# Architecture of an Expert System for Composite Document Analysis, Representation, and Retrieval

**Edward A. Fox and Robert K. France**
*Department of Computer Science, Virginia Tech*

## ABSTRACT

*The CODER (COmposite Document Expert/extended/effective Retrieval) project is a multi-year effort to investigate how best to apply artificial intelligence methods to increase the effectiveness of information retrieval systems handling collections of composite documents. To ensure system adaptability and to allow controlled experimentation, CODER has been designed as a distributed expert system. The use of individually tailored specialist experts, coupled with standardized blackboard modules for communication and control and external knowledge bases for maintenance of factual world knowledge, allows for quick prototyping, incremental development, and flexibility under change. The system as a whole is being implemented under UNIX as a set of MU-Prolog and C modules communicating through pipes and TCP/IP sockets.*

KEYWORDS: ***information retrieval, artificial intelligence, distributed expert system, knowledge bases, blackboard architecture, lexicon construction***

## INTRODUCTION

As the world's pool of information, particularly of machine-readable text, rapidly expands, it becomes increasingly necessary to engage the help of

Project funded in part by grants from the National Science Foundation (IST-8418877), the Virginia Center for Innovative Technology (INF-85-016), and by an AT&T equipment contribution. An earlier version of this paper was presented at the Third Annual USC Computer Science Symposium; Knowledge-Based Systems: Theory and Applications, Columbia, South Carolina, March 31–April 1, 1986.

*Address correspondence to Edward A. Fox, Department of Computer Science, Virginia Tech, Blacksburg, Virginia 24061.*

International Journal of Approximate Reasoning 1987; 1:151–175

52 Vanderbilt Ave., New York, NY 10017 0888-613X/87/$3.50

computers to control and manipulate it. Initial attempts at computer-aided information storage and retrieval (ISR) made centralized databases accessible to a large community of users [1] but focused principally on performance and have achieved only moderate levels of effectiveness [2]. Today, end users often prefer to search for themselves [3], using gateways, front ends, intermediaries, and interfaces [4], or aided by powerful microcomputers attached to optical disk stores. These end users need more effective and adaptable tools such as have been investigated by the research community [5]. CODER (*CO*mposite *D*ocument *E*xpert/extended/effective *R*etrieval) is a research system intended to address these needs through the mechanisms of knowledge-based and goal-directed artificial intelligence (AI) techniques.

## Problem Description

The CODER system is aimed at investigating issues of meaning representation and the effective matching of user needs with relevant (passages of) documents. Although the SMART system has been evolving for more than 25 years with similar objectives [6], recent experience with reimplementing a modern version [7] and with using its latest form [8] suggests than an AI-based architecture would make further development and experimentation much easier. Questions in key subject areas that could then be studied (along with references to related work) include:

COMPOSITE DOCUMENTS

1. Can composite documents that include text, factual information, and references to other documents [9] be effectively analyzed [10] so that entire documents or appropriately sized passages [11] can be retrieved?
2. Can document analysis and modeling improve with findings about abstract document structure [12], message composition [13], office modeling of documents and other objects [14, 15], document formatting [16], and related standards [17]?

EFFECTIVE RETRIEVAL

3. Can effective retrieval methods suitable for the growing number of full text databases [18] be developed [2] using automatic techniques [19]?
4. Because the overlap between results of different retrieval methods is small [20], can overall effectiveness increase by use of several? Can retrieval systems be tailored to different users' understandings of relevance?
5. Can rule-based processing allow more effective combination of bibliographic information about documents [21] with other factual and content components than has been achieved with statistically based approaches [22]?

6. Can a heuristic approach allow selection for each query of the most appropriate search strategy (e.g., choice between clustered versus inverted file searching, according to findings in [23]), the best retrieval approach (e.g., extended Boolean [24] versus vector space [25] versus probabilistic [26, 27]), and the fastest method for identifying good documents [28]?

AI METHODS

7. Is the logic programming paradigm in general [29, 30] and the Prolog language in particular [31] mature enough to use for natural language analysis [32], the rule-based processing commonly used in expert system development [33], and general AI programming [34] in a large, complex system?
8. Is the blackboard model [35] of a distributed expert information-providing mechanism [36] suitable for an ISR system?

KNOWLEDGE REPRESENTATION

9. In light of the many knowledge representation schemes suggested for information retrieval [37], can computationally tractable ones [38] be developed [39]?
10. In particular, are frames [40] useful in representation and reasoning [41] about document content in a way that can aid information retrieval [42]?
11. Can temporal data be suitably represented and used [43] in retrieval?

COMPUTATIONAL LINGUISTICS

12. Can linguistic analysis aid information retrieval [44], not only through improved analysis of queries [45] but also in document analysis [46] through skimming [47, 48] or far more robust and detailed analysis [49, 50] of more than a constrained sublanguage [51]?
13. Can machine-readable dictionaries [52] support expansion of the small lexicons used to date in text analysis systems [53]?

HUMAN-COMPUTER INTERFACE

14. Does current knowledge about human-computer interaction [54] and information retrieval [55] allow development of interfaces that can adapt to individual needs and preferences?
15. Can information retrieval systems satisfy some of the needs for tutoring systems by making books [56], encyclopedias [57], and other reference works more accessible?
16. Does a graphics-based interface [58] where problem formulation, query construction, term expansion, feedback, browsing, and profile-based filtering are all interwoven in a highly interactive human-computer dialogue [59] lead to more effective and pleasant retrieval?

### Related Work

Several research investigations are related to the CODER project. The earliest use of expert system methods in the retrieval area was probably in the CONIT system [60]. The closest contemporary effort is development of I³R by Thompson and Croft [61]. I³R differs by being coded as a monolithic system in Lisp, interfaced with a database system, aimed to explore retrieval methods that access a statistically analyzed document collection, and implemented using fewer but more complex experts. Yet like CODER, I³R is built around a blackboard that coordinates retrieval processing. The RUBRIC system, which uses a rule-based approach whereby queries become small knowledge bases [62], incorporates a variety of techniques for combining evidence [63] that have been included in CODER. TOPIC is also of interest, as it attempts to parse documents to condense their content and identify important concepts [46]. The more ambitious Project Minstrel, applying retrieval and AI methods of office modeling, is based on a comprehensive knowledge representation facility [15].

Although CODER incorporates many ideas from other research efforts, it is unique in its aim and scope. CODER provides a unified paradigm for the entire process of information storage, representation, and retrieval based on a tailor-made encoding of knowledge (see [64] and the section on knowledge administration below) and a flexible architecture designed to support the storage and manipulation of that knowledge. This article concentrates on the architectural issue; the interested reader is referred to France and Fox [64] for more details on how knowledge is used.

## ARCHITECTURE

CODER is organized as an integrated system for document entry, analysis, storage, retrieval, and display. It should be adaptable as a standalone system, as a server for interactive or batch entry of documents or queries, or as an intelligent intermediary to another database system. The following discussion relates to the most comprehensive case: standalone implementation.

In keeping with design principles of modularity and object-oriented programming, CODER is made up of four different types of objects differentiated by their use of knowledge (see Figure 1). *Experts* are specialists in particular restricted domains pertinent to the tasks at hand. They communicate with each other only through *blackboards*, which serve as holding areas for session-specific knowledge. Each blackboard is the external knowledge source for a *strategist* that also has a local knowledge base of planning rules to coordinate the activities of experts in the community. *External knowledge bases* store information of common interest to several experts. Because they deal only with factual world knowledge, they require only limited inference abilities. Finally, there are

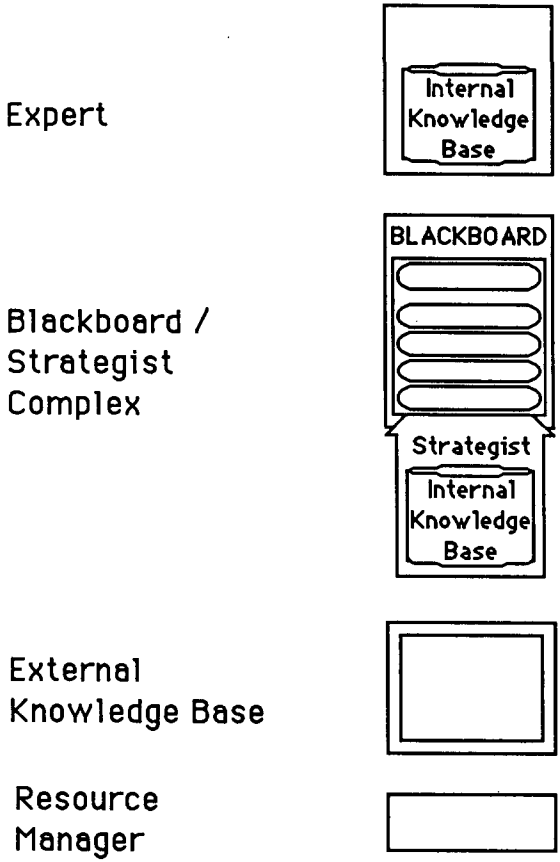


**Figure 1.** CODER Object Classes

*resource managers* mediating between low-level machine structures and the abstract representations used by the rest of the system. These may be implemented in a procedural language, as they do not require special knowledge or inference capabilities.

The internal structure of CODER is shown in Figure 2. The central region or "spine" includes external databases for documents and terms, along with the knowledge administration complex. The resources of the spine are shared by two expert communities, one for document analysis and one for retrieval. From an external perspective the system wraps so that users (shown at either end of the figure) are inside the system; they can both enter documents and retrieve them, possibly in an integrated fashion. Each user communicates with a resource manager specialized to his or her preferred interface, which in turn communicates with a group of translation specialists to effect a two-way dialogue between the user and the rest of the system. The interaction of these specialists with each

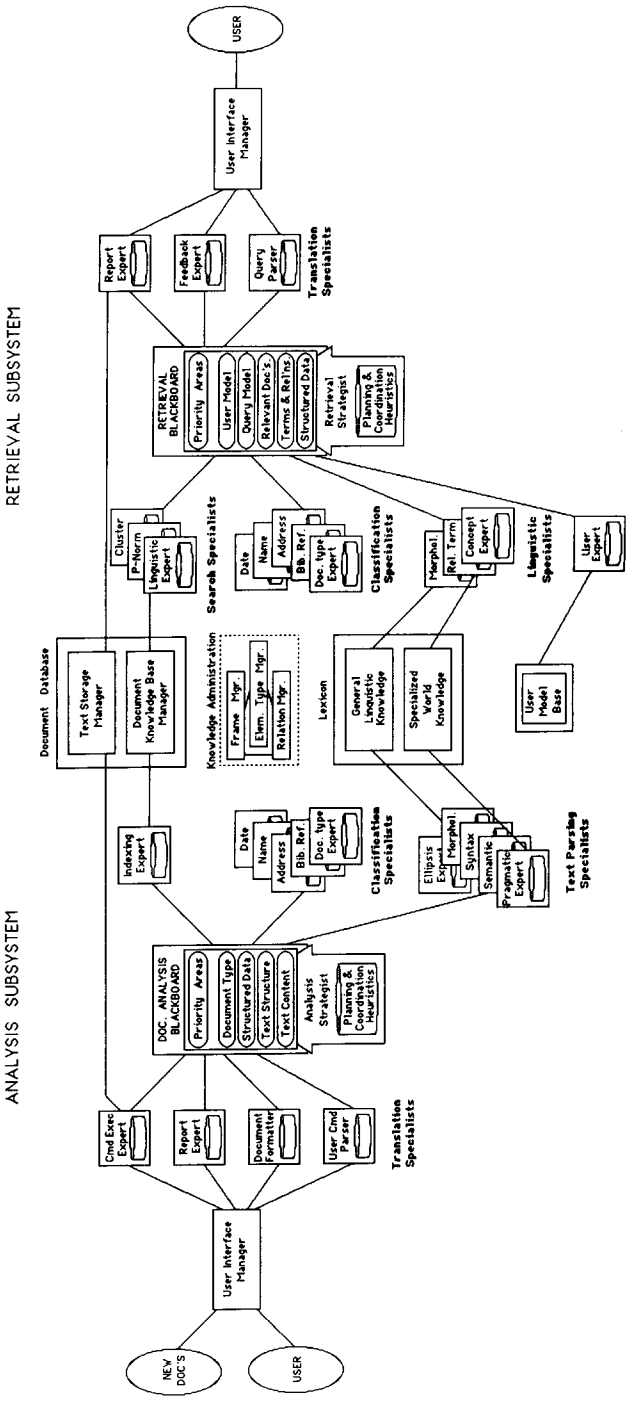


**Figure 2.** Overview of the CODER System

other and with other experts is mediated by a blackboard. Each expert community may also reference additional external knowledge bases, such as the user model base. Attached to each blackboard and coordinating all the activities of the subsystem is a strategist.

The overall operation of CODER is shown in Figure 3. Because one or more parts of CODER can be assigned to separate processors, it is logical to view the system as made up of groupings of modules needed for common functions. For example, one user might be entering new documents so the system can analyze and store them, while other users are searching and retrieving documents. In both these cases, state information about the progress of the system's services for a user is maintained entirely on the blackboard involved. Finally, at the

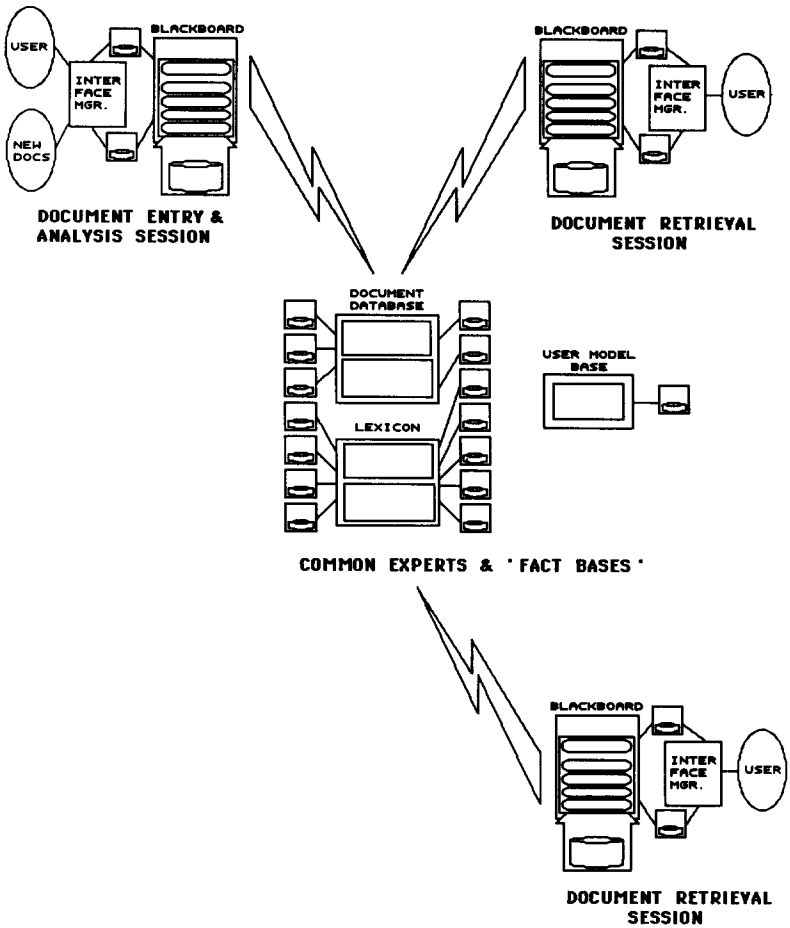


**Figure 3.** Overview of System Operation in a Distributed Environment

center of the figure are the shared experts and external knowledge bases involved in supporting these tasks.

The sections that follow provide more detailed information about the various CODER components.

## Knowledge Administration

Knowledge in CODER is partitioned both *horizontally* between the two subsystems and among the modules of each subsystem, and *vertically* along what Sterling [65] refers to as the "logical levels of problem solving." The top level of this second division is the goal-oriented planning knowledge that guides the session strategists. In the current CODER implementation this *strategic knowledge* is encoded in rules for recognizing and reacting to stages in the problem tasks. Actual steps in the problem solution are carried out by the experts in each community using *tactical knowledge* of how to accomplish their designated tasks. Finally, the characteristics of the problem universe are represented as *world knowledge* in the external knowledge bases.

Strategic and tactical knowledge are stored locally in the modules that use them. The same is not true for world knowledge. Facts about the world provide the premises from which the experts reason about their tasks, and facts and hypotheses about the world make up the problem-state descriptions that inform the strategists' decisions. Thus, the factual representation language used in CODER to encode world knowledge also serves as a *lingua franca* for communication among the modules that make up each subsystem community. This language, defined in Figure 4, is itself made up of three levels. Elementary data types include distinct sets of names for entities of different sorts, as well as such familiar primitives as character, integer, and atom. Frames provide a facility for building definite descriptions of entities according to prototypical descriptions drawn from a tangled hierarchy of classes. And relations are predications on those entities, either ascribing accidental properties to them or describing relations among them.

Relations are familiar to AI programmers by analogy to Prolog predicates. CODER relations differ from Prolog predicates in having specified type signatures (arity and types on arguments) and algebraic attributes (whether they are transitive, symmetric, and so forth). Elementary data types are also familiar; restricted data types are defined through a type of restriction polymorphism [66]. The semantics of frames, however, may require some explanation. The subsumption relation given in Figure 5*a* defines the inheritance hierarchy that can be specified for frame types. The frame with no slots subsumes all other frames, and two frames are equivalent if they subsume one another.

Figure 5*b* defines the matching of frame objects, in terms of the types of the various slots and their values. Frame object A matches frame object B if every filled slot of A matches a filled slot of B, where elementary objects match

```
relation  ::= relation_name (argument) +

   argument ::= relation
              | frame
              | elementary_object
              | quantifier argument

frame     ::= frame_name  (slot_name slot_filler) *

   slot_filler ::= frame
                 | elementary_object
                 | quantifier slot_filler


elementary_object ::= (quantifier)  (primitive_object   restriction)

    quantifier ::= list_of   |   non_empty_list_of
                 | set_of    |   non_empty_set_of
                 | integer
```

**Figure 4.** Definition of Factual Knowledge Representation Formalism

```
subsumes(ancestor_frame, descendent_frame) ≡
   slot_list(ancestor_frame, anc_list),
   slot_list(descendent_frame, desc_list),
   ∀x (x ∈ anc_list ⊃ ∃y (y ∈ desc_list ∧ name(x)=name(y) ∧
        subsumes(type(x), type(y)) )).
```

**Figure 5a.** Semantics of Frames: Frame Subsumption

```
match(frame1, frame2) ≡
   slot_list(type(frame1), list1) ∧
   slot_list(type(frame2), list2) ∧
   ∀x (x ∈ list1 ∧ has_value(frame1, x, v) ⊃ ∃y (y ∈ list2 ∧
      name(x)=name(y) ∧ has_value(frame2, y, r) ∧ match(v, r) )).

match(elt1, elt2) ≡
   elt1 = elt2.
```

**Figure 5b.** Semantics of Frames: Frame Matching

if and only if they are equal. This asymmetric matching is based only on filled slots, so objects of differing types can still match. Matching of frames is computationally inexpensive and mirrors whether the two objects, or two descriptions of the same object, are indeed similar. By defining suitable frame types, the knowledge administrator can describe the various entities to be handled in a particular CODER system. Experts can then instantiate objects of these types and store them on the blackboard or in external knowledge bases.

Consistent use of this formalism is maintained system-wide by the modules of the knowledge administration complex (see Figure 6), which include type managers for each component of the language. For example, the frame type manager keeps track of the classes suitable for describing entities and the

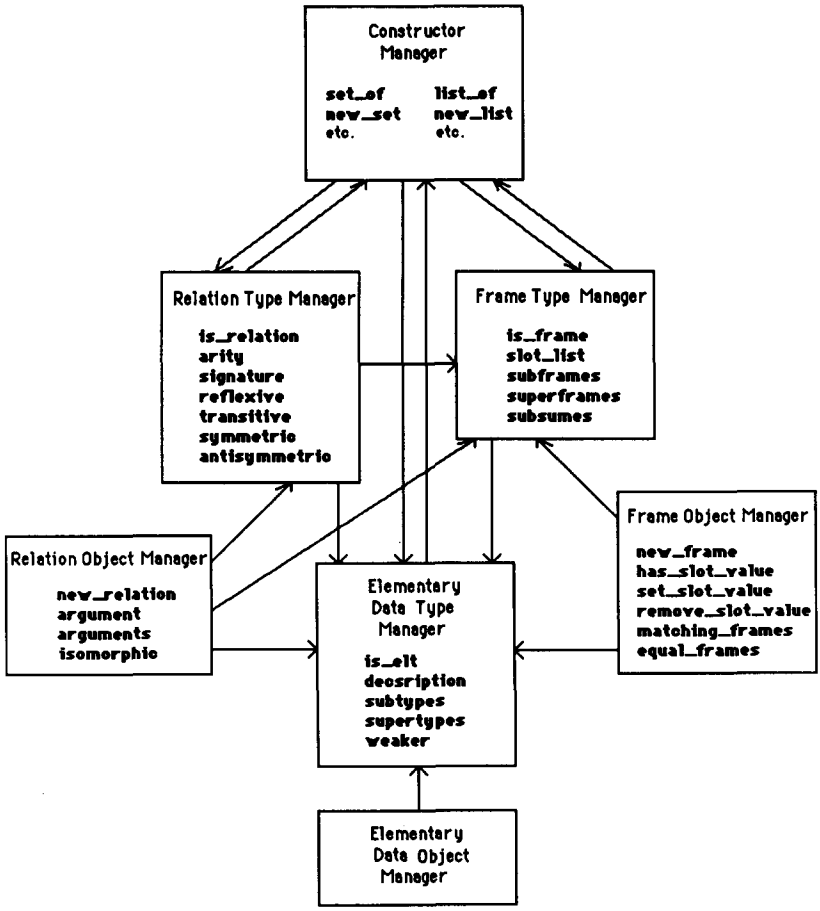


**Figure 6.** Internal Structure of the CODER Knowledge Administration Complex (arrows indicate dependencies)

attributes appropriate to each class; the relation type manager keeps track of the relations existing among entities and the characteristics of each relation. New types are added only by an external system administrator rather than by the modules of the system: deciding what sorts of types should be recognized by the system is part of the knowledge engineering involved in constructing CODER. Knowledge objects, by contrast—concrete expressions in the representation language—are created, modified, and destroyed dynamically during the operation of the system.

### External Knowledge Bases

Whereas the knowledge administration complex aids with control of the types of knowledge representations employed in a particular system, the external knowledge bases (EKBs) provide storage and access to large numbers of facts. The document knowledge base, the lexicon, and the user model base are all EKBs that each maintain knowledge about a particular class of objects as specific statements of fact.

The functionality of external knowledge bases is specified by the operations shown in Figure 7. Formally, propositions entered into a fact base are required to be *ground instances* of logical relations known to the system; that is, to involve neither unbound variables nor meta-terms. These propositions are added to an external knowledge base as single statements but may be retrieved in either of two ways. The knowledge base may be queried with a skeletal fact, that is, a fact containing one or more variables, and will return the set of all facts in the knowledge base that match the skeleton. Alternately, the knowledge base may be queried with an object (an elementary datum, a frame, or a relation) and will return the set of all facts involving that object. In addition, a knowledge base may be queried about the *number* of facts that match an object or a skeletal fact:

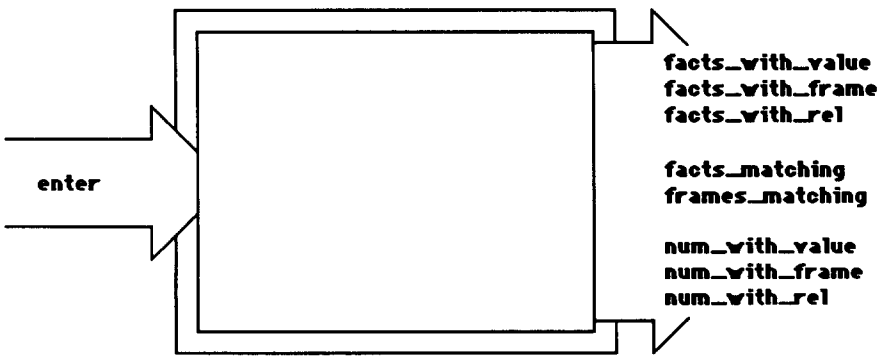


**Figure 7.** The Functionality of an External Knowledge Base. An EKB has no implementation independent internal structure.

this information can be used by the querying entity for statistical purposes or simply to avoid receiving excessively large sets of facts.

The lexicon maintains knowledge about terms in the language. It can be conceptually divided into two parts, one of general linguistic knowledge and the other of specialized world knowledge particular to the collection of documents employed. Although knowledge from both conceptual halves may be recalled for a given request, tagging the knowledge in this way promotes portability by allowing knowledge of general use to be decoupled from the pragmatics of a given document collection and reused in other applications. Construction of the current CODER lexicon following these principles is highlighted in Figure 8. The initial loading of facts portrayed at the top of the figure is from one large machine-readable dictionary [67]. Table 1 describes the various relations initially derived from the more than 80,000 headwords present. Further analysis such as of the definitions (see c_DEF) should lead to additional relations that would be more directly usable for parsing.

The document knowledge base maintains facts about the documents as assertions relating a document (passage) and a knowledge structure. A simple attached resource manager provides storage and retrieval for raw document text. Together these modules constitute the document database. Finally, there is a

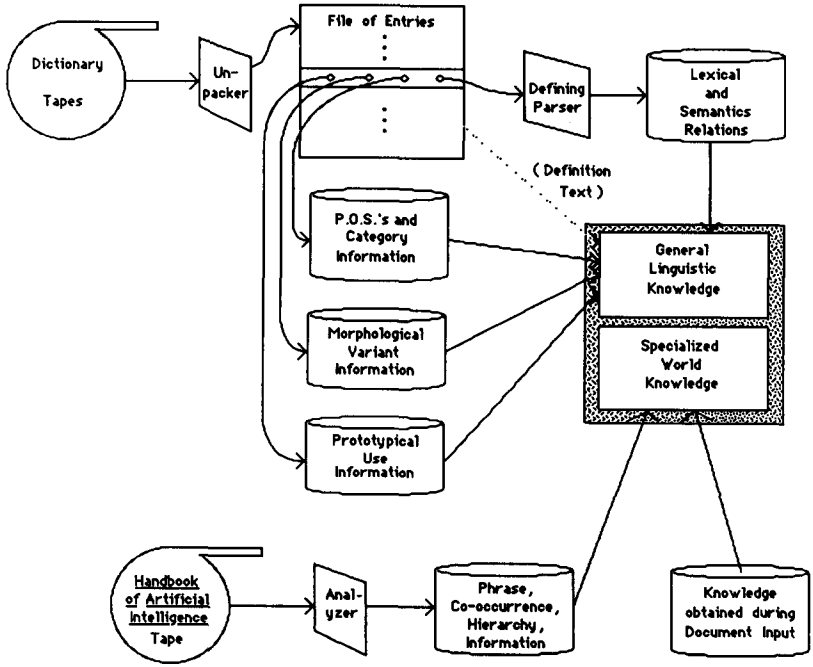


**Figure 8.** Construction of the CODER Lexicon

**Table 1.** Relations abstracted from the *Collins Dictionary of the English Language* [70] tapes

| | |
|---|---|
| c_ABBREV | Abbreviation of headword |
| c_ALSO_CALLED | Headword also commonly called this |
| c_CATEGORY | Category (semantic label) of headword |
| c_COMPARE | Compare to another headword and sense(s) |
| c_DEF | Definition of headword |
| c_DEF_NUM | Number of (up to) 80-character blocks of definition |
| c_HEADWORD | Headword entry |
| c_MORPH | Morphological variant of headword (including part of speech) |
| c_NLAST | Rest (e.g., first/middle name) of proper noun headword |
| c_PAST | Past form of headword |
| c_PLURAL | Plural of headword (sometimes just the ending) |
| c_POS | Part of speech |
| c_RELADJ | Related adjective to headword |
| c_SAMP | Example of headword in context |
| c_SINGULAR | Singular form of headword (sometimes just the ending) |
| c_SYLL | Syllabification of headword |
| c_USAGE | Usage notes providing guidance on usage of headword |
| c_USAGE_NUM | Number of (up to) 80-character blocks in usage note |
| c_VAR_SPELL | Variant spelling(s) (if any) |
| c_VAR_SYLL | Syllabification of variant spelling(s) |

user model base of facts about individual users. These include reports of occurrences during a single session and general statements about the user, such as the type of information that has proven relevant in the past, background knowledge particular to or supplied by the user, and common characteristics of relevant documents. This body of knowledge about users, and the bodies of knowledge discussed above, inform the system's response in intelligently analyzing and retrieving documents for a particular individual.

## Experts

Conceptually, an expert is a specialist in a certain restricted domain pertinent to the task at hand. Experts are designed to be implemented in relative isolation from one another: no expert has knowledge of the internal behavior of the other experts in the community, and all experts communicate with the community strictly through the given blackboard. Part of the specification of an individual expert is the set of predicates that it may view in a given blackboard area and the (possibly overlapping) set of predicates that it may post back. Obviously, there must be agreement among the expert implementors on the structure and bounds of those predicates if the experts are to work together. What each expert does

with those predicates, however, and what internal knowledge and processes it uses to produce new hypotheses are left to the implementor of the individual expert. Each expert can therefore be built in the way that best takes advantage of the characteristics of its particular domain of expertise.

An expert has only two requirements for its operation: it should be knowledge driven, and it must recognize the appropriate commands from the strategist scheduler. The first is philosophical in nature: it is part of the CODER design that the complexities of the system tasks be realized in the knowledge required for their execution rather than the process of execution itself. For experts this implies that expertise be represented as explicit knowledge, separate from whatever engine manipulates it. The knowledge in the expert, moreover, is constrained by system design to be on a higher level than factual knowledge: either rules for finding and manipulating factual knowledge in an external knowledge base or facts that relate to classes of objects in the problem universe. The second requirement is pragmatic: for the strategist to schedule their activity properly, experts must go through a canonical cycle of operations.

The typical CODER expert consists of a communications interface, a local knowledge base, and an inference engine (see Figure 9). The interface provides for communication with the blackboard and optionally with external knowledge sources such as resource managers or EKBs. The local knowledge base contains the particular expertise necessary for the proper execution of the expert's tasks; the inference engine is chosen to best execute those tasks. Possible engines include both forward-chaining and backward-chaining rule interpreters, frame-based classification engines, and pattern-matching engines. These engines could then be associated with different rule bases, classification trees, and similarity measures, respectively, to produce specialized experts in a variety of disciplines.

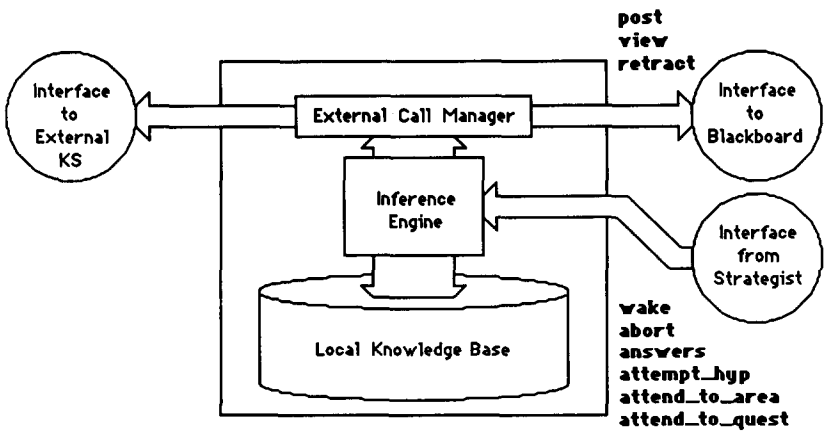


**Figure 9.** Canonical CODER Expert Showing Internal Structure and Functionality of Interface with Blackboard/Strategist Complex

Some examples of expert knowledge bases and inference techniques are given in Table 2. Recent research supports the view that it is possible to build engines that cover a broad range of problems without falling into the computational trap of general inference (see [38] for a formal analysis of this effect).

This method of problem decomposition is particularly well suited to the tasks of document analysis and retrieval, where the relevance of a given document to a given information need is influenced by many factors. Assigning an expert to a

**Table 2.** Knowledge bases and inference types for some sample experts

| Expert Name | Local Knowledge Base | Inference Engine |
|---|---|---|
| Date | Mappings from different natural representations for dates into a canonical internal representation | Forward chaining (rule based) |
| Bibio. Ref. | Different types of biblographic entities and clues to recognize them; lexical conventions for representations of biblio. entities in text and in bibliographies | Classificational |
| Doc. Type | Different types of documents (both hard and soft types) and clues to recognize them and their component fields | Classificational |
| Morphology | Declension, conjugation and case-changing rules for English; irregular morphological variants (or how to find them in the lexicon) | Hard coded |
| Related Term | Methods and metrics for navigating relation networks in the lexicon | Relational |
| Cluster | Heuristics for finding document passages in the database that are similar to those identified by the user as close to current needs | Special purpose |
| P-Norm | Methods to transform a fact-based representation of a search request to a p-norm representation and conduct a search | Hard coded |
| Linguistic | Methods and metrics for identifying documents in the database that share linguistic substructures with the retrieval request | Relational |

small area of specialization and decoupling it from the remainder of the system, however, has additional advantages. First, the development of the expert is separated from that of the surrounding system. Interaction problems, normally a plague of AI systems, are thereby kept to a minimum. In addition, the experts are kept small, so problems of rule interaction *within* the expert are minimized. Furthermore, tasks that are found to require too much complexity can be further subdivided according to the areas of expertise required to solve them, until they are reduced to manageable size.

## Blackboard/Strategist Complex

A blackboard is an area for communication among experts [35]. This communication takes place through posting and reading hypotheses in specialized subject areas. In CODER blackboards (see Figure 10), a specialization of this process provides a means for asking and answering questions, which are

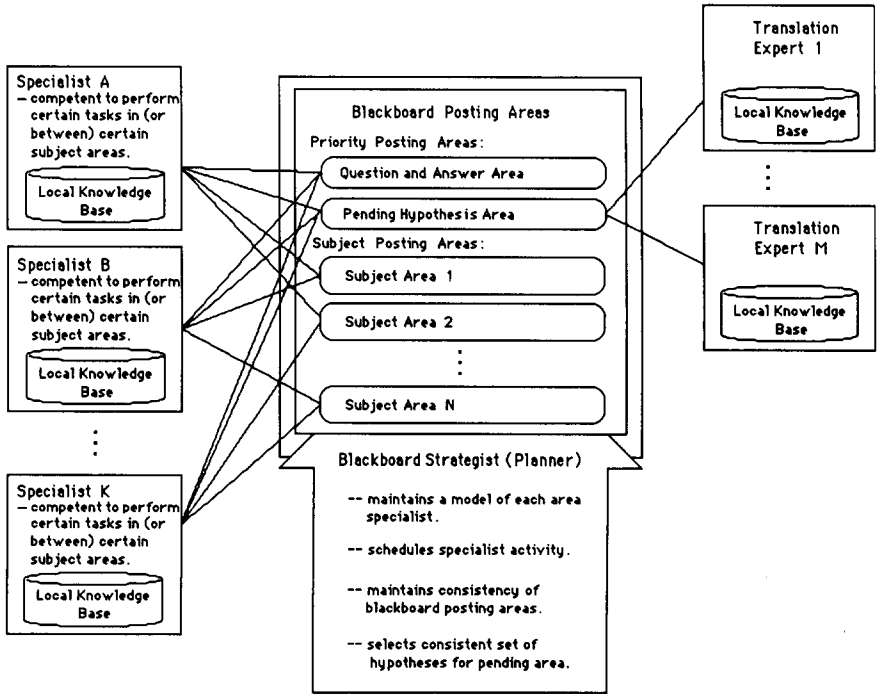


**Figure 10.** Blackboard/Strategist Complex Showing Mapping of Experts in the Immediate Community to Blackboard Areas

contained in a separate area of the blackboard. The importance of this type of communication was noted convincingly by Belkin and colleagues [36]. In addition, the CODER blackboards provide a special area, maintained by the blackboard strategist, containing a small set of consistent hypotheses of high certainty. This pending hypothesis area is available for read access by all experts and thus, indirectly, by the outside world. It provides an instantaneous picture of the "consensus" of the blackboard: what the system as a whole hypothesizes about the problem under consideration at any moment.

A hypothesis is a higher-order knowledge structure built on the factual knowledge forms supported by the knowledge administration complex. It consists of five parts:

1. The fact hypothesized.
2. The identifier of the expert hypothesizing it.
3. The confidence that the expert has in it (which can be assigned by different methods for different types of hypotheses, according to whatever knowledge aggregation scheme is appropriate for the set of constraints and knowledge sources at hand).
4. The hypothesis identifier.
5. The dependencies on other hypotheses.

This latter information, apart from aiding selection of the set of pending hypotheses, allows truth maintenance functions to be performed within the blackboard subject areas. If an expert withdraws a hypothesis, for instance, or radically changes the associated confidence level, this information makes it possible to schedule tasks to reconsider the dependent hypotheses.

Monitoring the blackboard for this class of event is one function of the blackboard strategist, shown in the lower part of Figure 11. Because the logic task scheduler's rules governing truth maintenance are independent of the particular predicates involved in the facts hypothesized, this function is independent of the application domain of the blackboard community. The strategist also monitors the blackboard for domain-specific events and conditions that trigger new processing. These categories of function are kept separate in the strategist, so the truth maintenance function can be transported to other tasks. Both task schedulers in the strategist have been designed as rule interpreters, as neither the strategies involved in truth maintenance nor those involved in information analysis or retrieval are yet well understood.

The final component of the strategist is a dispatcher of the tasks identified by the other components. On the basis of the mix of tasks scheduled by the truth maintenance and task oriented components, it attempts to make optimal use of all available machine resources by issuing appropriately timed commands to the experts in the blackboard community. This allows different groups of experts to be active at different phases in the community task, but also allows experts outside the currently active group to be called up to answer a question or to reconsider a hypothesis. In an ideal environment with one processor per expert,

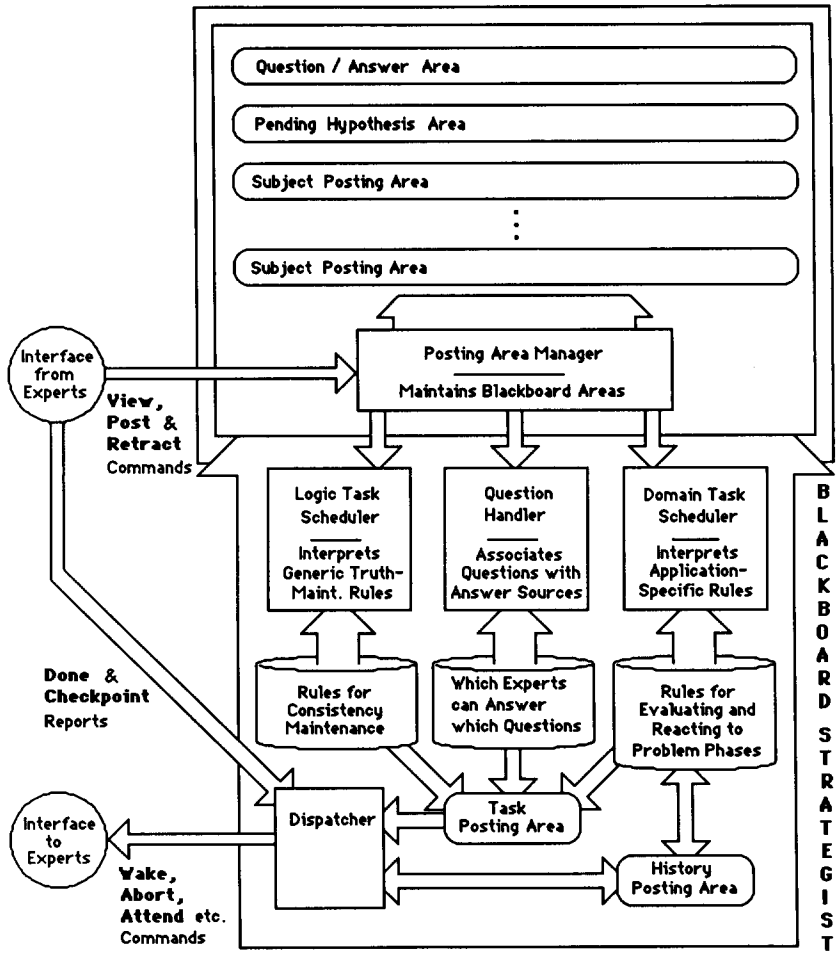


**Figure 11.** Internal Structure of Blackboard/Strategist Complex

such dispatching duties would be kept at a minimum, and the normal cycle of activities of each expert would ensure highly parallel processing.

## IMPLEMENTATION STATUS AND FUTURE WORK

Early work on the CODER system concentrated on design and on preparation of the knowledge to be loaded into the external knowledge bases. A test collection was needed that would allow investigation of the many questions of interest, so the decision was made to collect all issues of *AIList Digest*, an electronic mail publication distributed from the DARPA Internet to many

networks, beginning with the first one edited by Kenneth Laws in 1984. As of May 1987, roughly 13 megabytes of data, including over 6500 messages by many different authors in widely differing formats, have been collected. To provide domain knowledge relevant to this collection as a supplement to the general English lexicon, *The Handbook of Artificial Intelligence* has also been obtained in machine-readable form. Queries and relevance judgments on this test collection have been captured using the SMART system.

Experimentation in natural language processing is best supported by a large, comprehensive lexicon. The most efficient construction approach was to reformat machine-readable dictionaries and to parse entries into suitable structures. Because the G.&C. Merriam Company and the Longman Group Limited both refused to provide their dictionaries, it was decided to use four separate dictionaries obtained from the Oxford Text Archive [68–71] so that the resulting lexicon could be made freely available to other researchers. Initial efforts focused on the largest of these, the *Collins Dictionary of the English Language*.

Development of CODER is taking place in the UNIX[1] environment. Pipes and TCP/IP sockets [72] allow intermodule and intermachine communication. Thus, procedural modules like interface managers can be coded in C or C++, a dialect supporting the class-object paradigm [73]. MU-Prolog was selected as the AI implementation language, as it includes a clause indexing facility for medium-size collections of facts or rules, and two types of database support [74]. The first scheme uses hashing, and the second employs a two-level superimposed coding scheme [75] that performs well for partial matches [76] and can easily support large Prolog databases [77]. MU-Prolog also has tools for information hiding, interfacing with the UNIX operating system, and reducing dependence on rule ordering and extra-logical operations.

Implementation of the modules of the CODER system began early in 1986. At the end of the summer of 1986, the knowledge administration and blackboard/strategist complexes were nearly complete, the communication routines were well underway, interface managers using CURSES and SUN-Windows packages had been prototyped (see Figure 12), a p-norm search routine had been tested, and a first version of the document-type specialist developed. Further coding according to the detailed specifications given in France [78] should lead to a working prototype in 1987.

Subsequent efforts will aim first at demonstrating the feasibility of using CODER for document analysis and retrieval and at comparing different approaches to see if CODER will indeed simplify experimentation regarding the application of AI methods to ISR problems. Much of this work, however, will require development of a more refined lexicon, specification of heuristics regarding appropriate retrieval methods for particular types of queries, and

[1] Trademark of AT&T Bell Laboratories.

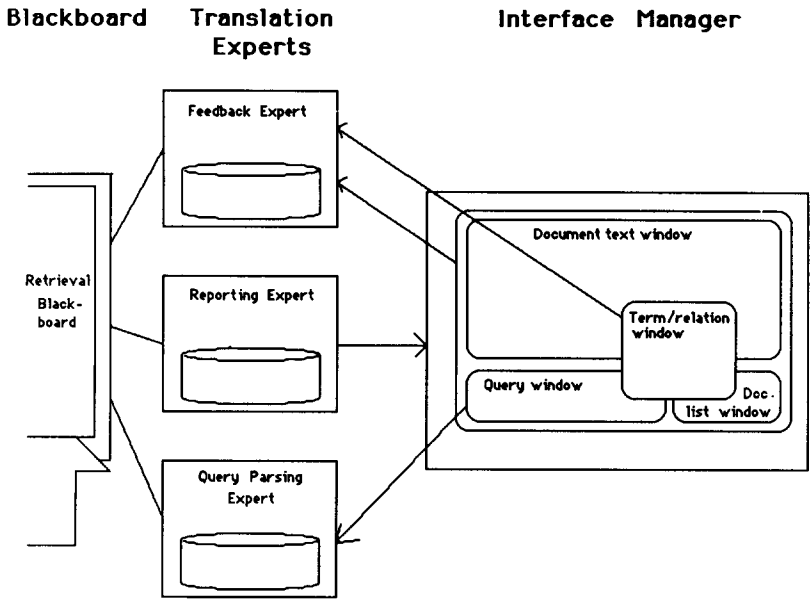


**Figure 12.** Retrieval Subsystem User Interface

thorough integration of user models into a truly interactive system for satisfying information needs. It is hoped that initial success with CODER will be followed by a long period of productive research and experimentation that will contribute to the human knowledge base about ISR.

## ACKNOWLEDGMENTS

Numerous students enrolled during the last two years in Virginia Tech's two-quarter graduate sequence on information storage and retrieval have played a role in the development of CODER. For an MS project, Robert Wohlwend made an initial version of the Prolog lexicon from the typesetting tapes of one major dictionary. Research assistants Mary Beth Weaver and Qi Fan Chen are currently involved in system implementation. Pat Cooper and Joy Weiss have provided secretarial support.

The Oxford Text Archive and the Melbourne University Department of Computer Science have supplied tapes with dictionaries and Prolog interpreters, respectively. The staff of the SUMEX Computer Project at the Stanford University Medical Center, with permission of the authors and William Kaufman, provided the machine-readable version of the *Handbook of Artificial Intelligence.* Kenneth Laws has helped assure the project of a complete backfile of *AIList Digest* issues.

## References


1. Neufeld, M. L., and Cornog, M., Database history: From dinosaurs to compact discs, *J. Am. Soc. Inf. Sci.* 37(4), 183–190, 1986.
2. Blair, D. C., and Maron, M. E., An evaluation of retrieval effectiveness for a full-text document-retrieval system, *Commun. ACM* 28(3), 289–299, 1985.
3. Ojala, M., Views on end-user searching, *J. Am. Soc. Inf. Sci.* 37(4), 197–203, 1986.
4. Williams, M. E., Transparent information systems through gateways, front ends, intermediaries, and interfaces, *J. Am. Soc. Inf. Sci.* 37(4), 204–214, 1986.
5. Fox, E. A., Information retrieval: Research into new capabilities, in *CD-ROM: The New Papyrus* (S. Lambert and S. Ropiequet, Eds.), Microsoft Press, Redmond, WA, 143–174, 1986.
6. Salton, G., The SMART system 1961–1976: Experiments in dynamic document processing, *Encyclopedia of Library and Information Science* 28, 1–36, 1980.
7. Fox, E. A., Some considerations for implementing the SMART Information Retrieval System under UNIX, TR 83-560, Dept. of Computer Science, Cornell Univ., 1983.
8. Buckley, C., Implementation of the SMART Information Retrieval System. TR 85-686, Dept. of Computer Science, Cornell Univ., 1985.
9. Fox, E. A., Composite document extended retrieval: An overview, *Research and Development in Information Retrieval, Eighth Annual Int. ACM SIGIR Conference.*, Montreal, 42–53, 1985.
10. Fox, E. A., Analysis and retrieval of composite documents, *ASIS '85, Proceedings of the 48th ASIS Annual Meeting,* 54–58, 1985.
11. O'Connor, J., Answer-passage retrieval by text searching, *J. Am. Soc. Inf. Sci.* 31(4), 227–239, 1980.
12. Kimura, G. D., A Structure Editor and Model for Abstract Document Objects, Dissertation, Tech. Report No. 84-07-04, Univ. of Washington, Dept. of Computer Science, 1984.
13. Babatz, R., and Bogen, M., Semantic relations in message handling systems: Referable documents, Paper presented at IFIP Working Group 6.5 Symposium, 1985.
14. Horak W., and Kronert, G., An object-oriented office document architecture model for processing and interchange of documents, *Proceedings of the Second ACM-SIGOA Conference on Office Information Systems,* 152–160, 1984.
15. Harper, D. J., Dunnion, J., Sherwood-Smith, M., and van Rijsbergen, C. J., Minstrel-ODM: A basic office data model, *Inf. Proc. & Mgmt.* 22(2), 83–107, 1986.

16. Peels, A., Janssen, N., and Nawijn, W., Document architecture and text formatting, *ACM Trans. Office Inf. Sys.* 3(4), 347–369, 1985.

17. Rauch-Hindin, W., Upper level OSI protocols near completion, *Mini-Micro Systems* (18)9, 53–66, 1986.

18. Tenopir, C., Full-text databases, *ARIST* 19, 215–246, 1984.

19. Salton, G., Another look at automatic text-retrieval systems, *Commun. ACM* 29(7), 648–656, 1986.

20. Katzer, J., et al., A study of the overlap among document representations, *Inf. Tech.: Res. & Dev.* 1(4), 261–274, 1982.

21. Bichteler J., and Eaton III, E. A., The combined use of bibliographic coupling and cocitation for document retrieval, *J. Am. Soc. Inf. Sci.,* 31(4), 278–282, 1980.

22. Fox, E. A., Extending the Boolean and Vector Space Models of Information Retrieval and P-Norm Queries and Multiple Concept Types, Dissertation, Cornell Univ., University Microfilms Int., Ann Arbor, Mich., 1983.

23. Voorhees, E. M., The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval, Dissertation, TR 85-705, Dept. of Computer Science, Cornell Univ., 1985.

24. Salton, G., Fox, E. A., and Wu, H., Extended boolean information retrieval, *Commun. ACM* 26(11), 1022–1036, 1983.

25. Salton, G., Wong, A., and Yang, C. S., A vector space model for automatic indexing, *Commun. ACM* 18(11), 613–620, 1975.

26. Robertson, S. E., and Sparck Jones, K., Relevance weighting of search terms, *J. Am. Soc. Inf. Sci.* 27(3), 129–146, 1976.

27. Van Rijsbergen, C. J., *Information Retrieval,* 2nd ed., Butterworths, London, 1979.

28. Buckley, C., and Lewit, A. F., Optimization of inverted vector searches, *Research and Development in Information Retrieval, Eighth Annual International ACM SIGIR Conference*, Montreal, 97–110, 1985.

29. Kowalski, R. A., *Logic for Problem Solving,* Elsevier North-Holland, New York, 1979.

30. Genesereth, M. R., and Ginsberg, M. L., Logic programming, *Commun. ACM* 28(9), 933–941, 1985.

31. Clocksin, W. F., and Mellish, C. S., *Programming in Prolog,* 2nd ed., Springer-Verlag, New York, 1984.

32. Pereira, F., Logic for Natural Language Analysis, Tech. Note 275, SRI International, 1983.

33. Helm, A. R., Marriott, K., and Lassez, C., Prolog for expert systems: An

evaluation, *Proceedings of the Expert Systems in Government Symposium,* 284–293, 1985.

34. Bobrow, D. G., If Prolog is the answer, what is the question? Or what it takes to support AI programming paradigms, *IEEE Trans. Software Eng.* 11(11), 1401–1408, 1985.

35. Nii, H. P., Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures, *AI Mag.* 7(2), 38–53, 1986.

36. Belkin, N. J., Hennings, R. D., and Seeger, T., Simulation of a distributed expert-based information provision mechanism, *Inf. Tech.: Res. Dev. Applications* 3(3), 122–141, 1984.

37. Smith, L. C., and Warner, A. J., A taxonomy of representations in information retrieval system design, in *Representation and Exchange of Knowledge as a Basis of Information Processes* (H. J. Dietschmann, Ed.), North-Holland, New York, 31–49, 1984.

38. Levesque, H. J., A fundamental tradeoff in knowledge representation and reasoning, *Proceedings of the Fifth CSCSI National Conference,* London, Ontario, 141–152, 1984.

39. Patel-Schneider, P. F., Brachman, R. J., and Levesque, H. J., ARGON: Knowledge Representation Meets Information Retrieval, Proceedings of the First Conference on Artificial Intelligence Applications, December 1984, Denver, CO. Washington, D.C.: IEEE Computer Society Press; 280–286, 1984.

40. Minsky, M., A framework for representing knowledge, *The Psychology of Computer Vision,* (P. Winston, Ed.), McGraw-Hill, New York, 1975.

41. Fikes, R., and Kehler, T., The role of frame-based representation in reasoning, *Commun. ACM* 28(9), 904–920, 1985.

42. Patel-Schneider, P. F., Small can be Beautiful in Knowledge Representation, Proceedings of the IEEE workshop on Principles of Knowledge-Based Systems. Denver, CO, 11–16, 1984.

43. Zarri, G. P., An outline of the representation and use of temporal data in the RESEDA system, *Inf. Tech.: Res. Dev. Applications* 2(2/3), 89–108, 1983.

44. Sparck Jones, K., and Kay, M., *Linguistics and Information Science,* Academic Press, New York, 1973.

45. Sparck Jones, K., and Tait, J. I., Automatic search term variant generation, *J. Doc.* 40(1), 50-66, 1984.

46. Hahn, U., and Reimer, U., Heuristic text parsing in "Topic": Methodological issues in a knowledge-based text condensation system, in *Representation and Exchange of Knowledge as a Basis of Information Processes* (H. J. Dietschmann, Ed.), North-Holland, New York, 143–163, 1984.

47. DeJong, G., An overview of the FRUMP system, in *Strategies for Natural*

*Language Processing,* (W. G. Lehnert and M. H. Ringle, Eds.), Lawrence Erlbaum, Hillsdale, N.J., 149–176, 1982.

48. Mauldin, M. L., Thesis proposal: Information retrieval by text skimming, unpublished manuscript, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Penn., 1986.

49. Riesbeck, C. K., Realistic language comprehension, in *Strategies for Natural Language Processing,* (W. G. Lehnert and M. H. Ringle, Eds.), Lawrence Erlbaum, Hillsdale, N.J., 435–454, 1982.

50. Selfridge, M., Integrated processing produces robust understanding, *Comp. Ling.* 12(2), 89–106, 1986.

51. Sager, N., Sublanguage grammars in science information processing, *J. Am. Soc. Inf. Sci.* 26(1), 10–16, 1975.

52. Amsler, R. A., Machine-readable dictionaries, *ARIST* 19, 161–209, 1984.

53. White, C., The linguistic string project dictionary for automatic text analysis, *Proceedings of the Workshop on Machine-Readable Dictionaries,* SRI, Menlo Park, Calif., 1983.

54. Borgman, C. L., Psychological research in human-computer interaction, *ARIST* 19, 33–64, 1984.

55. Sewell, W., and Teitelbaum, S., Observations of end-user online searching behavior over eleven years, *J. Am. Soc. Inf. Sci.* 37(4), 234–245, 1986.

56. Weyer, S. A., The design of a dynamic book for information search, *Int. J. Man-Machine Stud.* 17(1), 87–107, 1982.

57. Weyer, S. A., and Borning, A. H., A prototype electronic encyclopedia, *ACM Trans. on Office Info. Syst.* 3(1), 63–68, 1984.

58. Frei, H. P., and Jauslin, J. F., Graphical presentation of information and services: A user oriented interface, *Inf. Tech.: Res. Dev.* 2(1), 23–42, 1983.

59. Oddy, R. N., Information retrieval through man-machine dialogue, *J. Doc.* 33(1), 1–14, 1977.

60. Yip, M.-K., An Expert System for Document Retrieval, MS Thesis, M.I.T., 1979.

61. Thompson, R. H., and Croft, W. B., An expert system for document retrieval, *Proceedings of the Expert Systems in Government Symposium,* IEEE, 448–456, 1985.

62. McCune, B. P., et al., RUBRIC: A system for rule-based information retrieval, *IEEE Trans. Software Eng.* SE-11(9), 939–945, 1985.

63. Tong, R. M., et al., A rule-based approach to information retrieval: Some results and comments, AAAI-83: Proceedings of the National Conference on Artificial Intelligence. Washington, DC, 411–415, 1983.

64. France, R. K., and Fox, E. A., Knowledge structures for information retrieval:

Representation in the CODER project, *Proceedings of the Second IEEE Expert Systems in Government Conference,* McLean, Va., 135–141, 1986.

65. Sterling, L., Logical levels of problem solving, *Proceedings of the Second International Logic Programming Conference,* Uppsala Univ., Uppsala, Sweden, 231–242, 1984.

66. Cardelli, L., and Wegner, P., On understanding types, data abstraction, and polymorphism, *ACM Computing Surveys* 17(4), 471–522, 1985.

67. Wohlwend, R. C., Creation of a Prolog Fact Base from the Collins English Dictionary, MS Report, Dept. of Computer Science, Virginia Tech, Blacksburg, Va., 1986.

68. Cowie, A. P., and Mackin, R., *Oxford Dictionary of Current Idiomatic English. Volume 1: Verbs with Prepositions & Particles,* Oxford UP, Oxford, England, 1975.

69. Cowie, A. P., Mackin, R., and McCaig, I. R., *Oxford Dictionary of Current Idiomatic English. Volume 2: Phrase, Clause & Sentence Idioms.* Oxford U.P., Oxford, England, 1983.

70. Hanks, P. (Ed.), *Collins Dictionary of the English Language,* William Collins, London, 1979.

71. Hornby, A. S., (Ed.), *Oxford Advanced Dictionary of Current English,* Oxford, U.P., Oxford, England, 1974.

72. Leffler, S. J., Fabry, R. S., and Joy, W. N., A 4.2BSD interprocess communication primer, in *ULTRIX-32, Supplementary Documents,* vol. III, 3-5–3-28, 1984.

73. Stroustrup, B., *The C++ Programming Language,* Addison-Wesley, Reading, Mass., 1985.

74. Naish, L., *MU-Prolog 3.2db Reference Manual,* Dept. of Computer Science, Univ. of Melbourne, Melbourne, Australia, 1985.

75. Sacks-Davis, R., and Ramamohanarao, K., A two level superimposed coding scheme for partial match retrieval, *Info. Syst.* 8(4), 273–280, 1983.

76. Sacks-Davis, R., Performance of a multi-key access method based on descriptors and superimposed coding techniques, *Info. Syst.* 10(4), 391–403, 1985.

77. Ramamohanaro, K., and Shepherd, J., A Superimposed Codeword Indexing Scheme for Very Large Prolog Databases, Tech. Report 85/17, Dept. of Computer Science, Univ. of Melbourne, Melbourne, Australia, 1985.

78. France, R. K., An Artificial Intelligence Environment for Information Retrieval Research, MS Thesis, Dept. of Computer Science, Virginia Tech, Blacksburg, Va., 1986.