ON DESIGNING AND QUERYING A MATERIAL INFORMATION SYSTEM

Oliver Radfelder[A], Martin Gogolla[A] and Volker Behr[B]

[A]Universität Bremen, FB 3, Informatik
Postfach 330440, D-28334 Bremen

[B]Verbund Werkstoff Labor Bremen
Fahrenheitstr. 7, D-28359 Bremen

Abstract

We present an information system which has been developed in cooperation with a composite material development laboratorium and other industrial partners. The system enables people interested in this field to get knowledge about project participants and their offers. The main purpose of the development was to let all the involved people participate regardless of their technical skills. One precondition to achieve this goal was the development of an easy-to-learn graphical query language. The implementation has been led by the demand for platform independence and the necessity to involve the project participants in the maintenance of the data.

Keywords: Information System, Database System, Visual Query Language, Distributed System

1 Introduction

The processing of structured data has always been a main subject in computer science. The increasing availability of computers and the theoretical foundations of databases in recent decades were accompanied by a deep change of the users and users' demands. Not only the well educated computer scientists need a tool to maintain and prepare data for others, but increasingly, end-users want to be able to satisfy their information needs on their own. This need can hardly be fulfilled by complex textual query languages. Instead, a lot of different visual methods have been investigated for graphical query languages.

Additionally, an increased amount of information and data has been developed and stored while on the other hand computers are becoming more popular and available. But a vast amount of information typically stored becomes obsolete immediately or shortly after its release. Consequently, the maintenance of data has to be regarded as extraordinarily important. For example, product information systems which are delivered on CD are sometimes out of date instantly after they were released. Client/Server systems can solve this problem efficiently, if the required infrastructure is available and the developers do not have to deal with different platforms.

In our approach, we put special emphasize on three aspects: (1) We developed a simple query language, (2) our mapping of objects to relations differs from others, in the way we store the object schema in relations and (3) the coexistence of structured and semi-structured data in one database.

In our project we had to develop a system which should at first enable all the actors, who are somehow concerned with composite material, to exchange information efficiently and second offer an easy-to-use interface to a composite material database. Composite material consists of two ore more distinct components with different properties to achieve a new superior quality. For example, carbon reinforced polymers are composite material. Actors in our project can even be end-users who are searching for a product made of composite material. As a result we had to deal with the fact, that people, who are not as familiar with composite material as scientists or technicians are, would be using the system. That was the reason why we directed our approaches to the design of a flexible but easy-to-learn graphical query language.

Graphical query languages for information systems have thoroughly been discussed in the literature. [1] was one of the first attempts which analyzed querying in a non-textual way. The basic ideas of that approach are the foundations of many commercial database systems. [2] presents visualization techniques for data represented by graphical symbols and changing according to the underlying database changes. [3] was designed with a special attention to expressiveness and high reusability. Hy+, presented in [4, 5, 6] was based on hypergraphs. OdeView [7] is the graphical front-end of the OODBMS Ode. [8] introduces a concept to build the user-computer interaction on different interaction paradigms. [9] concentrates on recursive queries in the context of visualization in deductive databases. A very general query language developed with the goal to integrate other languages is presented in [10]. Pesto [11] is a general object browser for an OODBMS written in TCL/TK. Further studies discuss visual query languages in the context of semantic data models [12, 13], ER-models [14], graph-oriented models [15] and the ODMG object model [16]. Special applications as hypertext systems [17] or temporal databases [18] have also been influencing the development of graphical database languages. Apart from the approaches mentioned, there are numerous other works on graphical query languages.

Another area, where a lot of work has been done but which is also treated in this paper, concerns the mapping of objects to relations. One line in this field is represented by work about transparent persistence of programming language objects by means of relational databases. SMAC [19], for example, uses DB2 to make C++ objects persistent. The other line is provided by work on the mapping of business objects into relational databases [20] and by the development of special patterns [21]. This second line is much more influenced by the need to provide access to objects in a database manner, which is often realized by mediators [22]. Mediators map objects into the relational schema and offer methods to retrieve objects. Thus, the object schema is spread over (1) the database and (2) the mediator. One of the most promising approaches to provide persistence to objects by means of relational databases together with a query facility, seems to be CORBA [23] in combination with the Corba Query Service [24].

The structure of the rest of this paper is as follows. In Sect. 2 we present the database schema in order to discuss the general requirements presented in Sect. 3. Section 4 deals with the basic ideas of our query language as well as with its application by means of examples. In Sect. 5 we will discuss our implementation with special emphasis on the query language. The paper ends with some concluding remarks.

2 Basic Structure of the Data

For the development of the overall system it was necessary to structure the information to be stored and to develop a conceptual database schema. Since we wanted to work on a uniform schema from the very beginning, we had to integrate two different aspects of our system, namely a part for the material database and a part for the document data. Figure 1 shows the schema as a UML [25] class diagram.

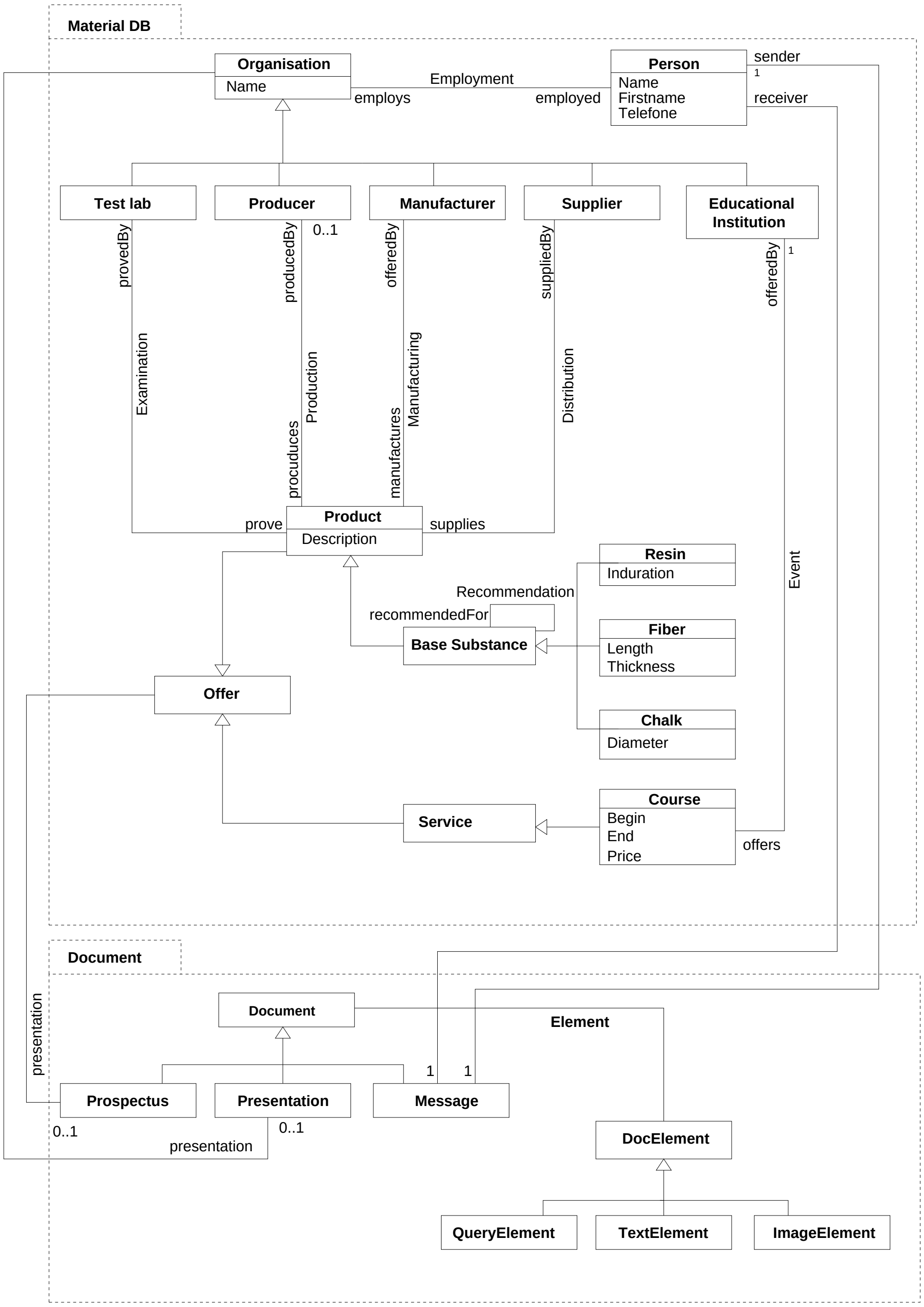


Figure 1: Database Schema in UML Notation

2.1 Material Database

The material database offers information about organizations and their products. Users will be given the opportunity to search for offers from several organizations with special criteria. Organizations in the domain of composite material are offerers of products on one side and offerers of services on the other side. Offers are separated in an inheritance hierarchy which starts from products and services and branches to special classes like fiber in order to allow for specialized attributes.

The classes *Product* and *Producer* are connected by an association, where the producers end is limited to 0 or 1. This expresses that a product is associated to maximally one producer. In a similar way, the association *Event* expresses that a course is offered by exactly one educational institute. The associations *Manufacturing*, *Examination*, and *Distribution* are not limited by multiplicities. This determines that a product may be sold by more than one supplier.

The reflexive association *Recommendation* at *Base Substance* expresses, that there might be special combinations of products which fit together.

2.2 Document Data

In addition to the structure of the material database, the database will consist of a certain amount of less structured data. The expected presentations of offers and organizations are collections of text, pictures and queries. These collections are documents in the database and they are associated with single organizations or offers. These associations will be used later to combine structured results of queries with the less structured information provided by the offerer.

Messages are also modeled as subclasses of documents. They are always associated to at least two instances of persons; one as the sender and one or more as the receiver(s).

2.3 Requirements

The resulting system has to deal with a lot of different requirements. Since the system has been developed to help people to communicate with other people from various domains we had to assume that there might be extremely different environments. This is valid for the technical equipments as well as for the skills of users with respect to computer use and knowledge about composite materials. The different groups producer, supplier, manufacturer , and educational institute are to be joined to communicate. There are three main aspects following from these assumptions:

Different platforms: The application on which the end-users will work must operate on as many platforms as possible. The Macintosh of the manufacturer has to be supported as well as the producers Windows PC and the Unix environment of the university.

Different knowledge: The need for complex information by a highly competent technician has to be fulfilled. But the demands of a customer, who does not know much about composite material and who wants to find precise information in a short time and with little effort, must be satisfied as well.

Maintenance of the data: Structured data must be maintained by the individual actors as far as possible. A system like ours can only be useful, if as much organizations as possible are involved. But beyond a certain level, the input of data cannot be handled by only one organization.

We addressed these issues in the following manner:

- The first subject is solved by adopting the client/server principle. The server runs (1) a platform specific database system and (2) a special application realizing the complete communication with the individual client. On the client side further applications provide interfaces to formulate queries, to send messages or to present an organization or a company.
- The demand for gathering information on a single subject with differing complexity is fulfilled by the users' possibility to define the complexity of a query result on their own. Instead of giving the users predefined queries, they can define them by themselves.
- Each participant is classified in at least one of the subclasses of the class *Organization*. Thus he has the possibility in his application to insert structured data into the database. Suppliers and producers have to be able to insert special products like fiber with their special attributes into the database.

These issues are accompanied by the requirement to keep the use of the client application as easy as possible. We tried to reach that goal by providing the users a complete visual environment to let them compose documents. The user has a free workspace, representing a sheet of paper on which he places document elements. Proceeding this way, *TextElements*, *ImageElements* and *QueryElements* are combined to *Documents*. The *QueryElement* has special significance. Since we do not want the user to deal with a textual notation to formulate a query, one of the central elements of the system is the graphical query language.

3 Graphical Queries

The query language is used whenever information is to be retrieved. This refers especially to the system's core, namely the material database.

The basis of a graphical query is the database schema. Having in mind graphical specification languages like EER [26] and UML, the schema is transformed into a form which allows to specify queries by navigation along specializations, associations and from classes to attributes. This process starts with the selection of one entry class. Entry classes are ordinary classes which are not allowed to have superclasses. The general process of searching information could be described as follows: Choose an entry class from the database schema; specialize it to the interesting context and extend it by the interesting properties, i.e. attributes.

Our query language uses the central properties of object-oriented systems. Instances of classes have property ends, these are determined by the classes. An instance of a class has for each attribute at least one value which is publicly known by the attribute name. For each association the class participates in, an instance has a set of references to instances of the opposite class. These are known by role names. Instances of subclasses possess all properties of their superclass.

For example, an instance of the class *Organization* has the properties *Name*, *employs* and *presentation* while an instance of the class *Supplier* additionally has the properties *supplies* yielding a set of *Products*.

3.1 Formulating a Query

In order to define a query, the entry classes are placed next to each other so that the user can select one with the mouse. Starting from that class, the result set can be restricted by specialization. There are also other means for doing such a restriction. Classes are presented as rectangles with a triangular mark in the upper left corner while attributes and roles are rectangles without any mark (see Fig. 2). This distinction is necessary, because the

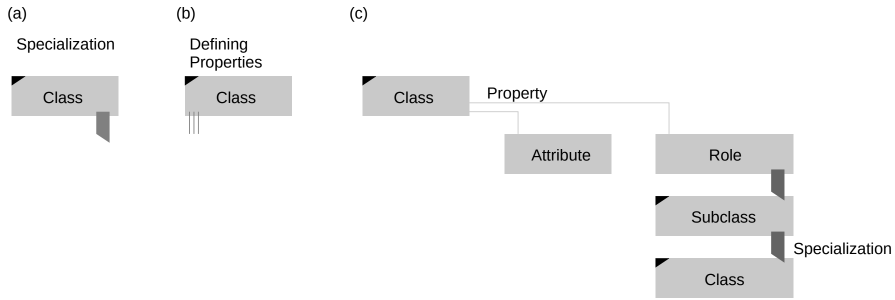


Figure 2: Concepts of the Query Notation

user can click on the class symbol to restrict the result set or to extend the property set of the individual objects in the result set. When the mouse pointer is on the right half of the class symbol a rhombus shaped symbol occurs at the bottom of the class which indicates a specialization(Fig. 2(a)). When the mouse pointer is on the class symbol's left half the property symbol occurs as a symbolic attribute link (Fig. 2(b)).

3.2 Specialization

When the user presses the mouse button while the specialization symbol appears, the system switches to specialization mode. While being in this state, all possible classes for extension are shown underneath the current class. One of them can be chosen and after that a new class symbol is shown for the selected class. In the next state, that one is the new current class.

This procedure expresses that the result set from now on does not contain all the instances of the general class but only those which are also instances of the more special one.

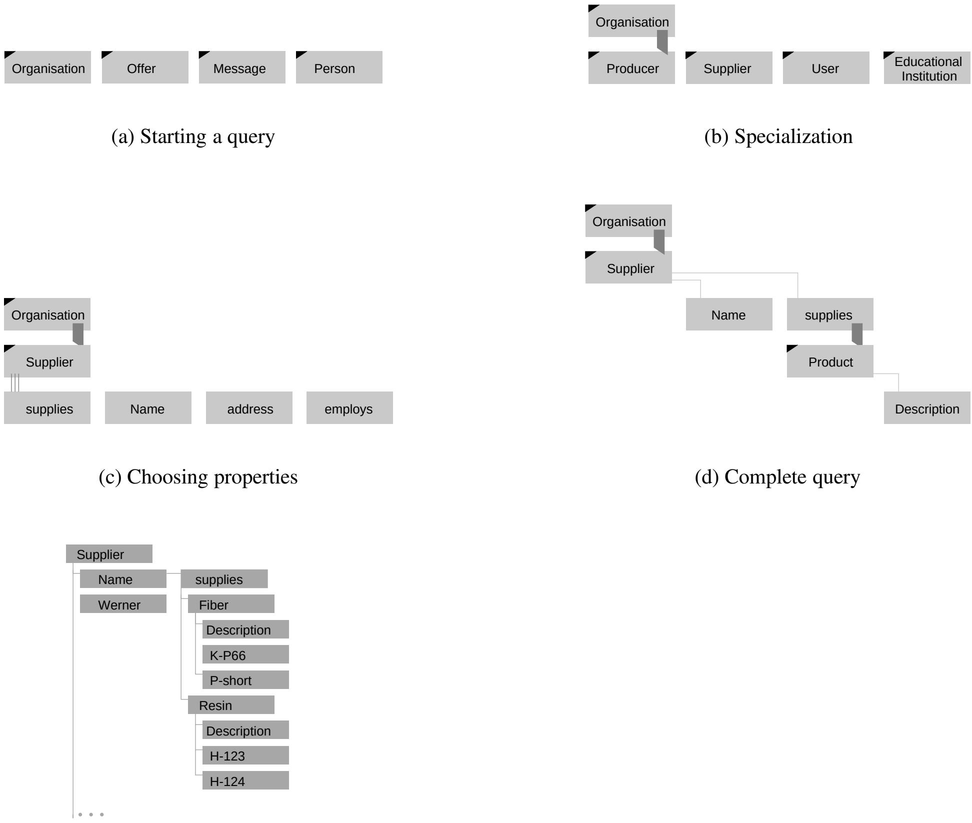
3.3 Selecting Properties

When the user presses the mouse, while the property symbol is visible, the system switches to the property state and shows the direct and the indirect attributes underneath the current class symbol. Additionally, the roles of associated classes are shown as rectangles. Attributes and association roles together are the *properties* of the class. One of these properties can be chosen. Selected properties are presented on the underneath right of the class symbol and they are connected with the class by a line. If the role of an associated class has been chosen, the role symbol of the associated class is presented underneath the current class (Fig. 2(c)).

This procedure extends the state of the result set elements: Attributes extend the state by values while associations extend the state by a set of instances playing a role in the respective context.

3.4 Example

We want to show the procedure of defining a query by means of an example: Imagine a user is looking for suppliers. The result should contain the suppliers with their names together with the offered products. Among the entry classes *Organization*, *Offer*, *Message* and *Person* (Fig. 3(a)) the user chooses the first one. When specializing it to *Supplier* (Fig. 3(b)) he restricts the result set to all the *Organizations* which are registered as *Suppliers*. The properties *Name* and *supplies* are selected one by one (figure 3(c). Eventually the *Product*



(a) Starting a query

(b) Specialization

(c) Choosing properties

(d) Complete query

(e) Result Set

Figure 3: Interactive Developing of a Query

from the association *Production* is extended by the attribute *Description* (Fig. 3(d).

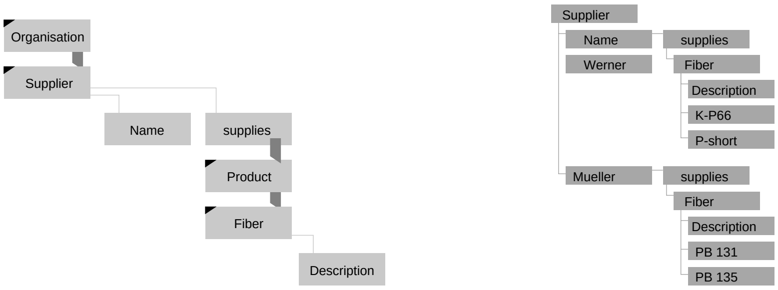
The result set consists of all the suppliers with their names and the products they supply with their names (Fig. 3(e)).

3.5 Further Specialization Of a Query Result

If the result set was extended by means of roles the class which appears underneath the role can be treated as if it were an entry class. An example for extending a class by a property was already used in Fig. 3(d). But it is also possible to specialize the resulting class if it has subclasses. The effect is a result set containing only those elements which have at least one link to an instance of the specialized class.

3.6 Example

We are only searching for suppliers of fibers. The result should be the suppliers with their names and the offered *fibers* with their *Description*. To achieve this, we extend the query of Fig. 3 by specializing *Product* to *Fiber*. Now, the attribute *Description* is a property of the class *Fiber*.



(a) Specialization of Roles

(b) Result Set

Figure 4: A Query with Specialization of Associated Classes

The result set now consists of the *fiber suppliers* only, with their *Names* and the *Fibers* they supply together with their *Description* (Fig. 4(b)).

4 Implementation

In the previous section we explained the query language by means of an example. But our system is parameterized by the database schema, which means it is also applicable to another but similar application. In addition to a relational database, the total system consists of three main applications: the *client* for the users, the *server*, and the *schema generator* (Fig. **??**). The architecture is led by the idea of the *three-tier architecture* [27]. The client application communicates only with the server application and only that one communicates with the relational database. Both applications are written completely in Java [28]. Consequently the client application can run in all environments which provide a virtual Java machine. This means that different implementations do not have to be developed. Java has been chosen for the server application due to two main aspects: (1) the Server application has also to be platform independent and (2) Java offers a well-defined interface to relational databases. A major advantage of this approach is the exchangeability of both server platform and database system.

Client: The client needs the structure of the schema in Fig. 1 for displaying the query elements. When the user specializes a class or chooses a property, the system must offer applicable elements. Since the server knows the schema, the client does not have to know it by itself. The server communicates it to the client.

Server: The server has three main functions: (1) to communicate the schema to the clients, (2) to transform object-oriented queries into relational ones and (3) to transform relational results back into object-oriented form and send them back to the client. To fulfill these requirements the server has to know the database schema. One more job for the server is to send *Documents* on requests to the clients.

Schema generator: Based on an abstract description of the object-oriented schema, the transformation into a relational schema is automated due to two reasons. The first one is the loss of semantically relevant information during the process of transformation from UML to the relational schema, which are necessary for the graphical queries. This metadata is also stored in the database and it is much easier to hold them consistent, if the transformation runs automatically. The second reason is the possibility to modify the schema during the development process.

Starting with the schema generator, we want to show the creation of the database and the cooperation of the single components.

Schema construction: The schema generator reads a list of commands from a file and creates relations according to a simple strategy. The commands used to describe the schema are:

```
class(name),attribute(class,name,type),sub(class1,class2),
association(name,class1,class2,role1,role2)
```

A selected part of the schema in Fig. 1 which only consists of the classes *Organization*, *Supplier*, *Product* and *Fiber* and the association *Distribution* would be constructed as follows:

```
class(Organization)
attribute(Organization,OName,varchar(20))
class(Supplier)
sub(Supplier,Organization)                -- specialization
class(Product)
attribute(Product,Description,varchar(20))
class(Fiber)
sub(Fiber,Product)
association(Distribution,Product,Supplier,
            supplies,suppliedBy)          -- association
entry(Organization)                       -- entry class
```

The schema generator creates a set of relations by applying the following rules: Each class becomes a relation which holds the object identity. Each attribute becomes a relation with the object identity and the value. An association results in a relation, which links two object identities. In addition to these relations, the schema generator creates one relation *schema* for semantic information. The relation *schema* holds the class names, the association names the class/subclass relations, the associations of attributes and associations to the classes, and the names of the entry classes.

Start of the server: When the server starts, the first thing to do, is to query the relation *schema* and to transform it into an object-oriented form.

Start of the client: When the client application starts, the first thing to do, is to contact the server and to get the database schema. During the running session, the user defines queries as it is described in the previous section. The client sends theses queries to the server and gets the results sent back.

Transformation: When the server gets a query as an object, it has to be transformed into an SQL statement. The query in Fig. 4(a) would be transformed into

```
SELECT OName.name, Description.name,
FROM   Supplier,Organization_OName,OName,Fiber,
       PD,Description,PD,SP
WHERE  Supplier.oid=Organization_OName.oid1 AND
       OName.oid=Organization_OName=oid2 AND
       Fiber.oid=PB.oid1 AND Description.oid=PB.oid2 AND
       Supplier.oid=LP.oid1 AND Fiber.oid=LP.oid2
```

5 Conclusion

We developed a system which can be used by companies and people from the domain of composite material to query a material database according to their special needs. They do not have to learn a complicated textual language. The embedding in a document framework connects the areas product presentation, information exchange and database queries.

Our future work will be dedicated to the question, how to extend the query language in order to define more complex queries without neglecting the desired clarity and simplicity. We will, for example, analyze how to visualize complex constraints.

References

[1] M. M. Zloof. Query-by-example: A database language. *IBM System Journal*, 16(4):324–343, 1977.

[2] Dennis Fogg. Lessons from a 'living in a database' graphical query interface. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 14(2):100–106, 1984.

[3] M. Angelaccio, T. Catarci, and G. Santucci. QBD*: A graphical query language with recursion. In *Proceedings of the Third International Conference on Human-Computer Interaction*, volume 2 of *Designing and Using Human-Computer Interfaces and Knowledge Based Systems; Graphics*, pages 384–391, 1989.

[4] Mariano P. Consens and Alberto O. Mendelzon. Hy^{+}: a hygraph-based query and visualization system. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(2):511–516, June 1993.

[5] M. P. Consens, I. F. Cruz, and A. O. Mendelzon. Visualizing queries and querying visualizations. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 21(1):39–46, March 1992.

[6] M. Consens and A. Mendelzon. Graphlog: a visual formalism of real life recursion. In *Proc. 9th ACM Symp. on Principles of Database Systems*, pages 404–416, 1990.

[7] R. Agrawal, N. H. Gehani, and J. Srinivasan. OdeView. the graphical interface to Ode. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 19(2):34–43, June 1990.

[8] T. Catarci, S. K. Chang, M. F. Costabile, S. Levialdi, and G. Santucci. A multi-paradigmatic visual environment for adaptive access to databases. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems – Adjunct Proceedings*, Short Papers (Talks): Visual Languages and Virtual Reality, pages 7–8, 1993.

[9] F. Staes and L. Tarantino. OOQBE*: An intuitive graphical query language with recursion. In *Proceedings of the Fifth International Conference on Human-Computer Interaction*, volume 2 of *III. Media*, pages 603–608, 1993.

[10] K. Vadaparty, Y. A. Aslandogan, and G. Özsoyoğlu. Towards a unified visual database access. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(2):357–366, June 1993.

[11] Michael J. Carey, Laura M. Haas, Vivekananda Maganty, and John H. Williams. PESTO : An integrated query/browser for object databases. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, pages 203–214, Mumbai (Bombay), India, 3–6 September 1996. Morgan Kaufmann.

[12] Kenneth J. Goldman, Sally A. Goldman, Paris C. Kanellakis, and Stanley B. Zdonik. ISIS: Interface for a semantic information system. In Shamkant B. Navathe, editor,

Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, pages 328–342, Austin, Texas, 28–31 May 1985.

[13] Manoochehr Azmoodeh and Hongbo Du. GQL, a graphical query language for semantic databases. In M. Rafanelli, J. C. Klensin, and P. Svensson, editors, *Proceedings of the 4th International Working Conference on Statistical and Scientific Database Management*, volume 339 of *LNCS*, pages 259–277, Berlin, June 1989. Springer.

[14] R. Elmasri and J. Larson. A graphical query facility for ER databases. In *Proc. 4th Int. Conf. Entity-Relationship Approach*, Chicago, IL, 1985.

[15] J. C. Kwak and S. Moon. Object query diagram: An extended query graph for object-oriented databases. In Ephraim P. Glinert and Kai A. Olsen, editors, *Proceedings of the 1993 IEEE Symposium on Visual Languages*, pages 44–48, Bergen, Norway, August 1993. IEEE Computer Society Press.

[16] Manoj Chavda and Peter T. Wood. Towards an ODMG-compliant visual object query language. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 456–465, 1997.

[17] Richard Gary Epstein. A graphical query language for hypertext database systems. In *Proceedings of Graphics Interface '89*, pages 47–54, June 1989.

[18] V. Kouramajian and M. Gertz. A Visual Query Editor for Temporal Databases. In M. P. Papazoglou, editor, *Proceedings of the OOER'95, 14th International Object-Oriented and Entity-Relationship Modelling Conference*, volume 1021 of *Lecture Notes in Computer Science*, pages 388–399, Gold Coast, Australia, December 1995. Springer-Verlag.

[19] B. Reinwald, S. Dessloch, M. Carey, T. Lehman, and H. Pirahesh. Making real data persistent: Experiences with SMRC. In Michael Atkinson, D Maier, and Verónique Benzaken, editors, *Proc. of the Int. Workshop on Persistent Object Systems (Tarascon, France, 1994)*, Workshops in Computing, pages 202–216, Berlin, 1995. Springer-Verlag.

[20] Peter Coad and Edward Yourdon. *Object-Oriented Design*. Prentice Hall, Eaglewood Cliffs, New Jersey, 1 edition, 1991.

[21] Kyle Brown and Bruce G. Whitenack. Crossing chasm. In *Pattern Languages of Program Design, PLoP '96*, Champain Urbana, USA, September 1996.

[22] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, March 1992.

[23] Object Management Group. The Common Object Request Broker: Architecture and specification. Accessible at `ftp://omg.org/pub/CORBA`, December 1993.

[24] CORBAServices: Common object service specification: Query service specification, November 1997.

[25] G. Booch, J. Rumbaugh, and I. Jacobson, editors. *Unified Modeling Language (UML) version 1.1, `http://www.rational.com`*. 1997.

[26] Martin Gogolla. *An Extended Entity-Relationship Model – Fundamentals and Pragmatics*, volume 767 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.

[27] Barry Cornelius. Using CORBA and JDBC to produce three-tier systems. *ACM SIGPLAN Notices*, 33(4):44–52, April 1998.

[28] Ken Arnold and James Gosling. *The Java Programming Language*. The Java Series. Addison-Wesley, Reading, MA, second edition, 1998.