Information Technology: Research and Development (1983), *2* (43–60)

DOCUMENT RETRIEVAL AS A DATABASE APPLICATION

I. A. MACLEOD AND R. G. CRAWFORD

Department of Computing and Information Science, Queen's University, Kingston, Ontario K7L 3N6, Canada

(Received 14 December 1982)

ABSTRACT

This paper outlines the capabilities and the underlying implementation strategy for a modern DRS. The three main DBMS models are examined and their suitability as vehicles for DRS implementation are analysed. The advantages and disadvantages of the DBMS approaches are discussed. Finally we look at possible future scenarios for DRS implementations. In particular the implications of utilizing universal data languages are examined together with the likely future impact of electronic filing systems.

1. INTRODUCTION

It is possible to identify three classes of information system. By 'information system' we mean a system to facilitate the storage and subsequent retrieval of documents. These are database management systems (DBMS), document retrieval systems (DRS) and electronic filing systems (EFS). (Document retrieval systems are more usually referred to as information retrieval systems. As the other systems also perform information retrieval, the less ambiguous terminology is preferred here.) DBMS are generally used to manage formatted records, that is, information that is well structured. DRS are normally applied to the handling of relatively unstructured text. EFS are currently not well defined but generally may be said to simulate traditional office operations such as filing, so that there is an hierarchical organization corresponding to filing cabinets, drawers, folders and so on.

As noted above, DBMS are usually applied to highly structured records. Moreover a particular DBMS handles many different record types. Retrieval tends to be deterministic. That is, a record is accessed only if a particular field contains a specific value. Further, an important function of a DBMS is the ability to access records of different types where the relationship between two types might be dependent on the relation of the values of two fields.

DRS, on the other hand, generally handle records of a single type. At least one field, and sometimes this is the only field, contains free text. A most important part of a DRS is the index. This may be created from manually supplied keywords, in which case retrieval tends to be deterministic, or it may be automatically supplied from the free text by some sort of indexing program. In the latter case, index terms

0144-817X/83/01 0043-18 $03.00

are often weighted with the weights intended to measure the likelihood of a particular term applying to a particular record or document. Retrieval in this type of environment tends to be probabilistic and the retrieved output is normally ranked according to its predicted correspondence to the user query. However it should be noted that the first type of retrieval environment is not really deterministic since it is generally impossible to specify, using index terms, exactly what a document is 'about'. Consequently, in DRS humans tend to interact heavily with the system as they try to optimize the amount of relevant material to be retrieved by iteratively modifying their queries. This contrasts with the DBMS situation where queries tend to be well formulated and are not usually modified as the result of a retrieval operation (although retrieval may lead to the formulation of new queries).

In an office environment, the purpose of an EFS is to store and retrieve information in files where a file is a collection of, presumably, related documents. EFS are not yet currently in general use and are most often found in the context of mail systems. The two major capabilities which would seem to be necessary in an EFS, and not generally found in the other types of information system, are the ability to organize information hierarchically and to be able to store records of different types within the same file. Additionally there will be the need for the types of retrieval capability provided in both the other systems. There is a great deal of well-structured data, as for example, the contents of forms. At the same time there are usually significant quantities of unstructured material such as letters, proposals, reports, and so on.

A DBMS does basically just what its name implies, namely manage data. The management is more than just organizing records. It is also the maintenance of structural information which helps to relate different pieces of information stored in different parts of the database. The problem in a DBMS environment is not how to retrieve the information in a particular record, but rather to extract information that may be scattered across a number of different record types. For example, we might have in database files the following three record types:

AUTHOR, ADDRESS;
AUTHOR, TITLE;
KEYWORD, TITLE;

We may wish to obtain information such as : 'all the addresses of authors working in an area described by the following keywords'. Here the information can only be found if the various records involved have been, or can be, connected together in some way. In a DBMS we describe such connections as well as defining the structure of the records themselves by means of a *schema.* The 'art' of retrieving in a DBMS then involves using the connections defined in the schema to bring together diverse pieces of information.

In a DRS on the other hand, queries are much less precise and are generally concerned with retrieving individual records whose textual content is 'about' a topic in which the retriever is interested. The art here is in describing the topic. A DRS provides tools and facilities to assist in this process.

DBMS and DRS have tended to evolve separately from both historical and functional reasons. This has resulted in systems which are somewhat less general than they might be. With the accessibility of software systems to an ever increasing range of users, there is a growing awareness that this situation should be amended. It is the opinion of a growing number of researchers in the field that there should be a

unification of the capabilities of DBMS and DRS (Crawford and Macleod, 1978; Slonin *et al.*, 1978; Atkinson, 1979; Croft, 1982). For example, it is generally not possible to perform word searches within unstructured text in a DBMS, nor is it usually possible in a DRS to handle records of more than one type simultaneously. Impetus for further work in this area is being provided by interest in the development of EFS and office systems. One of the keys for any sort of effective computerized office system will be an integrated information system where both document retrieval and database management operations can be carried out side by side. In the remainder of this paper we will look at document retrieval as a DBMS application. In particular we will take a typical retrieval system and look at its underlying implementation. We will then see how a similar retrieval system might be organized within various database models.

2. STRUCTURE OF RETRIEVAL SYSTEMS

In this section we will look at the overall structure of retrieval systems. As we noted above, a DRS provides aids in tracking down particular documents. The most important of these is the index. This is essentially a list of words associated with each document. Because words are untidy pieces of data in that they get misspelt, sometimes have more than one spelling, have grammatical variants and so on, it is normally the practice for a DRS to provide some sort of simple string matching capability. Sometimes there are also mechanisms to relate words together as with synonym dictionaries and thesauri. Some words may be excluded as index terms and are placed in a special file often called a stop-word list.

Thus, although document retrieval is primarily concerned with the retrieval of a single record type, namely a 'document', other record types implicitly exist within a DRS. However the relationships between the record types are rigidly defined and their physical structure is predefined. An occasional exception is the principal record, the document, where the individual fields may be named by the system builder and where the number of such fields may be varied between applications. Furthermore, the ancillary records may only be retrieved in certain contexts, if at all. For example, there is often a separate 'browse' command for retrieving words from an index and often no way of retrieving words from a stop-word list.

As an illustration of a typical modern commercially available system, we will look at STAIRS, an IBM product (IBM). The 'classic' retrieval systems such as ORBIT (System Development Corporation) and MEDLINE (McCarn, 1978) provide little that is not available in STAIRS. Indeed BRS (Bibliographic Retrieval Services), another popular system, is very similar to both of these, except for syntactic variations. It is implemented on top of STAIRS and is basically a simplified subset. STAIRS is not simply a retrieval system but rather provides capabilities for constructing databases and indexes. It includes a query language for subsequent retrieval operations. More interestingly, STAIRS is claimed by some to provide a database management system, although it is, at best, a somewhat primitive example of the genre.

A STAIRS document contains two types of information.

1. Information for which an index can be constructed and which corresponds to the main text of a document.
2. Ancillary information which corresponds to what might be called the attributes of the document, such as date of publication, number of pages and so on.

These attributes tend to be numeric though they need not be. These pieces of information are termed 'formatted fields'. Since a document rarely consists of a single piece of text, STAIRS provides the ability to name 'paragraphs' within the text. The word paragraph is slightly misleading in this context however, since the things that are named are usually distinct sections of the document such as the abstract, the authors, references and so on. Thus basically a STAIRS document consists of a set of fields of which some are considered to be free text and others to be formatted information.

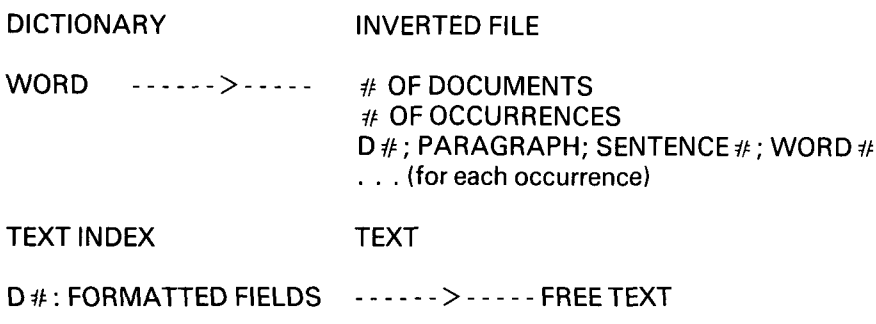

FIG. 1. STAIRS structure

Figure 1 illustrates the internal structure of STAIRS. A utility program is used to build the database. The resulting structure not only associates words with documents but also keeps track of the positions of words within a document and the number of occurrences of the word. This latter statistic enables probabilistic searches to be performed where the documents retrieved are ranked according to the probability of their being relevant to the original query terms. The DICTIONARY contains a list of every unique word in the text. Associated with every word is an entry in the INVERTED FILE containing a count of the number of documents containing the word; a count of the total number of occurrences in the database; and for each occurrence a specification as to where the word occurs in terms of document number, paragraph, sentence and position within the sentence. The TEXT INDEX contains the formatted fields together with a pointer to the location of the text which is contained within the TEXT FILE. Only one document type is permitted in a particular database. The DBMS-like behaviour, which is sometimes over generously attributed to STAIRS, derives from the ability of the system to perform retrieval based on the values in the formatted fields.

Corresponding to the three major data structures, the dictionary plus inverted file, the text index and the text, STAIRS provides three separate modes of retrieval: SEARCH for the dictionary file, SELECT for the text index and BROWSE for the text (additionally, the text index may be BROWSEd but not the dictionary).

Retrieval using the free text words is performed by entering SEARCH mode. Once in this mode, the usual Boolean queries may be entered. Each query is preceded by a statement number which can then be used to reference the results of a search. A term can refer to a set of words with the same stem by terminating the stem by a '$'. For example,

```
SEARCH
1: INFORMATION AND RETRIEVAL
2: 1 AND INTERACT$
```

This is the same basic capability found in most commercial systems. Additionally, a limited type of phrase search is permitted, to take advantage of the fact that the positions of words are recorded in the inverted file. Another capability is that of specifying the paragraph in which the word should occur. For example, to retrieve documents containing 'information retrieval' in the paragraph called 'TITLE', we would type:

3: INFORMATION.TITLE ADJ RETRIEVAL.TITLE

The word positions are also used in highlighting search terms in the retrieved documents when they are displayed.

Consider, as a further example:

4: INTERACTIVE INFORMATION RETRIEVAL

This would retrieve all documents containing at least one of these terms. (A missing operator is interpreted as 'OR'.) Alternatively searching can be done in 'RANK' mode.

RANK
5: INTERACTIVE INFORMATION RETRIEVAL

In this last example, the retrieved documents would be ordered according to a ranking algorithm based on the frequency of occurrence of the words in each document. (There are actually five ranking algorithms from which the user must choose.) Strangely enough, despite having a BROWSE command, STAIRS only allows browsing through the dictionary while in search mode and then by means of a special 'sub-query', called 'root', as for example, writing 'root x to z', which would list all dictionary words alphabetically contained with 'x' to 'z'.

To retrieve using the formatted fields, it is necessary to switch to SELECT mode. Here a set of relational operators may be used to qualify the values of particular fields of retrieved documents as in the following examples:

6: 1 DATE > 1980
7: 6 TYPE EQ PAPER AND PAGES < 30

SELECT mode is normally, but not necessarily, applied after search mode.

Entering the BROWSE command allows documents to be displayed. It can be used to display the results of a search, pages within a document or any document, identified by its position, within the database. For example, to list the result of the last search we would type:

BROWSE 7 ALL

The 'ALL' specifies that all fields are to be displayed.

MEDLINE (McCarn and Leiter, 1973; McCarn, 1978) contains some interesting features not found in STAIRS. In particular it permits an hierarchical dictionary so that it is possible to have index terms organized according to their 'breadth'. However, the hierarchical classification is biased towards biomedical documents and is not well integrated into the normal search mechanism. Of more interest is the

ability to perform string searches on documents already retrieved. This allows phrase searching for example. It is a more general though less efficient mechanism than the 'ADJ' operation of STAIRS.

This brief description of STAIRS is intended simply to illustrate typical characteristics of retrieval systems. The syntax is extremely simple and the types of operations which can be carried out are very limited, although less so than in other systems. Facilities are provided to make iterative searching straightforward. An essential point to note is that STAIRS differentiates very strongly between formatted fields and free text. Each is stored differently and each is accessed using a different set of commands. It is this lack of integration that has led to the separate development of DRS and DBMS and it is the basic problem to be solved if a satisfactory unified system is ever to be evolved.

3. DRS IN THE CONTEXT OF THE THREE MODELS

As we noted earlier, databases generally consist of collections of different types of records. One of the major tasks of a DBMS is to maintain relationships among the different types. To this end, various *models* have been developed and each DBMS is constrained to a particular model. The development of these models has been influenced by the extent to which they reflect reality although they all tend to be somewhat artificial. At the present time there are three major models: hierarchic, network and relational. Respectively, these see the world organized as a tree, a graph and tabularly. In setting up a database for processing within a DBMS based on one of these models, it is necessary to describe the relationships among the various record types so that they correspond to the underlying model. This description is called the *schema.* (Actually, there are several levels of schemata, for example *see* Date, 1981: 17–24. For the sake of simplicity, we will assume the existence of only one.) As the models correspond to particular types of data structure, languages have developed which are suited to the particular underlying data structures. Such languages are called *data sublanguages.* Sublanguage operations required to process one database are often quite different from those required to search another based on a different model. Sublanguages are normally embedded within a host language and are used procedurally. At a higher non-procedural level, we have *query languages* which, in principle at least, can be less influenced by the data model and in some cases approximate to natural language queries.

We will now look at each of the three models and attempt to evaluate the extent to which each is appropriate to DRS implementations. Two of these, the relational and network models, have already been applied in this context (Dattola, 1979; Macleod, 1979, 1981; Crawford, 1981).

As we discuss each of the three models, it is helpful to consider a simple bibliographic database. The data to be stored consist of the following fields for each document:

1. Document number
2. Title
3. Author*
4. Journal
5. Date
6. Keyword*

Fields marked with an asterisk are possibly repeating fields for any particular occurrence. Conceptually, we could view the database as consisting of records of all one type, i.e., records having the above fields. However, it is convenient in each of the models to separate certain of the fields into records of different types. In an actual bibliographic retrieval system, there are many more fields involved, such as journal volume and number, but these contribute nothing new in a conceptual way for our purposes.

One field that would present further problems is the abstract (or the actual text of the article). Current DBMS do not permit the kind of operations that would be necessary to, for example, check for the occurrence of a particular word within a field of text. For our consideration of the models, we can simply allow for the possibility that what we are referring to as keywords may have been automatically derived from the abstract of the document.

3.1 The relational approach

Intuitively, a *relation* is an unordered two-dimensional table in which each row represents a tuple and no two rows are identical (Codd, 1970). The columns of the table are called *attributes.* Each attribute is based on a *domain*, which is the set of possible values for the attribute.

It has been observed that certain collections of relations have better properties in updating than do other collections containing the same data. The theory of *normalization* provides a rigorous discipline for the design of relations that have favourable update properties. A series of normal forms for relations has been defined. Fundamental to this normalization is the notion of functional dependency. For a relation R, attribute B is said to be *functionally dependent* on attribute A if, at every instant of time, each A value in R has no more than one B value associated with it in R.

A relation R is in third normal form if it is in first normal form (is a flat table) and, for every attribute collection C of R, if any attribute not in C is functionally dependent on C, then all attributes in R are functionally dependent on C. Intuitively, we can describe the idea of *third normal form* as expressing the idea that each relation should describe a single concept or entity. (Higher normal forms have been defined, but are beyond the scope of this discussion.) Figure 2 shows a collection of third normal form relations for our example bibliographic database.

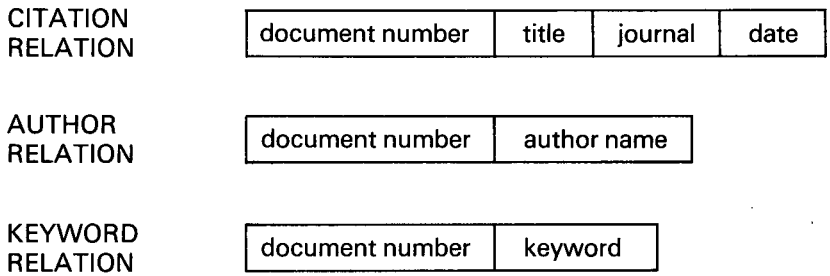

FIG. 2. Third normal form relations for bibliographic data

Several classes of languages have been defined for relations: relational calculus, relational algebra, mapping-oriented languages, and graphics-oriented languages (Chamberlin, 1976). One advantage of the relational model is seen in each of these

languages. That is, that conceptually symmetric queries are expressed and processed in a symmetric way. Thus, queries requesting 'all documents by a particular author' and 'all authors of a particular document' are handled similarly.

Further, we note that all associations between tuples (in different relations) are represented solely by data values in columns drawn from a common domain. That is, there are no external links; all associations are expressed by the values in the tables. In our example, authors and citations are related by the document number attribute (which is obviously on a common domain in each relation).

The advantages of the relational model for information retrieval may be summarized as follows.

1. The *simplicity* of the relational model is compelling. The user is able to view all aspects of the system in a clear, simple, coherent way.
2. There is a strong *theoretical basis* for the relational model. It is based on the mathematical theory of relations and on the first-order predicate calculus.
3. There is *consistency* of access to all information from the user point of view. Under other organizations, certain questions may be awkward to ask or require special language constructs.
4. The relational model permits a high degree of *data independence.* It is possible to eliminate the details of data storage and access methods from the user interface.

A further advantage, referred to above, relates particularly to the relational model as it applies to document retrieval. It is *natural* to view such structures as document collections, indexes and dictionaries as two dimensional and therefore as relations.

3.2 The network approach

Network data models are based on tables and graphs (Date, 1981: 70–73). The nodes of a graph usually correspond to the entity types, which are represented as tables. The arcs of a graph correspond to relationship types, which are represented as connections between tables. A comprehensive specification of a network data model has been published in the report of the Data Base Task Group (DBTG) of the Conference on Data Systems Languages (CODASYL) (CODASYL, 1971; Date, 1981: 387–446). The following discussion relates to this DBTG model.

In the DBTG-network data model, the data are represented by *records* and *links.* Record types are used to represent entity types and to specify the generic structure of the tables. Links are used to specify the relationship types and to specify the generic connections between record types. Links between record types in a graph must be functional. A graph representing record types and links which abides by this restriction is called a *data structure diagram.* Figure 3 shows a data structure diagram for our example bibliographic database.

Here nodes represent record types, and arcs represent functional links. In the DBTG-network data model these are called *set types.* In our example, we have been able, indirectly, to handle many-to-many relationships. For instance, in the case of Authors and Documents, we have introduced an intermediate record type (Doc/Auth) and two functional links. The many-to-many relationship of documents and keywords is handled similarly.

This structure permits symmetric queries to be handled in a symmetric manner. Thus, 'all documents by a particular author', and 'all authors of a particular

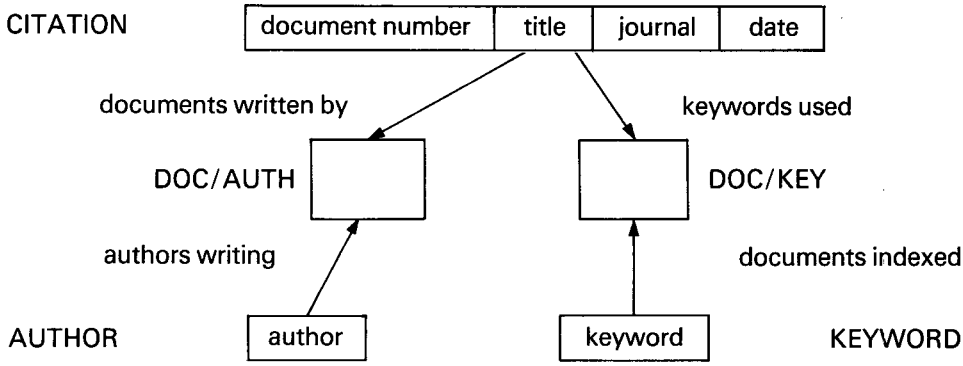

FIG. 3. Data structure diagram for bibliographic database

document', are processed similarly. However, the processing, involving the intermediate record type in each case, involves undue complexity. This is an inherent problem with the DBTG-network data model.

Further problems with this model involve the determination of which strategy may be best for handling a particular query. For instance, consider the query, 'Is Smith an author of document number 1009?'. To answer this we must determine whether there is an intermediate record occurrence between the document record for document number 1009 and the author record for Smith. There are two strategies for locating this occurrence, one that starts with the document record, and one that starts with the author record. How is the decision made as to which strategy to adopt?

In summary, it would seem that the network model, while providing facilities for modelling complex data, introduces unnecessary complexity as a basis for a DRS.

3.3 The hierarchical approach

As for the network data model, the hierarchical data model is a restricted case of a graph data model (Date, 1981: 275–386). However, for the hierarchical data model, even further restrictions are imposed. Stated most simply, the arcs in the data structure diagram representing an hierarchical database must form an ordered tree. This restricted data structure diagram is called a *definition tree.* A definition tree for the example bibliographic database is shown in Figure 4.

A node in a definition tree corresponds to an entity type. It is called a record type, is labelled, and is composed of one or more data items. Links are not labelled since they are necessarily unambiguously determined by their *parent* and *child* nodes.

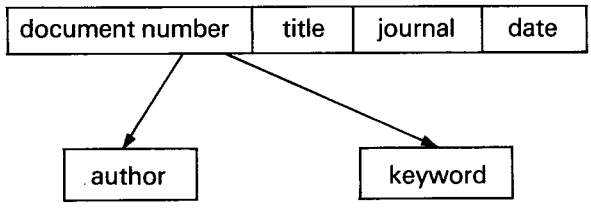

FIG. 4. Definition tree for bibliographic database

There are a number of difficulties with the hierarchic data model. Consider the queries for obtaining 'all authors of a particular document' and 'all documents by a particular author', which are apparently symmetric. The hierarchic structure leads to a loss of symmetry in the procedures required to answer these two types of request. This asymmetry, which is directly a function of the hierarchical structure, leads to unnecessary complications for the user.

Admittedly, an example in which the data are more naturally hierarchic would show the advantages of this model. In DRS, such structure occurs in a thesaurus and in a clustered document collection. Figure 5 shows the definition tree for clustered documents.

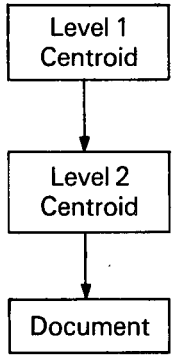

FIG. 5. Definition tree for clustered documents

Even when the hierarchy is natural, there is asymmetry in processing symmetric queries. Furthermore, many-to-many relationships, which are common, are not handled by the hierarchic model. In general, it is not a very appealing model to use as the basis for a DRS.

4. A COMPARISON OF DBMS AND DRS

While DBMS are based on rigid models, information retrieval systems follow rather *ad hoc* designs. This is principally because of the traditional practice of applying document retrieval within a single class of records. Since there is only one user-generated record type, there is no concept of a schema in a DRS. Further, because of the highly iterative way in which queries are developed in response to partial information, a great deal of effort is often expended on optimizing response times at the expense of data independence. In particular, the main index is often closely integrated with the actual documents of the database. This can be seen in the structure of STAIRS.

An advantage of a DBMS is that it provides for some degree of symmetry among the record types. The actual degree will depend upon the model chosen. Major implications of this are that all information can be retrieved and that the same query language is used to retrieve records of different types. On the other hand, the more general record retrieval capabilities of a DBMS require a much more sophisticated query language. A DBMS is a much larger and more complex piece of software than a DRS. This is one reason why they have been rarely considered as a vehicle for a DRS application. Other reasons include the lack of string matching, since DBMS

generally assume the existence of only 'well behaved' data, and the lack of simple mechanisms for carrying out iterative searches, the most common form of retrieval in a DRS.

We can identify five major components of a DBMS.

1. An underlying set of data structures.
2. A data model describing how the data structures are to be viewed.
3. A schema describing, in terms of the model, how an actual database is organized.
4. A data sublanguage allowing the database to be accessed from a procedural language.
5. A query language to process non-procedural queries.

The corresponding elements in a DRS are:

1. A single monolithic data structure.
2. None.
3. At best, a mechanism for specifying the number of fields in each document, and the names of each.
4. At best, a package to generate indexes.
5. A simple, easy to use, though somewhat inflexible query language.

Because of the way in which users retrieve information from a DRS and because the information is usually quite limited, no need has been foreseen for constructing a DRS around any particular model. Rather a data structure applicable to a specific data organization is built and a quite rigid query language is developed to access this data structure. Some systems, notably STAIRS, permit the database administrator to run a utility package to generate indexes. More commonly, systems assume that indexes have been generated externally prior to database creation. Because of the rigid nature of the underlying data structure, indexes must adhere to a rigidly predefined format so that it is not easily possible to allow users to construct their own index-building programs within the system.

In summary, DRS have many defects. Data structures are rigidly predefined and only one type of document can be handled within a particular database. The generation of internal data, such as indexes, is under limited user control. Query languages, while simple, have a very limited scope. DRS are usually stand-alone systems. That is, they cannot be accessed from any other system. In particular there is no interface to a procedural language to permit highly specialized processing operations.

The real fundamental difference between the DBMS philosophy and that of DRS is the idea of dynamic relationships. This is most prominent in the relational model where relationships are really established by the query language, but are also basic to all models. Relations can be established by the schema and are not predetermined by the physical storage organization. Furthermore, because the schema serves as the interface between the database and the query language, it is possible in a DBMS to have multiple views of the same data. That is, individual users or groups of users can have their own schema as long as it can be mapped into the basic schema. The main implication of this is that users can be restricted to certain fields of records and to certain types of records. Access to records may even depend on the values of certain fields of the records.

A further fallout from this difference is that updating is treated as a typical operation in DBMS whereas in DRS updating existing document records is not normally permitted other than through a regeneration of the database.

Thus it can be seen that in DBMS the emphasis is on *flexibility* after the database has been created. This is a natural evolution because of the amount of data involved and because of the number of relationships. It simply is not practical to regenerate the database every time some relatively minor view of it changes. Nor is it practical to provide physically separate subsets for differently authorized user groups, partially because of space requirements but, more importantly, because of the problem of handling updates.

The collection, indexing and entry of documents into machine-readable form is a resource-consuming task. With the exception of systems such as SPIRES (SPIRES, 1974), the designers of current DRS do not seem to have foreseen the possibility that individuals, or small groups of individuals, would wish to build and maintain their own databases. Note that there is a 'chicken-and-egg' situation here. It is sometimes claimed that it is not appropriate to implement a DRS within a DBMS because there is no need for the overhead of providing sublanguage and schema facilities. On the other hand, it could be said that DRS applications do not make use of such facilities because they do not exist, given that existing DBMS lack features which are essential for a DRS. However, with the increasing availability of computer technology, it is at least physically easier to enter small databases than once was the case and often they are available as a fallout of other data-handling operations. Furthermore, with the now almost universal use of time-sharing systems, computer terminals are much more accessible. Individuals now find it convenient to maintain computerized collections of records. The needs of these individuals are not met by existing DRS. (Nor for that matter are they met by existing DBMS. It is not uncommon to find individuals maintaining files of titles and using the string-matching abilities of text editors to perform retrieval.) The price of flexibility is a complex user interface and a large piece of software which is both expensive to acquire and to maintain. Furthermore there remains the one important respect in which DBMS are generally less flexible than DRS. This is in string handling. Most DRS permit some limited form of pattern matching to allow searches for partially specified strings. This is particularly necessary when working with natural language where spelling errors, spelling variants and grammatical forms are common.

In summary, the disadvantages of DBMS over DRS are their general inability to handle strings together with their overall complexity. The latter relates both to the software complexity and to the difficulty of phrasing simple iterative searches in a natural way within a complex query language. If these two main difficulties can somehow be avoided, then it would be much more attractive to implement a DRS on top of a DBMS. Note that within the DBMS context index creation can be handled quite cleanly by accessing the index-creation procedures through the data sub-language.

5. IMPLEMENTATION OF A DRS WITHIN A DBMS

A number of proposals have been made for the integration of document retrieval capabilities into a database management system. One suggests the network model (Dattola, 1979), and two others suggest the use of the relational model (Macleod, 1979; Schek and Pistor, 1982). Dattola does not appear, however, to have functionally integrated DRS and DBMS. What he suggests is that a DBMS be used to

store and manage documents indexed under very broad categories. More refined retrieval is provided using a separate retrieval system. Retrieval is seen as a two-stage process:

1. Documents are retrieved from the database using the broad document descriptors.
2. A SMART-like (Salton and McGill, 1983: 118–156) retrieval system is used to abstract relevant documents from this retrieved set.

While Dattola has used a network model for the original document storage, it does not appear that the network model was chosen for any reason other than availability.

Macleod (1979), on the other hand, proposes a single integrated approach. He uses the relational model and shows how many DRS operations can be carried out reasonably comfortably within it. The query language used is SEQUEL (or SQL) (Chamberlin and Boyce, 1974), and some suggestions are made for enhancements to the language in order to improve its suitability as a vehicle for DRS. The most important of these is an extension which would permit literal string values to be replaced by patterns. Another proposed extension is at the cosmetic level where a macro-facility is suggested in order to hide some of the less elegant SQL constructs.

A problem with the relational model derives from the concept of normalization. (Crawford, 1981, provides a more complete discussion of this aspect.) A logical collection of data does not necessarily correspond to a single tuple in a relation. For example, a data collection might consist of titles, authors and index terms, where a title may be associated with more than one author and more than one index term. A logical record would take the form:

Title; Author . . . ; Term . . . ;

As it stands, this type of record does not correspond to a tuple in a relation since it contains non-atomic values. The record must first be normalized so, for example, the record:

Title; A1, A2; T1, T2, T3;

would be represented by six tuples of the form:

Title; A1; T1;
Title; A2; T1;
Title; A1; T2;
Title; A2; T2;
Title; A1; T3;
Title; A2; T3;

There are still problems with this representation, particularly in updating, caused by the duplication of information within the relation. A preferable solution is to represent this information by a number of relations, as for example:

TITLES (Document #, Title)
AUTHORS (Document #, Author)
TERMS (Document #, Term)

So we would have:

TITLES:	AUTHORS:	TERMS:
D1; Title	D1; A1;	D1; T1;
	D1; A2;	D1; T2;
		D1; T3;

The number of tuples is not increased and in fact they occupy less space. The problem now is that the information has become scattered across three separate relations. The logical relationships among the separate tuples, maintained by the introduction of the new domain, Document #, are no longer obvious so that the conceptually simple act of retrieving what the user sees to be a single document may well entail join operations across a number of relations. (This problem can be partially solved by implementing an artificial relationship, sometimes called a view, which allows a group of physically separate relations to be viewed as a single relation. For example, SEQUEL takes this approach.)

Schek and Pistor (1982) too have suggested a relational-like approach. However, they have made somewhat more radical modifications to the original model. They make the not unreasonable suggestion that the unnormalized view is the natural one. Whereas Macleod has suggested hiding the explicit use of joins by using macros, Schek and Pistor suggest the retention of the original unnormalized information. In particular they suggest that unnormalized relations be permitted.

For example, suppose we want to list all titles by author A1 described by terms T1 and T2. In SQL we would write:

```
SELECT TITLE
    FROM TITLES, AUTHORS, TERMS X, TERMS Y
    WHERE AUTHOR = "A1"
    AND AUTHORS.D# = TITLES.D#
    AND TITLES.D# = X.D#
    AND X.D# = Y.D#
    AND X.TERM = "T1"
    AND Y.TERM = "T2"
```

The same query in STAIRS would be:

```
SEARCH
1  A1.AUTHORS AND T1.TERMS AND T2.TERMS
BROWSE 1 TITLE
```

In Schek's model it would be:

```
SELECT TITLE
    FROM DATABASE
    WHERE TERMS ⊆ {"T1", "T2"}
    AND AUTHORS ⊆ {"A1"}
```

Of the three queries, the first is clearly the least attractive while there is not a great deal to choose between the other two. The Schek model also handles adjacency of terms. The following query retrieves titles from documents containing T1 followed by T2.

```
SELECT TITLE
    FROM DATABASE
    WHERE TERMS ⊆{<*, "T1", "T2", *>}
```

Here the angle brackets denote a list of consecutive objects and the '*' is a 'match anything' symbol. Schek and Pistor also provide 'nest' and 'unnest' operators to transform relation from normalized to unnormalized and vice versa. What effectively they are doing, is providing a mechanism for transforming a set of attribute values into a string and then providing limited string or template-matching operations. A separate indexing mechanism is provided for the strings based on fragment indexing (Schuegraf and Heaps, 1976). In our earlier discussions we noted that STAIRS attempts the integration of text and formatted fields within a single system, with only moderate success. Schek's model represents a much more elegant solution to this problem.

6. THE IMPACT OF EFS

As we noted earlier, there are at least three classes of information system. Currently there is some activity in integrating the functionality of DRS and DBMS. However, it is likely that EFS will become ever more important with the spread of work stations. It would be foolish at this point of time to consider the integration of DRS and DBMS without also considering the likely impact of EFS. In an office environment both DBMS and DRS would seem to be potentially applicable. There is a great deal of well-structured data, as for example, the contents of forms. At the same time there are usually significant quantities of unstructured material such as letters, proposals, reports, and so on.

In our earlier discussions we advanced the view that of the three DBMS models, the relational model has the greatest potential as a vehicle for a DRS implementation. While the relational model has at first glance a number of appealing characteristics in comparison with the other models, there are a number of problems associated with it. Some we have already seen. Others occur when we come to consider the integration of EFS. There is no natural concept of hierarchy within relations. In many collections of data, hierarchies arise naturally. Filing systems are an example of such a system. This problem has been recognized for some time and some suggestions for overcoming it have been made by Codd (1979) in his extended relational model. However, the extensions do not map into natural tree operations.

A more general problem which is not specific to the relational model is that only a single type of record can be retrieved. This is not as bad as the DRS situation where only a single type of document can normally be stored in an active database, and it is not in general a severe restriction in typical database applications. However, in a conventional office environment it is normal to construct files containing documents of many different types. Document types might, for example, include letters, forms, résumés, reports, papers, and so on. These different documents may be brought together in related groups (e.g., envelopes, files, in-trays, stapled bundles). In addition to the external documents, internal documents will be created to describe and manage groupings (e.g., named file folders, circulation lists, calendars, envelopes). These internal documents may well become useful documents in their own right and there may be no logical distinction between the two.

People generally retrieve office documents in one of two ways. One is by a search

of the entire collection (usually facilitated by various indexing aids). The other is by accessing the internal documents used to manage groupings and performing a search within these (e.g., search within a file folder). Groups are often organized hierarchically (e.g., filing cabinets, drawers, file folders, stapled documents). This mode of operation is closely parallel to the way in which people retrieve information in a library environment. Documents are often hierarchically classified and people normally search within a particular classification. A number of DRS models have been developed along these lines (van Rijsbergen, 1979; Salton and McGill, 1983).

A model, albeit a somewhat imprecise one, has been suggested to handle these problems of hierarchy and type collection (Barnard and Macleod, 1982). It is based upon the idea of permitting the values of attributes in a relation to themselves be the names of relations. Somewhat similar ideas have been independently expressed by Schmidt and Jenkins (1982), who have developed a new model, not totally dissimilar to Schek's, but based upon array theory (More, 1975). A great deal of work, however, remains to be done in this area.

7. FUTURE TRENDS IN DOCUMENT RETRIEVAL

It is useful at this stage to summarize what we mean by a model. It is a set of object types, operators and integrity rules. The object types define the logical structure; the operators define what operations or transformations can be performed on the object types; and the integrity rules define any additional constraints on the data (such as the constraint of uniqueness of tuples in the relational model). There are many different models. It is probably safe to say that there is, for example, no single relational model but rather a collection of models based on the original relational model devised by Codd (1970). The SQL model, for example, differs from the original model in that its integrity rules permit duplicate tuples within a relation.

One of the reasons there is not only no single model, but rather a plethora of models, is presumably that no existing model is ideally suited to every database application. Nor is there any likelihood of such a model appearing in the near future. Criticisms can, and have, been made of all the models described earlier. Possibly a more promising approach than the pursuit of an elusive universal data model is to look at the underlying data structures and devise a common data representation which can be used for a variety of models. This approach has already been suggested by Date (1980) in his Unified Database Language. He has illustrated the feasibility of using the same data structures for hierarchical, network and relational implementations. However, it should once again be emphasized that any data model suited for free text must be capable of performing a range of string-matching operations. The development of the inverted index may have improved the operational efficiency of retrieval but a price has been paid in terms of the limited string-matching capabilities now generally provided.

Current DRS are typified by systems such as MEDLINE and BRS. These types of retrieval systems seem to be used primarily in conjunction with particular databases. A few more flexible systems, such as SPIRES, allow users to build and manage their own databases easily, but they are not widely available. This is probably one of the main reasons there has been little development into data models suited for textual data. A real need for future DRS development will be the availability in the marketplace of systems incorporating new ideas. Another motivation for the introduction of such systems would seem to be the increasing availability of machine-readable documents. There are two aspects to DRS.

1. The retrieval system itself.
2. The collection, indexing and general cleaning up which must be performed before a usable database can be obtained.

While it is premature to predict the demise of hard-copy communications, there is no doubt that the transmission of documents via electronic mail is increasing rapidly. There are many electronic mail systems, and many computer networks, for example ARPANET. With the corresponding decline in mass-storage devices, it is rapidly becoming both convenient and cost effective to maintain machine-readable copies of documents. With the ready availability of such documents, systems are going to have to evolve to facilitate the storage, filing, cataloguing and retrieval of a wide variety of documents of different types. Furthermore, a need can be seen for both large centrally maintained databases and small local systems, possibly interconnected. One of the major challenges of the next few years will be the design of fully integrated information systems which will be able to handle this type of information processing.

REFERENCES

Atkinson, M. P. (1979) Progress in documentation: Database systems. *Journal of Documentation 35*, 49–91.

Barnard, D. T. and Macleod, I. A. (1982) A methodology for the development of office information systems. *Proceedings of Session 82, The National Conference of the Canadian Information Processing Society.* pp. 127–134.

Bibliographic Retrieval Services, Inc. *BRS Bulletin*, Schenectady, New York.

Chamberlin, D. D. (1976) Relational data-base management systems. *Computing Surveys 8*, 43–66.

Chamberlin, D. D. and Boyce, R. F. (1974) SEQUEL: A Structured English Query Language. *Proceedings of the ACM-SIGMOD Workshop on Data Description, Access and Control.* pp. 249–264. New York: Association of Computing Machinery.

CODASYL Report. (1971) *Data Base Task Group of CODASYL Programming Languages Committee.* Report.

Codd, E. F. (1970) A relational model of data for large shared data banks. *Communications of the ACM 13*, 377–397.

Codd, E. F. (1979) Extending the data base relational model to capture more meaning. *Australian Computer Science Communications 1*, 5–48.

Crawford, R. G. (1981) The relational model in information retrieval. *Journal of the American Society for Information Science 32*, 51–64.

Crawford, R. G. and Macleod, I. A. (1978) A relational approach to modular information retrieval systems design. *Proceedings of the 41st Conference of the American Society for Information Science 15*, 83–85.

Croft, W. B. (1982) An overview of information systems. *Information Technology: Research and Development 1*, 73–96.

Date, C. J. (1980) An introduction to the Unified Database Language (UDL). *Proceedings of the 6th International Conference on Very Large Data Bases.* pp. 15–32.

Date, C. J. (1981) *An Introduction to Database Systems.* (3rd edn) Reading, Ma.: Addison-Wesley.

Dattola, R. M. (1979) FIRST—Flexible Information Retrieval System for Text. *Journal of the American Society for Information Science 30*, 29–35.

IBM World Trade Corporation. *Storage and Information Retrieval (STAIRS) Reference Manual.*

Macleod, I. A. (1979) SEQUEL as a language for document retrieval. *Journal of the American Society for Information Science 30*, 243–249.

Macleod, I. A. (1981) A data base management system for document retrieval applications. *Information Systems 6*, 131–137.

McCarn, D. B. (1978) Online services of the National Library of Medicine. *Proceedings of the 17th Institute of Electrical and Electronics Engineers (IEEE) Computer Conference.* pp. 48–53.

McCarn, D. B. and Leiter, J. P. (1973) On-line services in medicine and beyond. *Science 181*, 318–324.

More, T. (1975) *A theory of arrays with applications to databases.* Cambridge, Ma.: IBM Scientific Center (Technical Report G320-2106).

Salton, G. and McGill, M. J. (1983) *Introduction to Modern Information Retrieval.* New York: McGraw-Hill Book Company.

Schek, H. J. and Pistor, P. (1982) Data structures for an integrated data base management and information retrieval system. *Proceedings of the Eighth International Conference on Very Large Data Bases.* pp. 197–207.

Schmidt, F. and Jenkins, M. A. (1982) *Data systems design: The Nial approach.* Kingston, Canada: Department of Computing and Information Science, Queen's University (Technical Report).

Schuegraf, E. J. and Heaps, H. S. (1976) Query processing in a retrospective document retrieval system that uses word fragments as language elements. *Information Processing and Management 12*, 283–292.

Slonin, J., Maryanski, F. J. and Fisher, P. S. (1978) Mediator: an integrated approach to information retrieval. *Proceedings of the International Conference on Information Storage and Retrieval.* pp. 14–36.

SPIRES User's Manual. (1974) Stanford University.

System Development Corporation. *Highlights and Hints: ORBIT System Development Corporation Search.* Santa Monica, California.

Van Rijsbergen, C. J. (1979) *Information Retrieval.* (2nd edn) London: Butterworths.