An integrated fact/document information system for office automation

Esen A Ozkarahan and Fazli Can*

After a review of the needs of office automation, an integration model of an experimental system for testing various concepts is presented. This system aims to synthesize a relational database management system (DBMS) and document retrieval system (IR) capable of context sensitive full text searches. The IR system relies on a clustering subsystem for database partitioning and relation fragmentation. Conceptual data modelling, forms interface, and query execution in the distributed database constitute the support architecture of the integrated system. Underneath the support architecture, there is the integrated DBMS/IR datastructure and instruction set. Efficiency of the integrated system is discussed including a future extension that involves the RAP 3 database machine hardware. RAP 3 combines the term matcher and query resolver subsystems.

Keywords: computer networks, database management systems, database machines, distributed databases, document clustering, document retrieval, full text searching, integration of DBMS/IR, office automation

INTRODUCTION

The need to integrate various facilities of information processing is generally well recognized. This need is expressed more profoundly as the advances of the present-day computer technology are making things that could not be done in the past a reality today. The network information system foreseen by Salton[1] is described as a network of different information facilities such as the document (or information) retrieval (IR), database management (DBMS), data analysis, citation indexing, and text processing systems. Van Rijsbergen[2], in his statement of the future research problems, foresees the IR and DBMS integration and stresses the fact that such an integration will be driven by the needs of office automation. Also, there have been various studies in the DBMS area to augment a DBMS with some IR functionalities[3, 4, 5]. In the IR area, there are commercial and experimental systems such as DIALOG, STAIRS, BRS, MEDLARS, MEDLINE, ORBIT, The Information Bank, LEXIS, WESTLAW, SMART, etc., which are surveyed in[6, 7], that display the past and present state of the art in IR.

How to integrate IR and DBMS

As the users of the information processing community are provided with more capabilities, we see that the information structures and data manipulation requirements of applications are getting more and more sophisticated. This would mean that an information scientist responsible for data definition and creation will be faced with the representations of complex relationships of facts as well as documents. We can immediately realize that past efforts that simplified the integration problem by confining one system within the framework of the other or simply combining IR and DBMS in a semantically disjoint manner will not be sufficient, effective, or efficient within the context of the requirements of present applications. We cannot merely incorporate some string operations into a DBMS data language or store keywords as an attribute of a formatted database. We cannot implement DBMS

Department of Computer Science, Arizona State University, USA
*Department of Electrical Engineering, Middle East Technical University, Ankara, Turkey
Received: 12 March 1984

functionalities as file programs embedded within an IR system. Such an effort would lack the advantages of DBMS such as data independence, real time response, and *ad hoc* query formulation capability. On the other hand, the unique features of IR cannot be fitted into the deterministic, non-iterative nature of DBMS data manipulations.

The answer for the proper integration lies in a synthesis that will combine IR and DBMS by preserving their unique features in such a way that the advantages of both systems can be shared. In such a general purpose system, one would like to see conceptual data modelling, relational algebra power (or equivalent) of DBMS, and at the same time, automatic indexing and classification (clustering), partial and/or full text search, and query feedback features of IR.

Office automation

Office automation is a relevant target application that can tap the resources of an integrated information system — not only relevant but also with immediate prospects for exploitation in real life. We can define office automation as the pioneer of a (relatively) paperless society. This is because, with office automation, we are not merely automating manual systems of the present but introducing a new model for the office of the future. In such a model, we see electronic mailing, automatic forms handling, form driven database manipulation including full text processing. Because an automated office will embody workstations distributed throughout an organization, we can envisage a local network of workstations.

In the model of an automated office, we can describe the following: on the user resource network level, we have various modern office equipment linked to workstations. Workstations, in addition to their local intelligence, will be tied to network servers. User's interaction with the system will be via forms and electronic mail handling facilities. These facilities will be supported by a distributed environment of integrated information system onto which the forms processing system is mapped. More specifically, in this model, we see distributed databases in which DBMS and IR are integrated. For automated offices, besides formatted data of DBMS, processing of unformatted data in the form of full text searching is very important. The latter would fall within the realm of IR in which document classification, indexing and keyword based retrieval would be supported by the database server, whereas the full text processing capability would rest in workstations. In hardware description, a workstation can be a powerful microcomputer linked to the network server through the local network. The network server can be a powerful mini- or mainframe computer. One can think of cascaded local networks of such systems for larger environments.

The purpose of this research

The purpose of our research is to define a model of an integrated fact and document information system and to implement an operational experimental system in which various components of this model can be tested. We believe that the benefits of such an integration would be multifold. We cannot only gain insight into the integration issues and problems, but also experiment with the individual components of the system such as database design, network query execution, forms interface, and full text processing in conjunction with cluster based retrieval.

Because full text processing is important for specialized applications such as law, medicine, etc., we consider the inclusion of context sensitive, full text processing in the integrated information system model important. This does not preclude, however, keyword based document retrieval incorporating clustering at the network server level. In fact, in the proposed environment, the upper hierarchy of the unformatted retrieval rests within the database server.

As a tool for DBMS in our system, we will utilize the RAP relational DBMS (which will be introduced in the sequel) because of the following three important reasons:

- RAP relational DBMS goes beyond implementing string processing within DBMS. It supports associative, context sensitive, text search operations based on internal string processing as primitive commands of the DBMS.
- Besides its efficient software implementation, which can be compared with other existing systems, it has the option of being mapped onto the hardware of a database computer.
- Third but not least, it is the available facility which can easily be serviced and altered, if necessary, by ourselves.

In the remainder of this paper, we will present the model of the proposed integrated information system. In this model, we will see the abstract view of the query processing environment at various steps of processing from the input to the delivery of results. This will be followed by the description of the underlying support system structure. This involves the distributed database architecture, conceptual data modelling, and the handling of unformatted structures. Because of space considerations, we will try to keep the discussion at the descriptive level.

The support system structure will be followed by the description of the operational system that runs on the support system. The context sensitive full text operations using RAP will be demonstrated.

In the performance section, we will address the efficiency of the proposed integration model. While the performance of a pure software approach will reach a limit at some point, the potential performance improvements of text retrieval hardware will set the ultimate goal. This will be the topic of the final section.

INTEGRATION MODEL

In the integrated information system, a common framework has been established for the physical data-structure so that both formatted and unformatted data can be manipulated together under an instruction set that embodies DBMS as well as context sensitive, full text processing power. A given user request such as

> On the documents written by the authors who are referenced by the papers written by JOHN DOE after 1970, search for those with the phrase SEMANTIC DATA MODEL and AGGREGATION

would require both DBMS and IR specific operations in an interrelated sequence of processing steps (see below).

In the notation that will follow, a user query such as the one quoted will be denoted as Q, whereas the subsets of it dealing with DBMS and IR will be indicated by Q_R and Q_T respectively. Q_T is the portion of the request that deals with the context sensitive search such as the SEMANTIC DATA MODEL and AGGREGATION in the same sentence. While Q_T will be preserved for final processing in the workstations, a query called the system query Q_S will be derived from Q_T to initiate a search of the relevant clusters in the database. This search is carried out by the database server on the network in an effort to narrow the search space, before the full text search can be conducted.

In simple terms, a Q_S query will include those terms of Q_T that are included in the set of index terms of the collection with a possible inclusion of certain other components to increase system precision and/or recall. Figure 1 shows an abstract view of the query processing environment. With Q_S, a hierarchy of clusters will be searched and the corresponding data DQ_S will be returned when the cluster search is optimally terminated. The two resulting sets of data, DQ_R and DQ_S, corresponding to DBMS and IR operations, respectively, will be jointly processed by the integrated system to produce the user's response. The user, screening this response, may ask the system to repeat the operation by providing an indication of the relevant/irrelevant documents. With this information, Q_S will be modified and the previous cluster selection will be refined.

Mathematical model of query processing

The integrated information retrieval system, *IS*, consists of a set of hierarchies and is described by a 6-tuple:

$$IS = <R, D, Q, C, E, T>$$

where R represents a set of relations which contain structured data about the documents or document entities. The nature of the interrelationships among the structured entities will be described by means of a conceptual datamodel.

D represents the set of documents stored as unstructured entities. The relationships of these unstructured entities among themselves and/or with those of R also will be presented in the conceptual datamodel.

Q is a set of user queries. A given query, Q, is defined as

$$Q = \{Q_T\ Q_R\}$$

where Q_T is the part of the query that holds the context sensitive, document search specification.

Q_R is the part of the query that holds search specification on the $\{R, D\}$ set operable by the DBMS instruction set. As can be seen, the simplistic implication of $Q_R \rightarrow$ (IR on D) and/or (DBMS on R) is not always true. As shown in the example of the previous section, a search navigation threads through both R and D several times in an interrelated manner. This is an important point that shows that the integration is not achievable by a simple coexistence of two different systems in a semantically disjoint manner.

C is a hierarchical structure of clusters for D.

E is a mapping function, called the evaluation function, $E{:}Q \rightarrow 2^D$, used to find the relevant documents to a query. In reality, a subset of the range of this mapping is reached through the complex operations of clustering, building hierarchies of clusters, implementing search functions for these hierarchies, and using feedback. All these operations will be briefly discussed in the following.

T represents the terms used for the description of documents.

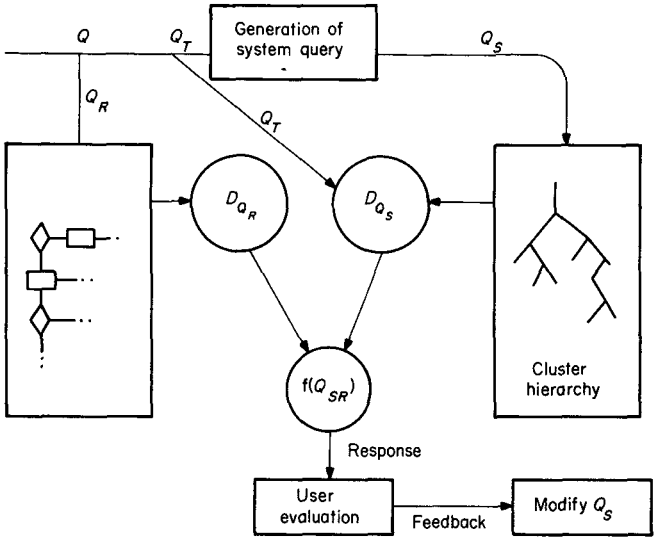

Figure 1. Abstract view of the query processing environment

Creation of system queries

As shown earlier, a query Q consists of two parts Q_T and Q_R where Q_R is the relational DBMS operable subquery and Q_T corresponds to the context sensitive, document retrieval subquery. Related with Q_T, there are a set of words, T_Q, such that:

$$T_Q \cap T \subseteq T_1, T_2, \ldots, T_n \text{ where } n = |T| \geq 0$$

We cannot expect all of T_Q to appear in the filtered search query (system query) for the simple reason that not all of the words would be included in T since there are conditions for a word to be an (index) term[8,9]. In Q_T, some terms will be used in the positive context, some others in the negative context, and some in both. For the reason that will be clear in what follows, Q_T, therefore, will be expressed as:

$$Q_T = Q_{TP} \cup Q_{TN}$$

where Q_{TP} and Q_{TN} correspond to the parts of Q_T that deal with the terms specified in the positive and negative context respectively, and

$$Q_{TP} \rightarrow |T_{TP} = T_1, T_2, \ldots, T_k|\ k \geq 0$$
$$Q_{TN} \rightarrow |T_{TN} = T_1, T_2, \ldots, T_l|\ l \geq 0, k, l \leq n$$
and $k = l$ is allowed.

In the filtered search query for Q_T, which will be called a system query Q_S, there may be positive and negative term specifications. If a term T_i is a member of both sets, that is:

$$T_i \in T_{TP} \text{ and } T_i \in T_{TN}$$

then T_i appears as a positive term in the system query. This selection is due to the fact that the information provided by the appearance of a term rather than its non-

appearance is more important[10]. Therefore, a system query, Q_S, is:

$Q_S = Q_{SP} \cup Q_{SN}$ where
$Q_{SP} = Q_{TP}$
$Q_{SN} = Q_{TN} - (Q_{TN} \cap Q_{TP})$

Hence, for the respective terms:

$$|T_{SP}| = |T_{TP}|, \ |T_{SN}| \leq |T_{TN}| \text{ and } |T_S| \leq |T_T|$$

This is reasonable since, as stated earlier, one cannot expect all the words used in a context sensitive search request to appear in the terms used for indexing.

Context sensitive Boolean query structure

The user query Q will be processed as follows:

- The query Q will be converted into a disjunctive normal form, i.e.,

  $$Q = (Q_{11} \Lambda Q_{12} \Lambda \ldots \Lambda Q_{1n_1}) V \ldots V (Q_{m1} \Lambda Q_{m2} \Lambda \ldots \Lambda Q_{mn_m})$$

  where each $Q_{ij} (i = 1, \ldots, m; j = 1, \ldots, n_m)$ may be a single word or a context sensitive, document retrieval operation of one of {*(A, B) in Sentence, A... B, A.n.B,* $<A, B>n$, *A???B, A*B, ...*} as will be discussed later. Furthermore, each Q_{ij} may be either in a positive or negative context (e.g., $\neg$ *(A, B) in Sentence* implies *(A, B)* $\neg$ *in Sentence.)*
- From the resulting query, a list of subqueries will be generated such that they will be in an ordered quadruple $< Q_R, Q_T, Q_S, Q_{SR}>$ where Q_R and Q_T are as explained before and Q_S and Q_{SR} are obtained in the following steps.
- The words used in Q_T will be searched in the thesaurus corresponding to the terms used in the clusters. Non-matching words will be dropped and the query, now left with only the terms, will be a system query Q_S.
- Concept hierarchies, continuous word phrases, and/or citation linkages will be the possible candidates of additions to Q_S if expansion due to recall and precision will be necessary in the course of the experiments.
- Q_{SR}, referring to Figure 1, is a further retrieval operation on DQ_R and DQ_S and the answer set returned by it can be expressed as:

  $$f(Q_{SR}) = f(\Pi DQ_R f_{QT}(DQ_S))$$

  In other words, $f(Q_{SR})$ is the data returned by a further retrieval function operating on the product of the datasets DQ_R and $f_{QT}(DQ_S)$, which correspond in turn to the data returned by Q_R and the context sensitive operations of Q_T executed on the dataset returned by the hierarchical cluster system search.
- Q_{SR} is repeated through the feedback loop (if necessary) until the user and/or certain performance indicators are satisfied.

In the above procedures, $f(Q_{SR}) \subseteq D$ will be always true.

Clustering subsystem*

The clustering subsystem uses a seed oriented, partitioning type classification based on the new concept called cover coefficient[8, 9]. The starting point is a document (m) by term (n) matrix, D, of m by n for m documents $(d_1, d_2, \ldots, d_m)$ in the collection; n is the total number of terms assigned to describe the collection. The vector entries can be binary or can use a weighting scheme. The process will produce ρ cluster partitions $\rho = \{C_1, C_2, \ldots, C_\rho\}$ where C_i will be a non-empty cluster and $C_i \cap C_j = 0$ for $i \neq j, 1 \leq i, j \leq \rho$ (i.e., no overlap). Each cluster C_i will have l_i documents $C_i = (d_{i_1}, d_{i_2}, \ldots, d_{i_{l_i}})$ such that $l_i \geq 1$ and

$$\sum_{i=1}^{\rho} l_i = m$$

The properties of the D matrix are:

- $\sum_{j=1}^{n} d_{ij} \geq 1 \quad 1 \leq i \leq m$, i.e., each document is described by at least one term.
- $\sum_{i=1}^{m} d_{ij} \geq 1 \quad 1 \leq j \leq n$, i.e., each term describes at least one document.

The D matrix is mapped into a cover coefficient matrix, C. The C matrix indicates the extent with which the documents of the collection cover each other. If a document is alone (i.e., unique in the collection), there will only be a diagonal entry for it in the C matrix. The C matrix is m by m, a document by document matrix. Each diagonal entry in this matrix is the decoupling (uniqueness) coefficient, whereas the sum of the off-diagonal entries yields the coupling coefficient of a document, corresponding to a row in the matrix, with the other documents in the collection. The coupling coefficient enables us to estimate the number of clusters needed for the collection and calculate the cluster seed power for the documents. Clusters start with the seed documents and the other documents are assigned to the seeds in such a way that a document joins the cluster whose seed covers it maximally.

The C matrix is formed as follows. Two matrices S and S' are defined from the D matrix as follows:

$$s_{ij} = d_{ij} / \left(\sum_{k=1}^{n} d_{ik}\right), \ s'_{ij} = d_{ij} / \left(\sum_{k=1}^{m} d_{kj}\right) \text{ for } 1 \leq i \leq m \text{ and } 1 \leq j \leq n$$

These normalizations can be interpreted as:

s_{ij}: significance of term-j (t_j) for document-i (d_i)
s'_{ij}: significance of d_i for t_j

The C and C' matrices are then obtained as:

$$C = S \times S'^{\tau} \text{ and } C' = S'^{\tau} \times S$$

where τ is the matrix transpose operation. The C and C' matrices are m by m and n by n, respectively. An element of C is obtained as:

$$c_{ij} = \sum_{k=1}^{n} s_{ik} \times s_{kj}'^{\tau} = \sum_{k=1}^{n} (\text{significance of } t_k \text{ in } d_i) \times (\text{significance of } d_i \text{ for } t_k)$$

Each c_{ij} is a cover coefficient that produces, as indicated earlier, the decoupling and coupling coefficients. The properties of the C matrix are:

- $0 \leq c_{ij} \leq 1$, $c_{ii} > 0$: a document may or may not be covered by other documents, it is certainly covered by itself.

*The reader may like to omit this section on first reading.

- $\sum_{j=1}^{m} c_{ij} = 1$ for all i, $1 \le i \le m$: each row sum is equal to 1 and the sum of all row sums gives the total documents, m, of the collection.
- For all i, j, $1 \le i, j \le m$, $c_{ii} \ge c_{ij}$ if the D matrix is binary; c_{ii} can be less than c_{ij} in a weighted representation. If d_i is unique, then $c_{ij} = 0$ and $c_{ii} = 1$ (or w_i, weight). A document is covered mostly by itself, others can cover it as much as itself.
- $c_{ij} = 0$ implies $c_{ji} = 0$, and $c_{ij} > 0$ implies $c_{ji} > 0$. However, in the latter case, it is generally $c_{ij} \neq c_{ji}$. Coverage is mutual, however generally, not symmetric.

The decoupling coefficient δ_i of d_i is c_{ij}, $1 \le i \le m$, and the coupling coefficient is $\psi_i = \sum_{i=1}^{m} c_{ij} = 1 - \delta_i$ where $i \neq j$. A document that shares a large number of terms with the other documents will have a large ψ_i but a low δ_i value. Coupling and decoupling can be computed for the entire collection as:

$$\delta = \sum_{i=1}^{m} \delta_i / m, \quad \psi = \left(\sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} \right) / m = 1 - \delta \text{ where } i \neq j.$$

The values of δ and ψ range between 0 and 1.

By using the C' matrix, we can cluster terms and produce the measures δ_i', ψ_i', δ', ψ' in thesaurus construction.

The theoretically implied number of clusters, n_c, needed for the collection can be produced as:

n_c = (decoupling coefficient of the collection) × (number of documents)

$$= \delta \times m = \left\lceil \sum_{i=1}^{m} \delta_i \right\rceil$$

(Similarly, the number of term clusters, n'_c is $\delta' \times n$)

The average number of documents per cluster, d_c, is given by $d_c = m/(\delta \times m) = 1/\delta$.

The cluster seeds are determined by a new concept called the cluster seed power. The cluster seed power, p_i, of document-i is obtained as $p_i = \delta_i \psi_i t_i$ where $t_i = \sum_{j=1}^{n} d_{ij}$ is the number of terms in the document description vectors. In p_i, δ_i contributes to the separation of clusters, ψ_i contributes to the connection among the members of a cluster, and t_i provides normalization.

The documents are assigned to the cluster seeds with respect to either a single pass or a multi-pass algorithm. In the single pass algorithm, what is done mainly is:

- Determine the first n_c cluster seeds from the first highest p_i.
- Repetitively do: if d_i is not a cluster seed then find the cluster which maximally covers it, i.e., $\max(c_{is_j})$ where s_j, $1 \le j \le n_c$, is the seed documents.
- For the remaining unclustered documents, either form a ragbag (common) cluster or compare the documents with the documents of the clusters already formed to find a maximal cover for each document and add the document into the respective cluster.

In this clustering process, the estimated number of documents in cluster-i initiated by the seed d_{s_i} is n_i which is given by:

$$n_i = (p_i / \sum_{k=1}^{n_c} p_k) \times m \text{ for } 1 \le i \le n_c$$

In the forming of centroids for the clusters, a centroid $G_i = (g_{i1}, g_{i2}, \ldots, g_{in})$, $1 \le i \le n_c$, is constructed by applying the state of existence rule for each g_{ij}. This rule states that a centroid entry will be 1, if $f_j^i \delta'_j \ge f_{j\text{avg}} \delta'$ holds, otherwise 0 where:

f_j^i is the frequency of t_j within the document vectors of cluster-i.
$f_{j\text{avg}}$ is the average number of occurrences of t_j within the cluster containing it. It is the document frequency of t_j (i.e., $\sum_{i=1}^{m} d_{ij}$) divided by the number of clusters whose document vectors contain t_j.
δ'_j is the diagonal entry of the C' (i.e., $S'^{\tau} \times S$) matrix for t_j, which indicates the uniqueness of t_j.
δ' is the overall decoupling coefficient of all the terms.

The existence rule can be made to emphasize term uniqueness in a varying manner by introducing a multiplier τ into the condition (i.e., $f_j^i \delta'_j \ge \tau f_{j\text{avg}} \delta'$).

In this clustering scheme, we have the advantage of knowing the number of clusters needed for the collection. The document distribution in the clusters is rather uniform and clustering is order independent. This is because the cover coefficient has no dependence on the arrival of documents. The complexity, similarity and stability analyses of the clustering scheme can be seen in other detailed publications[8, 9, 11].

Cluster hierarchy

Once the clusters are formed, the database server can search them by comparing the centroid vectors with that of Q_S returning DQ_S for the full text search. If, however, there are a very large number of clusters making the linear, single level, cluster search prohibitive we can construct a hierarchy of clusters (tree), HCT for a sublinear search. Also, in dynamic environments of very large databases where clustering becomes a bottleneck, we can select a sample of documents and construct a HCT of core clusters as discussed in[10, 12]. The remaining documents can be assigned to the nodes of this HCT by using the same search strategy applied for user queries.

In our case, we can easily compute the theoretically implied number of clusters, n_c, for the collection with a procedure of $O(mn_{\text{avg}})$ complexity[9], where n_{avg} is the average number of terms in the documents. In the search of the HCT, we may stop at an intermediate node, as will be seen in the search strategy to follow. However, we should get to the lowest level (leaf) cluster under that node to add the document being processed. In such cases, we can pick the cluster with the maximum cluster seed power for the seed document.

In the mathematical model, C represents a hierarchical type cluster structure for D with l levels. This is shown in Figure 2. In this structure, if a node c_i has n children $c_{i1}, c_{i2}, \ldots, c_{in}$, then the documents contained in c_i are:

$$D_{c_i} = D_{c_{i1}} \cup D_{c_{i2}} \cup \ldots \cup D_{c_{in}}$$

Furthermore,

$$D_{cij} \cap D_{cik} = 0 \text{ (null) for } j \neq k$$

which leads to

$$D_{ci} \cap D_{cik} = D_{cik}$$

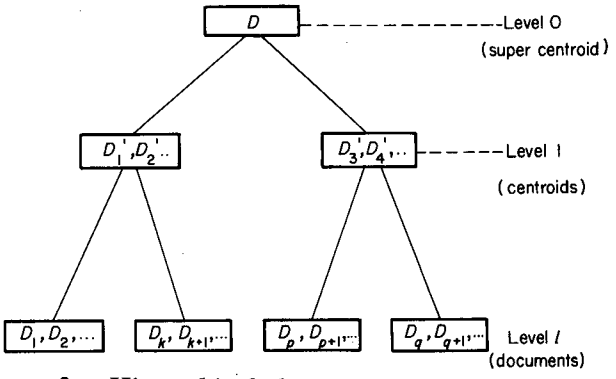

Figure 2. Hierarchical clustering

These properties hold for all levels because the clustering algorithms used are non-overlapping.

As to the terms, if T_{ci} is the set of terms used for the description of the centroid of cluster c_i, then $T_{cij} \cap T_{cik} = 0$ is not necessary true for $j \neq k$ since a given term can also be used in the description of documents in a different cluster. Similarly, $T_{ci1} \cup T_{ci2} \cup \ldots \cup T_{cin} = T_{ci}$ is not necessarily true for any i where $1 \leq i \leq l$ for an l level hierarchy because we cannot assume that all documents on both sides of the relationship are isomorphic.

The hierarchical document cluster implies a partial ordering relation. Assume R is a generalization relation for all the documents related with node-i. R will be the generalization of both the documents of the node as well as all the documents contained in the descendants of that node since the following conditions of reflexivity, antisymmetry, and transitivity will be applicable:

- Every node c_i is a generalization of itself by reflexivity, $c_i R c_i$.
- If $c_i R c_j$ and $c_j R c_i$ then this implies that $c_i = c_j$ (antisymmetry).
- If $c_i R c_j$ and $c_j R c_k$ then this implies that $c_i R c_k$ (transitivity).

Cluster search

The nodes of the cluster hierarchy and the system query, Q_S, will be represented as vectors equal in length to that of the document definition vectors. In the query definition vector corresponding to Q_S, each element can have one of the three possible values of 1, −1, and 0 corresponding to the use of the related term in the positive context, negative context, and non-use of it, respectively. As stated earlier, terms appearing in both contexts are assumed to be in the positive context. The search process will employ a top down search on a search tree. Depending upon the values of n_c and d_c, and the required precision/recall levels, the following two search strategies will be experimented with:

- Narrow search: at each node, the descendant that gives the highest value for the matching function is taken. The search terminates when none of the descendants of the processed node can exceed in match function value that of the node[10]. With this strategy, we always take a single branch out of a node.
- Broad search strategy: from a node, descend to children nodes that satisfy the match criterion (e.g., a similarity threshold). Stop when no child satisfies the match and send the parent to full text search. Accordingly, in this search, we may send multiple nodes from different branches of the tree to full text search.

Matching function

Two measures are being investigated for the matching function. One is the coupling function and the other is a similarity measure. The coupling function comes from the idea of the coupling coefficient in our clustering scheme. Accordingly, we can start with an m by n matrix D_{qc} whose first row would be the query vector and the remaining m-1 rows would correspond to the centroid vectors of the lower level subclusters of the node(s) identified in the search of the HCT. By obtaining the cover coefficient matrix C_{qc} corresponding to the D_{qc} matrix, we can produce the coupling values. A query would be covered by itself (i.e., $c_{qc_{ii}} > 0$) and the centroids identified as a result of the HCT search. Those $c_{qc_{ij}}$ that exceed a threshold would identify the centroids of the target clusters.

The second measure would be a similarity function of the following form:

$$SIM(Q_S, C) = \left(\sum_{j=1}^{n} F_1(q_j, c_j)\right) / \left(\sum_{j=1}^{n} (F_1(q_j, c_j) + F_2(q_j, c_j))\right)$$

where

$$F_1(q_j, c_j): \begin{cases} q_j \times g_j & \text{if } q_j > 0 \quad (1) \\ 0 & \text{if } q_j = 0 \\ \text{abs}(q_j) \times (|c| - g_j) & \text{if } q_j < 0 \quad (2) \end{cases}$$

$$F_2(q_j, c_j): \begin{cases} q_j \times (|c| - g_j) & \text{if } q_j > 0 \quad (3) \\ 0 & \text{if } q_j = 0 \\ \text{abs}(q_j) \times g_j & \text{if } q_j < 0 \quad (4) \end{cases}$$

In (1), we are saying that if a query term, q_j, position is a 1, then F_1 gives the number of documents containing that term under that centroid; g_j is the centroid entry. In (2), $|c|$ indicates the number of documents under the node, abs means absolute value and F_1 gives the number of documents that do not contain the negative query term. In (3), F_2 gives the number of documents in the nodes that do not contain the query term. In (4), F_2 gives the number of documents in the nodes that contain the query term. The *SIM* function is the conditional probability of hitting a one on the centroid vector of the HCT node given that a one was given in the query vector and hitting a zero on the centroid vector of the HCT node given that a minus one was given in the query vector.

Query feedback

The query feedback to be explored in this research differs in some respects from the other studies. This difference is motivated by the very fact that the integrated system will be concentrated on the context sensitive, search specifications of the type discussed in this paper. Accordingly, in this feedback scheme, the user query is kept fixed, but the system query, Q_S, is modified. Therefore, after each feedback process, a new cluster is chosen, which would be most related with the modified Q_S, and the documents are processed with respect to Q_T (and Q_{SR} later). The feedback process will depend on the

user choice although various measures of performance also will be employed. The user will have the choice of accepting the results or triggering the feedback loop by giving an indication of the relevant and irrelevant documents. In the refinement procedure of the feedback subsystem, the terms appearing in the relevant documents will be emphasized while those in the irrelevant documents are de-emphasized. The following criteria can be used for this purpose. If $|q_{ix}|$ denotes the number of appearances of term-i in set x where x can be one of system query, relevant documents (RL), or non-relevant documents (NR), then if

$$|q_{iS}^{p}| + |q_{iRL}^{p}| > |q_{iNR}^{p}| \quad ,$$

that is, if the sum of term-i's used in the positive context in the system query and those that appear in the relevant documents is greater than the number of positive context term-i's that appear in the non-relevant documents, then we would keep term-i in the refined query, otherwise we would drop it (or reduce its weight). Also, if $|q_{iS}^{N}| + |q_{iNR}^{N}| > |q_{iRL}^{N}|$, that is, if the sum of term-i's used in the negative context in the system query and those that appear in the non-relevant documents is greater than the number of negative context term-i's that appear in the relevant documents, then we would drop term-i (or reduce its weight) in the refined query.

The main goal of this feedback scheme is to preserve the context sensitive nature of the user query so that the advantages of full text search can be properly exploited.

SUPPORT SYSTEM STRUCTURE

Conceptual modelling and an example application

The universe of discourse of an integrated DBMS/IR application will be defined by means of the enhanced entity/relationship (E/R) model[13], which will allow representation of aggregates and generalizations as well as documents. These concepts of E/R and its implementation in a generalized environment are covered in our earlier studies[14, 15]. Presently, a data definition facility for an E/R interface is available that generates various system dictionaries. The diagrammatic representation of the E/R model uses rectangles to represent entities, diamonds for multidimensional relationships among entities, labels to indicate the type of relationships or mappings (e.g., one to many, many to many etc.), double lined rectangles for weak entities (or relationships), etc. In our case, an entity will correspond to a conceptualization whose representation will be in terms of formatted structures (i.e., a record type or specifically a relation) or a document whose representation will be in terms of unformatted structures. In the latter, the rectangle will be marked by an asterisk. For each datamodel, there corresponds a data definition language (DDL). Figure 3 shows the conceptualization of the example application environment. To give an appreciation of the information conveyed by this diagram, let us list some important relationships:

- AUTHOR is a weak entity, for him (her) to exist in the document database, he(she) should either have written a document or be referenced by at least one document.
- AUTHOR and DOCUMENT entities are related through the DOC-REF relationship in a many-to-many manner. That is, an author can reference several documents and can also be referenced by several documents.
- A specific journal is published by one publisher, yet that publisher can publish several journals.

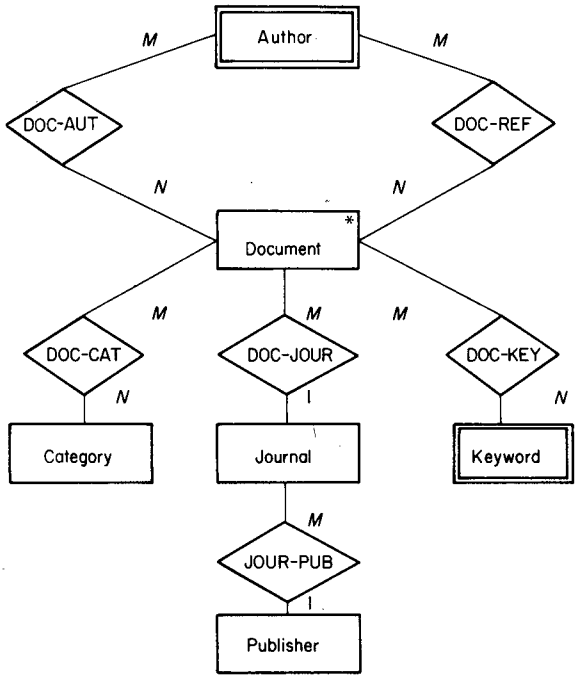

Figure 3. Conceptual representation of the integrated application environment

How to support unformatted data in database architectures

In the contemporary full text storage structures that store genuine text without inversion, sectors or page blocks are used as the units of storage of document files. We can think of tuples of normalized relations, 1024 bytes or longer, as the equivalent of those units in a stored relation. In fact, there is almost one-to-one correspondence between a (flat) file and a normalized relation. Therefore, in our system, we have made the following changes in the relation data type of the RAP.3 database computer so that both the formatted and unformatted structures can be supported. The main change allows a literal data type to be as long as a tuple itself (i.e., resulting in a unary relation). This provides us with a variable length literal data type, within a fixed length tuple, that stores the unstructured document attribute. The following is the format specification for text representation in the RAP.3 relations[16]:

- Each tuple contains one (or more) complete sentences.
- The first tuple of each document starts with a blank.
- As with standard typing rules, at least one blank should follow a punctuation mark.
- At the end of a sentence an end-of-sentence (EOS) marker is placed following the period. EOS must be followed with at least one blank.
- The last tuple of the document terminates with an end-of-document (EOD) character.

The following is the layout of a RAP.3 tuple:

D F	Mark Bits	D_1 Long text attribute (variable length)	ND_1	ND_2	D_4	. . .	D_n

Accordingly, a document file corresponds to a RAP.3 relation. This relation can hold one or more clusters of documents, or for large clusters also can be horizontally split. The associated storage overhead is the same as in file representations with the exception of one or two words per tuple reserved for tagging (marking) of data. Complete sentences from the documents are mapped into the $D1$ attribute of consecutive tuples.

The ND_1 and ND_2 attributes of the integer domain are needed for the internal use of the context sensitive, text retrieval operations. D_4 is the key attribute, which is a unique document identifier (DOCID). D_5 through D_n are the attributes reserved for formatted data. That is, in one relation, we can store both the formatted and unformatted data if they are needed for the convenience and efficiency of processing. Figure 4 shows the relational representation of the example application conceptualized in Figure 3. This relational representation conforms with the RAP.3 format specifications. According to Figure 4, the DOCUMENT entity is mapped into four tabular structures DOCUMENT, CITATION, ABSTRACTS, and SUB-HEADING. The rest of the mapping is one to one. In the list of relations, the key attributes are in bold characters. The ABSTRACT attribute holds the full text of the corresponding document's abstract. As shown in Figure 4, each tuple holds one or more sentences of the corresponding document and the TUPID attribute links multiple sentences (tuples) of one document. The unary relation DOCUMENT is added into the representation to increase the speed of database navigations (i.e., chain of mappings by semi-join). In this way, large unformatted data need not be kept in the system for the sake of the link attribute (i.e., DOCID).

DOCUMENT<**DOCID**>
CITATION<**DOCID**, TITLE, DATA, JCODEN, VOLUME, PAGES, *ND*1, *ND*2>
ABSTRACT<**DOCID, TUPID**, ABSTRACT, *ND*1, *ND*2>
SUB-HEADING<**DOCID**, STITLE, *ND*1, *ND*2>
AUTHOR<**AUTID**, NAME, NPAPER>
DOC-AUT<**DOCID, AUTID**>
DOC-REF<**DOCID, AUTID**, NREF>
KEYWORD<**KEYID**, KEY, KFREQ>
DOC-KEY<**DOCID, KEYID**, WEIGHT>
CATEGORY<**CATID**, CATCODE, CFREQ>
DOC-CAT<**DOCID, CATID**>
JOURNAL<**JCODEN**, JNAME, NPAPER>
DOC-JOUR<**DOCID, JCODEN**>
PUBLISHER<**PUBID**, PNAME, NPUB>
JOUR-PUB<**JCODEN, PUBID**>

Explanations of the abbreviated attribute names

DOCID	= Document identifier, uniquely identifies a document
JCODEN	= Journal code
TUPID	= Tuple identifier
STITLE	= Subtitle
NPAPER	= Number of papers written by an author
NREF	= Number of times an author is referenced in a document
AUTID	= Author identifier, uniquely identifies an author
KFREQ	= Number of times a keyword is used in the documents (keyword frequency)
CFREQ	= Number of times a category is used in the documents
NPAPER	= Number of pages in a published journal
NPUB	= Number of journals published by the publisher of a journal
*ND*1, *ND*2	= Full text search attributes

Figure 4. A relational representation of the conceptual structure

Language for the integrated DBMS and IR system

The language for the integrated system is the RAP.3 relational DBMS data language. This language is associative and high level (i.e., self-iterative). It contains the power of relational algebra and, in addition, there are commands for select and arithmetic update, select and compute aggregate function, etc. This language is also provided with the instructions for associative full text search.

To summarize, the following presents the main features necessary for integrating IR with DBMS:

(a) Domain type must support literals.
(b) Domain length must be variable for literal types.
(c) Efficient string search primitives must be incorporated.
(d) Sentence structure and adjacency must be recognized.
(e) A way of processing unformatted data with formatted structures must be found.
(f) Resolution capability must be added to the operations supported in (c) and (d).

We have been discussing items (a) and (b) so far and with regard to (c), (d), and (f), we will detail, in the following, the syntax and semantics of the IR related instructions as well as query resolution for the context sensitive, full text search operations. Item (e) has already been discussed in datastructure and the language aspects will be included in the following discussion.

As can be seen from the RAP.3 text mapping scheme, an important problem of text retrieval is the need to maintain text contiguity both in string searches and context resolutions. This problem has been solved, to a great extent, by the variable length text attribute feature of RAP.3. By this feature, some context resolution problems, such as splitting a text word between tuples (called term overflow) and passing overflow status and data to subsequent tuples to finalize the search, are inherently solved. However, there still remains the problem of context resolution for the text retrieval operations of the type $A \ldots B$ and $< A, B >n$, which correspond to variable adjacency in number of words and proximity, respectively. This is because the specified context might not be satisfied within a tuple. The necessary information for context resolution is passed from tuple to tuple by a feature referred to as *link passing*. The following gives the syntax definition of the new RAP.3 DBMS/IR Assembler

commands needed to perform the text search, and link pass operations:

MATCH (*tc*)[rel(atr1{,atr2{,atr3}}):qual]{[lit]}
MATCH __ WS (*tc*)[rel(atr1{,atr2}):qual]{[lit]}
MATCH __ WWC (*tc*)[rel(atr1{,atr2{,atr3}}):
qual]{[lit]}{[int]}
LINK __ PASS (*tc*1, *tc*2)[rel({atr1,}atr2)]

where

tc_i $1 \leq i \leq 13$ are the tag (mark) bits.
rel is the relation name of the document.
atr_i $1 \leq i \leq n$ (maximum tuple length is 1024 bytes) are the attributes of the relation corresponding to D_1, ND_1, ND_2, D_4, . . . , D_n in the RAP.3 tuple layout shown earlier.
qual Boolean qualification expression, examples of which can be found in the operations section.
int is a positive integer.
{} indicate options.
lit is the literal constant corresponding to the search pattern. The examples of search pattern are: *'AA'?'BB'*, **'A'??'BB'*,**'AA'??'BB'3?'C'* where * and ? correspond to variable length don't care (VLDC) and fixed length don't care (FLDC) character respectively, and int? indicates the repetitions of ?.

The first match instruction searches for the equality of the search pattern, in the text stored in atr_1 and *tc* marks all the qualifying tuples within the cells storing the document; atr_2 gives the beginning search offset within the text string (i.e., atr_1). If atr_2 is not specified, then the search offset is taken as 0. If atr_3 is specified, all occurrences of the search pattern within atr_1 are determined and the total number of occurrences is stored in atr_3, within each tuple. The difference of the second match instruction is that the first occurrence of the search pattern is found within the sentences of atr_1. The entire match should be contained within a sentence. The third match instruction finds the first occurrence of the search pattern, bypassing at most int number of words of the text. The value of int can be specified explicitly, or implicitly by atr_3.

During execution of the third match instruction, if the search pattern is not found and the search context specified by the word count is not satisfied within a tuple, then the *t*14 mark bit of that tuple is set. This specific case is called an overflow condition and can occur only during the execution of the MATCH __ WWC instruction, since the context of a match is always satisfied in the other types of the match instructions. The LINK __ PASS instruction takes one memory cycle to execute and resolves the overflow condition in those tuples marked for overflow. The resolution requires resetting the *t*14 mark bit (*tc*2 in the instruction syntax) of the tuples marked for overflow, setting *tc*1 bits of the subsequent tuples, and passing the unprocessed (i.e., remaining) count values into the predetermined attributes of the subsequent tuples.

In the other use of the LINK __ PASS instruction, when the literal attribute atr_1 is specified, if a tuple is *tc*2 marked and its atr_1 does not contain an end-of-text character, then the following tuple(s) of this document will be *tc*1 marked and their atr_2 is se to zero. This feature of the LINK __ PASS instruction is used to set the search context of the text retrieval operation *A* . . . *B*.

It is assumed that the don't care character will not match an end-of-sentence character. This semantic property of search patterns prevents the occurrence of an overflow during term matching.

Query resolution in RAP.3

In the RAP.3 text retrieval system, a great proportion of the context dependent query resolution takes place in the operation of the new text retrieval commands the basic task of which is term matching. As seen in the semantics of the instructions described above, sentence structure and adjacency are recognized as an integral part of the operations. Such resolution operations constitute a totally separable task in most other text retrieval systems. For the execution of the context sensitive, text retrieval operations of the more complex nature, however, more sophisticated resolutions are required. These operations are:

A and *B*	in sentence (specified context)
$< A.\ n.\ B >$	(directed proximity)
$< A, B > n$	(undirected proximity)
A??*B*	(fixed length don't care)
A . . . *B*	(immediate adjacency or variable number of words in between)
A * *B*	(variable length embedded don't care)

The resolution for these operations requires that small RAP.3 programs be written using both the DBMS and IR instructions of the RAP.3 language. In other words, these resolutions are embedded in the logic of RAP.3 programs in which MATCH and LINK __ PASS operations are directed[16]. To aid the user, all these programs are made general purpose, full text retrieval macros running under the macro processor RAPMAC. Appendix 1 presents a list of these macros, which also gives an idea of the type of context sensitive, full text operations supported in the system. Appendix 2, shows the algorithmic expansion of the 'Search (Match) *A* and *B* in sentence macro, which demonstrates resolution on the entire document relation level. More details on these can be found in reference[16]. One important note here, however, is that in a RAP instruction qualification, there is no practical limit on the number of predicates in a Boolean expression. This means that we can have a large number of terms specified for a text search. However, in terms of the macros listed in Appendix 1, this applies, at the present, to macros #3 and #4 only. The limitation on the number of predicates is determined by the buffer and parameter storage space set for the implementation of the RAP.3 DBMS.

Distributed database architecture

As we indicated at the beginning, a local computer network consisting of a network server and several workstations is the basic hardware architecture. More complex configurations can be obtained by cascading such networks. In the software counterpart, we can envision the centralized DBMS/IR as the basic software architecture. In this architecture, the network server acts as the central node. When local networks are cascaded, however, the software architecture can be expanded into a distributed DBMS/IR.

Figure 5 depicts the local computer network facility used for the experimental DBMS/IR integrated system.

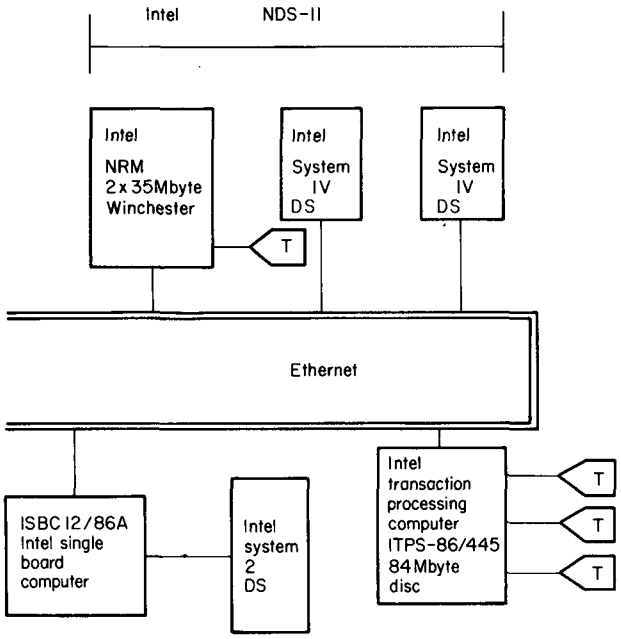

Figure 5. Local computer network

In this system, the Intel equipment is used. There is an NRM wih the three workstations (two SYSTEM-4's and a SYSTEM-2) in a star network based on Ethernet. In addition, on the same network, there is a single board computer attached to SYSTEM-2 and a transaction processing computer (iTPS). Currently, the RAP.3 DBMS/IR software and the RAPNET distributed database query execution monitor are operational on all the processors of the network. As a core application, a centralized DBMS/IR based on NRM with the three workstations is being carried out. The second phase will include iTPS as the network server with which a homogeneously distributed DBMS/IR system will be made operational. The third phase will include the networking of one VAX 11/780 Unix based system into the configuration. The Unix-C based versions of RAP.3 DBMS/IR and RAPNET software are nearing completion.

The distributed database (DDB) versions of RAP software were published earlier[15, 17]. In the homogeneous RAP DDB, RAP query programs are decomposed into subqueries and network execution is monitored with respect to a dataflow driven query graph[17]. In this way, parallelism inherent even in a single query program can be exploited. The RAPNET software is the monitor written for the homogeneous DDB. It is responsible for query decomposition, query graph construction, and distributed query execution on the network.

OPERATIONAL SYSTEM

Database creation and query execution

The RAP.3 DBMS/IR language can be used either as a stand-alone query language, or as a data sublanguage embedded in PLM/86 for the Intel based network processors or in the *C* language in the VAX 11/780 version. The simple example shown below defines and creates two relations and performs two simple retrievals, using the stand-alone version.

```
*JOB/*Creation of two relations*/
  AUTHORIZE<'U101'>
  RELATION<LOCATION (20)
        DEPARTMENT: LITERAL, 12, KEY
        FLOOR: INTEGER, 1><GRANT__
          READ>
   RELATION <EMPLOYEE (500)
        NAME:LITERAL, 25, KEY
        DEPARTMENT:LITERAL, 12
          INDEXED
        SALARY:INTEGER, 4><PRIVATE>
  LOCK <LOCATION>
        CREATE<LOCATION>RELEASE
          <LOCATION>
  LOCK <EMPLOYEE>
        CREATE<EMPLOYEE>RELEASE
          <EMPLOYEE>
        EQQ
  *DATA
        & LOCATION<'PERSONNEL' 1>
                  <'SALES' 2> & END
        & EMPLOYEE/*3 tuples follow*/
            <'KING' 'PERSONNEL' 4500>
            <"L'HOME" 'SALES' 1500>
            'JONES' 'SALES' 25000> & END

  *JOB  /*Query-A = Employees on 2nd floor*/
        AUTHORIZE<'U101'>
        MARK(T1)<LOCATION: FLOOR
          = 2>
        CROSS__MARK(T1)<EMPLOYEE:
            DEPARTMENT = LOCATION.
            DEPARTMENT><MKED(T1)>
            EQQ
  *JOB  /*Query-B*/
        /*Employees in the SALES department,
          and the one with the highest salary*/
        AUTHORIZE<'U101'>
        MARK(T1)<EMPLOYEE:
            DEPARTMENT 1='SALES'>
        READ<EMPLOYEE(NAME):
            MKED(T1)>
        MAX<EMPLOYEE(SALARY):
            MKED(T1)><REGF__4>
        READ<EMPLOYEE(NAME):
            SALARY = REGF__4>
        EQQ
```

As can be seen from the RELATION command defining a relation, the user can declare a primary key by the KEY keyword, which is repeated for compound keys. Also, the INDEXED option is used for directing the system to build a secondary key index for the attribute being declared (e.g., DEPARTMENT). GRANT__READ, PRIVATE, etc. are for security/integrity control. The CREATE command populates the database with respect to the RELATION declaration by inputing the data following *DATA (or from a disc file). We see examples of tagging selections, by the MARK; semi-joins, by the CROSS__MARK; associative and set oriented selection in the MARK, MAX (maximum), and READ instructions. Up to this point, we have not seen the use of full text retrieval. Let us point out that if we are interested in simple keyword based retrieval by using inverted indices we can use this formatted version of RAP. All we have to do is to declare a binary relation POSTINGS with the

KEYWORD and DOCADDRESS attributes and use a READ <POSTINGS(DOCADDRESS): KEYWORD = 'DATABASE'> instruction to retrieve document addresses of documents containing the term DATABASE from the POSTINGS relation. For merging and/or intersecting indices, we can either define an n'ary relation and say READ <POST(DOCADDRESS): KEYWORD1 = 'A' & (OR) KEYWORD2 = 'B'>, or equivalently, write the following program between the two binary relations, POST1 and POST2:

```
MARK (t1)<POST1: KEYWORD = 'A'>
CROSS_MARK(t2)<POST2: DOCADDRESS =
  POST1.DOCADDRESS> <POST1.MKED(t1)>
READ<POST2(DOCADDRESS): KEYWORD =
  'B' & MKED (t2)>
```

The second operation is a semi-join through the matching document addresses and the READ combines this mapping indicated by MKED ($t2$) with those tuples containing the keyword B. The result is a chain of selection, semi-join, and output using conjunctive selection (restriction). This scheme can be extended to include the concepts of weighted Boolean retrievals[18, 19] and ranked output[20] by using all the capabilities of the RAP instruction set.

DBMS/IR integrated retrieval

At this point, we can go back to the example application presented at the beginning. Based on Figures 3 and 4 and the query statement given at the beginning of the integration model, a possible execution sequence can be given as follows (using the relations of Figure 4 and with no regard to semi-join optimization):

1. SELECT AUTHOR with name JOHN DOE
2. SEMI-JOIN AUTHOR to DOC-AUT via AUTID
3. SEMI-JOIN DOC-AUT to DOC-REF via AUTID
4. SEMI-JOIN DOC-REF to DOC-AUT via DOCID
5. SEMI-JOIN DOC-AUT to ABSTRACT via DOCID
6. IR search for 'SEMANTIC DATA MODEL' AND 'AGGREGATION'
7. SEMI-JOIN ABSTRACT to DOCUMENT via DOCID
8. READ DOCUMENT

As can be seen in this query, operations 1 through 5, and 7, 8 are DBMS operations and 6 is an IR operation, and yet everything is linked and consequence of each other in the execution sequence. Although operation 6 is a single item in the list, it corresponds to a set of operations implemented as a RAP macro call which uses DBMS/IR instructions. To follow these, let us refer to Figure 1, which is the abstract representation of the integrated system. In this figure, Q represents the query program shown above (i.e., operations 1 through 8), Q_R corresponds to DBMS operations (operations 1 through 5, 7, 8) and Q_T to IR operation 6. In terms of the datasets and data manipulation functions, the following correspondences can be seen:

DQ_R: $(\text{DOC} - \text{AUT})_R$, DQ_S: $(\text{ABSTRACT})_R$ — subscript R indicates restriction — $f(DQ_R)$: step 5, $f(DQ_S)$: step 6, Q_{SR}: step 7, $(\text{DOCUMENT})_R = f(Q_{SR})$

The following is another example of a DBMS/IR operation at the RAP program level:

```
*JOB
/* . . . **1 –) Give the DOCID of the documents that
   contain RAP and DATABASE MACHINE within
   the same sentence. */
%MATCH$A$AND$B$IN$SENT'RAP', 'DATA-
BASE MACHINE', ABSTRACT, ABSTRACT,*
     ND1, T1, T3, T4
CROSS_MARK(T1) [DOCUMENT: DOCID =
   ABSTRACTS. DOCID][MKED(T1)]
READ[DOCUMENT(DOCID): MKED(T1)]
EOQ
```

Forms interface

Presently, there is a basic forms interface that creates forms, retrieves, updates, and deletes form instances. Specifically, a form corresponds to a RAP.3 relation. A form instance corresponds to a relation tuple. By entering values and parameters at the specified columns of an accessed form frame, the user can specify: selection and output corresponding to MARK and READ; semi-join corresponding to CROSS_MARK (with a form destination); arithmetic updates ADD, SUB, MUL or DIV; and the scalar aggregates SUM, COUNT, MAX, MIN, and AVERAGE.

It is the goal of the experimental system to enhance the forms interface with the functionality of context sensitive, full text operations, which would link to RAP macro generation, and user query feedback.

PERFORMANCE AND THE FUTURE

Efficiency of integration

The proposed integration, apart from its effectiveness and general purpose utility, which offers several important functionalities, is expected to be efficient because of the following main features:

- It uses an efficient implementation of a relational DBMS.
- IR features are integrated with DBMS at the primitive level.
- The IR primitives are based on the efficient string search operations and the context sensitive, full text instructions carry out query resolution.
- Clustering is used as an efficient IR index in addition to the secondary key indexing of the RAP.3 DBMS.
- Integration of DBMS and IR at the primitive level results in minimum layers of software, which contributes to overall system efficiency.
- There is no need for a separate computer for query resolution and the necessary communication of term matches since all of these functions are handled in the RAP.3 system.

To elaborate some of these further, we can point out the following. The RAP.3 DBMS system is a software emulation of a database computer on the present-day computers. This implementation is transparent to the user since the language is what the user sees and uses. This language is implemented in the emulator by using efficient

means of file organization. Relations are stored as hash addressed files based on keys. In addition, secondary key indices are constructed on the frequently needed attributes, as directed by the user.

As mentioned earlier, mapping of full text into long tuples where a long variable length literal attribute is allowed is not much different from the flat file implementations that use sector or page based indexing and storage. The document relation can be horizontally fragmented for access efficiency in much the same way as flat files are partitioned. In the distributed system architecture, fragments can be distributed according to user profiles.

In addition to secondary indexing, as directed by the INDEXED attribute, clustering is used as an effective means of document indexing. As opposed to indexing techniques such as *B*-trees, the indexing based on clustering uses semantic information, that is, coupling among the documents of a collection. A cluster can be a RAP.3 relation or a horizontal fragment of a larger relation. In the former, the relation can be further fragmented depending on the size of data. In any case, one can define a secondary index within a cluster because a cluster is a relation for which the INDEXED attributed can be declared.

As can be seen in the way the system is integrated, clustering is essential for the efficiency of the overall system. We will come back to this issue in the future hardware solutions for text retrieval.

The semi-join construct of the RAP.3 system (i.e., CROSS_MARK) is an important operation. In an integrated system where formatted and unformatted data are cross selected, the join operation must be implemented efficiently. The semi-join operation does not create a new relation to store the result and transmits a minimum amount of data between relations. The advantage of semi-join comes into the picture more importantly in distributed databases. As we have indicated earlier, office automation uses DDB as the underlying architecture.

We have also indicated the way of handling simple inverted file based keyword indices. Because indices are nothing but relations, one does not need to resort to an additional system to have a simple IR for certain applications.

Document retrieval hardware of the future

For a survey and analysis of various hardware proposals for document retrieval, one can start from the recently published summary[21]. In this section, we want to present the latest assessment of the work covered in the referenced survey with the addition of the RAP.3 database computer solution, which is not covered in that survey.

Text (document) retrieval related hardware varies from partial to reasonably complete solutions to the problem. In the partial category, we can mention the proposals on constructing fast index processors to speed inverted list intersection and merge. As can be realized, this approach is a component solution without altering the basic characteristics of conventional text retrieval.

In the complete system category, we see two major system components specialized for text retrieval, they are the term matcher and query resolver subsystems. Because the query resolver does not require much novel hardware and the system bottleneck rests in the term matcher, various proposals have concentrated on the term matcher. The following are the techniques used by the various proposals made to date:

- parallel comparators,
- associative memories,
- cellular arrays,
- finite state automata.

We can consider the first two together. In the parallel comparators approach, the term detector sits in series with the datastream, which is read off a mass memory such as disc. The data, in passing, are buffered in a window and simultaneously made available to a number of parallel comparators. The associative memory technique is a more advanced version of the former. Search terms are stored in an associative memory, which acts as a term matcher synchronous with the data. The advantage of this is the ability to store a large number of search terms, each of which corresponds to a comparator in the former approach. In reality, one uses a fast random access memory that emulates an associative memory. The majority of commercial systems of text retrieval hardware use this associative memory approach. Two points to realize in this approach are: the difficulty of implementing context sensitive, full text operations, especially those using fixed and variable length don't cares, and the serial nature of the system.

The cellular arrays approach aims to dynamically construct search patterns by using cells capable of comparing a particular character or pattern. The system is serial with data, and cells operate by communicating with their immediate neighbours so that when a match occurs a number of logically consecutive cells produce match signals. This approach requires too much dynamism and a large number of cells. The dynamic interconnection problem and data streaming for a large number of cells are non-trivial and costly issues.

The finite state automata (FSA) approach has been implemented in various experimental systems[21]. The clarity of the FSA abstraction and its efficient implementation do not go together. There have been various optimizations for FSA implementations ranging from constructing separate FSA's for the starter, sequential, and index states to efficient addressing of the state blocks. Recently, a partitioned FSA (PFSA) has been proposed with the aim of synchronizing with the datastream at disc speed using a minimum amount of buffer memory. A PFSA can be at more than one state at a time and a non-conflicting partitioning scheme of the state table has been found so that a PFSA can be cast modularly into VLSI. From this point on, we will refer to the VLSI based FSA or PFSA as the FSA filter. One needs a structure consisting of a ring of FSA filters and a match controller for each serial datastream[21]. For a complete system, the query resolver microcomputer, in addition to the standard I/O system, and the host computer are needed.

In the RAP.3 database machine/computer approach[22, 23], the entire integrated DBMS/IR system described thus far is supported with the RAP.3 database machine. As far as the user is concerned, there would be no difference between the software version running on the emulator, and the hardware version running on the actual machine, with the exception of further efficiency introduced by the specialized hardware.

The RAP.3 database machine uses parallel cells and parallel microprocessors in each cell. Each microprocessor executes firmware of query routines implementing DBMS and IR instructions. The entire device memory, which is equal to the union of cell memories, is made to work like an associative memory. The microprocessors of each cell execute query routines on the tuples of the cell memory in parallel. The difference between the associative memory approach discussed earlier and the RAP.3 (associative) database machine approach is that in the former, search terms are placed in the associative memory and the text is streamed through it, whereas in the database machine the text is stored in the device memory which is a quasi associative memory so that it can be large to accommodate bulk data. In the microprocessors of each cell, efficient string search is performed[16]. The complexity of this operation is of the order $O(m)$ character comparisons where m is the length of the literal text attribute in a RAP.3 tuple. The increase in efficiency due to hardware will be by a factor $(k \times n)$ where n is the number of parallel cells in the device and k is the number of parallel microprocessor (subcells) within each cell.

The advantage of the RAP.3 database machine approach is the ability to provide both a relational DBMS and the IR system in an integrated manner in the basic architecture and the integration of the query resolver which is an important component in the other systems.

To maintain a steady flow of data into the database machine, a disc cache, as in the Japanese fifth generation computer's database machine, or an alternate cell memory, as in the RAP virtual memory architecture, is needed.

In cases where physical device dimensions reach a limit, partitioning of very large databases into multiple discs so that each disc can be connected to a dedicated special purpose hardware would be a viable solution. This is what the datastreaming techniques discussed earlier and the FSA filter rely on for performance. There will be a factor of difference between the point where one duplicates an FSA filter and a database machine because of the difference in the cut-off points as determined by the sizes and speeds of the respective devices.

SUMMARY AND CONCLUSIONS

We have presented an integrated fact/document information system for office automation. We started with the need to synthesize DBMS and IR into a complete and general purpose system, in such a way that no one system would be reduced into the other, by losing its unique features and advantages. We have seen that the offices of the future will need and use services of DBMS and IR extensively, in a distributed computing environment.

An integrated model of the proposed experimental DBMS/IR system is presented with respect to query processing. In that model, the execution of Boolean queries of context sensitive, full text operations and their relationship to document clustering and query feedback are discussed. The clustering subsystem and its search strategy are introduced.

In the support system structure, conceptual data modelling, ways of supporting formatted and unformatted data together, and the language for the integrated DBMS/IR system are described. In the language, we have seen the context sensitive, full text primitives of the RAP.3 system that are associative and perform sentence and adjacency related query resolution. These structures and operations are embedded into RAP.3 relations and DBMS language, respectively.

The distributed database architecture is the next level of architecture when an office grows out of the bounds and capabilities of a single local area network that is managed centrally with a database server. In such a case, distributed query processing of the DBMS/IR system becomes necessary. The RAPNET software is the distributed query execution monitor of the RAP.3 DBMS/IR language. RAPNET constructs and executes a dataflow graph for distributed query execution.

In the operational system, we have seen how to create relations and to process simple inverted file based systems. Following that, the integrated DBMS/IR retrieval example is continued and the forms interface is outlined.

In performance, the efficiency of integration is discussed. Besides the effectiveness and utility of the proposed system, a better efficiency is achieved by reduction in software, lowering the integrated functionality towards the primitive hardware level, and combining the query resolver and text matching subsystems into a single system.

Clustering is used as an index at the semantic level and as a means for database partitioning and relation fragmenting.

In the discussion of hardware approaches, the RAP.3 database machine approach is indicated as a viable alternative to other current research such as the FSA filter. In the case of the RAP.3 hardware solution, all that needs to be done is to replace the underlying DBMS/IR RAP.3 software emulator with the actual RAP.3 hardware to further boost the performance of the integrated system. The sizes of the databases would dictate the issues such as database partitioning, using multiple hardware, and distributed databases. These are the common problems to all future hardware regardless of the specific organization of a given architecture.

ACKNOWLEDGEMENT

We gratefully acknowledge the computer network grant of INTEL Corporation of Arizona which made the configuration of Figure 5 a reality.

REFERENCES

1 **Salton, G** 'Some research problems in automatic information retrieval' *Proc. ACM SIGIR Conf.* (1983) pp 252–265

2 **Van Rijsbergen, C J** 'Information retrieval: New directions: Old solutions' *Proc. ACM SIGIR Conf.* (1983) pp 264–265

3 **Crawford, R G** 'The relational model in information retrieval' *J. Am. Soc. Inf. Sci. (USA)* Vol 32 No 1 (1981) pp 51–64

4 **Scheck, H J** 'Methods for the administration of textual data in database systems' *Proc. Joint BCS and ACM Symp.* (1980) pp 218–235

5 **Stonebraker, M** *et al.* 'Document processing in a relational database system' *ACM Trans. Office Info. Syst.* Vol 1 No 2 (1983) pp 143–158
6 **Salton, G and McGill, J** *Introduction to Modern Information Retrieval* McGraw Hill Book Company, USA (1983)
7 **Lancaster, F W and Fayen, E G** *Information Retrieval On-line* Melville Publishing Company, USA (1973)
8 **Can, F and Ozkarahan, E A** 'A clustering scheme' *Proc. ACM SIGIR Conf.* (1983) pp 115–121
9 **Can, F and Ozkarahan, E A** 'Two partitioning type clustering algorithms' *J. Am. Soc. Inf. Sci. (USA)* Vol 35 No 4
10 **Van Rijsbergen, C J** *Information Retrieval* (2nd edn) Butterworths, UK (1979)
11 **Can, F and Ozkarahan, E A** 'Similarity and stability analysis of the two partitioning type clustering algorithms' *J. Am. Soc. Inf. Sci. (USA)* Vol 35 No 5
12 **Van Rijsbergen, C J** 'Further experiments with hierarchical clustering in document retrieval' *Info. Storage and Retr.* Vol 10 (1974) pp 1–14
13 **Chen, P P** 'The entity relationship model — toward a unified view of data' *ACM Trans. Database Syst.* Vol 1 No 1 (1976) pp 9–36
14 **Dogac, A and Ozkarahan, E A** 'A generalized DBMS application in a database machine' *Proc. ACM SIGMOD Conf.* (1980) pp 133–143
15 **Ozkarahan, E A and Kerschberg, L** 'A heterogeneous distributed database system architecture incorporating data semantics and a relational database machine interface' Dept. Computer Science, Technical Report TR82–06, Arizona State University
16 **Ozkarahan, E A and Can, F** 'Integration of fact/document retrieval systems within a database machine' Dept. Computer Science, Technical Report TR–82–02, Arizona State University (1982)
17 **Ozkarahan, E A, Tansel, A U and Smith, K C** 'Database machine/computer based distributed databases' *Proc. 2nd Int. Symp. Distrib. Databases* (1982) pp 61–79, North Holland, The Netherlands
18 **Waller, W G and Kraft, D H** 'A mathematical model of a weighted Boolean retrieval system' *Info. Process. and Manage.* Vol 15 (1979) pp 235–245
19 **Buell, D A and Kraft, D H** 'A model for a weighted retrieval system' *J. Am. Soc. Inf. Sci. (USA)* (1981) pp 211–216
20 **Noreault, T, Koll, M and McGill, M J** 'Automatic ranked output from Boolean searches in SIRE' *J. Am. Soc. Inf. Sci. (USA)* (1977) pp 333–339
21 **Hollaar, L A** 'Hardware systems for text information retrieval' *Proc. ACM SIGIR Conf.* (1983) pp 3–9
22 **Ozkarahan, E A** 'Desirable functionalities of database architectures' *Proc. IFIP World Congr. Paris* (1983)
23 **Ozkarahan, E A** 'The implementations of the relational associative processor (RAP) and its system configurations' Dept. Computer Science, Technical Report, TR82–05, Arizona State University

APPENDIX 1

The text retrieval applications in the RAP system are programmed with the use of the RAP instruction set and nine available text retrieval macros that are listed in the following:

- MATCH$A: Finds any document that contains the word *A*.
- MATCHALLA: Finds any document that contains the word *A*. Different from MATCH$A because it searches for all occurrences of *A*.
- MATCHAOR$B: Finds any document that contains the word *A* and the word *B*.
- MATCHAAND$B: Finds any document that contains the word *A* and the word *B*.
- MATCHAANDNOTB: Finds any document that contains the word *A* but not the word *B*.
- MATCHAANDBIN$SENT: Finds any document that contains both the word *A* and the word *B* within the same sentence.
- MATCHAANYWORDS$B: Finds any document that contains the word *A* (either immediately or after an arbitrary number of words) followed by the word *B*.
- MATCHANWORDS$B: Finds any document that contains the word *A* followed by the word *B* within *n* words of the former.
- MATCH$THRESHOLD$OR: Finds any document that contains at least *n* of the *m* distinct words: $A_1, A_2, \ldots, A_m$; where $n \leq m$.

In the following, the meanings of the macro parameters are explained together with their actual use in the two example macro call statements.

$STRING_i$ ($i \leq 2$):	the strings to be matched (e.g., *A* and *B*) in the text.
REL:	the relation name containing the text to be searched.
LA:	the literal attribute of relation REL that corresponds to the literal domain that holds the text to be searched.
NA_i ($i \leq 2$):	the number of attributes of relation REL that are needed during the execution of the macro body.
T_i ($i \leq 5$):	the distinct mark bits of relation REL that are needed for the execution of the macro body. Before calling a text retrieval macro, all of the mark bits specified in the macro call statement should be cleared. After execution of the macro body, the first mark bit indicates (shown as T_1 in the call statements) the qualified tuples for the text retrieval operation.
QUAL:	the tuples for text retrieval are selected with respect to this qualifier which is a literal constant and contains a RAP qualification expression. If all the tuples of relation REL are to be considered, this parameter is omitted. A typical QUAL specification would be :MKED ($T1$).

The two macro call statements for the text retrieval operations corresponding to macros number 1, and 9 above are demonstrated in the following:

For Macro MATCH$A:
MATCH$A STRING, REL, LA, T_1, QUAL

For Macro MATCHAANDBIN$SENT:
MATCHAANDBIN$SENT $STRING_1$,
$STRING_2$, REL, LA, NA_1,
T_1, T_2, T_3, T_4, QUAL

After the execution of the macro body, the mark bits T_2 through T_3 remain clear.

APPENDIX 2

Search for A and B in sentence

In this text retrieval operation, a tuple containing the first term (e.g., A) may be considered more than once; for the current occurrence of A, the second term may not occur in the specified word proximity. In that case the unprocessed portion of the tuple will be considered for the occurrence of A again and, if found, the tuple will be re-examined for the occurrence of the second term within the same sentence. The algorithm is as follows:

1. Mark all the tuples of the text relation for an A match.
2. Set the search offset for the term A to zero in all the tuples.
3. Test if there are tuples to be considered for a pending A match, if not, go to step 11.
4. Reset the mark bits which were set in a previous A match operation.
5. Perform a match operation for term A on the tuples that may be eligible. If there is no match, go to step 11.
6. Discard the unqualified tuples at the end of the match operation, so that they do not take part in the possible re-executions of step 5.
7. Perform a match-within-sentence operation for the second term B on the tuples that were selected in step 5.
8. Test if any B match exists, if not go to step 3 (note that in a tuple, there might be more than one sentence and any one of these sentences may contain term A).
9. Mark all the tuples of the qualified documents and do not process them for an A match again.
10. Go to step 3.
11. Exit.