Journal of Intelligent Information Systems 7, 75–100 (1996)

Computing Access Relevance for Path-Method Generation in OODB and IM-OODB

ASHISH MEHTA amehta@quasar.poly.edu
Center for Applied Large-Scale Computing (CALC), Polytechnic University, Brooklyn, NY 11201

JAMES GELLER geller@homer.njit.edu
YEHOSHUA PERL
Institute for Integrated Systems, CIS Department, New Jersey Institute of Technology, Newark, NJ 07102

PETER FANKHAUSER
GMD-IPSI, Dolivostr. 15, D-6100, Darmstadt, Germany

Abstract. A *path-method* (PM) is a mechanism to retrieve or to update information relevant to one class, in an object-oriented database (OODB), that is not stored with that class but with some other class. The PM traverses a chain of classes and connections that ends at the class where the required information is stored. However, it is a difficult task for a user to write PMs. This is because it might require comprehensive knowledge of many classes of the conceptual schema. But a typical user has often incomplete or even inconsistent knowledge of the schema. Currently we are developing a system, called *Path-Method Generator* (PMG), which generates PMs automatically according to a naive user's requests. One algorithm of PMG uses numerical *access relevance* between pairs of classes as a guide for the traversal of an OODB schema. In this paper we define the notion of access relevance to measure the significance of the (indirect) connection between any two classes in an OODB and present efficient algorithms to compute access relevance. The manual PM generation in an interoperable multi object-oriented database (IM-OODB) is even more difficult than for one OODB since a user has to be familiar with several OODBs. We use a hierarchical approach for developing efficient online algorithms for the computation of access relevances in an IM-OODB, based on precomputed access relevances for each autonomous OODB. In an IM-OODB the access relevances are used as a guide in generating PMs between the classes of different OODBs.

Keywords: object-oriented database, path-method, path-method generator, traversal algorithms, access relevance, triangular norms, weighting functions, interoperable databases, interoperable multi-OODB, inter-OODB connection

1. Introduction

The task of retrieving information relevant to a class, that is stored with another class in an OODB, requires considerable knowledge of the structure of the OODB. In a large OODB this is difficult for a typical user who has incomplete knowledge of the conceptual schema. The mechanism for retrieving such distant information is a path-method. A *path-method* (PM) is a method which traverses from one class through a chain of connections (user-defined and generic relationships) between classes to retrieve relevant information from a distant class. The writing of such PMs requires traversal of the OODB schema. Currently, we are developing a system, called *Path-Method Generator* (PMG), which consists of a collection of traversal algorithms to generate desired PMs automatically. One algorithm of

the PMG (Mehta et al., 1993; Mehta, 1993) uses as a guide for the traversal *precomputed access relevances* between all the pairs of classes in the OODB schema.

Following, e.g., VML (Klas et al., 1991), GemStone (Butterworth et al., 1991), and ORION (Kim, 1990; Kifer et al., 1992), we are modeling an OODB schema as a directed graph. Classes are represented as nodes. Directly related classes are connected by a directed edge. Note that the directed graph of a schema may contain cycles. Our approach is enhanced as we assign an *access weight* from the range [0, 1] to each edge in the schema graph. The access weight of a connection from a class a_s to a class a_t is a measure of its significance according to the frequency of traversing this connection relative to all connections emanating from the class a_s. The access weights are assigned according to the frequencies of use of the connections accumulated during the operation of the OODB. Initially, frequency information is not available and access weights are approximated by the schema designer.

The significance of a path is measured by the *access relevance value* ARV(P). The ARV(P) of a path P is obtained by applying a triangular-norm (t-norm) (Fankhauser et al., 1991; Kracker, 1992; Zadeh, 1965; Klir and Folger, 1988) to the set of access weights of the edges of the path. For example, for the t-norm PRODUCT, the access relevance value of a path is the product of the access weights of all its edges. There exist several infinite families of t-norms and corresponding conorms (Schweizler and Sklar, 1961). However, in Bonissone and Decker (1986) it is empirically shown that for most practical purposes two to five different t-norms suffice. From these we have chosen PRODUCT.

The access relevance between non-adjacent classes a_s and a_t is a measure of the significance of the indirect connection from a_s to a_t. If several paths exist between a_s and a_t then we use the co-norm MAXIMUM to compute a single value. Thus, the *access relevance* AR(a_s, a_t) from a class a_s to another class a_t is the maximum access relevance value over all paths from a_s to a_t. Note again the difference between the two terms *access relevance value* ARV(P) and *access relevance* AR(a_s, a_t).

To relate our work to previous research, we mention the following relevant publications. The concept of *dynamic message forwarding plan generation* for incompletely specified global views of integrated databases is discussed in Neuhold and Schrefl (1988). In Bertino et al. (1992), OODB methods and OODB views are contrasted. Schema independent query formulation, i.e., finding proper terms defined in the schema from the terms contained in a user-query has been discussed in Kracker and Neuhold (1989). The PMG can be considered as a *powerful underlying traversal tool* for schema independent query formulation (Kracker, 1992), for dynamic derivation of personalized views (Neuhold and Schrefl, 1988), and in general as a retrieval/update mechanism for OODBs.

Other OODBs have used constructs similar to PMs, under a number of different names. Kim (1989) introduces an object-oriented query model based on query graphs. In Kifer et al. (1992) he extends his approach to a comprehensive treatment of path expressions. OQL, a query language for object-oriented databases (Alashqur et al., 1989) calls PMs in the fully formal treatment *intensional association patterns (with distinctively named associations)*. Intensionality thereby stands for schema objects, while extensional association patterns involve instances.

In Alhajj and Arkun (1993), a variant of PMs is referred to as *message expressions*. Their contribution is an object algebra that overcomes many problems of previous OODB query

models, such as violation of the closure property. The closure property requires that the result of a query has to have the same structural properties as the original database. (Note, however, that OQL (Alashqur et al., 1989) conforms to the closure property.) Kemper and Moerkotte (1990) introduces an indexing scheme, called *access support relations* for path expressions. Their query language is QUEL-like.

Query optimization for OODBs is combined with optimization for deductive databases in a single model in Cheiney and Lanzelotte (1992). This model permits the optimization for joins interleaved with path traversals. Query optimization has been extended to recursive queries (Lanzelotte et al., 1992), a useful feature for object-oriented engineering databases that deal with part hierarchies.

Our own work agrees with the previously mentioned approaches on the basic building blocks (path expressions), but it differs from them considerably in the following way. Most of the described research, except to some degree Kim's, comes out of the tradition of specifying *what* the user wants to retrieve, but now *how* it should be done. In this paper our queries are underspecified. Our user is assumed to be naive, so that he cannot even give a complete description of *what* he wants to retrieve. Rather he specifies only the endpoints of his path expression. The PMG then returns a suggestion of *what* the user wants to retrieve, while query optimizers find the best way of *how* to retrieve fully specified information. The PMG user can accept that suggestion, or ask for an alternative.

For interoperable databases the PM mechanism for supporting schema independent query formulation is even more important, as it is unrealistic to maintain a completely integrated schema which equally serves all users' needs. Rather, only a loosely coupled form of interoperable multi-database (Sheth and Larson, 1990) can be achieved by specifying simple cross-database relationships. In such interconnected schemas it is particularly difficult for individual users to combine information from multiple resources. We will discuss efficient algorithms to compute access relevance in an IM-OODB, i.e., each autonomous database is an OODB. However, the different OODBs may use different object-oriented database models. Connections between different OODBs are established by extending Cheiney and Lanzelotte (1992); Czejdo and Taylor (1991).

The automatic generation of joins in relational databases is an analogous problem to PM generation. There are two major approaches to this problem, the universal schema interface (Maier and Ullman, 1983; Maier et al., 1987) and the implicit join (Litwin, 1985). There is a fundamental difference between PMs in our approach and joins in both of the above approaches. PMs are properties of a class and are generated by following a sequence of connections of the schema. Once the PM is generated, its execution requires just the fast traversal of the connections which appear in the definition of the PM. On the other hand, joins are used to combine information stored in different relations, which may be quite large, and require a large overhead for deriving query-results. E.g., in relational databases to find all the sections taken by a student, we need to join at least three relations, Student, Transcript, and Sections. Hence the join operations require processing of a large amount of data.

A dramatic decrease in the amount of data accessed is achieved by using semijoins (see, e.g. (Elmasri and Navathe, 1989)). By optimizing the order of performing the join operations, a meaningful saving is obtained. Nevertheless, even with semijoins we have to process all the entries of some relationships, although not for all relationships. In an OODB on the other hand, once we find a student instance we connections, from student to transcript

and then from transcript to sections. This traversal does not require any information about transcripts of other students, or sections taken by other students. Thus, execution of PMs significantly reduces the overhead of joins and even semijoins in relational databases. A fast object reference technique, such as in the BeSS System (Biliris and Panagos, 1995), can improve execution of PMs even further.

The remainder of this paper is organized as follows. In Section 2, we discuss the notions of access weights and access relevance. In Section 3, we present an efficient algorithm for access relevance computation. In Section 4 we briefly describe the Path-Method Generator (PMG). In Section 5 we introduce the computation of access relevance in an Interoperable Multi-OODB. In Sections 6 and 7 we discuss the computation of access relevance in an IM-OODB containing two and many OODBs, respectively. Section 8 contains conclusions.

2. Access weight and access relevance in an OODB

2.1. A general OODB model

The presentation in this paper uses an abstract OODB model rather than a specific OODB model. The reason for this is to present the computation of access relevance between classes in a way that can be implemented on a variety of OODB systems. This general representation emphasizes the possibility of computing access relevance in an Interoperable Multi-OODB containing OODBs of different models. Although this abstract OODB model reflects a variety of existing OODBs, we use some of the terminology of Neuhold et al. (1989); Neuhold et al. (1990); Neuhold et al.; Geller et al. (1991); Klas et al. (1991), which we now summarize.

A class description consists of four kinds of properties: attributes, user-defined and generic relationships, and methods. Attributes specify values of a given data type. Methods specify operations defined for instances of a class. Relationships refer to other classes. We use the common term "connection" for user-defined and generic relationships. For a user-defined relationship the system has no additional semantics. Generic relationships are system-supported connections between classes. We consider two specialization generic relationships: *categoryof* (*roleof*) for the case where the superclass and the subclass are in the same (different) contexts. A set class S is always connected to a member class M. Every instance of S is a set that has as members instances of M. Two generic relationships, *memberof* and *setof*, connect S and M.

Let us consider a subschema of a university database (figure 1) and the directed graph $G(V, E)$ corresponding to it (figure 2). Figure 1 uses OODINI, a graphical language and tool for schema representation (Halper et al., 1992). A rectangle represents a class. A double line rectangle represents a set class and shares one corner with its member class, e.g., the classes **section** and **sections**. Note that the schema contains a second set class for **section**, called **crsections** which represents a set of sections of the same course. A thick arrow represents specialization generic relationships such as *categoryof*, and a thin arrow a user-defined relationship.

We will now define the formalism that is specific to this paper.

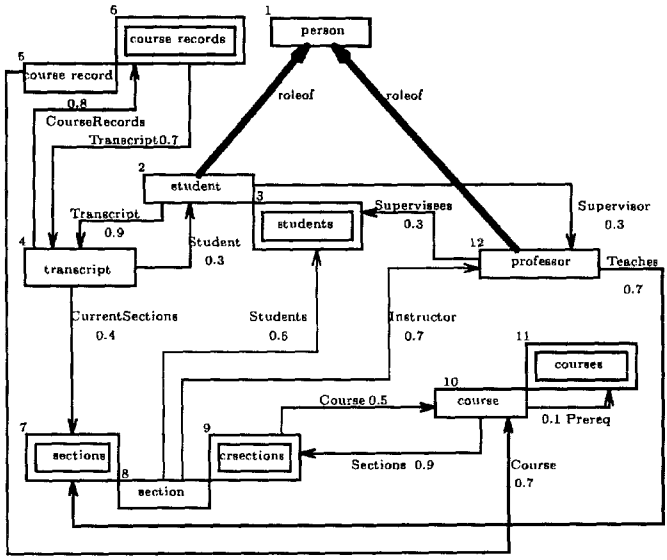


Figure 1. A subschema of a University Database.

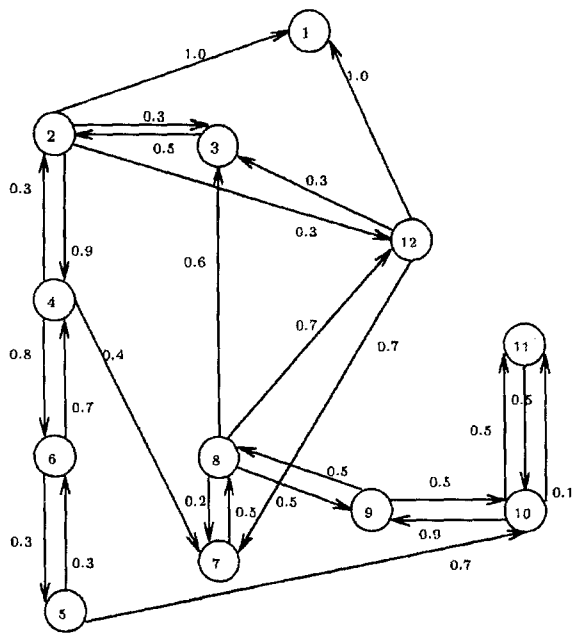


Figure 2. The subschema as a directed graph.

Definition 1. A *path-method* (PM) is a side effect-free method that consists of a chain of connections between classes, possibly terminated by an attribute. Execution of a PM with an instance as argument results in the retrieval of some object (or data) at the end of the chain which is a function of the argument. (If several objects or data are retrieved, then the set of objects is a function of the argument.)

E.g., in figure 1, if we want to know all sections that the advisor of a student X is teaching, we use a PM consisting of the connection Supervisor from **student** to **professor**, followed by the connection Teaches to **sections**.

Definition 2. The *access weight* of a connection between two neighboring classes is a number between 0 and 1 that characterizes the strength of this connection.

Intuitively, the access weights reflect how often connections have been used in the past, i.e., the strength is measured by frequency. We will use this strength to choose to which connection the PMG should advance. The access weights have to follow certain rules that will be introduced later on. A main problem of this paper is to find a technique for computing a number which measures the strength of a connection between two classes that are not directly connected, i.e., a generalization of access weights for pairs of distant classes.

Definition 3. The *access relevance value* $\mathrm{ARV}(P)$ of a path P is a number that characterizes the strength of P. It is computed from the access weights of all the connections on P.

Definition 4. The *access relevance* $\mathrm{AR}(a_s, a_t)$ from a node a_s to a node a_t is a number that characterizes the strength of the strongest path between those two nodes, taking into account all the paths from a_s to a_t.

Note that $\mathrm{AR}(a_s, a_t)$ is directional, because connections are directional. In other words, usually $\mathrm{AR}(a_s, a_t) \neq \mathrm{AR}(a_t, a_s)$. The *precomputed access relevances for a schema* consist of a table of the access relevances for every pair of nodes (a_s, a_t) such that there is at least one path from a_s to a_t.

Definition 5. The *Path-Method Generator* (PMG) is a system of algorithms that takes as arguments two nodes a_s and a_t, and that returns a path from a_s to a_t. In selecting this path it makes use of the precomputed access relevances for the schema. The resultant path is the "strongest" path that exists from a_s to a_t in this schema.

2.2. *Motivation*

We now describe briefly why the PMG needs to use access relevances. While traversing the schema to generate a PM, we start with the source class a_s and consider its different outgoing connections. Our observation has been that some connections in a database schema are more significant than others. Traversing the connections of a schema, giving priority to more significant connections, will in many cases produce PMs more correctly and efficiently than a uniform traversal. The problem is how to identify the "more significant" connections.

One may try to use the semantics of the different connections in the schema. However, one needs to specify a combination rule for the semantics of the connections along a path in order to derive the significance of a connection of two classes which are not directly related. Recent work (Fankhauser et al., 1991; Fankhauser and Neuhold, 1992) addresses this problem based on semantic resemblance between classes. However, the ideas used in Fankhauser et al. (1991); Fankhauser and Neuhold (1992) are not applicable to measuring the semantics of the connection between two indirectly related classes. The problem is that while semantic resemblance (Fankhauser et al., 1991; Fankhauser and Neuhold, 1992) can supply a semantic interpretation to the different adjacent edges, there is no known way to generalize this notion to paths. Thus, we have taken the approach introduced before.

The access weights associated with the connections emanating from a class are chosen to reflect the relative frequencies of traversal of each of the connections of the class during the operation of the OODB. These numbers should be accumulated during the operation of the OODB. In the beginning, when access frequency information is not available, a domain expert can suggest initial values for access weights. Further research will determine whether application (view) oriented frequencies will be needed to achieve acceptable success ratios for automatic path generation.

The simplest greedy traversal algorithm would be to choose at each node the outgoing connection of highest frequency. However, our experiments (Mehta et al., 1993) show that this algorithm, like many greedy algorithms (Horowitz and Sahni, 1989), lacks the look-ahead property necessary in many cases to create a desired PM. Thus, we use a measure that incorporates the access weights of all connections that make up a path. Our PMG algorithm decides on the connection to be traversed from a_s, based on the access weight of the connection to a neighboring class u and the access relevance from u to the target class a_t. These choices will be made for each node in the path traversal. This mechanism adds to the greedy traversal the necessary look-ahead property which dramatically improves the results of our PMG algorithm (Mehta et al., 1993). Our experiments (Mehta et al., 1993) show that traversal of a schema according to the following rules generates the desired PMs more successfully and more efficiently than a uniform traversal, e.g., depth first search, or breadth first search.

2.3. *Access weight assignment and computation of access relevance*

Each connection in the OODB is assigned an access weight W, where $0 \leq W \leq 1$. For the relationships *setof* and *memberof* and for user-defined relationships the access weight satisfies Rule 1.

Rule 1. The sum of the weights on the n outgoing connections of a class must conform to the following constraint: $\sum_{i=1}^{n} W_i = 0.5 * n$. From this sum, each connection is assigned a weight from the interval [0, 1], reflecting its relative frequency of traversal.

E.g., in figure 1, the class **transcript** has three relationships, CourseRecords to **course_records**, CurrentSections to **sections** and Student to **student**, with access weights 0.8, 0.4, and 0.3, respectively. Observe that $0.8 + 0.4 + 0.3 = 1.5 = 0.5 * 3$, as required by Rule 1.

The justification for Rule 1 is as follows. It is not sufficient to assign access weights which add up to 1 according to traversal probabilities of a connection. This would imply that the connections of a class with few connections are more significant than the connections of a class with many connections, which is not true. Thus, Rule 1 makes the values of access weights independent of the number of connections of a class. (For access weights missing in figure 1 refer to figure 2.)

While we hasten to stress that access weights are not probabilities (they do not sum to 1!) and not grades of fuzzy membership, we are still using t-norms to combine them. To stress the fact that we are talking about access weights and not probabilities, we will use the term "weighting function" (WF) instead of the term t-norm. We will use the weighting function PRODUCT. The weight of a path between two classes is the product of the access weights of all the edges in the path.

For an example, let us consider the paths from **professor** (12) to **course** (10). (1) The path p_1 represents the class sequence (**professor** (12), **sections** (7), **section** (8), **crsections** (9), **course** (10)). This class sequence will retrieve all the courses being taught by a professor. The ARV(p_1) is 0.0872. (2) The path p_2 represents the class sequence (**professor** (12), **students** (3), **student** (2), **transcript** (4), **sections** (7), **section** (8), **crsections** (9), **course** (10)). This class sequence will retrieve all the courses currently being taken by all the students supervised by a professor. The ARV(p_2) is 0.0068.

Following again techniques commonly used with fuzzy sets, we will use a co-norm to select one of the paths connecting the pair of nodes a_s and a_t. Here, we will use the MAXIMUM co-norm. In other words, we are applying the MAXIMUM function to the access relevance values of all paths from a_s to a_t to compute the access relevance from a_s to a_t.

Definition 6. The most relevant path from a_s to a_t is the path P with the maximum ARV(P) compared to all the other paths from a_s to a_t.

Lemma 1. *The ARV(P) of the most relevant path P from a_s to a_t is identical to the AR(a_s, a_t).*

Proof: As we are using the MAXIMUM co-norm to choose between different paths, the path P that will be selected to define the AR(a_s, a_t) value is the one with the maximum ARV(P). □

In the previous example p_1 is the most relevant path. Later on we will need the following property of a most relevant path.

Property 1: *For every pair of nodes there exists a simple (i.e., no cycles) most relevant path.*

Property 1 is implied by the fact that for our WF the deletion of a cycle from a path P can only increase the access relevance value of P.

The assignment of access weights to the relationships *roleof* and *categoryof* needs to reflect inheritance, i.e., each property of a superclass is available at the subclass at no extra cost. We consider two possible rules.

Rule 2a. Assign an access weight of 1.0 to each *roleof* and *categoryof* connection (see figure 2).

Such a value implies that the properties of the superclass are available at the subclass without decreasing the access relevance. However, Rule 2a has the following disadvantage. It enables the traversal of a specialization connection as a regular connection rather than an inheritance link. That is, it enables traversal which stops at the superclass as a target rather than continuing to use one of its properties. But there is no reason for such traversal since it does not lead to any meaningful information not available at the subclass. We would like to block traversals through specialization connections while still enabling inheritance. But an access weight of 1.0 enables such a traversal and gives it high priority.

Rule 2b. Assign an access weight of 0.0 to *categoryof* and *roleof* connections. Copy all the properties of the superclass to the subclass in the schema's underlying graph to allow for the effect of inheritance.

Rule 2b allows computation of access relevance exactly as discussed before, but it practically disallows unwanted traversals of specialization connections. One disadvantage of Rule 2b is that the schema graph becomes more dense. Thus, it increases the running time of the algorithms which compute access relevances. The schema visible to the user as well as the algorithms for computing the access relevance are not changed. Our experiments with the PMG algorithm (Mehta et al., 1993) show slightly better results for Rule 2b than Rule 2a for generating PMs. Thus, the choice between Rule 2a and Rule 2b is a tradeoff between accuracy and complexity where the difference is small in both dimensions. For figure 1 the graph is unchanged.

To summarize formally, we state that if a single path $P(a_s, a_t) = a_s(=a_{i_1}), a_{i_2}, a_{i_3}, \ldots, a_{i_k}$ $(=a_t)$ connects two nodes a_s, a_t, then the access relevance value of P is

$$\mathrm{ARV}(P) = \prod_{(1 \le r < k)} \left[W\left(a_{i_r}, a_{i_{r+1}}\right) \right]$$

If there are m paths $P_1, \ldots, P_m$ from a_s to a_t, then the access relevance of this pair of nodes is

$$\mathrm{AR}(a_s, a_t) = \max_{j=1}^{m} \mathrm{ARV}(P_j) = \max_{j=1}^{m} \prod_{((a_{i_r}, a_{i_{r+1}}) \in P_j)} \left[W\left(a_{i_r}, a_{i_{r+1}}\right) \right]$$

We define $\mathrm{AR}(a_s, a_s) = 1$. Maximizing the PRODUCT weighting function finds a path with the highest product of access weights of all its edges. We note that sometimes the user may be interested in the access relevance between a class a_s and an attribute atr_{a_t} of another class a_t. To handle this case we represent the connection between a class and one of its attributes by an edge with a given access weight.

Definition 7. We define $\mathrm{AR}(a_s, \mathrm{atr}_{a_t})$ for a pair of a class a_s and an attribute atr of another class a_t as follows:

$$\mathrm{AR}(a_s, \mathrm{atr}_{a_t}) = \mathrm{WF}[\mathrm{AR}(a_s, a_t), W(a_t, \mathrm{atr}_{a_t})]$$

Note that if $W(a_t, \mathrm{atr}_{a_t}) = 1$ then for WF = PRODUCT $\mathrm{AR}(a_s, \mathrm{atr}_{a_t}) = \mathrm{AR}(a_s, a_t)$.

3. Algorithms for computing access relevance in an OODB

We will now describe an algorithm PRODUCT_AR for the PRODUCT weighting function which computes access relevance from a source class to all the classes in the schema. This algorithm is a variation of the well-known nearest neighbor greedy algorithm of Dijkstra (e.g. (Aho et al., 1983)) which solves the single source shortest path problem. To find the access relevances for all pairs of classes in a schema of n classes we need to apply PRODUCT_AR n times, once for each class as a source class. PRODUCT_AR finds the access relevance AR(s, v) from a source class represented by node s to every other class v and stores it in an array ARS. In other words, the notation ARS[v] stands for the stored value of AR(s, v).

The algorithm assumes without loss of generality that all nodes are labeled with integers: $V = \{1, 2, \ldots, n\}$. It maintains a set S of nodes whose maximum access relevance from the source is already computed. Initially, S contains only the source node $\{s\}$. At each step, we add to S a node $u \in V - S$ of maximum access relevance. (If there are several, we choose one at random.) A path from s to a node v is called *special* if all its nodes (except possibly v itself) belong to S.

At each step of the algorithm, we use the array ARS to record the maximum access relevance value of a special path to each node. In each step, after u is chosen to be inserted into S, a new special path P to v, $v \in V - S - \{u\}$ containing u may result, that has an ARV(P) > ARS[v]. Hence, we update ARS[v] for each node $v \in V - S$ as follows. We are using a two-dimensional array W, where $W[i, j]$ is the access weight of the edge (i, j). If there is no edge (i, j), then $W[i, j] = 0$. ARS[v] is the maximum of two values: (1) The old ARS[v] and (2) ARS[u] $*$ $W[u, v]$ representing the access relevance of a special path containing u as the last node before v. Once S includes all nodes, all paths are "special", so ARS[v] will hold the maximum access relevance from the source to each node $v \in V$.

```
    Procedure PRODUCT_AR (IN s: node, OUT ARS: array[1..n] of REAL)
    begin
(1)     S := {s};
(2)     for each node v other than s do
(3)         ARS[v] := W[s, v];
(4)     for i := 1 to n − 1 do begin
(5)         choose a node u in V − S such that
                ARS[u] is a maximum;
(6)         add u to S;
(7)         for each node v in V − S do
(8)             ARS[v] := max (ARS[v], ARS[u] * W[u, v])
        end
    end;
```

Let us apply PRODUCT_AR to the graph of figure 2 assuming Rule 2b, although figure 2 was constructed according to Rule 2a. The source is node 12 (**professor**). In steps (2)–(3), $S = \{12\}$, ARS[3] = 0.3, ARS[1] = 0.0, ARS[7] = 0.7, and for the rest of the entries i of the array, ARS[i] = 0. Note that ARS[1] = 0.0 using Rule 2b, though it is shown as 1.0

in figure 2. In the first iteration of the **for**–loop of lines (4)–(8), $u = 7$ is selected as the node with the maximum ARS[u]. Then we set ARS[8] = max(0, 0.7 * 0.5) = 0.35. Other values of the array ARS do not change. See the sequence of the ARS values in Table 1.

If we compute the array ARS for every starting node s, then the result will be a two-dimensional matrix AR[s, v]. The value stored in every array element AR[s, v] is then identical to the result of computing AR(s, v). The results of PRODUCT_AR (Table 1) appear in row 5 in the matrix AR in Table 2, showing the access relevance for each pair $(5, v)$, $v \in V$, of nodes. We will now discuss the validity of the PRODUCT_AR algorithm.

Table 1. Computation of PRODUCT_AR on graph of figure 2.

Iteration	S	u	New value of ARS
Initial	{12}	–	ARS[3] = 0.3, ARS[1] = 0.0, ARS[7] = 0.7
1	{12, 7}	7	ARS[8] = 0.35
2	{12, 7, 8}	8	ARS[9] = 0.175
3	{12, 7, 8, 3}	3	ARS[2] = 0.150
4	{12, 7, 8, 3, 9}	9	ARS[10] = 0.088
5	{12, 7, 8, 3, 9, 2}	2	ARS[4] = 0.135
6	{12, 7, 8, 3, 9, 2, 4}	4	ARS[6] = 0.108
7	{12, 7, 8, 3, 9, 2, 4, 6}	6	ARS[5] = 0.032
8	{12, 7, 8, 3, 9, 2, 4, 6, 10}	10	ARS[11] = 0.044
9	{12, 7, 8, 3, 9, 2, 4, 6, 10, 11}	11	–
10	{12, 7, 8, 3, 9, 2, 4, 6, 10, 11, 5}	5	–
11	{12, 7, 8, 3, 9, 2, 4, 6, 10, 11, 5, 1}	1	–

Table 2. Access relevance matrix AR for graph of figure 2.

	1	2	3	4	5	6	7	8	9	10	11	12
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.3	0.9	0.216	0.72	0.36	0.18	0.136	0.151	0.076	0.3
3	0.0	0.5	1.0	0.45	0.108	0.36	0.18	0.09	0.068	0.076	0.038	0.15
4	0.0	0.3	0.12	1.0	0.24	0.8	0.4	0.2	0.151	0.168	0.084	0.14
5	0.0	0.095	0.189	0.21	1.0	0.3	0.154	0.315	0.63	0.7	0.35	0.221
6	0.0	0.21	0.084	0.7	0.3	1.0	0.28	0.14	0.189	0.21	0.105	0.098
7	0.0	0.15	0.3	0.135	0.032	0.108	1.0	0.5	0.25	0.125	0.063	0.35
8	0.0	0.3	0.6	0.27	0.065	0.216	0.49	1.0	0.5	0.25	0.125	0.7
9	0.0	0.15	0.3	0.135	0.032	0.108	0.245	0.5	1.0	0.5	0.25	0.35
10	0.0	0.135	0.27	0.122	0.029	0.097	0.221	0.45	0.9	1.0	0.5	0.315
11	0.0	0.068	0.135	0.061	0.015	0.049	0.11	0.225	0.45	0.5	1.0	0.158
12	0.0	0.15	0.3	0.135	0.032	0.108	0.7	0.35	0.175	0.088	0.044	1.0

Theorem 1. *In the PRODUCT_AR algorithm, ARS[v] contains at all times the highest access relevance of a special path from node s to node v, for every node* $v \in V$.

Proof: Due to space limitations we need to omit the proof, which can be found in Mehta (1993). □

The running time of the PRODUCT_AR algorithm is $O(n^2)$. Let e be the number of edges in the graph. If $e \ll n^2$ then we might do better by using an adjacency list representation of the directed graph and using a priority queue implemented as a heap (Aho et al., 1983) to organize the nodes in $V - S$. Choosing and deleting a maximum access relevance node from S in lines (5) and (6) takes $O(\log n)$ time. This operation is repeated n times yielding $O(n \log n)$ time. The loop of lines (7) and (8) can then be implemented by going down the adjacency list for u and updating the access relevance in the priority queue. At most a total of e updates will be made, each at a cost of $O(\log n)$, so the total time is now $O(e \log n)$, rather than $O(n^2)$. Thus, the running time of PRODUCT_AR algorithm is $O(e \log n)$. This running time is considerably better than $O(n^2)$ if $e \ll n^2$, as it is for a typical OODB schema whose graph representation is a sparse graph.

4. The PMG algorithm to generate path-methods

4.1. User interaction with the PMG

Before describing the PMG we shall clarify our assumptions about the interaction between the user and the PMG. We assume that the user supplies to the PMG a pair of (source, target). Although the user does not know the exact traversal path from the source to the target, s/he perceives the proper interpretation or "semantics" of the PM s/he desires and can give a verbal description of it. As we will see later, such a verbal description is an inappropriate input to PMG, because it is not usually clear how to utilize the information contained in it. For the same pair (source, target) there may exist several possible semantics. E.g., for (student, {course}) we list several possible interpretations. (1) The courses already taken by a student. (2) The courses currently taken by a student. (3) The courses taught by all current instructors of a student. (4) The courses offered by the supervisor of a student.

Let us enumerate the paths corresponding to these 4 interpretations. (1) (**student, transcript, course_records, course_record, course**). (2) (**student, transcript, sections, section, crsections, course**). (3) (**student, transcript, sections, section, professor, sections, section, crsections, course**). (4) (**student, professor, sections, section, crsections, course**).

Different users may want different PMs for the same pair. Thus, there is no sense in talking about a "right" or "wrong" PM for a given pair. Instead, we talk in this paper about the *desired* PM. A PM is generated by the PMG system and the user can verify whether it is the PM desired by him. The PMG system does not guarantee that the generated PM has the desired semantics, as this task seems to be beyond the capabilities of an automatic traversal. It is much easier for a user to verify a PM than to traverse the schema on his own and find it. By developing a PMG system which generates the desired PM in most cases

at the first try, the system will serve as an important tool to support queries in an OODB. In the few other cases, the PMG system enables the user to perform subsequent traversals, incorporating the feedback she gained, to find the desired PM in almost all remaining cases (see Mehta et al., 1993; Mehta, 1993).

4.2. Path-method generation algorithm

In Mehta et al. (1993) and Mehta et al. (1995) we present in detail the PMG algorithm using access relevance to generate PMs and compare its results to those of several other algorithms. In this paper we just review this algorithm briefly to motivate the study of efficient algorithms for computing access relevances which is the subject of this paper.

The PMG algorithm has two *required* parameters, the source s and the target t of the desired PM. The algorithm is basically a traversal algorithm from the source s to the target t. It maintains a stack containing the classes along the current path, which provide the class sequence of the PM once t is reached. The stack is also used for backtracking in case the traversal cannot continue forward.

The traversal starts at s and repeats in each iteration the same operation for the node u at the top of the stack. In each step we consider all outgoing neighbor classes of u. To select the next class in the PM we consider for each outgoing neighbor class v of u the access weight $W[u, v]$ and the access relevance matrix entry $AR[v, t]$. We apply to these two numbers a weighting function (PRODUCT) and select out of the neighbors the class v which maximizes the return value of the weighting function. If the selected class v is identical to the target t, the traversal is completed. Otherwise the next iteration follows, where u is set to the class v that was selected.

We now demonstrate the application of the PMG algorithm for the pair (**student** (2), **course** (10)). The class **student** (2) has 4 outgoing connections to **person** (11), **students** (3), **professor** (12), and **transcript** (4). We calculate now the value of $W(u, v) * AR(v, t)$, using the PRODUCT weighting function for each of the neighbors v. The ARs are taken from Table 2 as calculated for the schema of figure 1.

$$W[2, 1] * AR[1, 10] = 0 * 0 = 0 \quad \text{(Note: We use Rule 2(b)).}$$
$$W[2, 3] * AR[3, 10] = 0.3 * 0.076 = 0.0228$$
$$W[2, 12] * AR[12, 10] = 0.3 * 0.088 = 0.0264$$
$$W[2, 4] * AR[4, 10] = 0.9 * 0.168 = 0.1512$$

Thus, **transcript** (4) is selected as the next class in the PM and is pushed onto the stack. Now transcript has 3 outgoing connections to neighbors: **student** (2), **course_records** (6), and **sections** (7). The class **student** is already on the stack, so it is not a candidate. We compute $W[u, v] * AR[v, t]$ for the other classes.

$$W[4, 6] * AR[6, 10] = 0.8 * 0.21 = 0.168$$
$$W[4, 7] * AR[7, 10] = 0.4 * 0.125 = 0.05$$

Hence **course_records** (6) is selected as the next class in the PM and is pushed onto the stack. The class **course_records** has two outgoing connections, to **transcript** and to

course_record. The class **transcript** is already on the stack. Hence **course_record** is selected as the next class in the PM and is pushed onto the stack. It has only one unused outgoing connection to a neighbor, the class **course**, which is the target t of the desired PM. Hence, the PM is completed and its class sequence is (in reverse order): (**student**, **transcript**, **course_records**, **course_record**, **course**). This is the PM of interpretation (1) from Section 4.1, i.e., all the courses already taken by a student.

The user sees the PM generated by the algorithm and determines if it has the semantics intended. What happens if this is not the case? Then he applies the algorithm again, providing some optional parameters. There are two optional parameters: forbidden classes and required classes. Suppose, for example, that the user is interested in all the courses the student is taking now (interpretation 2 from Section 4.1). Looking at the first PM returned by the algorithm he tries to identify where it went wrong. He sees that the offered PM contains the class **course_records** which holds courses already taken. Then he can mark **course_records** as a forbidden class. By this choice he forces the algorithm to avoid this class when reconstructing the PM, resulting in the following PM: (**student**, **transcript**, **sections**, **section**, **crsection**, **course**). This corresponds to the desired interpretation (2).

Another option to obtain this PM is by picking a required class which must belong to the PM. For interpretation (2) the class **sections** is such a class since to obtain the current courses a student is taking we need to access the sections he is taking. Now, instead of searching for a path from **student** to **course** we search for a path from **student** to the required class **sections** and a path from **sections** to the target class **course**. The concatenation of these two paths actually yields the same PM as interpretation (2).

5. Computing access relevance in an interoperable multi-OODB

So far we have discussed path traversal in a single database. To automate path traversal in interoperable databases means to automate it across database boundaries which is even more important for a naive user. This is because not every possible query which needs a combination of data from multiple databases can be anticipated and represented in a predefined view. Rather, only simple cross-database relationships via joinable attribute domains can be determined. Thus, PMs are needed to help the user in navigating along these relationships.

Let us consider for example a university environment which typically contains several units. Each unit has its own autonomous OODB which contains necessary information for its day to day operations. In addition, these OODBs are interoperable, because it is necessary for one unit to access information from other units. An IM-OODB for a university environment, which consists of five OODBs, is shown in figure 3. It consists of an *admissions* OODB, a *registration* OODB which was discussed in the previous sections, a *departmental* OODB, a *finance* OODB and a *library* OODB.

In an IM-OODB one component OODB can retrieve information from another component OODB. For example, the *registration* OODB retrieves information from the *departmental* OODB about every professor's teaching sections. It retrieves information from the *admissions* OODB about the admission status for new students, and from the *finance* OODB about overdue payments before a student may register each semester.

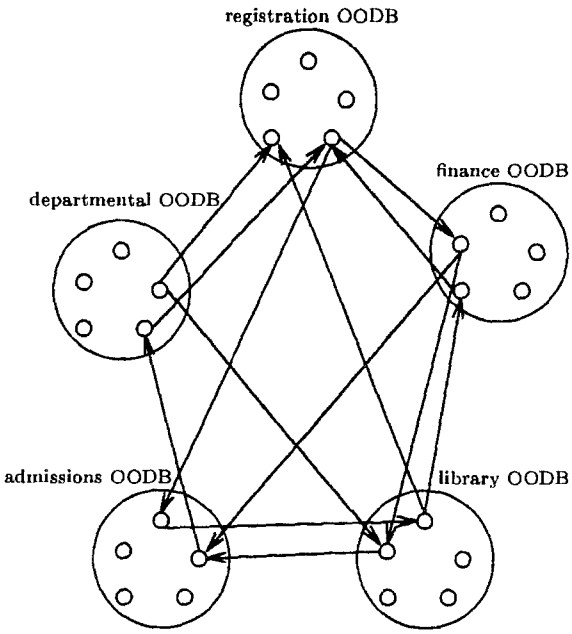


Figure 3. An interoperable multi-OODBs system.

We have shown only a few classes of each component OODB in figure 3, but in reality the number of classes in each OODB is large. Let us assume that each component has n classes. All-pair computation of access relevances for each component OODB requires computation of n^2 values. As we have k component OODBs it is necessary to compute kn^2 values. If these component OODBs are interoperable, there are totally kn classes which requires the computation of $(kn)^2$ values, a number much larger than kn^2. Even if we subtract the kn^2 values that were already computed, the combination into a Multi-OODB still requires to compute and store $k(k-1)n^2$ values, a number that is large compared to the number of values stored for all individual databases. Therefore, it is not practical to simply extend our approach from Sections 2 and 3 to IM-OODBs. Thus, we apply a hierarchical approach. While the internal values for each component OODB are precomputed, the values between pairs of classes from two different OODB components will be computed on the fly based on the precomputed access relevance of each component OODB and the access weights of the relatively few connections between pairs of classes from different OODBs.

6. An IM-OODB containing only two OODBs

In this section we discuss the computation of access relevances in an IM-OODB containing only two databases. For this special case we present an algorithm which is more efficient than in the general case. Furthermore, this case will serve as an introduction to the more complex general case in Section 7.

To explain how to realize a connection between two component OODBs, we follow Czejdo and Taylor (1991). The problem is that a class in an autonomous OODB cannot have pointers to a class in another OODB. The solution of Czejdo and Taylor (1991) selects two classes, one in each OODB, that represent the same real world objects. In their example the two classes represent social security numbers, one with dashes and one as integers. For both of these classes they define a class in the IM-OODB schema. Then a correspondence is defined between the two IM-OODB schema classes and implemented as a mathematical transformation capable of transforming an instance of one class to an instance of the other class. In this way, we avoid pointers for instances from different IM-OODBs. The connection between the two classes of the two OODBs is realized by a PM (in our terminology) from one class to its IM-OODB schema class and on through the transformation to the IM-OODB schema class of the other class and then to the other class.

In Czejdo and Taylor (1991) every item of information is represented as a class. However, in our abstract model as well as in many other models (e.g., VML (Martin, 1991), ONTOS (Klas et al., 1991), ObjectStore (Lamb et al., 1991)), most items of information which are stored with a class are represented as attributes. For example, the social security number of a person will be an attribute of the class person. Thus, we have to modify the solution of Czejdo and Taylor (1991) as follows. We pick two classes, one in each OODB, representing the same real world object, e.g., in our upcoming example **dep.professor** and **reg.professor** to be represented in the IM-OODB schema. The dot notation is used to distinguish classes of different OODBs. The mathematical transformation between these two classes in the IM-OODB schema is based on the correspondence of their appropriate attributes, e.g., an attribute representing the name or the social security number of the professor.

However, in our model we can have a connection between two classes, one in each OODB, *even if they do not represent the same real world object.* If both classes have corresponding attributes representing the same real world information, the correspondence can be realized based on the mathematical transformation of the attributes of the two classes, even though the classes do not represent the same real world object. This enables more flexibility in establishing connections between different OODBs, as seen in the following example.

In figure 4 the class **course** of the *registration* OODB has an attribute *dept_name* which represents the department that offers this course. The class **department** in the *departmental* OODB has an attribute *name*. Presumably the department names used in these two databases are identical. Thus, we can have a PM with the class sequence (**course**,

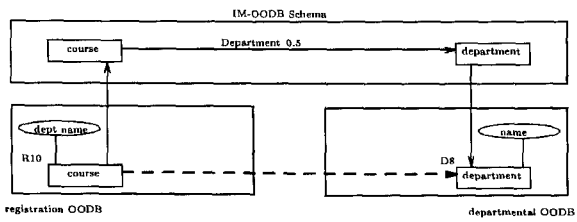


Figure 4. How to realize connections between two component OODB.

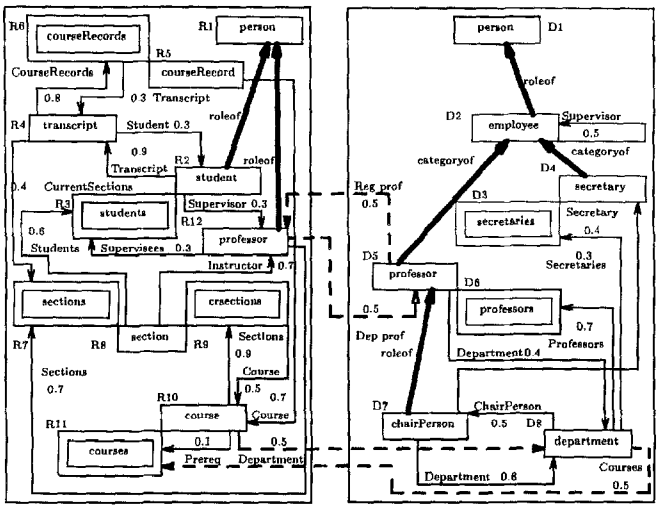


Figure 5. An IM-OODB containing the registration and the departmental OODB.

im-course, **im-department**, **department**), where classes **im-course** and **im-department** are defined in the IM-OODB schema. The endpoints of this path do not represent the same real world objects. Nevertheless, an attribute-pair (*dept_name*, *name*) can be used for implementing the transformation in the IM-OODB schema.

Two small subschemas of the *registration* OODB (figure 1) and the *departmental* OODB and their corresponding graphs are shown in figure 5 and figure 6. Both the OODBs have a class **professor**. Two PMs between these two classes, Dep_prof and Reg_prof, consist of the class sequences (**reg.professor**, **im-reg.professor**, **im-dep.professor**, **dep.professor**) and (**dep.professor**, **im-dep.professor**, **im-reg.professor**, **reg.professor**), respectively. An attribute-pair (*name*, *name*) can be used for establishing the correspondence between instances. The PM Department for the class **course** is described in figure 4. There is a PM Courses defined from **department** to **courses** (figure 5). The attribute *dept_name*, is also defined for **courses** and has a non-nil value only for instances of a set of courses of the same department. Thus, the same attribute-pair (*dept_name*, *name*) can be used for correspondence between such instances of the class **courses** and instances of the class **department**. There may be instances of the class **courses** representing a set of courses which have several department names, e.g., prerequisite sets. Such instances are not considered for correspondence with the class **department**.

We will now define the problem of computing access relevances in an IM-OODB. In a typical IM-OODB, a component OODB is developed first, and later on it is added to the IM-OODB. We assume that all-pair access relevances for each individual OODB is precomputed with the algorithms discussed in Section 3. In the following discussion we will define several terms needed to compute access relevance in an IM-OODB. Assume two independent databases OODB_i and OODB_j. Denote by a_p the classes of OODB_i and by b_q the classes of OODB_j, where $i \neq j$, i.e., $a_p \cap b_q = \emptyset$.

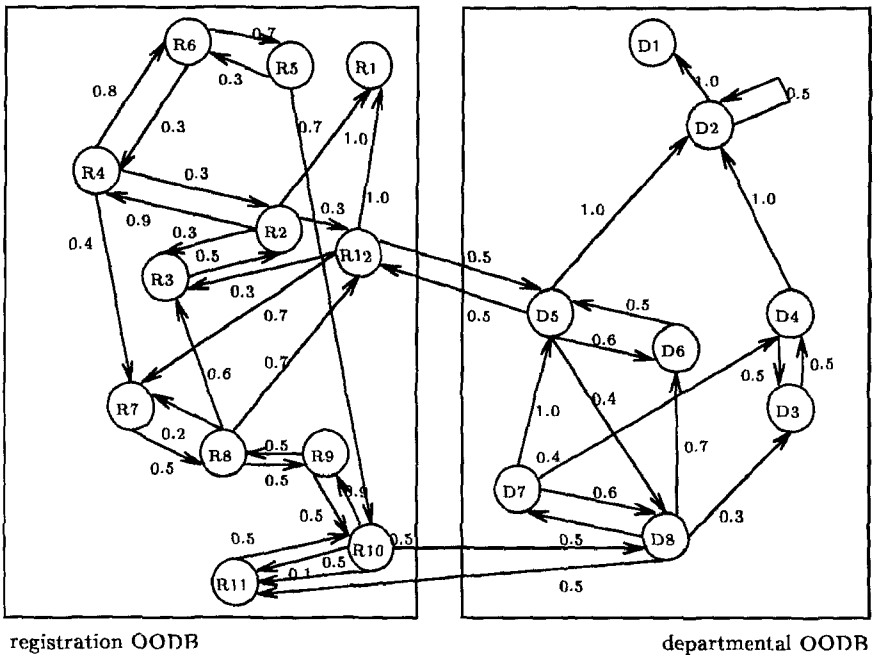


Figure 6. The registration and departmental schemas as a directed graph.

Definition 8. A connection from a class $a_p \in \text{OODB}_i$ to another class $a_q \in \text{OODB}_i$ is called an *intra*-OODB *connection.*

Definition 9. A connection from a class $a_p \in \text{OODB}_i$ to another class $b_q \in \text{OODB}_j, i \neq j$, is called an *inter*-OODB *connection.*

As discussed earlier, such inter-OODB connections are realized as PMs. Their weights are determined by Rule 3 since they should not interfere with weights of intra-OODB connections.

Rule 3. The sum of the weights on the outgoing inter-OODB connections of a class conforms to the following constraints: (1) $W_i \in [0, 1], 1 \leq i \leq n$. (2) $\sum_{i=1}^{n} W_i = 0.5 * n$, where, n is the number of outgoing inter-OODB connections from this class. Each inter-OODB connection is assigned a weight reflecting its relative frequency of traversal.

Definition 10. For each component OODB_i, a class $a_p \in \text{OODB}_i$, which has an inter-OODB connection *or* is referred to by an inter-OODB connection is called a *contact class.*

We will assume that there are relatively few inter-OODB connections and contact classes in IM-OODBs. Considering the difficulties in defining such classes (Biliris and Panagos, 1995) this is a realistic assumption. In figure 5, the relationship Transcript of **student** to

transcript is an intra-OODB connection. The PM Department of the class **course** to the class **department** is an inter-OODB connection. The classes **course** and **department** are examples of contact classes.

Definition 11. Let a_s and a_t be classes in OODB_i. A path $P(a_s, a_t) = a_s(=a_{i_1}), a_{i_2}, \ldots, a_{i_k}$ $(=a_t)$ using only classes of OODB_i is called an *intra*-OODB path.

Theoretically, there may exist a most relevant path between two classes of the same OODB going through classes of another OODB. But we will not consider such paths since it contradicts the autonomy assumption (Sheth and Larson, 1990) of the OODBs.

Definition 12. Let a_p and b_q be classes of OODB_i and OODB_j, $i \neq j$, respectively. A path $P(a_p, b_q)$, is called an *inter*-OODB path.

In figure 5, the path of the class sequence: (**student**, **transcript**, **sections**) is an intra-OODB path, while the path of the class sequence: (**section**, **reg.professor**, **dep.professor**, **department**) is an inter-OODB path.

Definition 13. Let $P(a_p, b_q)$ be an inter-OODB path from class $a_p \in \text{OODB}_i$ to class $b_q \in \text{OODB}_j$, $i \neq j$. In general, such a path may contain several inter-OODB connections. An inter-OODB path containing only one inter-OODB connection is called a *direct inter-OODB path.*

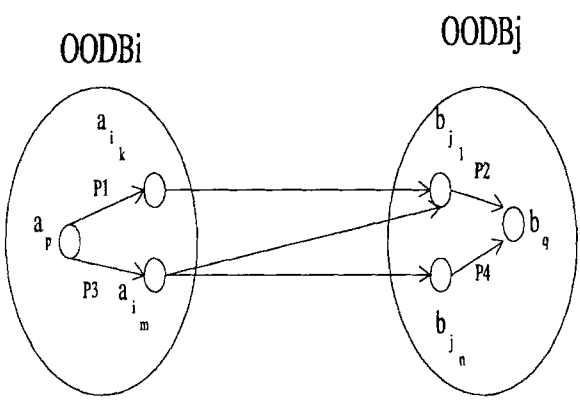
Note that in an IM-OODB containing only two OODBs we shall assume first that an inter-OODB path is a direct inter-OODB path since other kinds of paths traversing back and forth between the two OODBs are very unlikely to have a reasonable interpretation. However, such paths will be considered in the next section.

A direct inter-OODB path P has the form $a_p(=a_{i_1}), a_{i_2}, \ldots, a_{i_k}, b_{j_1}, b_{j_2}, \ldots, b_{j_l}(=b_q)$ where a_{i_m}, $1 \leq m \leq k$ are classes of OODB_i and b_{j_n}, $1 \leq n \leq l$ are classes of OODB_j. Hence, $(a_{i_r}, a_{i_{r+1}})$, $1 \leq r < k$ and $(b_{j_r}, b_{j_{r+1}})$, $1 \leq r < l$ are intra-OODB connections and (a_{i_k}, b_{j_1}) is the only inter-OODB connection in $P(a_p, b_q)$. The access relevance value $\text{ARV}(P(a_p, b_q))$ for a weighting function WF is defined as

$$\text{ARV}(P) = \text{WF}\big(\text{AR}(a_p, a_{i_k}), W(a_{i_k}, b_{j_1}), \text{AR}(b_{j_1}, b_q)\big)$$

The access relevance from a_p to b_q is defined by maximizing the access relevance value $\text{ARV}(P)$ over all paths $P(a_p, b_q)$, that is, over all the paths $P(a_p, a_{i_k})$ and all the paths $P(b_{j_1}, b_q)$, for all inter-OODB connections (a_{i_k}, b_{j_1}) between all possible contact classes $a_{i_k} \in \text{OODB}_i$ and all possible contact classes $b_{j_1} \in \text{OODB}_j$.

$$\begin{aligned}\text{AR}(a_p, b_q) &= \max_z \text{ARV}(\text{P}_z) \\ &= \max_{(\text{all inter-OODB connections}(a_{i_k}, b_{j_1}))} \text{WF}\big(\text{AR}(a_p, a_{i_k}), W(a_{i_k}, b_{j_1}), \text{AR}(b_{j_1}, b_q)\big)\end{aligned}$$



	1	2	3	4	5	6	7	8
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	0	0.25	1	0.5	0	0	0	0
4	0	0.5	0.5	1	0	0	0	0
5	0	0.5	0.12	0.06	1	0.6	0	0.4
6	0	0.25	0.06	0.03	0.5	1	0	0.2
7	0	0.5	0.2	0.4	0.25	0.5	1	0.6
8	0	0.18	0.3	0.15	0.35	0.7	0	1

Figure 7. Computation of access relevance, Table 3: AR for departmental OODB.

We assume that using the efficient algorithms of Section 3, access relevances for OODB_i and OODB_j are already computed and stored. All-pair access relevances for OODB_i (OODB_j) are stored in a matrix AR_i (AR_j). Thus we have a simple algorithm to compute $\text{AR}(a_p, b_q)$ as follows (see figure 7):

procedure Compute_AR_IM_OODB (a_p, b_q : class);
begin
(1) $\text{AR}[a_p, b_q] := 0$;
(2) **for** each inter-OODB connection (a_{i_k}, b_{j_1}) such that
(3) $a_{i_k} \in \text{OODB}_i$ and $b_{j_1} \in \text{OODB}_j$ **do**
$\text{AR}[a_p, b_q] := \max(\text{AR}[a_p, b_q], \text{WF}(\text{AR}_i[a_p, a_{i_k}], W[a_{i_k}, b_{j_1}], \text{AR}_j[b_{j_1}, b_q]))$
end

Suppose we want to find the access relevance between **student** and **department**. In step 1, $\text{AR}[a_p, b_q]$ is set to zero. In the **for**-loop of step 2, for each inter-OODB connection we try to find the maximum access relevance value. Two access relevance matrices for *registration* OODB and *departmental* OODB are shown in Table 2 (Section 3.1) and Table 3, respectively. The steps of algorithm *Compute_AR_IM_OODB* for computing AR (for WF = PRODUCT) from **student** to **department** using two alternative inter-OODB connections (R12, D5), and (R10, D8) are shown below.

1. AR[R2, D8] := 0
2. AR[R2, D8] := max (AR[R2, D8], WF (AR[R1, R12], W[R12, D5], AR[D5, D8]))
:= max (0.0, WF (0.3, 0.5, 0.4)) := max (0.0, 0.06) := 0.06
3. AR[R2, D8] := max (AR[R2, D8], WF (AR[R2, R10], W[R10, D8], AR[D8, D8])
:= max (0.06, WF (0.151, 0.5, 1.0)) := max (0.06, 0.075) := 0.075

The complexity of this algorithm is $O(c)$ where c is the number of inter-OODB connections from OODB_i to OODB_j. Typically in IM-OODBs c is a constant or a sublinear function of n, such as $\log n$ or $\sqrt{n}$. Hence this is a very efficient algorithm, which is appropriate for online computation.

7. An IM-OODB containing many OODBs

Consider, for example, an IM-OODB with 5 OODBs: OODB_A, OODB_B, OODB_C, OODB_D, OODB_E. Denote a class of OODB_A (OODB_B, OODB_C, OODB_D, OODB_E) by a_i (b_m, c_l, d_k, e_j), respectively. Consider an inter-OODB PM from a_p to b_q which involves classes of all 5 OODBs. This is an indirect inter-OODB PM. The corresponding path P in G can have, for example, the form

$$\begin{aligned}&\big(a_p(=a_{i_1}), a_{i_2}, \ldots, a_{i_v}, e_{j_1}, e_{j_2}, \ldots, e_{j_w}, d_{k_1}, d_{k_2}, \ldots, d_{k_x}, c_{l_1}, c_{l_2}, \ldots, c_{l_y},\\ &\quad b_{m_1}, b_{m_2}, \ldots, b_{m_z}(= b_q)\big).\end{aligned}$$

This path involves 4 inter-OODB connections: (a_{i_v}, e_{j_1}), (e_{j_w}, d_{k_1}), (d_{k_x}, c_{l_1}), and (c_{l_y}, b_{m_1}). Others are intra-OODB connections. The access relevance value ARV(P(a_p, b_q)) for a weighting function WF is

$$\begin{aligned}\text{ARV}(P(a_p, b_q)) = \text{WF}\big(&\text{AR}(a_{i_1}, a_{i_v}), W(a_{i_v}, e_{j_1}), \text{AR}(e_{j_1}, e_{j_w}), W(e_{j_w}, d_{k_1}), \text{AR}(d_{k_1}, d_{k_x}),\\ &W(d_{k_x}, c_{l_1}), \text{AR}(c_{l_1}, c_{l_y}), W(c_{l_y}, b_{m_1}), \text{AR}(b_{m_1}, b_{m_z})\big)\end{aligned}$$

The access relevance from a_p to b_q is defined by maximizing access relevance values ARV(P) over all paths $P(a_p, b_q)$. Suppose for the moment that all those paths actually traverse these 5 OODBs in the same order of the above mentioned P, i.e., (A, E, D, C, B), Then we obtain

$$\begin{aligned}\text{AR}(a_p, b_q) &= \max_z \text{ARV}(P_z(a_p, b_q))\\ &= \max \text{WF}\big(\text{AR}(a_p, a_{i_v}), W(a_{i_v}, e_{j_1}), \text{AR}(e_{j_1}, e_{j_w}), W(e_{j_w}, d_{k_1}), \text{AR}(d_{k_1}, d_{k_x}),\\ &\qquad W(d_{k_x}, c_{l_1}), \text{AR}(c_{l_1}, c_{l_y}), W(c_{l_y}, b_{m_1}), \text{AR}(b_{m_1}, b_q)\big)\end{aligned}$$

where max is taken over all possible inter-OODB connections (a_{i_v}, e_{j_1}), (e_{j_w}, d_{k_1}), (d_{k_x}, c_{l_1}), (c_{l_y}, b_{m_1}). The pair (a_{i_v}, e_{j_1}) stands for any pair of contact classes, such that a_{i_v} is in OODB_A, e_{j_1} is in OODB_E and there exists an edge between a_{i_v} and e_{j_1}. There might be several such edges, and maximization is done by selecting the one edge that leads to the largest overall access relevance value. The same applies to the other pairs of OODBs. The ARV(P)s are precomputed for each OODB. The access weights of the inter-OODB connections are known to the IM-OODB system.

If the number of such connections between every two OODBs is c then our optimization has to select from c^4 possibilities. Furthermore there are many more possible patterns of paths from OODB_A to OODB_B using different subsets (whose number is an exponential function of the number of OODBs) and different orders (whose number is a factorial function of the number of OODBs). Thus, it is not practical to extend the computations of the previous section to the case of many OODBs. Furthermore, paths may return to an OODB several times using each time different classes.

Thus, we shall look for a solution which involves modeling the graph representation $G(V, E)$ of the IM-OODB as an auxiliary small graph $H(U, D)$. For each OODB, U will

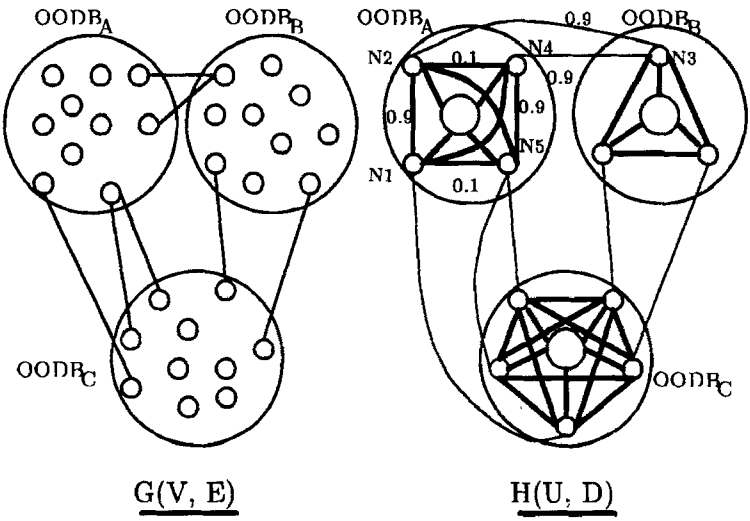


Figure 8. Two graphs G(V, E) and H(U, D) for IM-OODB with many OODBs.

contain a node for each contact node of the OODB and an extra center node representing a variable node of the OODB. This variable node will at each time represent another node (possibly a contact node) of the OODB. However, the graph H contains only one such node for each OODB, keeping the number of nodes of H relatively small. Each OODB is represented by a clique of its contact classes and a star with its variable class as center and its contact classes as end points. That is, for each OODB, D will contain "clique" edges connecting each pair of contact nodes and "star" edges connecting the variable node to each one of the contact nodes. In addition, D contains an edge for each inter-OODB connection of the IM-OODB (figure 8). The inter-OODB edges have given access weights. For clique edges we define the access weight as the access relevance in G between the two nodes of the edge, which are precomputed for each OODB, e.g., $W_H(e_{j_1}, e_{j_w}) = \mathrm{AR}(e_{j_1}, e_{j_w})$. While H is described in figure 8 as a bidirectional graph, since in general we can have a path from every node to every other node, the access weights are usually different for both directions unless the original OODB schemas are bidirectional. The access weight of a star edge varies. Whenever the center node a_i represents a specific node e.g., a_{i_1}, then the access weight of a star edge (a_i, a_{i_v}) is defined as the access relevance $\mathrm{AR}(a_i, a_{i_v})$ in G, i.e., $W_H(a_i, a_{i_v}) = \mathrm{AR}(a_i, a_{i_v})$. Note that when a star node a_i represents a contact node a_{i_v} then, by our definition in Section 2.3, $\mathrm{AR}(a_i, a_{i_v}) = 1$.

Definition 14. A path P between two center nodes in H is called *proper* if (1) all the rest of the nodes are contact nodes, (2) the sequence of edges except for the two end edges (which are star edges) consists of inter-OODB edges and clique edges such that no two clique edges are consecutive.

In general, the path will have inter-OODB edges and clique edges in alternating order, but a proper path may sometimes use only one contact node of an OODB rather than two, in which case this is the only node of the path which belongs to this OODB.

Lemma 2. *For every path Q between two center nodes in H there exists a proper path R such that $AR(R) \geq AR(Q)$.*

Proof: Due to space limitations we need to omit the following proofs, which can be found in Mehta et al. (1993). □

Lemma 3. *For every most relevant path P from a node a_p to a node b_q in G there exists a corresponding proper path Q in H from the center node a_i to the center node b_m, such that $AR(P) = AR(Q)$.*

Lemma 4. *For every proper path Q from center node a_i representing a_p to center node b_m representing b_q in H there exists a pair of nodes $a_p \in OODB_A$ and $b_q \in OODB_B$ and a corresponding path P from a_p to b_q in G such that $AR(Q) = AR(P)$.*

The omitted proof is based on the definition of H and works in reverse order to the previous proof.

Theorem 2. *A most relevant path Q connecting a pair of center nodes a_i and b_m in H which represent nodes a_p and b_q in different OODBs in G has a corresponding most relevant path P in G from a_p to b_q such that $AR(P) = AR(Q)$.*

The theorem implies that for calculating the access relevance from node a_p in OODB_A to node b_q in OODB_B we take the nodes a_i and b_m in H to represent a_p and b_q, respectively, and calculate the access relevance in H from a_i to b_m. In this way, for each pair of classes of different OODBs, we can find the access relevance by applying the single source algorithm from Section 3 (i.e., PRODUCT_AR) of complexity $\min(O(|U|^2, O(|D| \log |D|))$ which uses the precomputed access relevance of the given OODBs. Clearly $|U| \ll |V|$ holds here.

This calculation will yield a most relevant path in G taking into account the different possible subsets of OODBs traversed and their different possible orderings in the path, as edges of a most relevant path could be from different OODBs. Intuitively, we have replaced the graph G by the much smaller graph H and can apply to H exactly the same techniques that we used within a single OODB (Section 3).

The above calculation takes into account even the possibility of traversing through an OODB more than once. One such case occurs when there exists a most relevant path in H between two contact nodes of one OODB using at least one node from another OODB. To see this, refer back to Graph H in figure 8. Assume that we are looking for a most relevant path from N_1 to N_5. It is possible that such a path consists of $(N_1, N_2, N_3, N_4, N_5)$. When the access relevance between N_2 and N_4 was computed in OODB_A alone, the "shortcut" through N_3 was not available (see figure 8). However, such a path is considered when modeling with H. Another such case occurs when a most relevant path in H contains two clique edges of the same OODB, which are not consecutive in the path. Note that in such a case it may happen that the two intra-OODB paths corresponding to two clique edges of the same OODB share a non-contact node. But then the most relevant path contains a cycle which can be removed without decreasing the access relevance as explained regarding Property 1 (from Section 2.3).

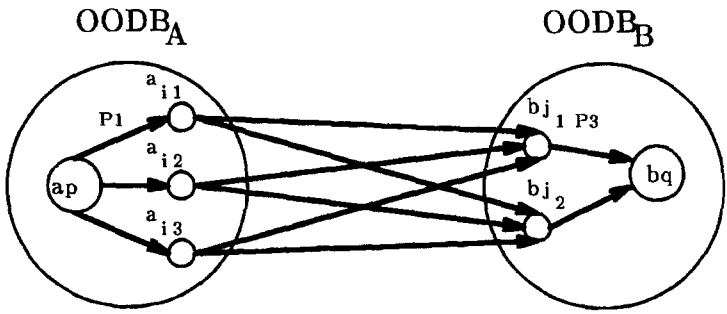


Figure 9. An efficient computation of access relevance.

We can further improve the efficiency as follows. We realize that a most relevant path from a_p to b_q starts and ends with a center node in H but contains no other center nodes. Furthermore, the access weights of the star edges in an OODB change with the choice of the source and target classes in the OODBs, but the rest of the access weights of D are independent of this choice. Thus, we define a subgraph $I = (U_1, D_1)$ of H where the star subgraphs are omitted, i.e., each OODB is represented in I only by the clique of its contact classes. Now, we can precompute the access relevance for all pairs of nodes in I by applying the single source algorithm of Section 3 $|U|$ times resulting in a complexity of $\min(O(|U_1|^3, O(|U_1||D_1|\log|D_1|))$. The result is stored in an access relevance matrix AR_I for I. Now a most relevant path between arbitrary classes $a_p \in \mathrm{OODB}_A$ and $b_q \in \mathrm{OODB}_B$ is represented as a concatenation of three paths $P1(a_p, a_i)$, $P2(a_i, b_m)$, and $P3(b_m, b_q)$ where a_i and b_m are contact classes of OODB_A and OODB_B, respectively. The path $P1$ ($P3$) is a most relevant path of OODB_A (OODB_B) and $P2$ is a most relevant path of I. Thus,

$$\mathrm{AR}(a_p, b_q) = \max_{(\text{contacts } a_i \in \mathrm{OODB}_A, b_m \in \mathrm{OODB}_B)} \mathrm{WF}(\mathrm{AR}(a_p, a_i), \mathrm{AR}_I(a_i, b_m), \mathrm{AR}(b_m, b_q))$$

Now all these access relevances are precomputed and $\mathrm{AR}(a_p, b_q)$ is found with complexity $O(c_A c_B)$, where c_A and c_B are the numbers of contact classes of OODB_A and OODB_B, respectively (figure 9). As explained in Section 6, these numbers are typically constants or sublinear functions of the numbers of classes in the OODBs. Thus, we achieve a very fast online algorithm for computing access relevances between classes of different OODBs.

8. Conclusion

In this paper we have discussed efficient algorithms for computing access relevances in an OODB as well as in an IM-OODB. We have used access relevance as a guide for PMG (Mehta et al., 1993), for automatic generation of PMs in OODBs, to support retrieving and updating distant information for a user lacking complete knowledge of the conceptual schema. We have introduced the notion of *access weight* as a measure of frequency of traversal of a connection during the operation of an OODB. Access weights are used to further define access relevance as a measure of the connection between two indirectly related classes. For

such an OODB, we have developed algorithms of complexity $O(n\ e\ \log n)$ for computing access relevance using the t-norm PRODUCT.

We have also discussed computation of access relevances in an IM-OODB. The connections between classes of different OODBs are realized using PMs. We started with a special case of an IM-OODB containing two OODBs and derived an online algorithm for computation of access relevance values. Then we used a hierarchical approach to derive an online algorithm for the computation of access relevances in an IM-OODB containing many OODBs. Both these algorithms use precomputed access relevances for each component OODB.

Acknowledgment

We gratefully acknowledge the valuable comments that we received during the process of anonymous review.

References

Aho, A., Hopcroft, J., and Ullman, J.D. (1983). *Data Structures and Algorithms*, Reading, MA: Addison-Wesley Publishing Company.

Alashqur, A.M., Su, S.Y.W., and Lam, H. (1989). OQL: A Query Language for Manipulating Object-Oriented Databases. In P.M.G. Apers and G. Wiederhold (Eds.), *Proceedings of the Fifteenth International Conference on Very Large Databases*, pp. 433–442.

Alhajj, R. and Arkun, M.E. (1993). A Query Model for Object-Oriented Databases. In *Proceedings of the Ninth International Conference on Data Enginering* (pp. 163–171). Los Alamitos, CA: IEEE Computer Society Press.

Bertino, E., Negri, M., Pelagatti, G., and Sbattella, L. (1992). Object-Oriented Query Languages: The Notion and the Issues. *IEEE Transactions on Knowledge and Data Engineering*, 4, 223–237.

Biliris, A. and Panagos, E. (1995). A High-Performance Configurable Storage Manager. To appear in *Proceedings of the Ninth International Conference on Data Engineering*, Taipei, Taiwan.

Bonissone, P.P. and Decker, K.S. (1986). Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-off Precision and Complexity. *Machine Intelligence Pattern Recognition*, 4, 217–247.

Butterworth, P., Otis, A., and Stein, J. (1991). The GemStone Object Database Management System. *Communications of the ACM*, 20, 64–77.

Cheiney, J.-P. and Lanzelotte, R.S.G. (1992). A Model for Optimizing Deductive and Object-Oriented DB Requests. In F. Golshani, *Proceedings of the Eighth International Conference on Data Enginnering* (pp. 385-392). (Ed.), Los Alamitos, CA: IEEE Computer Society Press.

Czejdo, B. and Taylor, M. (1991). Integration of Database Systems Using an Object-Oriented Approach. In *Proc. First International Workshop on Interoperability in Multi–Database Systems* (pp. 30–37). Kyoto, Japan.

Czejdo, B. and Taylor, M. (1991). Integration of Database Systems and Smalltalk. In *Proc. Symposium on Applied Computing*, Kansas City.

Elmasri, R. and Navathe, S.B. (1989). *Fundamentals of Database Systems*. New York, NY: Benjamin/Cummings Publishing Co.

Fankhauser, P., Kracker, M., and Neuhold, E.J. (1991). Semantic vs. Structural Resemblance of Classes. *SIGMOD Record, 'Semantic Issues in Multidatabase Systems'*, 34, 59–63.

Fankhauser, P. and Neuhold, E.J. (1992). Knowledge Based Integration of Heterogeneous Databases. In *Proc. of the IFIP TC2/WG2.6 Conference on Semantics of Interoperable Database Systems, DS–5*, Lorne, Victoria, Australia.

Geller, J., Perl, Y., and Neuhold, E.J. (1991). Structure and Semantics in OODB Class Specifications. *SIGMOD record, 'Semantic Issues in Multidatabase Systems'*, 34, 40–43.

Halper, M., Geller, J., Perl, Y., and Neuhold, E.J. (1992). An OODB Graphical Schema Representation. In *Proc. IDS92: Int. Workshop on Interfaces to Database Systems*, Glasgow.

Horowitz, E. and Sahni, S. (1989). *Fundamentals of Computer Algorithms*, Reading, MA: Computer Science Press.

Kemper, A. and Moerkotte, G. (1990). Advanced Query Processing in Object Bases Using Access Support Relations. In D. McLeod, R. Sacks-Davis, and H. Schek (Eds.). *Proc. of the Sixteenth International Conference on Very Large Databases*, pp. 290–301.

Kifer, M., Kim, W., and Sagiv, Y. (1992). Querying Object-Oriented Databases. In *Proceedings 1992 ACM SIGMOD International Conf. on Management of Data* (pp. 393–402). San Diego, California.

Kim, W. (1989). A Model of Queries for Object-Oriented Databases. In P.M.G. Apers and G. Wiederhold (Eds.). *Proceeding of the Fifteenth International Conference on Very Large Databases*, pp. 423–432.

Kim, W. (1990). *Introduction to Object-Oriented Databases*, Reading, MA: The MIT Press.

Klas, W., Neuhold, E.J., Bahlke, R., Drosten, K., Fankhauser, P., Kaul, M., Muth, P., Oheimer, M., Rakow, T., and Turau, V. (1991). VML Design Specification Document. Tech. Report, GMD-IPSI, Germany.

Klir, G.L. and Folger, T.A. (1988). *Fuzzy Sets, Uncertainty and Information*, Prentice Hall.

Kracker, M. (1992). A Fuzzy Concept Network Model and its Applications. In *Proc. FUZZ-IEEE '92* (pp. 761-768). San Diego.

Kracker, M. and Neuhold, E.J. (1989). Schema Independent Query Formulation. In *Proc. 8th Int. Conf. on Entity-Relationships Approach* (pp. 233–247). Toronto, Canada.

Lamb, C., Landis, G., Orenstein, J., and Weinreb, D. (1991). The Objectstore Database System. *Communications of the ACM*, pp. 50–63.

Lanzelotte, R.S.G., Valduriez, P., and Zait, M. (1992). Optimization of Object-Oriented Recursive Queries Using Cost-Controlled Strategies. In *Proc. 1992 ACM SIGMOD International Conference on Management of Data* (pp. 256-265). San Diego, California.

Litwin, W. (1985). Implicit Joins in the Multidatabase System MRDSM. *IEEE-COMPSAC*, pp. 495–504.

Maier, D., Rozenshtein, D., Salvater, S., Stein, J., and Warren, D. (1987). PIQUE: A Relational Query Language Without Relations. In *Information Systems*, 12(3), 317–335.

Maier, D. and Ullman, J.D. (1983). Maximal Objects and the Semantic of Universal Relation Databases. *ACM Transactions on Database Systems*, 8(1), 1–14.

Martin, R. (1991). ONTOS Overview. In *Proc. Executive Briefing on Object-oriented Database Management*, San Francisco.

Mehta, A. (1993). Algorithms for Generation of Path-Methods in Object-Oriented Databases. Doctoral Dissertation, CIS Dept., New Jersey Institute of Technology, Newark, NJ.

Mehta, A., Geller, J., Perl, Y., and Fankhauser, P. (1993). Computing Access Relevance for Path-Method Generation in OODBs and IM-OODB. Research Report CIS-93-02, CIS Dept, New Jersey Institute of Technology, Newark, NJ.

Mehta, A., Geller, J., Perl, Y., and Neuhold, E.J. (1993). The OODB Path-Method Generator (PMG) Using Precomputed Access Relevance. In *Proc. of the 2nd Int'l Conference on Information and Knowledge Management* (pp. 596–605). Washington DC.

Mehta, A., Geller, J., Perl, Y., and Neuhold, E.J. (1995). The OODB Path-Method Generator (PMG) Using Access Weights and Precomputed Access Relevance. (submitted for journal publication, under revision).

Neuhold, E.J., Perl, Y., Geller, J., and Turau, V. (1989). Separating Structural and Semantic Elements in Object-Oriented Knowledge Bases. In *Proc. Advanced Database System Symposium* (pp. 67–74). Kyoto, Japan.

Neuhold, E.J., Perl, Y., Geller, J., and Turau, V. (1990). A Theoretical Underlying Dual Model for Knowledge Based Systems. In *Proc. First Int. Conf. on Systems Integration* (pp. 96–103). NJ Morristown.

Neuhold, E.J., Perl, Y., Geller, J., and Turau, V. The Dual Model for Object-Oriented Databases. NJIT Technical Report CIS 91–30.

Neuhold, E.J. and Schrefl, M. (1988). Dynamic Derivation of Personalized Views. In *Proc. 14th International Conference on Very Large Databases, VLDB '88* (pp. 183–194). Los Angeles, CA.

Schweizler, B. and Sklar, A. (1961). Associative Functions and Statistical Triangle Inequalities. *Publicationes Mathematicae Debercen*, 8, 169–186.

Sheth, A.P. and Larson, J.A. (1990). Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), 183–236.

Zadeh, L.A. (1965). Fuzzy Sets. *Information and Control*, 8, 228–353.