SPIDER: A Multiuser Information Retrieval System for Semistructured and Dynamic Data

Peter Schäuble
Swiss Federal Institute of Technology (ETH)
CH-8092 Zürich (Switzerland)

Abstract

The access structure, the retrieval model, and the system architecture of the SPIDER information retrieval system are described. The access structure provides efficient weighted retrieval on dynamic data collections. It is based on signatures and non-inverted item descriptions. The signatures provide upper bounds for the exact retrieval status values such that only a small number of exact retrieval status values have to be computed. SPIDER's retrieval model is a probabilistic retrieval model that is capable to exploit the database scheme of semistructured data collections. This model can be considered as a further development of the Binary Independence Indexing (BII) model. The system architecture was derived systematically from a given set of requirements such as effective and efficient retrieval on dynamic data collections, exploitation of the database scheme, computed views, and the integration of information retrieval functionality and database functionality.

1 Introduction

The SPIDER project is aimed at building a next generation information retrieval system which meets the current needs of information management. The current needs are expressed by a list of requirements that were identified in projects dealing with technical documentation and medical reports (Berrut & Chiaramella, 1989), (Chiaramella et al., 1986), (Frei & Schäuble, 1991a), (Hoppe, 1992). The experiences made in these projects seem to indicate that future information retrieval systems will have the following salient features.

1. Vague queries (including relevance feedback) are processed both effectively and efficiently even when the data collection is modified frequently.
2. The retrieval of information from semistructured data collections is supported by an appropriate retrieval model which exploits the database scheme.
3. The system provides database functionality (persistency, concurrency, data independence, data integrity, data model including query language and views).
4. The system supports the derivation of both numeric and non-numeric data (computed views).
5. The system provides access to remote data in a heterogeneous environment.

The SPIDER project is guided by these requirements. In order to bridge the gap between these requirements and the state of the art of information retrieval technology we made the following new developments that are presented in this paper. First, a new access structure was developed which supports effective and efficient retrieval on dynamic data collections (Section 2). The access structure is based on signatures and non-inverted item descriptions. The signatures provide upper bounds for the exact retrieval status values such that only a small number of exact retrieval status values have to be computed. Second, we developed a novel probabilistic retrieval model that takes advantage of the database scheme (Section 3). This model can be considered as a further development of the Binary Independence Indexing (BII) model (Fuhr & Buckley, 1991). Third, we derived systematically a new system architecture from the requirements given above (Section 4). We will see that a new system architecture is required because conventional database management systems are difficult to

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
ACM-SIGIR'93-6/93/Pittsburgh, PA, USA
© 1993 ACM 0-89791-605-0/93/0006/0318...$1.50

enhance by information retrieval capabilities and information retrieval systems are difficult to enhance by database capabilities.

2 Weighted Retrieval of Dynamic Data

In this section, we present a retrieval method together with a novel access structure which supports *effective and efficient* retrieval from *large and dynamic* data collections. Effective retrieval is accomplished by *weighted retrieval* including relevance feedback (Harman, 1992), (Salton & Buckley, 1988b). A weighted retrieval method consists of an indexing method and a retrieval function. The indexing method determines the descriptions (i.e. the description vectors) of the stored data items (e.g. of documents) and the descriptions of the query items given by the users. The retrieval function matches the description of the query item q against the descriptions of the stored data items d_j. The result of a single matching consists of the retrieval status value $RSV(q, d_j)$ which is usually (up to an order preserving transformation) the estimated probability that d_j satisfies the user's information need expressed by q. A user is presented a list of data items that are ranked in an optimal order, i.e. they are ranked in decreasing order of their retrieval status values (Robertson, 1977).

Efficient retrieval is achieved by means of an appropriate *access structure* containing the precomputed descriptions of the stored data items. The existence of an access structure implies the existence of redundant data (up to 100 %) which, however, is conflicting the aim at efficient updatability; particularly, if every description of a data item is partitioned into small pieces (e.g. postings) that are spread over the whole access structure. Hence, effectiveness and efficiency are conflicting goals when the data collection is dynamic. Before presenting our approach, we briefly discuss existing approaches.

In information retrieval, *inverted files* are widely used to support weighted retrieval on static data collections (Harman et al., 1992). When the data collection is dynamic, inverted files are problematic; particularly, if long full-text documents are inserted, deleted, or modified frequently. An insertion (or deletion) of a data item requires as many modifications of postings in the inverted file as indexing features are contained in the document description. As a consequence, updating an inverted file is time consuming if large data items are updated frequently. In the case of infrequent updates, inverted files are highly appropriate; particularly, if an advanced buffering scheme is used as suggested in (Cutting & Pedersen, 1990).

Alternatively, *signatures* are appropriate to support efficient access to dynamic data collections (Croft & Savino, 1988), (Faloutsos, 1985), (Thanos, 1990). Every signature represents a complete document description which can be updated efficiently. As signatures are amenable to distributed environments retrieval is particularly efficient when performed on parallel processors (Pogue & Willet, 1987), (Stanfill & Kahle, 1986). Signatures, however, do not support document feature weighting and hence, their retrieval effectiveness is inferior to the retrieval effectiveness of a conventional fully weighted retrieval method.

In what follows, we present a retrieval method together with a new access structure facilitating both fast weighted retrieval and efficient updates of the access structure. Our access structure consists of *signatures and non-inverted descriptions* of data items. The signatures are used to compute approximate retrieval status values $RSV_0(q, d_j)$ first and the non-inverted data item descriptions are then used to determine the exact retrieval status values $RSV(q, d_j)$. The trick is that the approximate retrieval status values provide fairly tight *upper bounds* for the exact retrieval status values. We will see how we can take advantage of these upper bounds such that only a few exact retrieval status values have to be computed.

The retrieval method determining the approximate retrieval status values and the retrieval method determining the exact retrieval status values are given by the functions RSV_0 and RSV respectively. Let

$$\Phi := \{\varphi_0, \ldots, \varphi_{m-1}\} \quad (1)$$

be the indexing vocabulary (e.g. a set of terms) and let

$$D := \{d_0, \ldots, d_{n-1}\} \quad (2)$$

be the set of retrievable data items stored in the database (e.g. a document collection). We assume that the signatures consist of w bits.

$$\sigma(d_j) = (\sigma(d_j)[0], \ldots, \sigma(d_j)[w-1]) \quad (3)$$

Every indexing feature φ_i is assigned a bit position $p = h(\varphi_i)$ by means of a hash function $h : \Phi \rightarrow \{0, \ldots, w-1\}$. The function h specifies a signature $\sigma(d_j)$ for every data item d_j by setting the bit at position p iff d_j contains a feature φ_i which is hashed to this position.

$$\sigma(d_j)[p] := \begin{cases} 1 & \text{if } \exists \varphi_i \in d_j : h(\varphi_i) = p \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

We define the approximate and exact retrieval status values by

$$RSV_0(q, d_j) := \quad (5)$$

$$\frac{1}{|\vec{d_j}|} * \sum_{\varphi_i \in q : \sigma(d_j)[h(\varphi_i)]=1} nff(\varphi_i, q) * idf(\varphi_i)^2$$

$$RSV(q, d_j) := \frac{1}{|\vec{d_j}|} * \sum_{\varphi_i \in \Phi} nff(\varphi_i, q) * idf(\varphi_i)^2 * nff(\varphi_i, d_j) \quad (6)$$

where the normalized feature frequency $nff(\varphi_i, d_j)$ is determined by the feature frequency $ff(\varphi_i, d_j)$, i.e. the number of occurrences of φ_i in d_j, and the inverse data item frequency $idf(\varphi_i)$ is determined by the data item frequency $df(\varphi_i)$, i.e. the number of data items containing φ_i.

$$nff(\varphi_i, d_j) := \frac{ff(\varphi_i, d_j)}{\max\{ff(\varphi_h, d_j) \mid 0 \leq h < m\}} \quad (7)$$

$$idf(\varphi_i) := 1 - \frac{\log(1 + df(\varphi_i))}{\log(1+n)} \quad (8)$$

The size $|\vec{d_j}|$ of the description vector $\vec{d_j}$ is defined as usual.

$$|\vec{d_j}| := \sqrt{\sum_{i=0}^{m-1} (nff(\varphi_i, d_j) * idf(\varphi_i))^2} \quad (9)$$

The retrieval method given by RSV differs from a standard retrieval method (tf*idf weights and cosine) in that $nff(\varphi_i, d_j)$ is used rather than $ff(\varphi_i, d_j)$. The corresponding difference in retrieval effectiveness is neglectable. Experiments with test collections (CACM, MED, CRAN) have shown that the differences in average precision at three recall points are less than 0.4 %.

It is easy to show that $RSV(q, d_j) = 0$ if $\sigma(q) \wedge \sigma(d_j) = 0$. Furthermore, from the fact that $\varphi_i \in d_j$ implies $\sigma(d_j)[h(\varphi_i)] = 1$ and from $nff(\varphi_i, d_j) \leq 1$ follows

$$RSV(q, d_j) \leq RSV_0(q, d_j). \quad (10)$$

Let q be the user's query and let k be the number of data items the user wants to retrieve. The top k exact retrieval status values are computed efficiently by the algorithm shown in Figure 1. Upon termination of the first for-loop, the set J contains at least the indices j of those data items d_j which have a positive approximate retrieval status value. Upon termination of the while-loop the indices of the desired top k data items are contained in J_k if at least k exact retrieval status values are positive.

The evaluation algorithm shown in Figure 1 is based on an *access structure* that specifies the following functions.

```
procedure eval(q, k, var J_k);
begin
  J := ∅;
  for all d_j ∈ D do
    if σ(q) ∧ σ(d_j) ≠ 0 then
      compute approximate RSV_0(q, d_j);
      J := J ∪ {j};
    end;
  end;
  sort {RSV_0(q, d_j) : j ∈ J} by determining
    a bijective function
    j : {0 ... |J| −1} → J, r0 ↦ j(r0)
    such that RSV_0(q, d_{j(r0−1)}) ≥ RSV_0(q, d_{j(r0)})
    for r0 = 1, ..., |J| −1;
  r0 := 0; J_k := ∅; d_{j(|J|)} := −∞;
  while (r0 < |J|) and (|J_k| < k) do
    compute exact RSV(q, d_{j(r0)});
    INC(r0);
    for all j(r) ∈ {j(0), ..., j(r0 − 1)} − J_k do
      if RSV(q, d_{j(r)}) ≥ RSV_0(q, d_{j(r0)}) then
        J_k := J_k ∪ {j(r)};
      end;
    end;
  end;
end eval;
```

Figure 1: Query evaluation.

tions.

$$\sigma : d_j \mapsto \sigma(d_j) \quad (11)$$

$$\pi : d_j \mapsto \{(\varphi_i, nff(\varphi_i, d_j)) : \varphi_i \in d_j\} \quad (12)$$

$$idf : \varphi_i \mapsto idf(\varphi_i) \quad (13)$$

$$size : d_j \mapsto |\vec{d_j}| \quad (14)$$

The functions σ and π determine the signatures and the non-inverted data item descriptions respectively. The functions idf and $size$ provide scaling and normalization factors. The function values $\sigma(d_j)$ and $\pi(d_j)$ are determined completely by the data item d_j itself. The function values $idf(\varphi_i)$ and $size(d_j)$, however, depend on d_j as well as on other data items. Fortunately, the variation of $idf(\varphi_i)$ and $size(d_j)$ is very small if the data collection is sufficiently large. Thus, these two quantities have to be recomputed only if the domain of the data collection is shifting. In our system, this recomputation of idf and $size$ is performed within a long transaction (see Section 4).

For a rough time analysis it is sufficient to consider the computation of the approximate retrieval status values while scanning the signature file. We assume that the

n	=	$20,000,000$	=	number of data items
Δt	=	$1\mu sec$	=	time to process 32 bits of a signature while computing approx. RSV_0
Q	=	8	=	average no. of indexing features per query
P	=	16	=	number of processors

Figure 2: Sample parameters.

signature file is partitioned into 32 bit columns which are scanned simultaneously by P processors. Given the parameters shown in Figure 2, we obtain by

$$t = \frac{Q * n}{P} * \Delta t \quad (15)$$

an approximate response time of $t = 10$ seconds. This rough time analysis seems to indicate that 16 modern workstations may achieve response times that are only one order of magnitude slower than the response times achieved on a Connection Machine CM2 with 65,536 processors. A CM2 needs 1.268 seconds to search a document collection of the same size (Stanfill, 1992, p. 481). In terms of effectiveness, our retrieval method performs clearly better than the CM2 method because we are able to include both query feature and document feature weighting whereas in the CM2 approach, only the query features a weighted (Salton & Buckley, 1988a).

The storage requirements of our approach are not particularly moderate, since both a signature file and non-inverted item descriptions are needed. Compressing the signature file may deteriorate considerably the update efficiency. The descriptions of the data items, however, can be compressed with little loss of update efficiency. At this moment, it is not clear whether compression techniques developed for inverted item descriptions (Moffat & Zobel, 1992) are also appropriate to compress non-inverted item descriptions.

3 Weighted Retrieval of Semistructured Data

In this section, we present a novel probabilistic retrieval model for semistructured data. We call a data collection *semistructured* if there exists a database scheme which specifies both normalized attributes (e.g. dates or employee numbers) and non-normalized attributes (e.g. full text or images). Normalized attributes require the conversion of the attribute values into a canonical form such that the usage of the equality operator is meaningful. Such conversions are usually performed manually and since they are expensive, information seekers encounter an increasing number of semistructured data collections.

In contrast to conventional probabilistic retrieval models, our model takes into account the *database scheme.* We will see that our model is able to distinguish to which attributes the features belong. For instance, "Zurich" may belong to the address of a first person and at the same time, "Zurich" may belong to the name of the insurance of a second person. Clearly, these two persons should not be considered as similar even though they have a common feature.

Our retrieval model is complementary to Norbert Fuhr's model designed for fact and text retrieval (Fuhr, 1990), (Fuhr, 1992). We assume a data collection containing many different types of data items such that usually the user does not know which types contain relevant data items. Norbert Fuhr, on the other hand, assumes a single type of data items each consisting of several facts (fact part) and of a single piece of text (text part). Norbert is focusing on vague search conditions related to the facts whereas we are focusing on an automatic and weighted preselection of types containing relevant data items.

Our probabilistic retrieval model is based on the data model of the functional query language FQL* (Schäuble & Wüthrich, 1992). The building blocks of FQL* are sets of objects and functions mapping objects (possibly tuples of objects) to bags of objects (bags of tuples of objects). Such sets of objects and such functions can be grouped together to form *heterogeneous algebras.* A heterogeneous algebra consists of a family of sets (called phyla) and of a set of functions each mapping a Cartesian product of some phyla to a phylum (Birkhoff & Lipson, 1970). Furthermore, these heterogeneous algebras are arranged within a *hierarchy* by means of a partial order relation. This relation is often called subtyping, e.g. in QUEST (Cardelli, 1989). In what follows, we first define which algebras may belong to such a hierarchy and second, we define the partial ordering of these algebras.

1. Every algebra $\underline{X} = \langle X; \alpha_0, \alpha_1, \ldots \rangle$ consists of a set of functions $\alpha_h : X \rightarrow D_h$ and a set X in addition to the sets D_h which are not mentioned for simplicity in the structure $\underline{X}$. The common *domain* X of the functions α_h consists of object identifiers (OID's). The functions $\alpha_h : X \rightarrow D_h$ are called *attributes* as they determine the attribute values (properties) of the objects $x \in X$. Every attribute is assumed to have a unique name. Hence, attributes having different names are always considered as different attributes even though they may be identical as functions.

2. Given two algebras $\underline{X} = \langle X; \alpha_0, \alpha_1, \ldots \rangle$ and $\underline{Y} = \langle Y; \beta_0, \beta_1, \ldots \rangle$, we define that $\underline{X}$ is a *subalgebra* of $\underline{Y}$ (written as $\underline{X} \leq \underline{Y}$) iff $X \subseteq Y$ and $\{\alpha_0, \alpha_1, \ldots\} \supseteq \{\beta_0, \beta_1, \ldots\}$. The relation $\leq$ is a partial ordering (transitive, reflexive, and antisymmetric) representing the hierarchy of the algebras.

We say that an algebra $\underline{X}$ contains an object x iff the domain X of the algebra $\underline{X}$ contains x. Since an object may belong to different algebras, our retrieval method determines for every object as many retrieval status values as they are algebras containing this object. In other words, every retrievable data item $d_j = (\underline{X}, x)$ is specified by an algebra $\underline{X}$ and by an object identifier x. Thus, the set of retrievable data items is finite and it is defined explicitly by the hierarchy of algebras.

Given a hierarchy of algebras as described above, we will use the following notation. Let $\underline{X} = \langle X; \alpha_0, \ldots, \alpha_{k-1} \rangle$ be one of these algebras. Then, the set

$$A(\underline{X}) \quad := \{\alpha_0, \ldots, \alpha_{k-1}\} \tag{16}$$

consists of those attributes α_h that belong to the algebra $\underline{X}$. The set

$$\Phi \quad := \{\varphi_0, \ldots, \varphi_{m-1}\} \tag{17}$$

contains the features that can be derived from an attribute value, e.g. reduced terms. Finally, we define the following two sets for every data item $(\underline{X}, x)$.

$$\Psi(\underline{X}, x) \quad := \bigcup_{\alpha_h \in A(\underline{X})} \{(\alpha_h, \varphi_i) \mid \varphi_i \in \alpha_h(x)\} \tag{18}$$

$$\Phi(\underline{X}, x) \quad := \quad \{\varphi_i \mid \exists \alpha_h : (\alpha_h, \varphi_i) \in \Psi(\underline{X}, x)\} \tag{19}$$

The feature sets $\Phi(\underline{X}, x)$ are independent of the database scheme whereas the sets $\Psi(\underline{X}, x)$ do depend on the database scheme.

Assume that a person is interested in retrieving data items $(\underline{Y}, y)$ that are similar to $(\underline{X}, x)$. For we adopt the probability ranking principle by Robertson (Robertson, 1977) and we rank the data items $(\underline{Y}, y)$ in decreasing order of the retrieval status values $RSV(\underline{X}, x, \underline{Y}, y)$. The retrieval status value $RSV(\underline{X}, x, \underline{Y}, y)$ represents (up to an order preserving transformation) the probability that $(\underline{Y}, y)$ is relevant to a person who is interested in finding data items that are similar to $(\underline{X}, x)$. In conventional text retrieval, the estimation of the retrieval status values is usually based on features only. Ignoring the attributes may be inappropriate when retrieving information from *semistructured* data collections as shown in the following example.

OID	0	2
name	A. Meyer	C. Maier
address	Paradeplatz, Zurich	Bahnhofstr., Luzern
insurance	Winterthur Vers.	Berner Allgemeine
diagnosis	arthrosis	arthrosis

Table 1: Attributes of the algebra *Patient*.

OID	1
name	B. Meier
address	Bahnhofstr., Winterthur
bank	UBS Zurich

Table 2: Attributes of the algebra *Physician*.

Example: Given are the algebras *Patient* = ⟨*Patient*; *name*, *address*, *insurance*, *diagnosis*⟩ and *Physician* = ⟨*Physician*; *name*, *address*, *bank*⟩. The former contains two objects (OID = 0 and 2) and the latter contains one object (OID = 1). The corresponding attribute values are given in Tables 1 and 2. In the following we list the feature sets $\Phi(\underline{X}, x)$ of the objects shown in Tables 1 and 2.

$$\begin{aligned}
\Phi(\underline{Patient}, 0) &= \{\text{Meyer, Paradeplatz, Zurich,} \\
&\quad \text{Winterthur, Vers., arthrosis}\} \\
\Phi(\underline{Physician}, 1) &= \{\text{Meier, Bahnhofstr., Winterthur,} \\
&\quad \text{UBS, Zurich}\} \\
\Phi(\underline{Patient}, 2) &= \{\text{Maier, Bahnhofstr., Luzern,} \\
&\quad \text{Berner, Allgemeine, arthrosis}\}
\end{aligned}$$

Assume that a user is interested in finding data items that are similar to $(\underline{Patient}, 0)$. When ignoring the database scheme, we find that $\Phi(\underline{Patient}, 0)$ and $\Phi(\underline{Physician}, 1)$ have two common features (i.e. "Winterthur" and "Zurich") whereas $\Phi(\underline{Patient}, 0)$ and $\Phi(\underline{Patient}, 2)$ have only one common feature (i.e. "arthrosis"). In the former case, the common features "Winterthur" and "Zurich" are meaningless because they were derived from unrelated attributes (i.e. from *insurance* and *address* and from *address* and *bank* respectively). In the latter case, the common feature "arthrosis" is meaningful as it was derived from the same attribute (i.e. from *diagnosis*). [End of Example]

In what follows, we present our new probabilistic re-

trieval model which does not only cope with unrelated attributes (like *address* and *insurance*) but also with related attributes (e.g. like *diagnosis* and *therapy*) The underlying probability space has been adopted from Fuhr's BII model (Fuhr & Buckley, 1991). In this probability space $\langle \Omega, P \rangle$, an event $(\underline{X}, x, \underline{Y}, y) \subseteq \Omega$ consists of individual uses of the item $(\underline{X}, x)$ and of the item $(\underline{Y}, y)$.

Let $(\underline{X}, x)$ be the query item, i.e. the user is searching for data items $(\underline{Y}, y)$ that are similar to $(\underline{X}, x)$. According to the probability ranking principle, the data items $(\underline{Y}, y)$ should be presented to the user in decreasing order of the probabilities $P(R \mid \underline{X}, x, \underline{Y}, y)$. Analogously to Fuhr's BII-model, the event R consists of those pairs of individual uses of the two data items $(\underline{X}, x)$ and $(\underline{Y}, y)$ where $(\underline{Y}, y)$ is considered as *relevant* to $(\underline{X}, x)$. In order to determine these probabilities, several *assumptions* are made. These assumptions correspond bijectively to the assumptions from which the original BII-formula for unstructured data can be derived. See (Fuhr & Buckley, 1991) or (Schäuble, 1992) for the derivation of the BII-formula for unstructured data. In the case of semistructured data, we obtain

$$
\begin{aligned}
P(R \mid \underline{X}, x, \underline{Y}, y) = \quad & (20)\\
& c(\underline{X}, x) * P(R \mid \underline{Y}, y)\\
& * \prod_{\alpha_h \in A(\underline{X}) \cap A(\underline{Y})} \frac{P(R \mid \alpha_h, \underline{Y}, y)}{P(R \mid \underline{Y}, y)}\\
& * \prod_{\varphi_i \in \Phi(\underline{X}, x) \cap \Phi(\underline{Y}, y)} \frac{P(R \mid \varphi_i, A(\underline{X}), \underline{Y}, y)}{P(R \mid A(\underline{X}), \underline{Y}, y)}\\
& * \prod_{\psi \in \Psi(\underline{X}, x) \cap \Psi(\underline{Y}, y)} \frac{P(R \mid \psi, A(\underline{X}), \Phi(\underline{X}, x), \underline{Y}, y)}{P(R \mid A(\underline{X}), \Phi(\underline{X}, x), \underline{Y}, y)}
\end{aligned}
$$

where $c(\underline{X}, x)$ depends on the query item $(\underline{X}, x)$ but not on the data item $(\underline{Y}, y)$.

For practical retrieval, the factors occurring in the BII-formula are approximated. The factors of the first product of (20) are approximated by

$$
\frac{P(R \mid \alpha_h, \underline{Y}, y)}{P(R \mid \underline{Y}, y)} \approx \exp(\beta^2 * iaf(\alpha_h)^2) \qquad (21)
$$

where the inverse algebra frequency $iaf(\alpha_h)$ is defined as follows.

$$
\begin{aligned}
& iaf(\alpha_h) := \qquad (22)\\
& 1 - \frac{\log(1 + \text{<no. of algebras containing } \alpha_h\text{>})}{\log(1 + \text{<total no. of algebras>})}
\end{aligned}
$$

Note that the approximation is exact in the case where α_h is contained in every algebra. The factors of the second product of (20) are approximated as follows.

$$
\begin{aligned}
& \frac{P(R \mid \varphi_i, A(\underline{X}), \underline{Y}, y)}{P(R \mid A(\underline{X}), \underline{Y}, y)} \approx \qquad (23)\\
& \exp(nff(\varphi_i, \underline{X}, x) * idf(\varphi_i)^2 * nff(\varphi_i, \underline{Y}, y))
\end{aligned}
$$

The normalized feature frequency $nff(\varphi_i, \underline{X}, x)$ is determined by the feature frequency $ff(\varphi_i, \underline{X}, x)$, i.e. the number occurrences of the feature φ_i in the attribute values of the data item $(\underline{X}, x)$.

$$
\begin{aligned}
& nff(\varphi_i, \underline{X}, x) := \qquad (24)\\
& \frac{ff(\varphi_i, \underline{X}, x)}{\max\{ff(\varphi_h, \underline{X}, x) \mid 0 \leq h < m\}}
\end{aligned}
$$

The inverse data item frequency $idf(\varphi_i)$ is defined analogously to the inverse document frequency.

$$
\begin{aligned}
& idf(\varphi_i) := \qquad (25)\\
& 1 - \frac{\log(1 + \text{<no. of data items containing } \varphi_i\text{>})}{\log(1 + \text{<total no. of data items>})}
\end{aligned}
$$

The factors of the third product of (20) are approximated as follows.

$$
\begin{aligned}
& \frac{P(R \mid (\alpha_h, \varphi_i), A(\underline{X}), \Phi(\underline{X}, x))}{P(R \mid A(\underline{X}), \Phi(\underline{X}, x))} \approx \qquad (26)\\
& \exp(\gamma^2 * iaf(\alpha_h)^2 * idf(\varphi_i)^2)
\end{aligned}
$$

We finally assume $P(R \mid \underline{Y}, y) = c_0$ for all data items $(\underline{Y}, y)$. When defining

$$
\begin{aligned}
& RSV(\underline{X}, x, \underline{Y}, y) := \qquad (27)\\
& \log(P(R \mid \underline{X}, x, \underline{Y}, y)) - \log(c_0) - \log(c(\underline{X}, x)),
\end{aligned}
$$

we obtain the following retrieval status values where the constants β and γ have to be determined experimentally.

$$
\begin{aligned}
RSV(\underline{X}, x, \underline{Y}, y) = \quad & (28)\\
& \beta^2 * \sum_{\alpha_h \in A(\underline{X}) \cap A(\underline{Y})} iaf(\alpha_j)^2\\
& + \sum_{\varphi_i \in \Phi(\underline{X}, x) \cap \Phi(\underline{Y}, y)} \frac{w_i(\underline{X}, x) * w_i(\underline{Y}, y)}{\sqrt{\sum_{i=0}^{m-1} w_i(\underline{Y}, y)^2}}\\
& + \gamma^2 * \sum_{\psi \in \Psi(\underline{X}, x) \cap \Psi(\underline{X}, x)} iaf(\alpha_h)^2 * idf(\varphi_i)^2
\end{aligned}
$$

where

$$
\begin{aligned}
w_i(\underline{X}, x) &:= nff(\varphi_i, \underline{X}, x) * idf(\varphi_i), \qquad (29)\\
w_i(\underline{Y}, y) &:= nff(\varphi_i, \underline{Y}, y) * idf(\varphi_i). \qquad (30)
\end{aligned}
$$

This retrieval formula is a *generalization* of the classical retrieval formula determining the cosine of vectors

consisting of tf*idf-weights. This can easily be seen by assuming that every data item is assigned a single text attribute α_0. In this case, $iaf(\alpha_0) = 0$ and the first sum and the last sum in (28) vanish. The remaining sum is the classical cosine measure.

4 The System Architecture

Future retrieval systems will provide database capabilities including transaction management, recovery after failure, and a powerful data model. The traditional way to integrate information retrieval functionality and database functionality is to build an additional retrieval layer on top of an existing database system, e.g. on top of IRIS (Croft et al., 1992) or on top of ONTOS (Harper & Walker, 1992). We consider this traditional approach as questionable for the following reasons. First, existing databases do not have appropriate access structures to support weighted retrieval. Second, the underlying data model is usually inappropriate to express weighted retrieval, probabilistic behavior, and computed views because they are mostly based on finite Boolean algebras. Third, the performance required for weighted retrieval is far beyond the performance of most existing database systems. As pointed out in (Cattell & Skeen, 1992), simple lookups using in-memory structures can be performed in microseconds whereas databases typically respond to queries 100,000 times slower. The architecture of most databases is inherently disk oriented and hence, it is inappropriate for efficient weighted retrieval (including relevance feedback) which requires that the computation of the retrieval status values is performed in the main memory. Switching from disk orientation to main memory orientation cannot be achieved by simply increasing the cache size (Salem & Garcia-Molina, 1990). These are the arguments why we believe that a *new system architecture* is needed to integrate information retrieval functionality and database functionality.

In this section, we derive a suitable system architecture from the requirements stated in the introduction. We start with the requirement that the system has to support the derivation of numeric and non-numeric data from the stored data (e.g. document indexing). The derived data is often called a *computed view*. Examples of computed views are data item descriptions (e.g. for weighted retrieval), spectrograms (e.g. for editing audio recordings), or images recovered from compressed data. Computed views require the *interpretation* of attribute values. The interpretation of attribute values contradicts a basic assumption that governed the design and implementation of DBMS's in last decades. This assumption says that the DBMS should treat the elements of the database as uninterpreted objects (Chandra & Harel, 1980). We abandon this assumption and we postulate explicitly an architecture where attribute values can be interpreted efficiently within the kernel of the system, i.e. within the query evaluator.

- The efficient interpretation of attribute values is supported within the query evaluator.

Given that certain attribute values must be interpreted within the query evaluator, the next question is how is the interpretation formulated. It seems that many relevant interpretations such as document indexing, signal processing, and compression are most easily formulated in a procedural way. Hence, our system has to be extensible by external functions that can be written in an appropriate programming language. We require an *extensibility* that is both simple and flexible. By simple we mean that programming an external function does not require any knowledge about the internal data structures of the DBMS (e.g. about the query tree) nor about internal processes (e.g. about the transaction manager). By flexible we mean that external functions can actually "compute" a result possibly by means of special purpose hardware (e.g. by a DSP-board for signal or image processing).

- The system is extensible in a simple and flexible way.

Since we require a simple extensibility, the transaction manager cannot impose any restrictions on the external functions. A straightforward approach to avoid any interferences between the transaction manager and the external functions is to use a *serial scheduler* which does not allow overlapping transactions. When using a serial scheduler, however, long transactions such as recomputing $idf(\varphi_i)$ and $|\vec{d_j}|$ block other transactions for a long time. In many applications of retrieval systems, it is acceptable if long write transactions block other write transactions because the long write transactions can be executed when no other write transactions have to be executed quickly. On the other hand, read transactions should never be blocked by write transactions. This is achieved by *a sequence of identical query evaluators* working in parallel. In this way, one query evaluator may perform a long write transaction while the other query evaluators are available to execute read transactions.

- Concurrency is achieved by a sequence of identical query evaluators each processing one transaction after the other (serial scheduling).

Weighted retrieval on semistructured data requires that the corresponding read transactions are executed within extremely short time intervals. Such short response times cannot be achieved by a conventional architecture of a DBMS as explained in Section 1. Instead

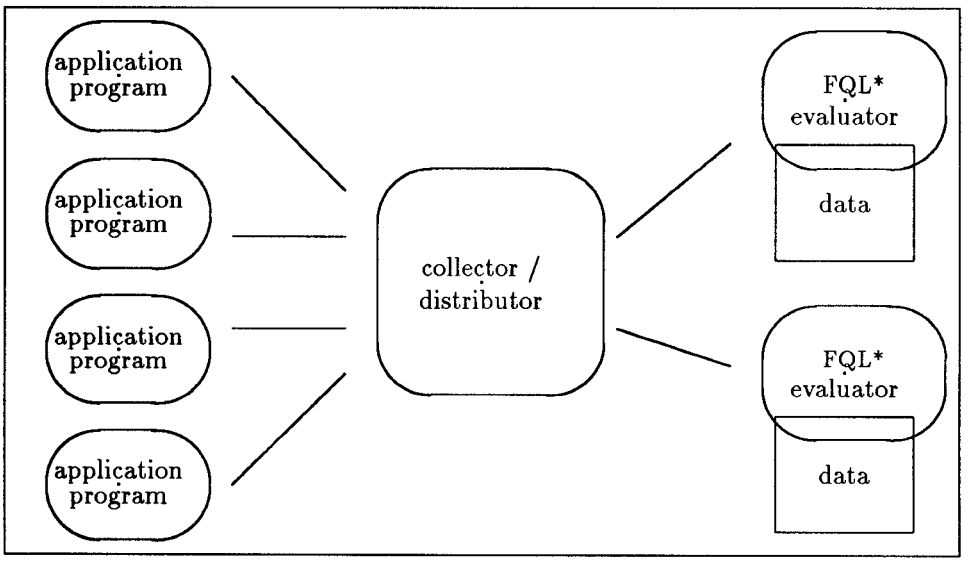

Figure 3: SPIDER's system architecture.

we will rely on an *interpreter architecture* that is particularly well suited for short response times. The availability of large memories facilitates that at least the structured data can be kept in the main memory whereas parts of the unstructured data (long texts, audio, video, images) will still be in the secondary or tertiary storage.

- The query evaluators are organized as fast interpreters.

In a distributed environment where application programs and query evaluators are working in parallel, an appropriate communication protocol has to be defined. In order to achieve simplicity, we require that this communication protocol and the query language be identical. Since the communication protocol between the application programs and the query evaluators is a limiting factor of what can be retrieved in which form, a query language is required that has an outstanding expressive power. As shown in (Chandra, 1988), relational query languages are based on finite Boolean algebras which are inappropriate to express probabilistic behavior. But expressing probabilistic behavior is required, since the interpretation of text (e.g. probabilistic indexing (Fuhr, 1989)), the interpretation of images (e.g. by OCR (Govindan & Shivaprasad, 1990)), and the interpretation of audio (e.g. speech recognition (Lee, 1989)) are inherently probabilistic. Furthermore, finite Boolean algebras are inappropriate for the derivation of numeric data from the stored data. Thus, instead of a relational query language we will use the functional query language FQL* that is appropriate to express probabilistic behavior as well as numeric computations (Schäuble & Wüthrich, 1992). FQL* has an outstanding expressive power. It was shown to be more expressive than SQL, more expressive than Datalog with stratified negation, and even more expressive than fixpoint query languages. Thus, each query formulated in one of these languages (e.g. SQL) can be translated into an FQL* query.

- The functional query language FQL* is used as a query language and as the communication protocol between the application programs and the evaluators.

Based on the five architectural features mentioned above, we have developed the server-client architecture shown in Figure 3. The server consists of a *collector/distributor* which is collecting requests from the application programs. The requests are buffered in a request queue. The *query evaluators* are also running as clients. They fetch the buffered requests and interpret them. This architecture we have derived systematically from our list of requirements is similar to the architecture of the TPK prototype running on two DEC Fireflies (Li & Naughton, 1988). TPK is also based on serial scheduling and it also manages identical copies of the database in the main memory.

A read transaction is executed by one evaluator whereas a write transaction is executed by all evaluators to guarantee consistency. More precisely, a write transaction is executed by the first evaluator first. Upon its termination it is executed by the next evaluator in the sequence of query evaluators and so on. In this way, all write transactions are propagated in the same order

through the sequence of query evaluators. Hence, the database of every evaluator was or is still equal to the current database of the last query evaluator.

For *recovery after failure* we adopt the logging and checkpointing strategies of the system M (Salem & Garcia-Molina, 1990). The collector/distributor maintains a log file such that the effects of the write transactions can be reconstructed in the case of a failure. The so-called checkpoint process is copying regularly the data collection of the first evaluator to the disk. Having copied successfully the data collection, the checkpoint process inserts checkpoints into the log file. In this way, only transactions executed since the last checkpoint have to be considered when reconstructing the data collection after a failure.

The availability of external functions facilitates a flexible *access to remote data*. For instance, a relation of a remote database can be represented by an external FQL* function that is mapping the primary keys to tuples containing the corresponding attribute values. When such an external function is called, an appropriate communication channel is established, a query is generated, the required data is then retrieved from the remote database, and finally, the retrieved data is converted into an FQL* function. In this way, accessing remote data is accomplished by encapsulating the communication, the querying, and the conversion into external functions.

In this section, we derived a system architecture which is fairly different from the architecture of a conventional DBMS. Nevertheless, the system seems to meet well the requirements of many applications such as management information systems, office information systems, or clinic information systems (Frei & Schäuble, 1991a).

5 Outlook

This paper is a step towards the SPIDER retrieval system we have started to implement. Ongoing activities include performance evaluations of our access structure and the design and implementation of the FQL*-evaluators. The "implementation" of the BII-formula for semistructured data collections is fairly simple because it can easily be formulated in FQL*. The FQL*-evaluators and the formulation of information retrieval related queries will be the topic of a forthcoming paper. The estimation of the retrieval effectiveness of the BII-model for semistructured data may be more difficult, since we lack a semistructured test collection. The same difficulties were encountered in the IOTA and RIME projects (Chiaramella et al., 1986), (Berrut & Chiaramella, 1989). We hope to overcome this problem by means of the usefulness measure which needs only incomplete and therefore inexpensive relevance assessments (Frei & Schäuble, 1991b). The results of the ongoing activities will show if SPIDER will meet the requirements stated at the outset of the introduction.

Acknowledgements: The author would like to thank Gabriela Keller, Daniel Knaus, and Daniel Kohli for working on the implementation of the SPIDER system. In particular, I thank Christian P. Jacobi and Daniel Knaus for discussing various aspects of SPIDER's architecture. Finally, I would like to thank Daniel Knaus once more for working on another access structure that influence indirectly the access structure presented in this paper.

References

Berrut, C., & Chiaramella, Y. (1989). Indexing Medical Reports in a Multimedia Environment: the RIME Experimental Approach. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 187–197.

Birkhoff, G., & Lipson, J. D. (1970). Heterogeneous Algebras. *Journal of Combinatorial Theory*, 8,115–133.

Cardelli, L. (1989). Typeful Programming. Technical Report 45, DEC Systems Research Center, Palo Alto.

Cattell, R. G. G., & Skeen, J. (1992). Object Operations Benchmark. *ACM Transactions on Database Systems*, 17(1),1–31.

Chandra, A., & Harel, D. (1980). Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21(2),156–178.

Chandra, A. K. (1988). Theory of Database Queries. In *Symposium on Principles of Database Systems (PODS)*, pp. 1–9.

Chiaramella, Y., Defude, B., Kerkouba, D., & Bruandet, M. F. (1986). IOTA: a Prototype of an Information Retrieval System. In *ACM SIGIR Conference on R&D in Information Retrieval.*

Croft, W. B., & Savino, P. (1988). Implementing Ranking Strategies Using Text Signatures. *ACM Transactions on Information Systems*, 6(1),42–62.

Croft, W. B., Smith, L. A., & Turtle, H. (1992). A Loosly-Coupled Integration of a Text Retrieval System and an Object-Oriented Database System. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 223–232.

Cutting, D., & Pedersen, J. (1990). Optimization for Dynamic Index Maintenance. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 405–411.

Faloutsos, C. (1985). Access Methods for Text. *ACM Computing Surveys*, 17(1),49–74.

Frei, H. P., & Schäuble, P. (1991a). Designing a Hypermedia Information System. In Karagiannis, D., editor, *DEXA '91 Conf.*, pp. 449–454. Springer-Verlag.

Frei, H. P., & Schäuble, P. (1991b). Determining the Effectiveness of Retrieval Algorithms. *Information Processing & Management*, 27(2–3),153–164.

Fuhr, N. (1989). Models for Retrieval with Probabilistic Indexing. *Information Processing & Management*, 25(1),55–72.

Fuhr, N. (1990). A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In McLeod, D., Sacks-Davis, R., & Schek, H., editors, *International Conference on Very Large Data Bases*, pp. 696–707, Los Altos, CA. Morgan Kaufman.

Fuhr, N. (1992). Integration of Probabilistic Fact and Text Retrieval. In *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 211–222.

Fuhr, N., & Buckley, C. (1991). A Probabilistic Learning Approach for Document Indexing. *ACM Transactions on Information Systems*, 9(3),223–248.

Govindan, V. K., & Shivaprasad, A. P. (1990). Character Recognition—A Review. *Pattern Recognition*, 23,671–683.

Harman, D. (1992). Relevance Feedback Revisited. In Belkin, N., Ingwersen, P., & Pejtersen, A. M., editors, *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 1–10.

Harman, D., Fox, E., Baeza-Yates, R. A., & Lee, W. (1992). Inverted Files. In Frakes, W. B., & Baeza-Yates, R. A., editors, *Information Retrieval, Data Structures & Algorithms*, pp. 28–43. Prentice-Hall, Englewood Cliffs, NJ.

Harper, D. J., & Walker, D. M. (1992). ECLAIR: an Extensible Class Library for Information Retrieval. *The Computer Journal*, 35(3),256–267.

Hoppe, J., editor (1992). *Integrated Management of Technical Documentation, The System SPRITE*, volume 1 of *Research Reports ESPRIT*. Springer-Verlag.

Lee, K. F. (September, 1989). Hidden Markov Models: Past, Present, Future. In *European Conference on Speech Communication and Technology*, pp. 148–155.

Li, K., & Naughton, J. (1988). Multiprocessor Main Memory Transaction Processing. In *Int. Symp. Databases Parallel Distributed Systems*, pp. 177–187.

Moffat, A., & Zobel, J. (1992). Parametrised Compression for Sparse Bitmaps. In Belkin, N., Ingwersen, P., & Pejtersen, A. M., editors, *ACM SIGIR Conference on R&D in Information Retrieval*, pp. 274–285.

Pogue, C., & Willet, P. (1987). Use of Text Signatures for Document Retrieval in a Highly Parallel Environmen. *Parallel Computing*, 4(3),259–268.

Robertson, S. E. (1977). The Probability Ranking Principle in IR. *Journal of Documentation*, 33(4),294–304.

Salem, K., & Garcia-Molina, H. (1990). System M: A Transaction Processing Testbed for Memory Resident Data. *IEEE Transactions on Knowledge and Data Engineering*, 2(1),161–172.

Salton, G., & Buckley, C. (1988a). Parallel Text Search Methods. *Communications of the ACM*, 31(2),202–215.

Salton, G., & Buckley, C. (1988b). Term Weighting Approaches in Automatic Text Retrieval. *Information Processing & Management*, 24(5),513–523.

Schäuble, P. (1992). A Tutorial on Information Retrieval. Information Retrieval Course 1991/92, ETH Zurich.

Schäuble, P., & Wüthrich, B. (1992). On the Expressive Power of Query Languages. Technical Report 173, ETH Zurich, Department of Computer Science.

Stanfill, C. (1992). Parallel Information Retrieval Algorithms. In Frakes, W. B., & Baeza-Yates, R. A., editors, *Information Retrieval, Data Structures & Algorithms*, pp. 459–497. Prentice-Hall, Englewood Cliffs, NJ.

Stanfill, C., & Kahle, B. (1986). Parallel Free-Text Search on the Connection Machine System. *Communications of the ACM*, 29(12),1229–1239.

Thanos, C., editor (1990). *Multimedia Office Filing. The MULTOS Approach*. North-Holland, Amsterdam.