DOLORES:
A System for Logic-Based Retrieval of Multimedia Objects

Norbert Fuhr, Norbert Gövert, Thomas Rölleke
University of Dortmund, Germany

Abstract We describe the design and implementation of a system for logic-based multimedia retrieval. As high-level logic for retrieval of hypermedia documents, we have developed a probabilistic object-oriented logic (POOL) which supports aggregated objects, different kinds of propositions (terms, classifications and attributes) and even rules as being contained in objects. Based on a probabilistic four-valued logic, POOL uses an implicit open world assumption, allows for closed world assumptions and is able to deal with inconsistent knowledge. POOL programs and queries are translated into probabilistic Datalog programs which can be interpreted by the HySpirit inference engine. For storing the multimedia data, we have developed a new basic IR engine which yields physical data abstraction. The overall architecture and the flexibility of each layer supports logic-based methods for multimedia information retrieval.

1 Introduction

New multimedia applications like digital libraries, video-on-demand or electronic kiosks are reaching the end user. Thus, the development of multimedia information systems is a growing area of research. A crucial issue in many of the applications is content-oriented access to multimedia objects.

Most multimedia information systems are based on (object-oriented) database management systems. For content-based retrieval, however, these systems are not adequate, since they hardly offer any support for performing uncertain inference.

On the other hand, current information retrieval (IR) approaches lack the support for multiple abstraction levels. In the database field, there are semantic data models (like e.g. the entity-relationship model) as high-level models which allow for a more application-oriented modeling of the domain under consideration. Next, there is the logical level (e.g. the relational model) which includes logical (i.e. descriptive) query languages (e.g. SQL). At the bottom layer, there is the physical level which deals with access structures (e.g. indexes) and the algorithms operating on them. The concept of data independence ensures a clear separation between all three levels, making e.g. query formulation at the logical level independent of the organization at the physical level. Looking at IR, we see that so far, no approach covering these three abstraction levels has been described. In classic IR systems like e.g. SMART ([Buckley 85]) or INQUERY ([Callan et al. 92]), there is only a basic logical level where documents typically are represented as sets of weighted terms, and queries are either like documents or Boolean combinations of terms. This logical structure is mapped one-to-one onto the physical level, only organized in inverted files for speeding up query processing. However, problems arise with this organization when the set of terms is not fixed in advance, e.g. with phrases or compound words. Multimedia retrieval, e.g. typical methods for similarity-based image retrieval, also cannot be performed within this traditional framework.

Another shortcoming of current IR approaches is the poor suitability of the underlying classical models (originally developed for unstructured text documents) for multimedia environments, namely for three major reasons:

1. Since text retrieval typically only considers the presence/absence of terms, it is logically founded on propositional logic, where each term corresponds to a proposition, to which a document assigns truth values (see [Rijsbergen 89]). In multimedia IR, however, we have to deal with e.g. temporal or spatial relationships which cannot be expressed in this logic.
2. Classic IR models treat documents as atomic units. On the other hand, since multimedia documents comprise different media, they are always structured documents. Through the additional use of links, hypermedia documents have an even more complex structure. Thus, document nodes linked together may form a single answer item.
3. When combining the knowledge of linked hypermedia nodes, these nodes may contain contradictory information, which cannot be handled properly by most models.

In this paper, we present a new logic-based approach for hypermedia[1] retrieval which remedies the shortcomings of classical IR models. Our system DOLORES (*Do*rtmund *l*ogic-based *o*bject *re*trieval *s*ystem) is based on a multi-layered architecture which corresponds to the different data abstraction levels mentioned above. The major contributions of this paper are the following:

- We show how several advanced approaches to hypermedia retrieval can be integrated within a single IR system.
- A multi-layered system architecture for hypermedia retrieval is devised, ranging from a graphical user interface for object-oriented query formulation and result display to the low-level data structures as used by most classical text retrieval systems.

[1] We regard hypermedia documents as the most general case, subsuming multimedia, structured and hypertext documents.

- In order to bridge the gap between logic-based IR and classical retrieval algorithms and data structures, we have defined a logical level for basic IR systems; this allows for physical data independence, i.e. the logical level is independent of physical design issues (e.g. presence or absence of indexes) which are more related to efficiency.

In the remainder of this paper, we first give a survey on the overall architecture of DOLORES and describe its underlying multimedia retrieval model. Then we present a probabilistic object-oriented logic for realizing this model, which uses probabilistic Datalog as inference mechanism. The underlying basic IR engine is described in section 5. A description of the implementation and some application examples are given in section 6. Finally, we summarize our results and give an outlook on further work.

2 General approach

2.1 System architecture

In order to cope with the shortcomings of classical IR models in the context of hypermedia IR, we have developed a logic-based approach; logic-based IR allows for more complex inferences and flexible retrieval strategies. For coping with hypermedia retrieval, we have devised probabilistic Datalog as a combination of uncertain inference with a restricted form of predicate logic (i.e. horn clause predicate logic without function symbols); Datalog has the advantage that it can be processed efficiently even on large databases. Contradictory information is handled by extending probabilistic Datalog to a probabilistic four-valued logic allowing for "unknown" and "inconsistent" as additional truth values. Using this logic as basic inference mechanism, we have designed POOL (Probabilistic Object-Oriented Logic) as a high-level logic for retrieval of hypermedia objects.

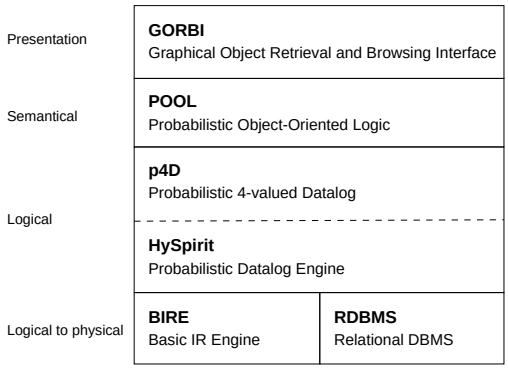

Figure 1: Overall architecture of DOLORES

For integrating these different approaches, we have designed a multi-layered architecture for IR systems, where we have a presentation layer above three layers corresponding to the different data abstraction levels (see also figure 1).

Presentation: The top level of our system consists of a graphical object retrieval and browsing interface (GORBI). This component supports dictionary browsing, graphical query formulation, result survey and display of hypermedia objects. (Due to space limitations, this component is not described further in this paper).

Semantical: Our approach is based on the FERMI multimedia retrieval model as semantic data model (see below). For implementing this model, we use POOL.

Logical: The corresponding layer consists of two sublayers: POOL programs are transformed into probabilistic four-valued Datalog, which in turn is mapped onto probabilistic (two-valued) Datalog (pD). Our basic inference engine HySpirit executes pD programs.

Logical to physical: The facts HySpirit operates on are stored in a new basic IR engine (BIRE) which provides physical data abstraction. For retrieval, this system offers a set of logical predicates which are independent from the actual physical data structures and algorithms used for implementing these predicates. In addition, facts also can be stored in a relational database management system.

2.2 The FERMI multimedia retrieval model

There are three different views for multimedia documents, namely the logical, layout and content view (see e.g. [Meghini et al. 91]). The logical view deals with the logical structure of the document (e.g. chapters, sections and paragraphs) and the layout view with the presentation of the document on an output medium (e.g. pages and rectangular areas within a page). The content view addresses the semantic content of a document, and thus is the focus of IR.

The FERMI multimedia model (FMM) presented in [Chiaramella et al. 96] considers a content structure which is closely related to the logical structure (in contrast to classical IR models treating documents as atomic units). Thus, the answer to an IR query may not only return whole documents, but also substructures according to the logical structure. For this purpose, the FMM uses a representation for the logical structure which focuses on those elements which are important for retrieval, thus neglecting issues dealt with by other models (e.g. ODA, SGML) which also relate to other tasks (e.g. authoring, presentation).

A database is a set of documents, and each document is a tree consisting of typed structural objects (e.g. book, chapter, section, paragraph, image), where the leaves contain single-media data. Hypermedia documents also contain links between different nodes (possibly from different documents). Nodes are assigned attributes, which can be either standard attributes (e.g. author or creation date) or so-called index expressions describing the content of a node. The latter are initially assigned to the leave nodes only, where the indexing language depends on the media type. For example, for text, there are languages for describing the physical, the structural and the symbolic content, whereas for images, there is in addition also a spatial and a perceptive content.

Depending on the class of an attribute, attribute values may be ascending or descending along the hierarchy. For example, the authors of different nodes are propagated upwards (involving an attribute-specific merge operation), whereas the publishing date of a complete document is propagated downwards. The index expressions assigned to leave nodes are also propagated upwards. Like any data model, the FMM also supports typing of nodes, links and attributes.

Retrieval in this model follows the logical approach, i.e. search for document nodes n which imply the query

q. In order to consider the hierarchical structure of documents, the overall retrieval strategy is to search for the smallest possible units (i.e. low-level nodes) fulfilling this criterion. This strategy is supported by the upward propagation of index expressions. The original formulation of the FMM uses predicate logic as basic retrieval logic, whereas in [Lalmas 97], a refinement of the retrieval model (but restricted to propositional logic) using Dempster-Shafer theory is described.

3 A probabilistic object-oriented logic

3.1 Description

The motivation behind the development of POOL ([Rölleke 98]) was the need for a logic for retrieval of structured objects, like e.g. hypermedia documents. Its major features are the support of nested objects and the combination of a restricted form of predicate logic with probabilistic inference.

Objects in POOL have an identifier and a content, which is a POOL program. Objects with a nonempty content are also called *contexts*, because the logical formulas forming the content first are valid only within this object/context. Propagation of this knowledge into other contexts is performed via the process of augmentation (see below). A program is a set of clauses, where each clause may be either a context, a proposition or a rule. In the following example, we have several nested contexts: an article `a1` consisting of two sections `s11` and `s12`, where the latter again contains two subsections `ss121`, `ss122`. A *proposition* is either a *term*[2] (like `image`), a *classification* (e.g. `article(a1)`) or an *attribute* (e.g. `s11.author(smith)`). Propositions also may be negated (e.g. `not presentation`) or assigned a probability (e.g. `0.6 retrieval`).

```
a1[ s11[ image  0.6 retrieval  presentation ]
    s12[ ss121[ audio indexing ]
         ss122[ video  not presentation ] ] ]
s11.author(smith)
s121.author(miller) s122.author(jones)
a1.pubyear(1997)
article(a1) section(s11) section(s12)
  subsection(ss121) subsection(ss122)
docnode(D) :- article(D)
docnode(D) :- section(D)
docnode(D) :- subsection(D)
mm-ir-doc(D) :- docnode(D) &
                D[audio & retrieval]
german-paper(D) :- D.author.country(germany)
```

A *rule* consists of a head and a body, where the head is either a proposition or a context containing a proposition; the rule body is a conjunction of subgoals, which are propositions or contexts containing a rule body. In the example program shown above, the first three rules state that articles, sections and subsections are document nodes. Next, we classify documents talking both about `audio` and `retrieval` as `mm-ir-doc`. We also allow for path expressions in rule bodies as shown in the last rule, which is a shorthand for `D.author(X) & X.country(germany)`.

A *query* consists of a rule body only, e.g. `?- D[audio & indexing]`, for which `ss121` would be an answer.

A basic assumption underlying POOL is that clauses only hold for the context in which they are stated. For example, the query `?- D[audio & video]` cannot be answered by an atomic context, since `audio` occurs in `ss121` and `video` in `ss122`. For dealing with content-based retrieval of aggregated contexts, POOL uses the concept of *augmentation*. For this purpose, propositions are propagated to surrounding contexts — similar to upward propagation of attribute values in the FMM. This way, the content of a context is augmented by the content of its components. Thus, the last query would be fulfilled by the context `s12`. However, augmentation also may lead to inconsistencies: when we ask `?- D[image & video]` and combine contexts `s11` and `s12`, then we get a contradiction with respect to `presentation`. In classical logic, this inconsistency would allow us to infer anything. In order to avoid these problems, POOL is based on four-valued logic (as described in [Rölleke & Fuhr 96]), treating only `presentation` as inconsistent and yielding `a1` as correct answer to the last query. We describe the four-valued logic in some detail in section 4.1.

Since we have no inference engine which implements POOL directly, we map POOL programs onto probabilistic datalog programs, for which we have the HySpirit inference engine presented in section 4.1.

3.2 FMM and POOL

From the description of POOL given above, it is obvious that it is more general than the FMM. The FMM poses a number of reasonable restrictions on hypermedia documents (e.g. that only leaf nodes have a content, or the type hierarchy on document nodes) which are not present in POOL. On the other hand, both approaches deal with nested objects, and the retrieval strategy of the FMM is already integrated in POOL.

The only feature of FMM that we have to model explicitly in POOL is propagation of attribute values. This can be achieved by formulating an appropriate rule for each attribute, depending on the propagation direction, e.g.

```
D.author(A) :- D[S] & docnode(D) & docnode(S) &
               S.author(A)
S.pubyear(Y) :- D[S] & docnode(D) & docnode(S) &
                D.pubyear(Y)
```

The first rule performs (recursively) upward propagation of author names: When a document node `D` contains a subnode `S`, then any author of `S` also is an author of `D`. In a similar way, the second rule yields downward propagation of the publication year.

4 Probabilistic Datalog

4.1 Description

Probabilistic Datalog (pD) is an extension of ordinary (two-valued) Datalog (2D). On the syntactical level, the only difference is that with facts and rules, also a probabilistic weight may be given, e.g.

```
0.7 docterm(d1,ir).  0.8 docterm(d1,db).
link(d2,d1).
0.5 related(D,D1) ← link(D,D1).
about(D,T) ← docterm(D,T).
about(D,T) ← related(D,D1) , about(D1,T).
q1(X) ← about(X,ir) , about(X,db).
```

Informally speaking, the probabilistic weight gives the probability that the following predicate is true. In our example, document `d1` is with probability 0.7 about IR and with probability 0.8 about databases (DB). The first rule states that two documents are semantically related

[2] Throughout this paper, we use the word "term" in the typical IR meaning, not in the usual logical meaning. Logically, terms stand for argument-free predicates.

(with probability 0.5) if there is an explicit link in between. The rule for q1 searches for documents dealing with both of these topics. Assuming that index terms are stochastically independent, we can compute a probability of $0.7 \cdot 0.8 = 0.56$ for q1(d1). As a more complex example, consider the case of d2 involving the second rule for about(D,T) stating that a document is about a term if it is related to another document indexed with this term. Thus, we retrieve document d2 with probability $0.5 \cdot 0.7 \cdot 0.8 = 0.28$ (The relatedness between d1 and d2 is the same probabilistic event for both terms).

In addition to independent events, pD also supports disjoint events. Besides modeling imprecise attribute values, this feature makes it possible to use linear retrieval functions like e.g. in the vector space model. For this purpose, we can treat query terms as disjoint events, e.g.

```
0.6 queryterm(q2,ir).  0.4 queryterm(q2,db).
ret(Q,D) ← queryterm(Q,T), docterm(D,T).
?- ret(q2,D)
```

Here we state that with probability 0.6, the query q2 is about ir and with probability 0.4, it is about db. Searching for documents which have terms in common with the query, we retrieve d1 with probability $0.6 \cdot 0.7 + 0.4 \cdot 0.8 = 0.74$.

In [Fuhr & Rölleke 98], we describe the extension of probabilistic Datalog to a probabilistic four-valued logic. Let us first consider deterministic four-valued Datalog (4D). In the model-theoretic semantics of 2D, an interpretation only contains the (unnegated) atoms of the Herbrand base. In contrast, we now assume that for 4D, an interpretation may contain both positive and negated atoms. Thus, for a specific fact, a model may contain the positive atom, the negated atom, both the positive and the negated atom or none. This corresponds to the four truth values true (T), false (F), inconsistent(I) and unknown (U). In this logic, rules are viewed as conditional facts, not as implication formulas. A rule like $p \leftarrow q$ is interpreted such that the positive atom p is in the model M, given that $q \in M \wedge (\neg q) \notin M$. Thus the precedent q must be true (and not inconsistent), from which we conclude that the consequence is true or possibly inconsistent (in case there is evidence for the negative fact as well).

Based on this interpretation, we map 4D programs onto 2D such that for each predicate q in 4D, we have two predicates p_q and n_q in 2D, where the first gives the positive information (true or inconsistent) and the latter the negative information (false or inconsistent) for q. Thus, we are also able to infer negative facts or state them explicitly. For example, the 4D program

```
¬ person(fido).
student(X) ← person(X), enrolled(X).
¬ student(X) ← ¬ person(X).
```

is mapped onto

```
n_person(fido).
p_student(X) ← p_person(X), ¬ n_person(X),
               p_enrolled(X), ¬ n_enrolled(X).
n_student(X) ← n_person(X), ¬ p_person(X).
```

Queries in 4D are mapped onto four separate queries in 4D, namely one for each truth value in 4D. E.g. for the 4D query ?- student(X), we get the instances yielding a truth value of false (i.e. fido) by means of the following query:

```
?- n_student(X), ¬ p_student(X).
```

4D uses an implicit open world assumption. This fits with basic IR requirements, since we typically know what a document is about, but it is difficult to state for sure that a document is certainly not about a specific topic. Thus, our approach would yield unknown as truth value when a user asks for a topic which is not explicitly covered by a document. On the other hand, when we have to deal with document attributes (e.g. authorship), then a closed world assumption (cwa) should be used. This behavior can be achieved by stating a cwa in 4D, e.g. #cwa(author(D,A)). This declaration is translated into the 2D rule

```
n_author(D,A) ← ¬ p_author(D,A).
```

Thus, the 4D program

```
docterm(d1,ir).  author(d1,smith).
```

yields unknown for ?- docterm(d1,db), but false for ?- author(d1,miller).

Probabilistic four-valued Datalog (p4D) differs from 4D only in the additional specification of the probabilistic parameters, i.e. we have to note three probabilities (for true/false/inconsistent), from which the probability of unknown can can be derived as the complement to 1.

4.2 POOL and probabilistic Datalog

In order to implement an inference engine for POOL, we map POOL onto p4D. This mapping is fairly straightforward. POOL propositions are mapped onto predicates with an additional argument, namely the id of the context to which the proposition belongs:

- Terms are mapped onto the predicate term(Term, Context).
- Classifications are represented by the predicate class(Class,Instance,Context).
- Attributes are transformed into the predicate attr(Attribute,Object,Value,Context).

The aggregation structure of contexts is represented by means of facts for the predicate part(D,P), where the first parameter denotes the surrounding context and the latter the embedded one. Probabilities and negations in POOL can be mapped directly onto the corresponding p4D notation. By element-wise transformation, we can map propositions as well as rules, queries and the context structure of a POOL program into the corresponding p4D program.

For augmentation, we have to add a few rules to the resulting program. Instead of deterministically propagating statements of a context to all surrounding contexts, we use a probabilistic version based on the notion of accessibility: It is only with a certain probability that a context has access to (comprises) the content of its embedded contexts. For this purpose, we define an accessibility predicate: for a pair of contexts (c_1, c_2), it gives the probability that the content of context c_2 is contained in context c_1. The strength of the relationship and the stochastic dependencies between different pairs can be defined in arbitrary ways, depending on the actual application. Here we consider a simple solution only, where all direct subcontexts of a supercontext are accessed independently and with the same probability. Thus, we can derive the accessibility relationship from the part relation:

```
0.6 acc(D,P) ← part(D,P).
0.6 acc(D,P) ← part(D,P1), acc(P1,P).
```

The concept of augmentation also can be applied for dealing with hypermedia retrieval. When there is a link from node n_1 to n_2, then the content of n_2 should be considered as being contained in n_1 in a similar way as that of any subnode of n_1. In order to implement this retrieval strategy, we extend the definition of the acc predicate such that it also considers links (formulated as link attribute in POOL) between document nodes:

```
0.4 acc(D,L) ← attr(link,D,L,C).
0.4 acc(D,L) ← attr(link,D,L1,C), acc(L1,L).
```
Then we can formulate the following rules for augmenting both positive and negative propositions:
```
term(T,C) ← acc(C,C1), term(T,C1).
¬ term(T,C) ← acc(C,C1), ¬ term(T,C1).
class(Cl,I,C) ← acc(C,C1), class(Cl,I,C1).
¬ class(Cl,I,C) ← acc(C,C1), ¬ class(Cl,I,C1).
attr(A,O,V,C) ← acc(C,C1), attr(A,O,V,C1).
¬ attr(A,O,V,C) ← acc(C,C1), ¬ attr(A,O,V,C1).
```
As an application of these rules, consider the following POOL program:
```
d1[ s1[audio indexing]  s2[image retrieval]
    s3[video  not retrieval] ]
```
Based on an open world assumption, here the query `?- D[audio & indexing]` would retrieve `s1` with probabilities 1/0/0/0 (for true/false/inconsistent/unknown). For `d1`, the corresponding probabilities are 0.6/0/0/0.4 only, due to the definition of accessibility. So the more specific document node gets a higher probability than its supernode in case the subnode already implies the query. Thus, we have implemented a probabilistic version of the FMM retrieval strategy. On the other hand, asking `?- D[audio & image]` yields only `d1` as possible answer, with probabilities 0.36/0/0/0.64: only when both sections are accessible, the document implies the query, otherwise the result is unknown. Inconsistency gets involved when we ask e.g. `?- D[video & retrieval]`. Here `s3` yields false and `s2` yields unknown; `d1` returns an inconsistent value when both `s2` and `s3` are accessible; thus, the probabilities for `d1` are 0/0.24/0.36/0.40.

Like p4D, POOL is based on an implicit open world assumption. However, it is possible to state closed world assumptions. In principle, one could choose arbitrary units for applying a cwa, e.g. objects (i.e. assuming that we know everything about a specific object), specific combinations of objects and types of propositions (e.g. we know all authors of this document) or specific attributes/classifications (e.g. we know all document authors or all books). Among these possibilities, the last choice seems to be most reasonable, thus we support it in POOL. Attributes and classifications can be closed by stating that the cwa should be applied to them, e.g.
```
#cwa(author). #cwa(article).
```
For incorporating these declarations in the retrieval strategy, we map them directly onto appropriate statements and rules in pD, since the cwa mechanism of p4D (which refers to predicates) is not appropriate in our case. Thus, we generate facts stating that the cwa holds for specific attributes and classifications, e.g.
```
cwa(author).  cwa(article).
```
Now we only need two rules for applying the cwa:
```
n_class(Cl,I,C) ← cwa(Cl), ¬ p_class(Cl,I,C).
n_attr(A,O,V,C) ← cwa(A), ¬ p_attr(A,O,V,C).
```

5 The basic IR engine

5.1 Description

In the description of pD given above, we have assumed that the indexing facts (like e.g. `0.7 docterm(d1,ir)`) are stored explicitly, e.g. in a relational database or an IR system. In principle, it would be fairly easy to implement such an interface to an IR system. However, not all basic queries to an IR system can be answered this way. In fact, the retrieval functionality of current IR systems is not appropriate for interfacing to a logical retrieval engine: there are several weaknesses of current IR systems which make them inappropriate for being integrated in a logic-based IR engine:

Physical data dependence: The retrieval functions offered are not independent from the availability of access paths: Most systems only allow for queries which can be answered by accessing the inverted file; when there is a possibility for scanning document texts directly, then other operators have to be used in the query. In order to achieve physical data independence (like in any database management system), query formulations should be independent from the presence or absence of indexes — the physical structure only affects efficiency, which should be kept separately from the logical query formulation.

Inappropriate query operators: When searching for phrases instead of single words only, then most IR systems provide special operators (e.g. for proximity search) in order to process this request based on the information available in its inverted lists. From a logical point of view, we just want to search for a phrase without caring about implementation details; the system itself should use the proximity information or syntactic structure in order to assign an indexing weight.

Propositional logic only: Most IR systems are based on the assumption that index terms are independent (or disjoint) propositions. Thus, they have problems even with real-world text retrieval needs: Users want to search in the text in different ways, e.g. by phonetic similarity (also for proper names) or for similar compound words — e.g. the German word *Donaudampfschiffahrtsgesellschaft* (Danube steamship company) should also be matched by *Donauschiffahrt* (Danube shipping). The concept of similarity plays an even more important role in multimedia retrieval e.g. most image retrieval methods are similarity-based.

In order to solve these problems, we apply the concept of attributes with vague predicates as introduced in [Fuhr 90] (and extended to text retrieval in [Fuhr 92]). A vague predicate is similar to a builtin predicate like in most database query languages, but instead of a Boolean value only, it returns a probability when it compares two values (e.g. "Jones" $\approx$ "Johnson"). For text retrieval, we assume that one argument of the vague predicate is a search term (e.g. a phrase) and the other one is the text of a document node which we check for the occurrence of this term. Analogous methods are used e.g. in image retrieval, where search for images with similar colors, texture or contours is based on similarity measures comparing the query image with any image in the database (see e.g. [Flickner et al. 95]).

These ideas lead us to the development of a new basic IR engine (BIRE) which is designed for supporting logic-based IR. Whereas the logical components of DOLORES are rather general, BIRE is restricted to hypermedia retrieval according to the FMM. Thus, it manages document nodes and offers vague predicates for searching the attribute values (both content and other attributes) of these nodes. However, although all nodes of a hypermedia document are stored in BIRE, the functionality of BIRE is restricted to single nodes. Functions operating on whole documents are implemented at the higher levels of DOLORES by means of logical rules. The BIRE interface to the logical level consists of a set of (binary) predicates, each applying a specific vague predicate to a specific attribute of document nodes (e.g. stem search,

phrase search and full word search on node texts, equality and phonetic similarity on author names).

Similar to IR systems like ECLAIR [Harper & Walker 92] or FIRE [Sonnenberger & Frei 95], BIRE is based on an object-oriented design (figure 2 shows the class diagram in UML [Fowler & Scott 97] notation); however, only BIRE implements physical data independence[3]. A `database` contains a set of document nodes (`doc-node`), where each node has a unique node number `nodeno`. This node number is used as external reference in the logic-based parts of DOLORES. There are different classes of nodes (which we have omitted here), e.g. leaf nodes for different media and non-leaf-nodes, as well as nodes with different sets of attributes. A `doc-node` is an aggregation of node attributes (`node-attr`). A node attribute has a name (`attr-name`) and a value `V` which is derived from the document node. Each node attribute corresponds to a database-wide `attribute` which manages all corresponding values of all nodes. An `attribute` is mainly an aggregation of vague predicates with two arguments, namely a node number `N` and an attribute value `V`, giving the probability that the predicate holds for value `V` in the document node with number `N`. These predicates form the interface to the logical levels of DOLORES. Depending on the data type of the attribute values, there are different classes of attributes with different sets of predicates (thus implying an inheritance hierarchy on data types, as described in [Fuhr 96]). For example, for the data type *english_text*, there are at least predicates for full word search, stem search and phrase search. For the data type *person_name* (e.g. for the attributes author or editor), there would be a predicate supporting phonetic search. In each case, there is also an equality predicate for directly accessing an attribute value.

Logically speaking, each argument of a predicate may be either free (**f**) or bound (**b**) — thus, there are in principle four different methods which have to be invoked on the procedural level. However, we do not allow for both arguments to be free (e.g. give me all words in all documents), and not all predicates allow for the attribute value to be free (e.g. give all phrases occurring in a document, or all names which are phonetically similar to the author names of a document). Thus, each `predicate` provides at least the methods `fb(V)` and `bb(N,V)`. The former corresponds to the standard retrieval method in classical IR system in that it searches for all node numbers where the predicate holds for the specified attribute value `V`. The latter returns the probability that the predicate holds for value `V` in the node with number `N`; this is implemented by means of the method `pt(N,V)` of the subobject `pred-test`. In addition, some predicates (class `v-predicate`) provide the method `bf(N)` giving all values for which the predicate holds in node `N`. For example, for equality on author names, there would be such a `v-predicate`, but not for phonetic similarity (the system can only decide whether or not a given name is phonetically similar to one of the authors', by means of `fb(V)` or `bb(N,V)`).

In order to describe how physical data independence is achieved in BIRE, we give an overview on the retrieval and indexing process for the standard retrieval method.

The retrieval task is implemented by the method `fb(V)` of class `predicate`, which in turn invokes `ps(V)` in class `pred-search`. Here we separate the logical and the physical level of our IR system: Depending on the availability and the usage of access structures, there are different subclasses of `pred-search`:

`ps-direct` uses an `index-structure` for direct search of the corresponding document numbers and probabilities. The latter class again has different subclasses for different types of index structures like inverted lists (e.g. for term search), B-trees (e.g. for numerical values) or spatial access structures for multidimensional values (e.g. for image retrieval).

`ps-indirect` uses in addition to an index structure also a `support-structure` for performing a search. For example, when there is an index on full word forms already, a search for word stems can be implemented by means of that index and a `support-structure` which returns all possible word forms for a given stem. These are used for invoking a search on `index-structure`.

`ps-filter` also makes use of an existing `index-structure` for a different predicate for the actual attribute. Here an index is used only as filter for possible answers, which are subsequently scanned. For example, a full word search with an `index-structure` for word stems could be implemented this way. Another example is signature-based retrieval.

`ps-noinx` uses no access structure at all. Instead, the document nodes are scanned directly.

Like for retrieval, also the indexing task for predicate `pred-search` is implemented such that it separates the physical level from the logical level. When a document node is inserted into a database the `index` methods for any of its `attributes` are called to pass node number and value to the corresponding database-wide `attribute` object. This object invokes the method `index(N,V)` for each of its `predicates`. For supporting the method `fb(V)` the corresponding method `psinx(N,V)` of class `pred-search` is called. As within `ps` for the retrieval tasks, the implementation of `psinx(N,V)` depends on the kind of index support:

`ps-direct` calls for each (document number, value)-pair the method `insert(N,V)` of `index-structure` for storing it. In `index-structure` the probabilities for tuples inserted are derived. For efficiency these probabilities are not computed directly after inserting an attribute value of a single document node. Instead, method `contextupdate` has to be called, at the latest when there is a `search` request for the actual index or when the index structure is closed.

`ps-indirect` relies on the fact that there is an `index-structure` for a different predicate and thus it has to update the `support-structure` only by inserting the appropriate pairs of values, e.g. (stem, full word form) for supporting stem search based on an index for full word forms.

`ps-filter` in most cases maintains no additional data structure, since it only uses the `index-structure` of a different predicate. (Signature-based retrieval can be implemented as a subclass which also maintains the signatures.)

`ps-noinx` maintains no additional data structure.

Retrieval and indexing for the other tasks are either implemented by use of the according `pred-search` object (`pred-test` just makes use of the `ps(V)` method; indexing is not required since `ps(V)` depends on the indexing for `pred-search`) or is done analogously to the implementation of `pred-search`: `pred-values` defines sub classes similar to `pred-search`. Just indexing has to be done inversely.

[3] In fact, data abstraction is one of the basic principles of object-oriented design, so it also should be realized in the systems developed using this method.

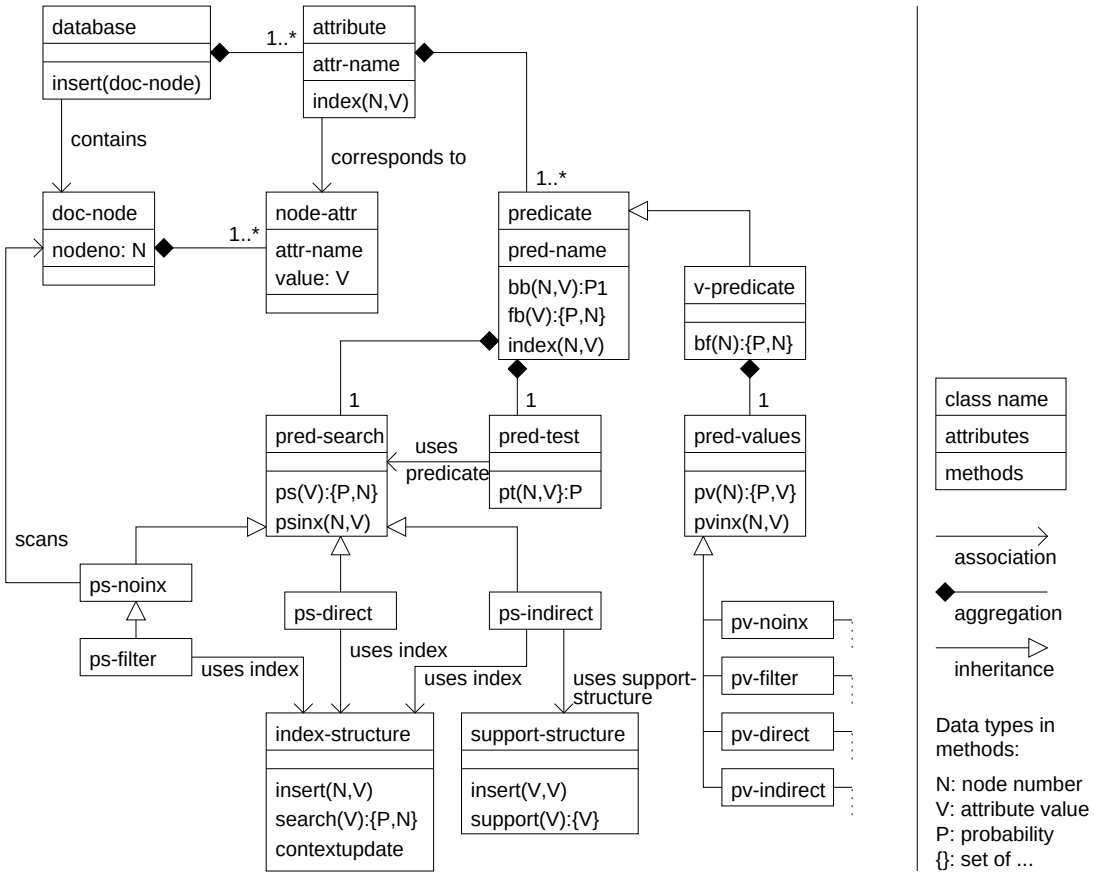

Figure 2: Object-oriented design of BIRE

5.2 Probabilistic Datalog and the BIRE

As pointed out above, the BIRE offers binary predicates as interface to the logical level of DOLORES. Furthermore, the current version of BIRE only manages positive information.

In general, the BIRE predicates are a combination of an attribute value and a vague predicate. For example, phonetic search on author names by means of a predicate `author-phonsim(N,A)` combines the attribute `author(N,X)` with a vague predicate `phonsim(X,A)` for phonetic similarity. We can also view the different text search predicates this way, where the attribute text yields the complete text of a document node and then there are different vague predicates for testing on the occurrence of full word forms, stems, phrases or compound words. Asking for the value of an attribute only (e.g. author names) is a special case involving equality as a "vague" predicate. Thus, the predicates offered by BIRE implement a join operation between the relation given the attribute value and an (intensional) relation representing the vague predicate. For example, consider a POOL query like `?- D.author.phonsim(jones)` which is equivalent to `?- D.author(X) & X.phonsim(jones)`. Mapping onto p4D generates the query `?- attribute(author,D,X,db) & attribute(phonsim,X,jones,db)`, and the equivalent in pD is

```
?- p_attribute(author,D,X,db) &
   p_attribute(phonsim,X,jones,db) &
   ¬ n_attribute(author,D,X,db) &
   ¬ n_attribute(phonsim,X,jones,db).
```

For efficiently processing the last query, the database interface of HySpirit must be able to recognize join operations that can be performed by the database system. For this purpose, term rewriting rules are used. Thus, we have

`p_attribute(author,D,X,db)` ⋈
`p_attribute(phonsim,X,A,db)` ⟶
`author-phonsim(D,A).`

Given the relational algebra equivalent of the last query, this rule can be applied.

Since BIRE treats content and node attributes the same way, but the mapping from POOL to pD makes a distinction between these concepts, we need additional rules for matching the BIRE interface. For text content, the default is that POOL terms are mapped onto word stems, namely by a rule

`term(T,D)` ← `text-stem(D,T).`

Other text search methods are represented as classifications in POOL, so we have to formulate rules like e.g.

`class(phrase,P,D)` ← `text-stem(D,P).`

6 Implementation and application

We have realized the complete DOLORES system. The graphical user interface is implemented as Java applet running in a WWW browser. The transformation of POOL programs into p4D and subsequently into pD is implemented in Perl. The HySpirit inference engine for pD (see [Rölleke & Fuhr 97]) is implemented in the object-oriented language Beta. HySpirit's evaluation algorithm is based on the magic sets strategy for modular

stratified programs ([Ross 94]). For accessing external data, there are interfaces to different relational database management systems and to BIRE. So far, the BIRE (implemented in Perl) only supports text retrieval and a limited number of vague predicates for attributes. Currently, we are working on the integration of similarity-based image retrieval methods.

Figure 3: Example images

Now we describe some applications of the DOLORES system. As an example for demonstrating the expressiveness of POOL, we used the system for semantic-based image retrieval. For this purpose, we have a collection of 650 images from the city of Paris at the beginning of this century. This collection was indexed manually by describing symbolic, the structural and the spatial view of each image. Therefore, an image is composed of objects with classifications and attributes. For example, the leftmost picture in figure 3 is described in POOL as follows:

```
p1[/* p1 structural */
   o7[/* o7 structural */
      o1[/* o1 structural */
         o2[] o3[] o4[] o5[] o6[]
         /* o1 symbolic */
         woman(o2) man(o3) cherub(o4)
         swan(o5) socle(o6)
         o2.represents(river) o2.represents(seine)
         o2.qualifier(naked) o2.position(sitting)
         o3.represents(river) o3.represents(marne)
         o3.qualifier(naked) o3.position(sitting)
         /* o1 spatial */
         o2.right_of(o3)
         o2.above_2D(o6) o3.above_2D(o6)
         o4.above_2D(o6) o5.above_2D(o6)
         o2.above_3D(o6) o3.above_3D(o6)
         o4.above_3D(o6) o5.above_3D(o6)]
      /* o7 symbolic */
      sculpture(o1)
      o1.material(stone) o1.represents(allegory)]
   /* p1 symbolic */
   parc(o7)]
/* database symbolic */
image(p1)
```

Asking for images where a woman is right from a man can be formulated in POOL as

```
?- image(I) &
   I[man(X) & woman(Y) & Y.right_of(X)]
```

As answer, we get the three images shown in figure 3. Since we search for images only, `p1` is retrieved (via augmentation). Had we omitted the restriction to images, we would have got `o1` in the highest rank, `o7` in a lower rank and `p1` even further behind.

Further applications of POOL (described in more detail elsewhere) are the following:

Large databases: In order to demonstrate the feasibility of our approach even for large databases, we applied our system to a classical text retrieval task. For this purpose, we used a part of the TREC collection, namely 12 months of the AP newswire data comprising 259 MB of text (about 85,000 documents). The index structure for stem search consumed additional 70 MB (in comparison to 2 GB when stored in a relational DBMS, see also [Fuhr & Rölleke 98])). As queries, the first 150 TREC queries were taken, but only terms occurring in less than 1000 documents were considered, thus leaving an average number of 16 query terms. It turned out that the response time of our system is proportional to the number of documents retrieved, whereas the number of query terms has only a minor effect. On average, the system outputs 30 documents per second (on a 170MHz Sun Ultrasparc with 64 MB main memory), where the performance bottleneck is the HySpirit engine.

Hypertext retrieval: In [Rölleke & Blömer 97], we describe the application of several retrieval strategies (formulated as logical rules) for the CACM collection, including strategies for considering hypertext links. Like other researchers (not using logic-based IR methods), we were able to improve retrieval effectiveness when using information about links between documents.

Image retrieval: Whereas the Paris collection was indexed manually, the IRIS system ([Hermes et al. 95]) performs automatic indexing of images. For the domain of landscape photos, IRIS detects basic concepts like e.g. water, sand, stone, forest, grass, sky and clouds. For each concept identified in an image, its location (as minimum bounding rectangle) and the certainty of identification are given. The system was applied to a database of 1200 images, of which 300 contain landscapes. After transforming the output of IRIS into POOL, we were able to ask queries for both content and spatial relationships (see [Fuhr & Rölleke 98]), e.g. searching for images with water (lake, river, sea) in front of stone (rocks):

```
?- D[water(A) & stone(B) & A.ylow(AY) &
     B.ylow(BY) & AY.less(BY)]
```

Here `xlow` gives the lower Y coordinate of the minimum bounding rectangle of an object, and `less` is a builtin method for comparing numerical values.

7 Conclusions and outlook

The DOLORES system presented in this paper combines several advanced concepts for the problem of hypermedia retrieval. Starting from a logic-based approach and combining it with the concept of data abstraction, we have designed a multi-layered system, thus achieving a

clear separation of the issues to be dealt with at different levels:

- POOL is an object-oriented logic for describing hypermedia objects. It supports aggregated objects, different kinds of propositions (terms, classifications and attributes) and even rules as being contained in objects. Based on a probabilistic four-valued logic, POOL uses an implicit open world assumption, allows for closed world assumptions and is able to deal with inconsistent knowledge.
- Probabilistic Datalog supports (probabilistic) predicates and rules as building blocks only. By using it as basic inference logic for implementing POOL retrieval, we have a non-procedural method for describing retrieval strategies, namely by specifying appropriate pD rules.
- BIRE is a basic IR engine that yields physical data abstraction. This feature makes it possible to support logic-based IR methods in a flexible way. On the other hand, we are able to integrate different access methods or algorithms within BIRE, without affecting the interface to the logical level.

In comparison to other IR systems, the representation and query language of DOLORES provides a much higher expressiveness, but for standard retrieval tasks, the efficiency is much lower. This situation is rather similar to the development of relational database management systems during the seventies when their performance was clearly inferior to that of hierarchical systems. Traditional IR systems are much like hierarchical database management systems in that they have fixed access paths, lacking physical data independence and an expressive query language. In fact, the technology used in DOLORES corresponds to deductive databases, thus yielding a tremendous progress in terms of expressiveness.

In order to increase the efficiency of DOLORES, we are working on the development of new query processing strategies which focus on the top-ranking elements of the answer (see e.g. [Pfeifer & Fuhr 95]) — in contrast to the magic sets evaluation strategy currently used in HySpirit which considers all objects yielding a nonzero probability of implying the query.

Our work is focused on the development of retrieval methods for hypermedia objects. In order to make full use of these methods, however, appropriate indexing methods have to be available. Most of today's indexing and retrieval methods are restricted to the syntactical level of multimedia objects (e.g. color, texture and contour for images), but the major part of user needs can be satisfied only by semantic-based methods. Since the application of manual indexing is hardly ever feasible, there is a clear need for further research on semantic-based indexing methods for multimedia data.

References

Buckley, C. (1985). *Implementation of the SMART Information Retrieval System.* Technical Report 85-686, Department of Computer Science, Cornell University, Ithaca, NY.

Callan, J.; Croft, W.; Harding, S. (1992). The INQUERY Retrieval System. In: *Proc. DEXA*, pages 78–83. Springer, Berlin et al.

Chiaramella, Y.; Mulhem, P.; Fourel, F. (1996). *A Model for Multimedia Information Retrieval.* Technical report, FERMI ESPRIT BRA 8134, University of Glasgow.

Flickner, M.; Sawhney, H.; Niblack, W.; Ashley, J.; Huang, Q.; Dom, N.; Gorkani, M.; Hafner, J.; Lee, D.; Petkovic, D.; Steele, D.; Yanker, P. (1995). Query by Image and Video Content: The QBIC System. *Computer 28(9)*, pages 23–32.

Fowler, M.; Scott, K. (1997). *UML Distilled. Applying the Standard Object Modeling Language.* Addison Wesley, Reading, Mass.

Fuhr, N.; Rölleke, T. (1998). HySpirit — a Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases. *Proc. EDBT* pages 24–38. Springer, Berlin et al.

Fuhr, N. (1990). A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In: *Proc. VLDB*, pages 696–707. Morgan Kaufman, Los Altos, Cal.

Fuhr, N. (1992). Integration of Probabilistic Fact and Text Retrieval. In: *Proc. SIGIR*, pages 211–222. ACM, New York.

Fuhr, N. (1996). Object-Oriented and Database Concepts for the Design of Networked Information Retrieval Systems. In: *Proc. CIKM*, pages 164–172. ACM, New York.

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

Hermes, T.; Klauck, C.; Kreyß, J.; Zhang, J. (1995). Image Retrieval for Information Systems. In: *SPIE Proc. Vol. 2420: Storage and Retrieval for Image and Video Databases III.* San Jose, CA, USA.

Lalmas, M. (1997). Dempster-Shafer's Theory of Evidence Applied to Structured Documents: Modelling Uncertainty. In: *Proc. SIGIR*, pages 110–118. ACM, New York.

Meghini, C.; Rabitti, F.; Thanos, C. (1991). Conceptual Modeling of Multimedia Documents. *IEEE Computer 24(10)*, pages 23–30.

Pfeifer, U.; Fuhr, N. (1995). Efficient Processing of Vague Queries using a Data Stream Approach. In: *Proc. SIGIR*, pages 189–198. ACM, New York.

van Rijsbergen, C. J. (1989). Towards an Information Logic. In: *Proc. SIGIR*, pages 77–86. ACM, New York.

Rölleke, T.; Blömer, M. (1997). Probabilistic Logical Information Retrieval for Content, Hypertext, and Database Querying. In: *Hypertext — Information Retrieval — Multimedia (HIM)*, pages 147–160. Universitätsverlag Konstanz.

Rölleke, T.; Fuhr, N. (1996). Retrieval of Complex Objects Using a Four-Valued Logic. In: *Proc. SIGIR*, pages 206–214. ACM, New York.

Rölleke, T.; Fuhr, N. (1997). Probabilistic Reasoning for Large Scale Databases. In: *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 118–132. Springer, Berlin et al.

Rölleke, T. (1998). *Probabilistic Logical Representation and Retrieval of Complex Objects.* University of Dortmund. Dissertation (in preparation).

Ross, K. (1994). Modular Stratification and Magic Sets for Datalog Programs with Negation. *Journal of the ACM 41(6)*, pages 1216–1266.

Sonnenberger, G.; Frei, H.-P. (1995). Design of a Reusable IR Framework. In: *Proc. SIGIR*, pages 49–57. ACM, New York.