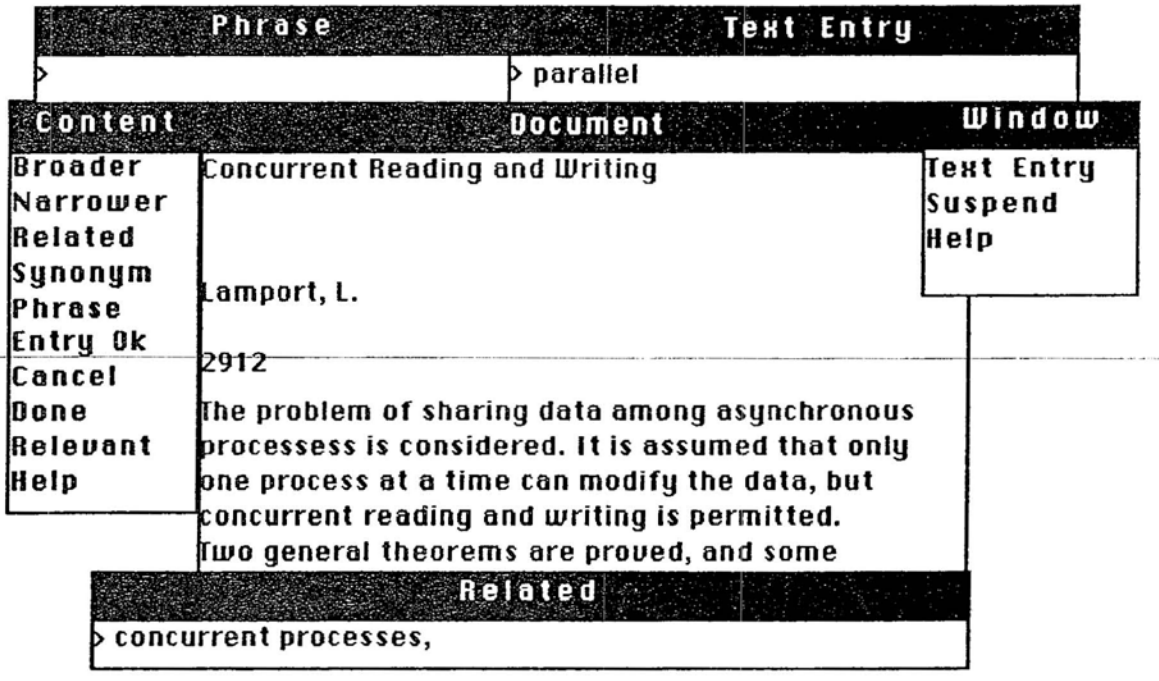

Figure 4.7: After selecting **`Entry OK`** from the Content menu, the phrase "concurrent processes" is transferred to the Related Window. The user then selects **`Phrase`** again from the Content menu and then **`Text Entry`** from the Window menu to allow him to enter the word "parallel."

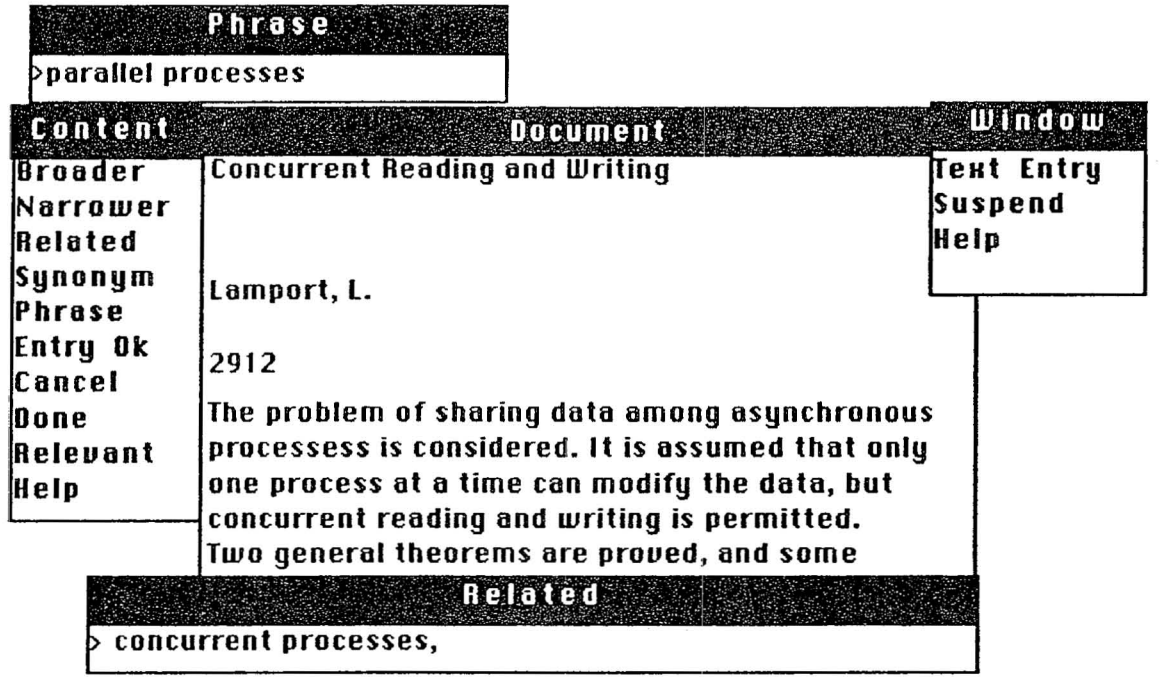

Figure 4.8: The user has keyed `return` in the Text Entry window (which then disappears), causing the word "parallel" to appear in the Phrase window, and has selected "processes" from the text, which also appears in the Phrase window.

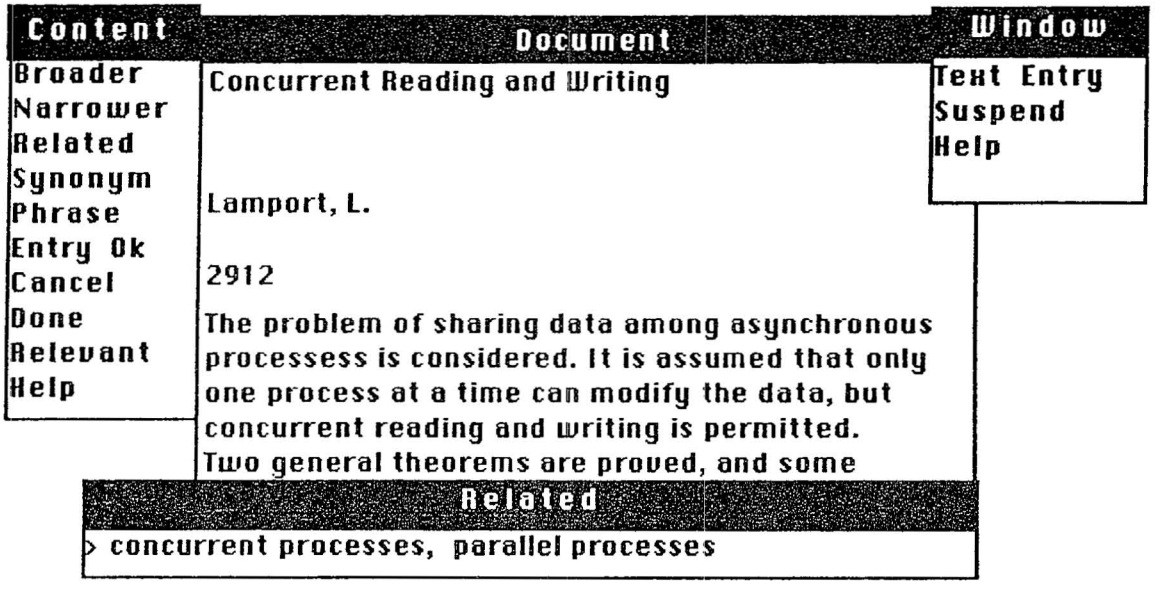

Figure 4.9: The user selects **Entry OK** from the Content menu, which causes the phrase to be transferred to the Related window. When the user selects **Entry OK** again, the Related window disappears and the domain knowledge is entered into the user's domain knowledge model.

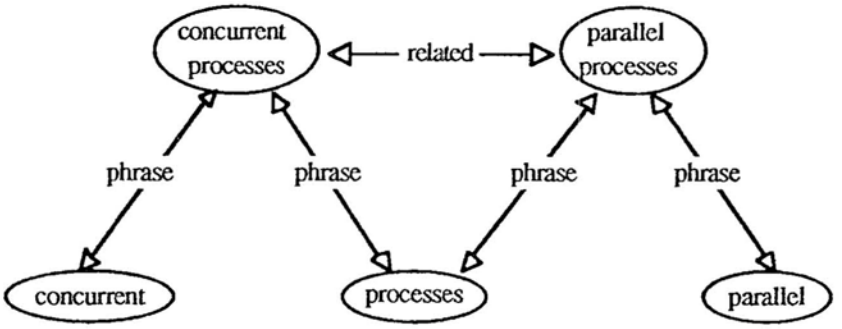

Figure 4.10: Representation of the domain knowledge added to the user's model

4.2.5.1.3 Document Level

The next level is the document level. In this system, documents are the fundamental units of information. It is in a document that the user will find the desired information. Each document is represented by a collection of concepts that indicate its content. These are generally derived from the title, abstract, and whatever keywords are supplied by the author. Because of the automatic indexing method [Porter 80] used to derive the document representations, they consist only of single word concepts (terms). The document level is, like the concept level, highly interconnected. Figure 4.11 shows the kinds of links that can be found. A document nearest neighbor link, which is similar to the concept nearest neighbor link, is used. It is based on the overlap of the terms contained in representations of two documents. Only the most highly related documents are linked by the nearest neighbor link. The algorithm used for generating these links was described in chapter two.

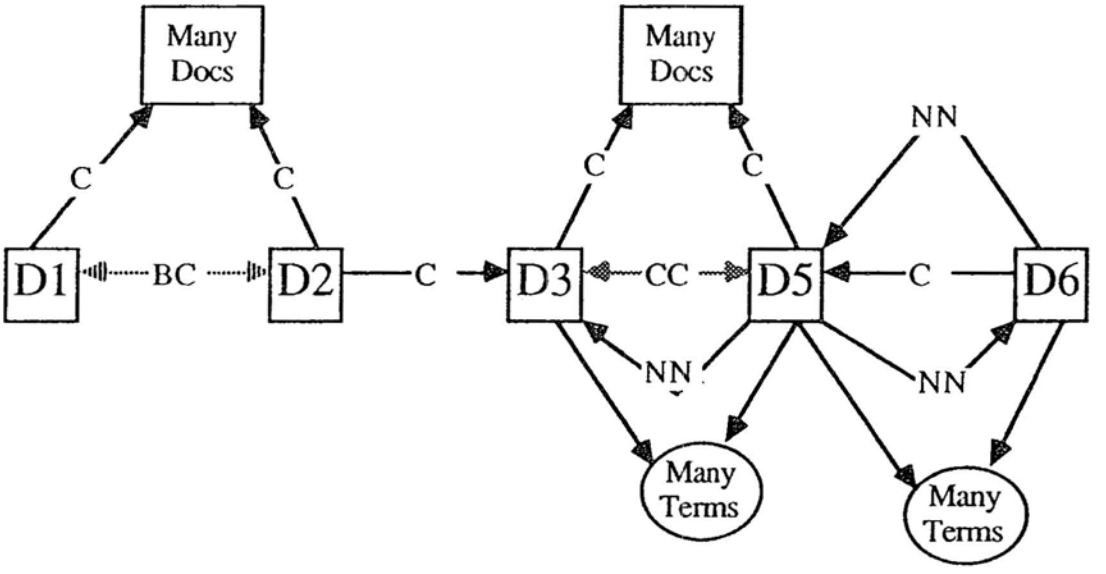

Figure 4.11: The Document level. The markings on the links are as follows: BC = Bibliographic Coupling, CC = Cocitation, NN = Nearest Neighbor, C = Citation.

Each document also contains citation links, which connect it to other documents; these are important since they represent author judgements of what other documents are related to the contents of their documents. They can be used directly to facilitate finding other documents, or they can be used to generate two other kinds of document-document similarity links called bibliographic coupling (BC) links and co-citation (CC) links [Salton 83, Mansur 80]. The first is based on the overlap of two documents' reference lists, and the second is based on the number of times that two documents appear together in reference lists of other documents. Due to some deficiencies in the test collection that is currently used by I^3R, the BC and CC links are not generated. Specifically, the citation information available for each document only references other documents in the collection. Consequently, accurate determination of bibliographic coupling links could not be made, since much many references were missing.

Even without bibliographic coupling and cocitation links, the document level even in the collection used is still highly interconnected. Figure 4.12 shows a region of the test collection, containing documents that deal with tree data structures. In a few cases, the

nearest neighbor links link the same documents as the citation links do. In others, they link documents whose connection by citations is more circuitous, or is not made at all.

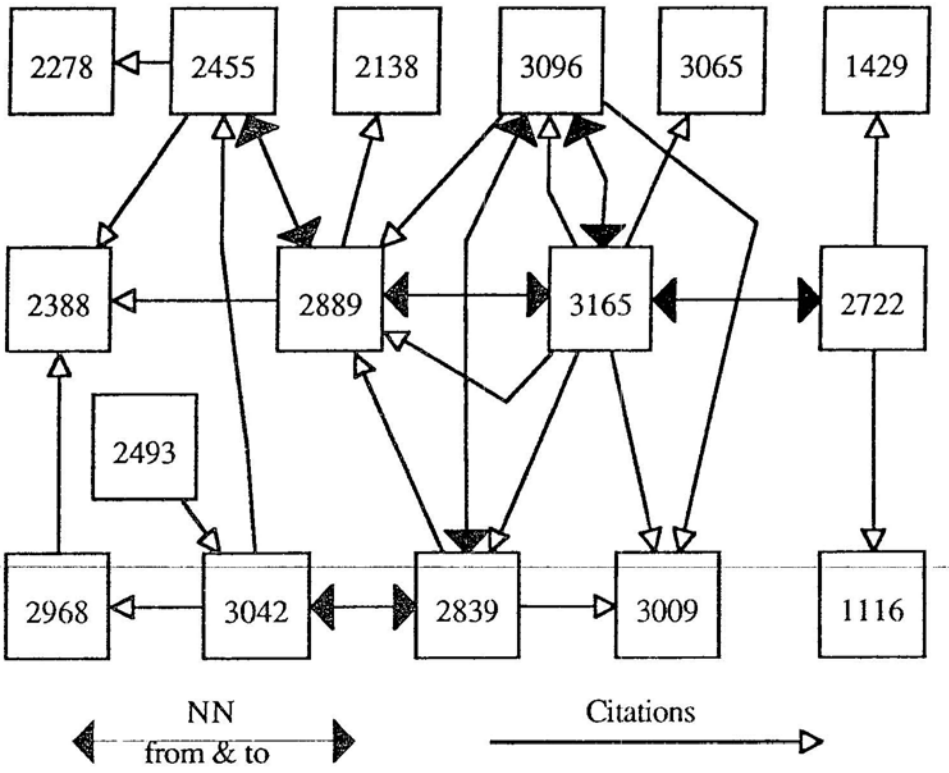

Figure 4.12: Document neighborhood taken from the CACM collection. This neighborhood will be used in the fourth scenario in chapter six.

4.2.5.1.4 Implementation of the Document level

The document level is implemented using VAX/RMS file structures. There are 9 files which are:

1. DOC_TERM – holds the basic document representations,
2. DOC_DOC_CITATION – holds the citation links,
3. DOC_DOC – holds nearest neighbor links,
4. DOC_TEXT – holds the full text of the document as well as the date,
5. DOC_AUTHOR – maps documents to authors,

6. TERM_DOC – hold the inverted file of #1,
7. TERM_TERM – holds term nearest neighbors,
8. TERM_PHRASE – maps the text of terms to the term numbers,
9. COLLECTION_INFO – holds information about the size of the collection and the most frequently occurring term.

These files are accessed directly by the search program. The rest of the system uses access programs that are written in C that can be called form Lisp. Originally, these files were stored as relations in a relational database, VAX/RDB. This database system, however, was very inefficient, causing the searches to take about five minutes. Searches in the current system take about 50 seconds.

4.2.5.1.5 Journal Issue Level and Above

The journal issue level is based on the observation that many journals will have from time to time whole issues devoted to a specific topic. These issues might be the proceedings from a particular conference, or tutorial articles on a specific area. Journal issues may also have sections which are topic specific. The primary motivation for including this level is to support browsing. A typical browsing heuristic is: "If this document is interesting, then are there any more documents in this issue that are interesting."

There are other possible connections above the journal issue level. The most obvious in a collection that has more than one source of information is connecting the journal issues together into journals. The journals can then be categorized by higher level classification structures like the Library of Congress system.

In the current system, the journal issues are not represented by a separate file structure. This information is available from the DOC_TEXT file which has fields representing the month and year that the document appeared.

4.2.5.1.6 The Test Collection

The test collection used to test the implementation of I^3R consists of 3204 documents taken from the Communications of the ACM, from the years 1960 to 1979.

4.2.5.2 User Histories

The other part of the long term memory is the user histories, which consists of two parts, the user's domain knowledge and the records of the user's previous searches. The user's domain knowledge is organized the same way as the global domain knowledge. The user record has the following structure.

```
(Defstruct (User-Record
                (:Conc-Name UR-)
                (:Predicate UR?))
           (User-Name nil)
           (Session-Records nil)
           (Domain-Knowledge))
```

This record is stored in the VMS file system in a file with the name `<username>.model`. The user name corresponds to what the user's account name is on the particular machine.

The session records contain information that was in the short term memory when the session was suspended or ended. Briefly, a session record consists of:

1. The stereotypes that were in effect when the session was closed. (These will be discussed in the next section on experts under the user model builder.)
2. The request model consisting of the concepts that were judged relevant, any relevant phrases, and the documents that have been judged relevant.
3. If the session was suspended, the browse map if there was one, and the state -history.

A session record has the following structure.

```
(Defstruct (Session-Record
                (:Conc-Name SR-)
```

```
   (:Predicate SR?))
(Date nil)
(Stereotypes nil)
(Initial-Need nil)
(Document-Evaluations nil)
(Term-Weights nil)
(Browsing-Path nil)
(Session-Complete))
```

4.2.6 Experts and Short Term Memory Structure

In the current implementation of I^3R, six of the eight specified experts were implemented. The natural language expert (NLE) was not implemented since the needed techniques are still under development [Croft 87]. The explainer (EXP) was not implemented since it represented a significant piece of work in itself and it was felt that the utility of the I^3R could be demonstrated without it. The browsing expert will be discussed in chapter five.

4.2.6.1 Control Expert (Scheduler)

4.2.6.1.1 Purpose

One of the major differences between the traditional blackboard system and I^3R is the purpose of the control expert or scheduler. In most blackboard systems, the purpose of control is to constrain the system's activity to processing those KS instantiations that will contribute the most to solution of the problem. In other words, the scheduler's purpose is to conserve the resource of time.

In information retrieval, the purpose of control is to constrain the course of a session, so that it appears logical and consistent to the user, rather than being chaotic. In this sense, the control function could be considered a dialogue manager. What the control

function determines is the relative importance of the activity of the system experts during a given part of the session.

The control function must be organized to provide a flexible dialogue with the user. The work by Belkin [83], Brooks [83], and Daniels [85] has shown that, while the same basic steps are accomplished in every session, the order may vary considerably. Consequently, the control function has to be organized so that it can handle this variability. Their analysis goes fairly deep, examining the changes in focus down to the utterance level. In I^3R these lower level utterances are determined by the individual experts. For example, the domain knowledge expert would decide what concepts should be presented to the user for approval, the control expert would determine when the domain knowledge expert should be engaged in this activity.

Because the actual requests for and presentations of information, which correspond to the utterances of a human intermediary, are determined by the different experts, the control expert does not have to engage in a sophisticated process to determine the course of a session. Therefore, the control expert can be implemented as a state/transition network with relatively few states that encodes the general plan or sequence of activities for the information retrieval process. In this network, there are two types of states, intermediate and leaf states. The leaf states are where priority orderings for the experts are determined. The transitions are also of two types, normal and exception. Normal transitions encode the standard path through the states; exception transitions are taken when the session is not proceeding as expected. Figure 4.13 shows the structure of the network. Selection of what transition to take is based on parameters derived from the stereotypes determined by the UMB and by completion of certain functions; for example, the user has entered his query.

Another interpretation of these states are that they are goals to be met. When all the goals are met then the session is complete. Associated with each state, then, are the criteria

to determine when that goal is satisfied. At present there are two criteria, the number of relevant documents expected to be found and the number of searches expected to find them.

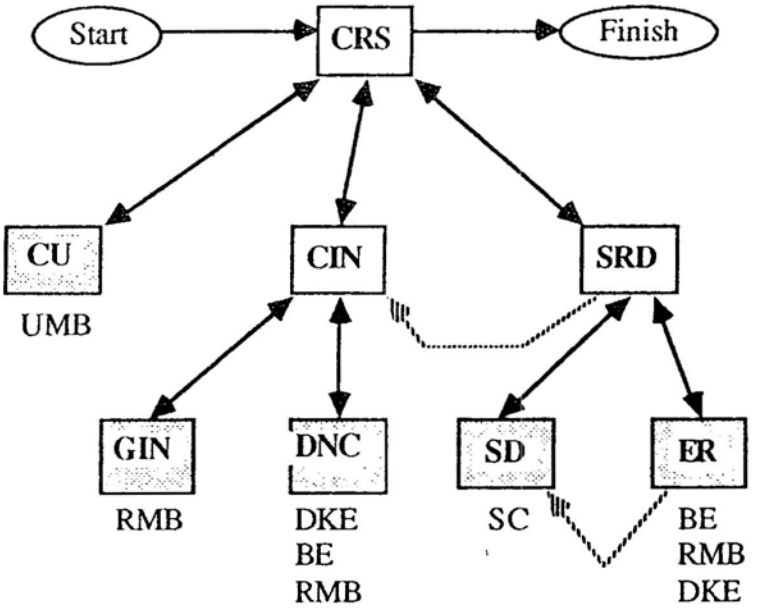

Figure 4.13: Control Expert States. Shaded states are leaf states where the relative priorities of the experts are established.

4.2.6.1.2 Conceptual Operation

The top level goal of Conduct Retrieval Session (CRS) simply represents the goal of the system, and it is met by the three subgoals of Characterize User (CU), Characterize Information Need (CIN), and Search for Relevant Documents (SRD) being satisfied, or the user indicating that he wants to quit or suspend the session.

The purpose of the first subgoal, Characterize User, is to allow the UMB to engage the user in a dialogue to determine what stereotypes apply to the user for the session. This allows the search and document expectations to be posted in the STM. Posting of these signals the satisfaction of the goal. Figure 4.14 shows the values that the control expert uses based on the user's stereotypes. These values are based on the following assumptions.

1. A domain expert will be able to specify the information that he is looking for precisely.

2. A domain novice will *not* be able to specify the information he is looking for with the same precision as an expert.

Because of the inability of the domain novice, it will require more searching to find relevant information. Furthermore, the domain novice may not recognize relevant documents. The values chosen reflect the abilities of these kinds of users.

Search Emphasis	**Domain Knowledge Expertise**	
	Novice	Expert
Exhaustive (recall oriented)	D = 15 S = 4	D = 20 S = 2
Selective (precision oriented)	D = 5 S = 2	D = 5 S = 1

Figure 4.14: Summary of control expert expectation values based on user stereotypes. D is the number of *relevant* documents expected, and S is the maximum number of searches needed to find the relevant documents

The next goal is Characterize Information Need (CIN) which is divided into two subgoals of Get Information Need (GIN) and Develop Need Context (DNC). The first goal (state) only allows the RMB to operate, during which the user enters his query in one of the query entry forms supplied by the RMB (see section 4.2.6.3). Completion of the state is marked by the internal form of the initial need being posted. The DNC state is characterized by an interaction between the domain knowledge expert (DKE) and the user where the DKE suggests, for user approval, additional concepts to be added to the developing information need. Any terms approved will be added to the request model by the RMB.

The browsing expert (BE) may also be active during this state, DNC. Whether or not it is depends on the user model and the history of the session. If control expert returns

to this state as a result of failure to retrieve the expected number of relevant documents in the number of searches allowed or if two searches in a row fail to retrieve any relevant documents, then browsing will be enabled. It will also be enabled if the user is categorized as a system expert. The state is completed when either the DKE has no more terms to suggest, or the user quits browsing.

Once the states of GIN and DNC are finished, then so is CIN, and the system passes to Search for Relevant Documents (SRD). If the search and relevant document expectations have not been met the control expert passes to the Search for Documents (SD) state where the search controller selects an appropriate strategy. When finished the SC posts the results in the STM and passes them to the interface manager (IM) for evaluation by the user.

The control expert (CE) then goes to the Evaluate Results (ER) state. Here the user will determine what documents and concepts are relevant. This information is of interest to the RMB, DKE, and SC. The user is also able to browse at this point depending on the context of the situation. If the user is browsing the RMB and the DKE will record judgements made during the process. The ER state is finished when either the user exits browsing after having evaluated at least one document as relevant or after having evaluated the search results.

After the results of the search are evaluated and the user has not found the expected number of documents the exception transition back to SD is taken, so that the SC can use the revised request model for a new search. If the expected number of searches has been taken and the expected number of documents has not been found, the CE will take the exception transition from SRD to CIN and then take the normal transition from CIN to DNC. This transition embodies the idea that the information need is still not defined sufficiently and needs further refinement by the DKE. If no new concepts are added the system will

suggest that the user browse if he has not already done so. If concepts are added, the CE will return to the SD to make use of the revised need model.

4.2.6.1.3 Implementation

The Control expert is implemented using Lisp symbols to represent the states (or goals) and rules to represent the transitions. State related information is kept on the property list of a state. This information includes the expert priority list for those states where the experts are operable. It also includes the substates that must be completed for the state to be completed; for example, the state CRS has three substates, GIN, DNC, and SRD that must be completed for it to be completed.

Since the transitions are implemented by rules, it is easy to add new transitions to the control expert. For example, say that the user model builder is expanded so that it also monitors how the user interacts with the system. At some point, the user model builder may wish to ask the user about how he works with the system, since the user's activity differs from that expected by the system based on the original model. A transition can be added that will take the system state back to the Characterize User state, so the user model builder can pose questions to make new determinations. Addition of this transition only requires a new rule that would respond to the conditions of the system and expectations set up by the UMB.

4.2.6.2 User Model Builder

The purpose of the user model builder is to determine where the user fits into the kinds of users that the system expects to find. Recognition of these stereotypical users is embedded in the rules that make up the UMB. The second purpose is to perform housekeeping functions for other experts by maintaining the user models kept in the long term memory.

In I^3R there are three kinds of information about the user that are important to maintain. The first is the user's domain knowledge. Since this information is primarily used by and manipulated by the DKE, the UMB simply performs the housekeeping functions of storing it at the end of a session and retrieving it at the beginning. The second kind of information is the characterization of the user along three lines, experience with computers, experience with the domain of the search topic, and interest in exhaustive or precise search. The third is the user's history, which is a record of the search sessions that the user has with the system. This information is used primarily by the request model builder, so here too the UMB performs housekeeping.

The primary reason for the inclusion of the user model builder is to demonstrate how a user model can be used to modify the behavior of the system. This is different from the use that user models were put to in GRUNDY, where the purpose was to assess the user and find books that would match or fit with that assessment. To this end, what the model should contain is determined by what behaviors are important to modify. Adaptability based on user models is another significant contribution of this thesis to the design and implementation of information retrieval systems.

The behaviors in I^3R that are important to modify are how the search controller responds to the information need, and how the interface changes with the respect to the ability and experience of the user. The second aspect is made up of a number of the kinds of choices that the user has for domain knowledge entry, the amount of information to be displayed on the two browsing maps, and how quickly the search controller will initiate a search while the user is browsing.

The UMB determines what stereotypes apply to the user by questioning him directly. It asks directly whether the user is interested in an exhaustive or a specific search to determine the interest in recall or precision. It poses a number of choices to determine the user's domain and system experience. These choices are for domain experience:

1. Know very little.
2. Have read a few news magazine articles on the subject.
3. Have read a few science magazine articles on the subject.
4. Have read a textbook on the subject.
5. Have read a few journal articles on the subject.
6. Have written journal articles on the subject.
7. Have written a textbook on the subject.

And for system experience are:

1. Seldom use a computer.
2. Use a word processor.
3. Own a personal computer.
4. Have never user an information retrieval system before.
5. Have used an information retrieval service before.
6. Frequently use an information retrieval service.

To these the user can answer yes or no; the default answer is no. If the user answer yes to questions 5, 6, or 7 of the first group, he is considered a domain expert. If he answers yes to 3, 5, or 6 of the second group, he is considered a system expert. From these determinations, the UMB posts on the STM on the `*User-Model*` place, its evaluations in the form of a list. For example, `((Domain Novice) (System Expert) (Search Recall))`.

4.2.6.3 Request Model Builder

The primary purpose of the Request Model Builder (RMB) is to keep track of the information provided by the user that pertains to the developing query. This includes the initial definition of the query, the single word concepts (also called terms) and multi-word concepts derived (called phrases) that the user considers important from the initial query, from concepts presented by the domain knowledge expert, from documents presented by

the search controller, and from browsing. Each term is stored in a hash table of records that is keyed by the term's number and its stem. The following is the record structure kept in the table.

```
(Defstruct
   (Term-Representative
      (:Conc-Name TR-)
      (:Predicate TR?))
   (Full-Text nil :Type String)
   (Stem      nil :Type String)
   (TermNo      0 :Type Integer)
   (Cfreq       0 :Type Integer) ; Collection Freq
   (Qfreq       0 :Type Integer) ; Query Freq
   (RelFreq     0 :Type Integer) ; Relevant Freq
   (User-Judgement nil)
   (Source nil)
   (Concept nil)
   (DK-Checked nil)
   (B-Recommended))
```

The RMB also performs stemming of any input text (see the example in chapter two), as well as providing different ways of initially specifying the information need. These specifications are:

1. free text,
2. a known document,
3. a simple Boolean query — terms occurring on the same line on the input form are implicitly ORed together, the separate lines are implicitly ANDed, and NOT is not allowed,
4. a complex Boolean query — an arbitrarily complex query using all three operators and parentheses.

Choices 3 and 4 are provided because many users are used to this form of specification or they may have previously developed queries on another system that they wish to transport. Boolean queries in either form would be processed in the following way [Croft 86a].

1. The component words are stemmed.

2. The query is transformed into a tree which is then used to generate candidate phrases for user approval. For example, the query, (parallel OR distributed) AND (processes OR programs) yields the phrases: parallel processes, parallel programs, distributed processes, and distributed programs.

3. The candidate phrases are presented to the user for evaluation. This step is required because the combinatorial nature of step 2 can produce phrases that do not make sense.

4. Phrases approved by the user are added to the request model.

The RMB also maintains a list of documents, on the STM place `*Doc-Evals*` that have been seen by the user. Each document is represented by a record:

```
(Defstruct (Document-Representative
              (:Conc-Name DR-)
              (:Predicate DR?)
            (ID 0 :Type Integer)
            (User-Judgement nil)
            (Nearest-Neighbors nil)
            (Terms nil)
            (B-Recommend nil))
```

Documents that have been judged relevant have their component terms retrieved and put in the appropriate field; documents that have been retrieved and are not judged relevant do not have their terms retrieved.

4.2.6.4 Domain Knowledge Expert

The DKE is one of the major functions in I^3R devoted to query refinement. Its responsibilities are twofold. The first is to build a model of the user's domain knowledge. The second is to search the available domain knowledge models for concepts that are related to those supplied by the user while describing his information need.

An underlying assumption in the operation of the DKE is that the user is the final authority on what is and is not relevant to his need. Therefore, the DKE acts as an advisor

to the user while the query is being developed. This means that the DKE will only suggest concepts for the user's approval, and will never automatically add any.

The DKE has, in the current design of the system, two sources in which to look for concepts, the global domain knowledge and, if present, the user's domain knowledge. There will be at least a minimal amount of information in the global domain knowledge consisting of term nearest neighbors. The order in searching for candidate concepts, is to search the user's knowledge before the global knowledge.

Searching for concepts proceeds by taking the terms in the request model that have not been checked previously by the DKE, and using them as entry points into the knowledge. From these entry points a modified form of spreading activation is performed. The links emanating from them are chosen in a specific order to find candidate concepts. The basis for the order is to find the words that are most likely to be, in the mind of the user, associated with the information that he is looking for.

The first links taken are the nearest neighbor links. These are relatively rare, therefore, if they exist it is very likely that the associated term is "about" the same topic. This not to say that the nearest neighbor is synonymous, but by reason of the association hypothesis (see section 2.2.2.3.2.3) it will be dealing with the same topic. The second link followed is the synonym link, since synonyms are defined to have the very same meaning. In effect, these words have to be interchangeable. These words increase the coverage of the request model, but do not expand meaning of it. Synonyms are especially important in fields where there are a number of synonymous terms for the same concept. For example, in graph theory, point, vertex, and node mean the same thing, and line, edge, and arc are also synonymous. The third link followed is the narrower link, which tends to make the query more specific. For example, a user may be interested in trees (botanical usage) and the DKE might find deciduous trees and coniferous trees as narrower terms. The user may or may not be interested in the distinction.

The next group of links to be followed tend to expand the query beyond its original meaning. The fourth link followed is the related link, which gets terms that bear some general association to another term or are a cross reference. The fifth link to follow is the phrase link that connects single word concepts to any phrases that they are members of. The last link that is followed is the broader-than link, which finds more general terms

This activation method differs from that used in most semantic link based systems. In those systems the purpose of the activation is to determine what relationship, if any, that the entry points have either to the other entry points or to some other designated set of points in the network. In earlier systems, such as Quillian's original one [Quillian 68], the activation sought to find any path between the entry points, and then to explain the path, thereby giving a description of the relationship of the points.

Each of the words used as an entry point is marked. If and when the DKE looks for additional terms, only those that have not been examined previously will be used as new entry points. Before the DKE further examines the domain knowledge for additional concepts , there is expected to be a search or some browsing activity providing some concept that can be used.

The other major activity of the DKE is updating of the user's domain knowledge model. This is done when the user is initially developing his query and when he is examining documents. During each of these activities, depending on the user model, the DKE will allow the user to enter different kinds of domain knowledge. A novice user is allowed to pick out phrases of interest, whereas an expert can enter all types.

Figures 4.6 through 4.9 show the interaction that the user has with the system to enter some domain knowledge. The actual operation of the system works in the following way. As the user selects the relationships, the full words are placed in list structures that reflect the relationships and words. These list structures are kept in fields of a record that is

part of every record that holds the content of what is displayed on the interface. This record is defined as follows.

```
(Defstruct
   (Common-Content-Info
       (:Conc-Name CCI-)
       (:Predicate CCI?))
   Display ;pointer to the display structure
   Relevant-Display
   Relevant-Display-Content
   DK-Display ; holder for DK acquisition displays
   DK-Type ;type of DK being acquired
   DK-Words ;words or phrases being related
   DK-New-Word ;holder for a word entered from keyboard
   DK-Phrase ;holder for phrase being built as part of
             ;another relationship
   Expanded ;a flag to indicate whether this node has
            ;been expanded in browsing
```

When a user selects a relationship to enter, the first thing that happens is a window is generated to display what the user is entering; the pointer to that display is held in DK-Display. This field may actually contain up to three display pointers, depending on what is being entered. This is the case of the situation shown in figure 4.7. When the user selects a word from the display, it is first pushed into DK-Word, and then displayed in the DK entry display that is either at the top or the bottom of the main document, concept, or query display. When `Entry OK` is selected the whole DK entry, for example, `(related distributed parallel)`, is pushed onto the list kept on `Relevant-Display-Content`. If `Cancel` is selected, the fields containing the information are cleared and the display is removed.

When `Done` is selected for a document or a concept the connections are sent from the interface manager to the system as a list of connections, as in:

```
((related (phrase concurrent processes)
```

```
                (phrase parallel processes))
    <other connections>)
```

After the domain knowledge is received by the DKE, each of the component phrases is stemmed and the words are entered into the domain knowledge with the original phrase put into the phrase field of the individual word record. Then, the phrases are put into the user's domain knowledge with the single words that make them up put into the phrase field of the record and the related phrase in the related field.

The only information that goes into the model is that which the user enters while examining a document or the query. The concepts that the user approves while he is evaluating the suggestions that the DKE makes go only into the request model. There is no migration of information from the global domain knowledge to the user's domain knowledge. The purpose of the user's model is to record the relationships he makes that are different from those in the global model.

4.2.6.5 Search Controller

The purpose of the search controller is to select the search technique or techniques that are appropriate given the user's interest in precision or recall, and the history of the session. This is an innovative feature of I^3R, since most systems are limited to a single search strategy. The system has two basic kinds of searches at its disposal, a probabilistic search based on the term independence model, and a cluster search. The cluster search in this system has a two variations depending on the links used to define the clusters. The primary cluster search uses nearest neighbor links; the other variation uses citation links. Other variations of the cluster search based on bibliographic coupling links, or cocitation links could be included, as well as searches based on different retrieval models such as the vector space model with the cosine correlation [Salton 68] or the extended Boolean model [Salton 83].

As has been mentioned previously, there is as yet no way to select retrieval strategies based solely on attributes of the query [Croft 84]. Therefore, other kinds of information must be used. This is the prime motivation for determining the user's interest in recall or precision. Besides the user's search interest, attributes of the search techniques should be taken into consideration, but there is little information to work with in relation to this. One piece of information is that cluster searches tend to retrieve different sets of documents than probabilistic searches for the same query [Croft 80] (see figure 2.1 and scenario three in chapter 6).

In order to make use of the available information heuristics have been developed to select what strategy to use in a particular situation. In developing these heuristics, knowledge of how human intermediaries perform searches is of no use, since their experience lies in manipulating Boolean queries in commercial systems. The heuristics of the search controller can be summarized as follows.

- Initial Searches
    - If the user is precision oriented, use a probabilistic search. The motivation for this is to use a well test search method as the basic technique.
    - If the user is recall oriented, initiate both a probabilistic and a citation cluster search. This will give the user the greatest number of documents to choose from. A large volume of documents is the goal in a recall oriented search.
- Subsequent Searches
    - If the previous search failed to retrieve more than two relevant documents, use the other search technique. The search had very low precision, so try another technique.
    - If the previous search was successful, more than 2 relevant documents, use it again with the modified request model. If a search has achieved a minimally acceptable performance, stick with it.
    - If two searches in a row fail to retrieve more than 2 relevant documents, signal this failure. The control expert will then put the system in a state where the browsing expert has priority and will suggest that the user browse. If all of the search techniques fail, then let the user do

some of the searching manually. This will cause the request model to be altered, so that the searches may work better later.

The search controller maintains a record of each search, so that it can keep track of all the documents that it has retrieved, the kind of search used, and the precision of the search.

The searches are implemented by means of a C program that performs all of the searches currently implemented. The search controller gets from the request model the term numbers, the term's collection frequency and occurrence in relevant documents, relevant phrases, and documents that have already been seen. This information, as well as what kind of search to perform is passed to the search program.

4.2.7 Interface Manager

The interface to the system is handled by an interface manager that operates independently from the rest of the system. The interface manager communicates with the rest of the system by placing messages on and reading messages from two places on the short term memory. It is primarily window oriented, and the following are the kinds of windows it supports:

- **System Messages Window** — displays textual messages from the system to the user.
- **Choices Window** — display choices to the user which may be selected by the mouse.
- **Query Entry Window** — a window produced by a text editor that allows the user to enter a query in a variety of different ways. These different ways are defined by forms that the user completes.
- **Menus** — there are two types, window and content. Window menus allow the user to manipulate a window by scrolling the contents, suspending it, etc. Content menus let the user make choices about the content of a window. For example, getting the bibliography of a document, or selecting the domain knowledge link that he wishes to use to connect two concepts.
- **Text Entry Windows** — these allow the user to enter character strings. Used in acquiring domain knowledge.

- **Document Window** — Shows the document title, author, abstract, and reference. Also used for displaying the query after it has been entered.
- **Document List Window** — shows the titles of documents. Used for search results, bibliography lists, and citation lists.
- **Concept Window** — displays the concept as well as all of the other words it is connected to.
- **Concept List Window** — displays concepts chosen by the domain knowledge expert to the user for approval.
- **Context Map** — Gives a graphic "road map" of where the user has been while browsing.
- **Neighborhood Map** — Shows the immediate neighborhood of a node in the Context Map.

The content of the windows is controlled by the experts via the content of the messages that they send. For example, depending on the UMB's categorization of the user, different choices for domain knowledge entry are made available by the DKE. A domain novice user is only allowed to select phrases he is interested in, whereas an expert can enter domain knowledge using any of the links and can enter text as well.

The interface manager is basically composed of two parts, one part to receive messages from the experts and display appropriate information, and another to receive information from the end user. The output portion of the IM runs as it is called from the main part of the system. This is done in several places. The first is during the part of the cycle when the Control Expert is determining what state the system is in. This allows it to pass any control information to the IM. The second place that the IM is called is after all the rules of the experts have been executed. This is when information that was the result of rule execution is passed to the IM for action. The messages passed to and from the IM have the following format:

```
(Defstruct (Inter-Process-Message
                (:Predicate IPM?)
                (:Conc-Name IPM-)
   (Message-No 0 :Type Integer)
   Msg-Id
```

```
Choices
Value-Type ;Choice-List, Done, Concepts, Text, etc.
Values)
```

These are placed in a list on an blackboard place named `To-IM`, and they are received back in a list on a place called `From-IM`. The interface basically takes each message from the place, `To-IM`, each cycle, decodes it and performs the action specified by the message-id. These actions consist of displaying information or changing parts of the interface, such as adding or removing choices from a menu, for example. Some of these actions provide a response back to the system and some do not. The system looks at the `From-IM` place on every cycle before the experts determine what rules to place on the agenda, so that they can act on information collected from the user.

The input part of the interface is primarily interrupt or event driven. Each mouse click or keystroke, depending on the window in which it occurs, causes an interrupt routine to be executed. These routines execute at a higher priority than the rest of the system, so the response to the user is fast. The only exception to this is the query input editor, which runs as a separate process while the rest of the system is suspended.

4.3 Implementation of the Blackboard System

Many of the design decisions in the way that the blackboard system was implemented were based on building the system inside of a single LISP image. More specifically, each expert does not run as a separate process communicating with a process that manages the blackboard. There are an number of reasons for this. As mentioned previously, on a DEC VAX minicomputer running the VMS operating system, a LISP process consumes a significant amount of resources. Running more than one process slows down the operation of the system considerably.

Descriptions of a blackboard architecture often make the statement that knowledge sources *look* for changes on the blackboard. This implies that they *actively* examine the

blackboard when they are not performing some other computation, and suggests a polling implementation. Polling in this situation would be a very inefficient implementation. A more accurate description would be that the knowledge sources *respond to changes* on the blackboard. This suggests an interrupt-driven or message-sending implementation, where the knowledge sources are idle until something of interest happens. The difficulty of this style of implementation is that the process that monitors the blackboard must know what interests the various knowledge sources. In the Hearsay II system, this information was simply that a hypothesis was posted on the level that a knowledge source examines. What to do with the notification was up to the KS. I^3R follows the message-sending view and provides the underlying process that runs the system with the information about what interests I^3R experts

4.3.1 Rule organization and execution

Because of the requirements to support explanation and incremental development, each expert in I^3R is implemented as a separate rule system. Rules can be interpreted as a condition/action pair or an antecedent/consequent pair. There is a subtle difference between the two interpretations. The first is an operational view and corresponds to the "`If-Then`" construct found in most programming languages.

The other interpretation is a logical view, and means that the antecedent and the consequent are related by modus ponens. This interpretation means that the rule is a specification that is interpreted by an underlying mechanism. The rules can be used in both directions. An example is the BNF specification of a programming language. If the specification is used as a generator, starting with the start symbol and working toward the terminals it is a top-down interpretation. If it used as a recognizer, working from the terminal symbols to the start symbol, it is bottom up interpretation.

In I^3R the rules were implemented using the condition/action interpretation. The primary reasons for this choice were the following:

1. Rules represent the high level control structure in each expert, where the actions are performed by algorithmic processes; for example, stemming or search. It was not intended to implement the entire operation of each expert using rules. In some cases, it would have been grossly inefficient to do so; for example, the search processes.

2. Ease of implementation with regard to the blackboard. Since it was desired to have more than one rule fire, if possible, during a major cycle of the system, use of commercially available systems was infeasible.

4.3.1.1 Rule form

The form of a rule is relatively simple, it is a 4-tuple consisting of:

```
(<expert name> <rule#> (<conditions>) (<actions>))
```

The expert name and the number simply indicate where the rule belongs. The actions and conditions are the meat of the rule. The conditions are a list of 4-tuples and the actions are a list of triples. The condition 4-tuple is:

```
(<BB-place> <action-name> <predicate-name> <arguments>).
```

The essential meaning of a condition is that if a certain action is taken on a specified place, then check the blackboard place with the given predicate and arguments. The action triple is:

```
(<BB-place> <action-name> <arguments>).
```

Similarly, this means perform the specified action on the BB-place with the specified arguments.

The actions and the place names provide the mechanism for notifying experts when something has happened that is of interest. At each blackboard place a list is kept, indexed by action names, of the actions that are interesting to particular experts. This information is extracted from the condition part when a new rule is added to the system. This list is accessed by the rule execution code when an action is performed by using the `bb-place` and `action-name` part of the condition. For example, the bb-place called `*Domain-`

`Knowledge*` would have on its list of actions an action called `Add-DK`. This action is interesting to the request model builder, since the new domain knowledge may contain words that are not already in the request model. When `Add-DK` was performed a message would be sent to the RMB of the form `(Add-DK *Domain-Knowledge*)`. This would cause all the rules of the RMB that have this action/place pair in any one of its conditions to become a candidate rule for selection.

4.3.1.2 Rule processing

When the rules are stored in the system for an expert, they are processed in the following way. First, each rule is stored in an array at the location that corresponds to its rule number. Then the condition part is broken up into individual conditions. These are used to form an inverted list of action-place indexes that point to the rules that contain them. To illustrate this, an expert, E1, has the following three rules:

```
(E1 1 ((P1 A1 <predicate><args>)
       (P2 A2 <predicate><args>)) (<action part>))
(E1 2 ((P1 A1 <predicate><args>)
       (P2 A3 <predicate><args>)) (<action part>))
(E1 3 ((P1 A1 <predicate><args>)
       (P3 A3 <predicate><args>)) (<action part>)).
```

When these rules are read in,each one is put in an array indexed by its rule number. The condition parts are then broken up into the following triples:

```
(P1 A1 1) (P2 A2 1) (P1 A1 2) (P2 A3 2) (P1 A1 3)
(P3 A3 3)
```

These are combined into lists:

```
((P1 A1) (1 2 3)) ((P2 A2) (1)) ((P2 A3) (2))
((P3 A3 (3))
```

These lists are put into a hash table, keyed by the place/action pair. These place/action pairs are also used to key operations on places back to the experts that are interested in them. For example place P1 would have on its action list the following:

```
((A1 (E1 <other experts>)) (An (<experts>))...)
```

So, when an action is performed on a place, the experts that have rules that have that place/action combination in their condition part can be notified, by means of a message, of that event.

When an expert is allowed to operate by the control expert, it looks at its list of incoming messages notifying it that some action has occurred on a place that it is interested in. For each one of these messages, the expert retrieves all of the rules, by means of the index table, that have the place/action pair in their condition part. The system forms a triple that consists of the rule number, the number of conditions it has in its condition part, and the number of messages that refer to it. After all the messages have been processed, the list of triples is sorted; the major criterion being the number of conditions in the condition part, and the first minor criterion being the number of messages that refer to the rule; the second is the rule number. The system then goes down the list and takes the first rule whose condition part is true and places it on the agenda for potential execution during that cycle.

This ordering is how the system performs conflict resolution. It is based on conflict resolution heuristics that give priority to the most constrained rule, the rule that is responding to the most activity in the system, and the numbering of the rules.

The actions and the predicates are implemented as Common Lisp functions. Some of these may call-out to programs written in C.

4.3.1.3 System operation

A detailed description of the system operation is the following.

1. The scheduler determines what state it is in, which specifies what experts can operate during this cycle of the system. This is done in the same way that the rest of the experts determine what rules will be eligible for execution. The stopping criterion for the scheduler is the attainment of a leaf state.

2. Any messages from the interface manager are put in the short term memory and appropriate action-place pairs are sent to the experts.

3. Each expert that is allowed to operate by the control expert looks on its list of messages that contains action-place pairs that correspond to condition parts of its rules.

4. Every rule that is indexed by an action-place pair is retrieved and is ordered by the rule conflict resolution criteria.

5. The rules are then checked in order to find the first rule that is true. This rule is placed on the agenda for execution.

6. Once the agenda is set, it is executed in the order determined by the scheduler in the following way.

    a. The first rule is executed.

        i. Any BB-place that has been altered is put on a list of altered places.

        ii. For each place that is altered a message consisting of an place/action pair is sent to each expert "interested" in that place and action.

    b. The rest of the rules on the agenda are executed in order.

        i. A rule is checked to see if any of the altered places are in its condition part. If so, then the condition part is rechecked to see if it is valid.

        ii. If it is valid, then it is executed in the same fashion as the first rule (steps 6.a.i & 6.a.2).

        iii. Get the next rule.

        iv. If any more rules, go to step 6.b.i.

7. Any messages for the interface manager are sent .

8. Goto Step 1.

4.3.2 Interface Manager

The interface manager (IM) represents a significant portion of the code that implements I^3R. In its initial design, it was to run as a process separate from the main part of I^3R. This proved infeasible, since running two Lisp processes on a single processor severely degrades performance. It was desired to implement the IM in Lisp to retain the flexibility and superior development environment that Lisp provides.

With these constraints in mind, the structure of the interface manager can now be described. Figure 4.15 presents a schematic view of the IM structure. The `*Display-Table*` holds pointers to all of the records that define a major window on the screen, which are displays such as documents, concepts search results, the context map, etc. Neither menus nor domain knowledge entry windows are considered major windows and are associated with a particular major window. The first six records hold pointers to the windows that the system always has, which are the system messages window, the questions window, the neighborhood map, the context map, query display, and the relevant documents display.

The other displays are lists of concepts, lists of documents, single concepts, or single documents. Each display is represented by a record that holds information related to the VMS UIS (User Interface System) display and a pointer to the content of the display. The content is either an array of pointers to documents or concepts, or is a single document or concept. The documents and concepts items are kept in hash tables, the concepts are kept in their respective global or user domain knowledge hash tables, and the documents are kept in a document hash table, which is keyed by document number.

Each display is represented by a record structure that keeps the size of the window, the associated interrupt functions, pointers to the menu displays, the coordinates of the object on the browsing map, and a pointer to the content of the display. The content is either a document, a document-list, a concept, or a concept list. The system messages window have no associated content and the choices displays are a special case of a menu display. The interface manager data organization is summarized in Figure 4.15.

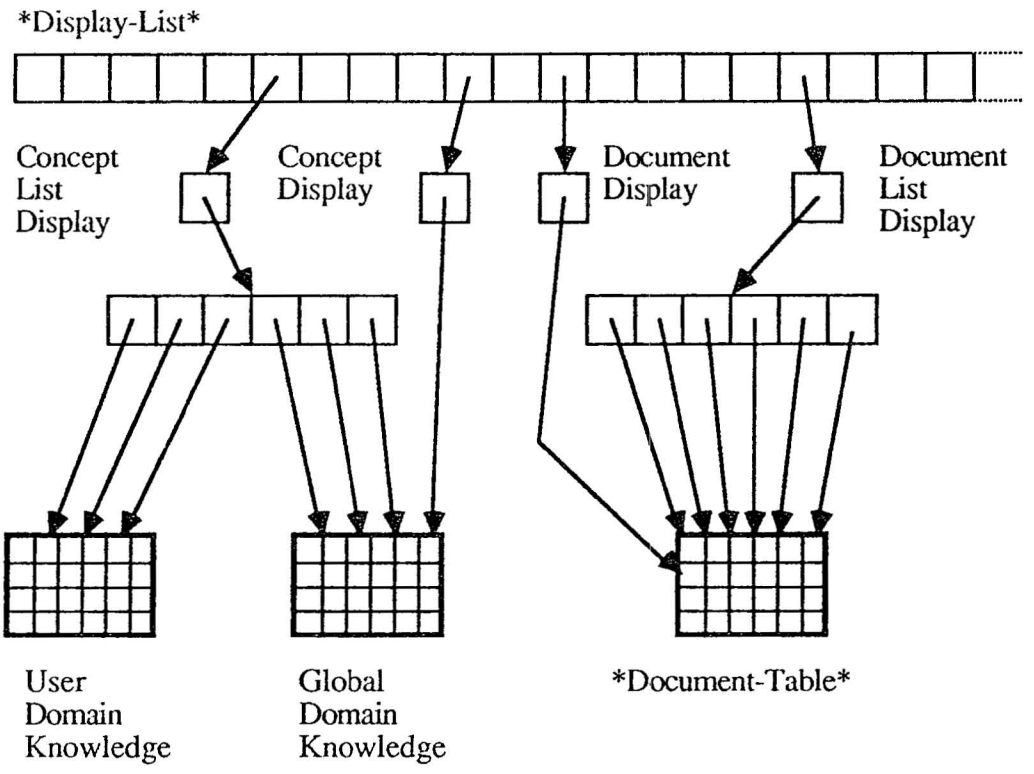

Figure 4.15: Organization of Interface Manager data.

4.4 Summary

In this chapter, an architecture for an intelligent interface for information retrieval has been presented. It meets the requirements established in the previous chapter in the following way. New large scale functions can be added easily by means of adding new experts to the system. This can be accomplished simply by adding the expert's name to the priority lists of the control expert in the appropriate states, and writing the rules. Each expert can be incrementally developed by adding or changing rules. Rules also make explicit the decision criteria, so that the operation of the system can be explained. The system provides different ways for the user to enter his query and allows him refine his query by having a flexible interface that supports the entry of domain knowledge and browsing. The system can use multiple search strategies to find candidate relevant documents. Finally the system allows the user to take complete control of a session by browsing.

CHAPTER 5

BROWSING EXPERT

5.1 Introduction

In this chapter, the browsing expert is examined. Browsing is an alternative method for both query refinement and information search. It gives the user direct access to the concept/document knowledge so that he may explore to find the information he needs. The end user is assisted in the browsing process by recommendations made by the expert and by graphical displays that provide the user with context, showing where he has been and the immediate neighborhood of his current location. Browsing, in concert with the automatic construction of the request model and multiple formal search techniques provide a synergistic retrieval environment that overcomes the disadvantages of using browsing as the sole means of information search.

5.2 Definition

Browsing is an informal or heuristic search through a well connected collection of records in order to find information relevant to one's need. The searcher evaluates the information currently being displayed to determine its value relative to the information he is seeking. Once this evaluation is made, the user then selects the next item to display and evaluate. Browsing is also a feedback process, but it differs from traditional relevance feedback in two major ways. The first difference is granularity; the user only examines one item at a time evaluating its relevance and selecting another item for view, instead of having to examine the results of an entire search. The second difference is the locus of control; it is the user that determines the items to be examined rather than the system.

5.2.1 Advantages

Bates [Bates 86] points out the advantages of browsing in the context of term selection by showing how it takes advantage of two cognitive capabilities. The first one is the greater ability to recognize what is wanted over being able to describe it. This concept is evidenced in the production artist renditions of criminals by means of "paste up" facial sections, where an individual picks out facial features that they recognize of the individual being sought. The second capability is being able to skim or perceive at a glance. This allows a searcher to evaluate rapidly a large amount of material, determining what is useful in it. Browsing makes use of these abilities by showing the user examples of information that match his current model, as expressed to the interface, for evaluation. A system that allows a user to browse and to do so quickly may provide information that the user wants and may not have been able to describe, and quite possibly all the information that the user needs. In this way, browsing addresses the content specification problem of query formulation described in chapter 2.

Besides being an alternative to a formal or parameterized search, browsing serves to acquaint a user, unfamiliar with a domain, with the structure of its information. This tutorial use helps those users that cannot find the right words to express their information need or that do not know how terms in a particular domain are used. This kind of browsing is dependent on having a high quality thesaurus, which may or may not be available. Some domains, like medical science, have very well defined structures that are embodied in thesauri such as MeSH (Medical Subject Headings). Other domains do not have such a readily available source of knowledge, but domain knowledge collected from domain experts can be a source for this kind of information.

5.2.2 Use in Other Systems

Because of these advantages, a number of systems use browsing as a means of finding information. These systems include hypertext and text retrieval systems, database systems [Motro 85], and object oriented programming systems [Goldberg 1983]. In the hypertext and text retrieval systems browsing has been used for both query formulation assistance and finding document information directly. Some prototype systems developed to tackle the query formulation problem are CANSEARCH [Pollitt 83], CALIBAN [Frei 83], and CoalSORT [Monarch 87]. These systems take an existing classification system and automate it to give a user online access in order to select terms for a query. The structure of the knowledge is organized as frames with various kinds of relationships, depending on the field to which the system was applied, connecting them. In both CALIBAN and CoalSORT the user is required to manually construct a query. CANSEARCH is oriented directly at producing a query. Consequently, it leads the user to a greater extent than the others to specify certain types of information. As the user makes evaluations and selections, the system includes them in the developing query.

There have been a number of text oriented systems that use browsing as a method of search. An early system is ZOG [McCracken 84], which is designed to be a general purpose human-computer interface. The fundamental mode of operation is menu selection with the basic unit of information being a screen-full of text called a frame. The information in a ZOG system is handbuilt using the built-in editor. The main organization is a tree, but there is no restriction on how frames may be linked. Search is done by traversing the frames until one finds the information one desires. ZOG's advantages are that it is easy to use, requires very little training, and is fast.

BROWSE [Palay 81] is a system built on top of ZOG and more oriented to document retrieval. The set of frames combined document abstracts with a concept classification hierarchy, author, and journal information. It overcomes some of the limitations of

ZOG by including a partial map of the frame structure. This map is limited to only the concept hierarchy. A further improvement is the addition of a search capability. At any time the user could make a selection to perform a parameterized search of the documents. However, the query for the search still has to be manually constructed by the user.

Browsing is also found in the hypertext or dynabook systems [Kay 77, Weyer 82, Trigg 83]. These kinds of systems are characterized by text units of approximately paragraph or page length connected by various kinds of links. One class of links organizes the text units hierarchically into subsection, section, chapters, and papers or books. Other kinds of links provide citation referencing and editorial commentary. TEXTNET [Trigg 83] maintains over 50 kinds of links. Once the user has entered the system, he is free to meander through the network examining the text. These systems also provide a sophisticated user interface, giving the user a number of ways to get information about the document that he is in.

A major problem with these systems is that browsing is, generally, the *only* way to find information in them. Either no facility or only a rudimentary one is provided to perform a search. Searching through that many units of text by browsing only would be a formidable task. In one case, the dynamic book [Weyer 82], an index structure similar to that of a book is provided so that the user can "jump" from one place to another. However, this prototype was constructed from a history text, and therefore, the amount of information and the subject matter was constrained.

THOMAS [Oddy 77] is an interface program that employs a different type of browsing. In this work, the system does not rely on a pre-existing complex, highly connected database of documents. Instead, a model is built by the system of the user's interest as the user evaluates the information presented to him. The model is different from the kind built by CANSEARCH in that it is domain independent. This aspect of THOMAS is

significant since it relieves the user of having to manually construct a query, other than the initial few terms.

Another difference in the operation of THOMAS from the previous characterization of browsing is that the system takes the initiative, after the user entered a few initial terms, in selecting what to show the user. Even though this would seem constraining, the system is quite flexible in its interaction. It can determine when the user is not making progress, and ask him to reevaluate previously seen abstracts. It is similar to the ZOG systems in that it presents the user with only one item of information, a document abstract, at a time for evaluation. However, the user can indicate that individual elements of the abstract are relevant, rather that only being able to evaluate the whole item.

5.2.3 Disadvantages

The disadvantages of browsing as a means of search are that given the complexity of the concept/document database it is very easy to get lost. This is one of the major problems of ZOG-like or hypertext systems. Once the user is deep into the database, he may have forgotten how he got there and has only the current frame for context. This results from these systems being "memoryless," and providing only a simple interface and a structure. Another disadvantage is that browsing is labor intensive. The user may have to examine many pieces of information before finding anything that is relevant. In I^3R this is overcome by the availability of formal search techniques that can be used after the user has judged a number of documents or concepts relevant.

5.2.4 User Heuristics

How does a user browse? How does he get started, and how does he determine what to view? In order to determine what item will be displayed next for evaluation, the

searcher uses heuristics. Some examples of browsing heuristics that a searcher might use in a document retrieval system are:

1. If the current document is interesting:
    a. What else has been written by its authors?
    b. Are any of its references interesting?
    c. Are any of the documents that reference it interesting?
    d Are any of the documents in the same journal issue, conference proceedings, etc. interesting?
    e. Are there any documents that are very similar to it in the database?
2. If the current term is interesting:
    a. Does it have any synonyms, narrower terms, etc.?
    b. What documents is it used in?

All of these heuristics depend on the kind of links maintained by the system. For example, if references are not maintained, then heuristics 1b and 1c cannot be used. The richer the set of links, the more ways that the user can move through the database. By having a rich set of links, the system is responsible for helping the user understand what the links mean, how they might be used, and where he is in the network formed by them.

Browsing can also be seen as a form of constrained spreading activation in a semantic net [Cohen 87]. The heuristics, in this case, constrain the choice of paths that the user selects from. Each path is evidence that a document or concept is related to another document or concept. For example, if two documents share a number of very common terms, these documents are likely to be related only on a very general level. If one of the documents cites the other, the likelihood of them being related is greater.

Since the concept/document database has a large variety of links, the user has many possible ways to navigate through the information. Furthermore, because of the way that the concept knowledge is fused with the document knowledge, the user is given much more latitude to find information of interest. The user can explore the structure of the do-

main knowledge in a variety of ways. For example, he can look at all the phrases that a particular word is used in, as well as how the concepts are used in the documents. This is in contrast to the BROWSE system [Palay 81] where the user is restricted to the tree structure and the few cross reference links of the hand-coded domain knowledge.

5.3 Browsing Operation

The Browsing Expert (BE) provides assistance to the user in three major ways. First, it makes recommendations about nodes connected to the current node that it considers likely to be useful. Second it remembers where the user has been, so that he can retrace his steps to return to interesting nodes that he has seen in order to pursue different paths. Third, in concert with the Interface Manager, it provides visual context, so that the user can avoid getting lost in the complex and potentially confusing structure of the database.

The BE, in a manner similar to the Domain Knowledge Expert, acts in an advisory capacity, allowing the user to be the final judge of the usefulness of the information displayed. The advice is given to guide and not restrict the user options; the user can always ignore the advice of the BE, and elect to go off in a direction of his own choosing.

5.3.1 Browsing Interface

The advice given by the BE is reflected by what is displayed on the browsing maps. These maps are generated by the IM and consist of the neighborhood map and the context map. The neighborhood map (figure 5.1) gives a picture of the nodes that are immediately adjacent to the current node of interest; the context map (figure 5.2) gives the view of a larger area around the node of interest, so that the user can get an idea of the path that he has been pursuing. Both maps consist of nodes representing concepts, documents, lists of documents, and connectors connected by links marked as to their type or frequency (in the case of document to concept links). Nodes are filled with different patterns indicating

whether they have been visited, recommended, or judged relevant. The shape of the node indicates what kind of item it represents; circles represent documents, squares represent lists of documents, octagons represent concepts, diamonds represent connectors, and over sized boxes represent reminders. Reminders are markings made by the user on the maps to indicate some node that he would like to come back to and examine at another time.

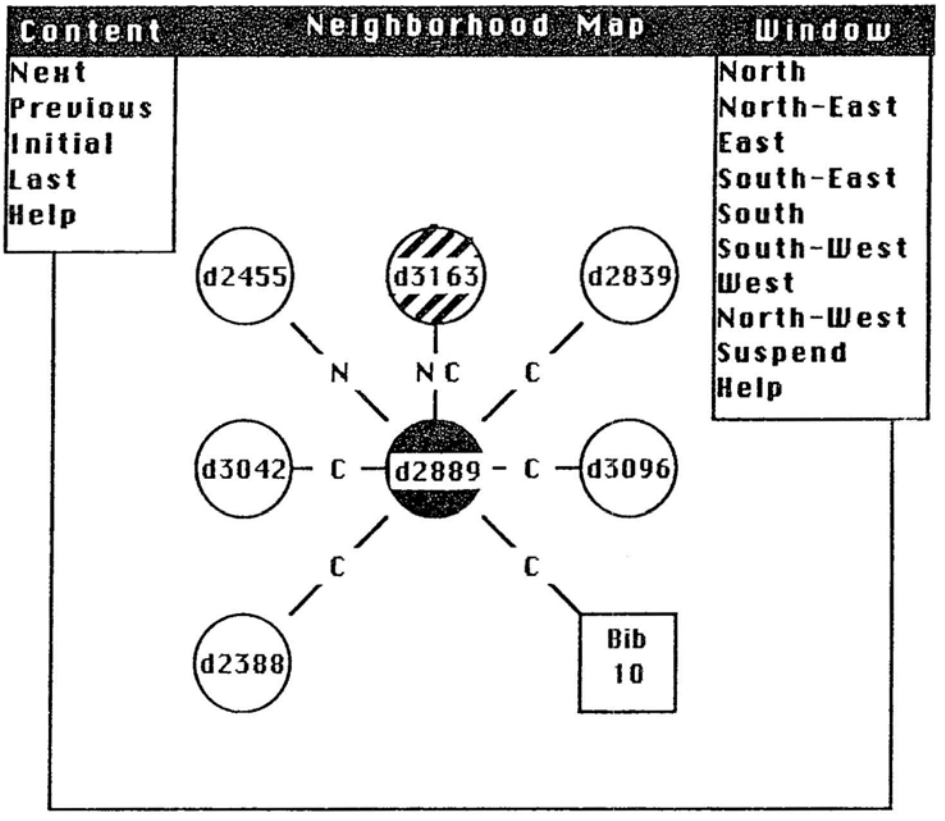

Figure 5.1: Sample Neighborhood Map. Markings on the links indicate the kind of link: C = Citation, N = Nearest Neighbor.

When the user views the node again, the reminder is removed. This gives a cue to the user if he is scrolling around on the maps that there is something that at one time caught his eye.

The possible number of links that can emanate from any node, representing a concept, is potentially large. A moderately frequent single word concept (also called a "term") may be found in as many as fifty documents, even in the CACM test collection. Besides this, a concept can be linked to many other concepts by the different links. For example, a

term such as "algorithm" in a computer science document collection could be a part of many phrases.

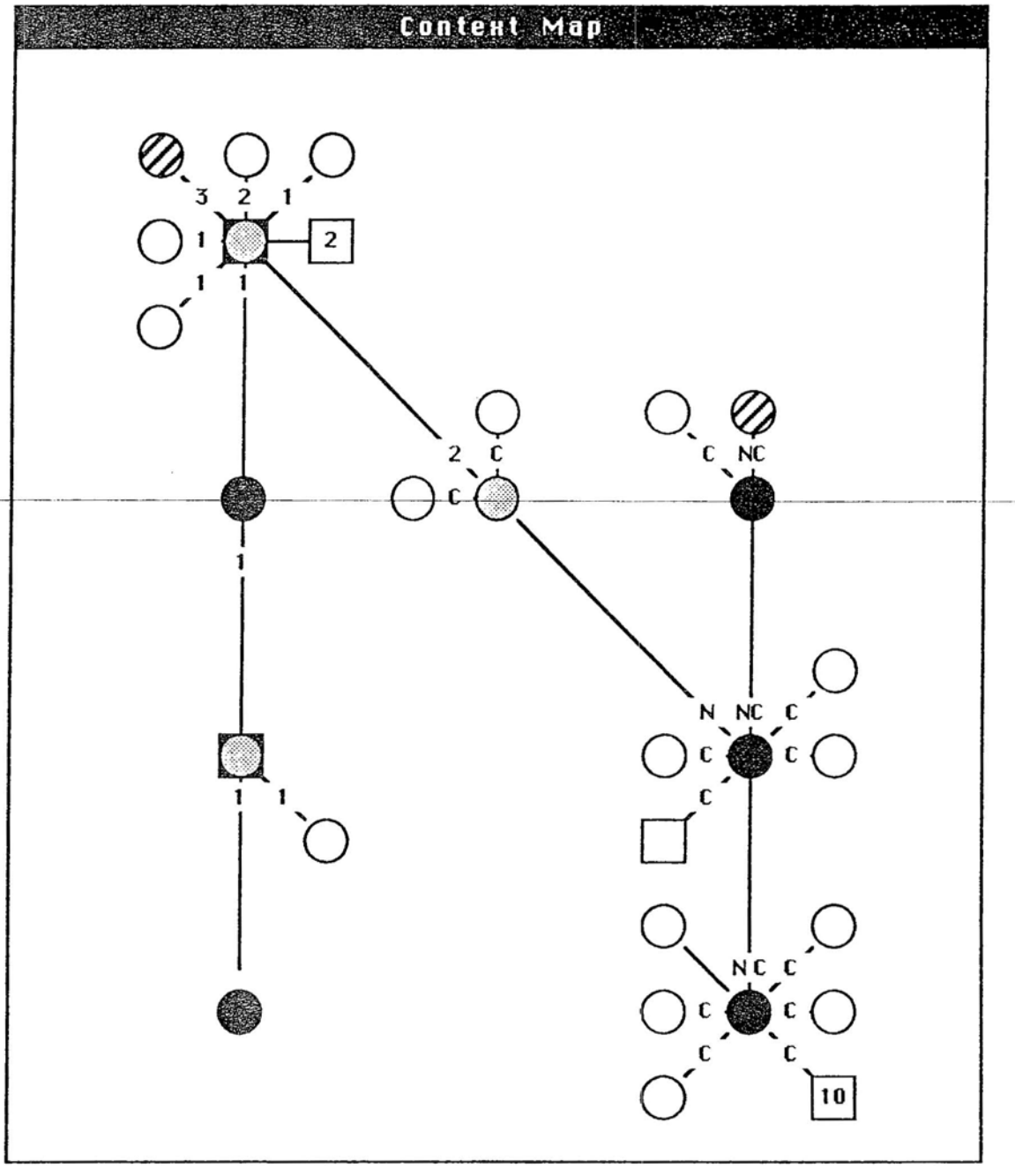

Figure 5.2: Sample Context Map.

The situation for documents is similar. With reference links, a document may have anywhere from just a few links, in the case of a short correspondence with one or two