Query Classes*

Martin Staudt, Matthias Jarke, Manfred A. Jeusfeld, Hans W. Nissen

Informatik V, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Germany

Abstract. Deductive object-oriented databases advocate the advantage of combining object-oriented and deductive paradigms into a single data model. Certainly, the query language in such a data model has to reflect the amalgamation because it works as the interface to the user and/or application program. This paper proposes a language to formulate queries as classes related to the schema classes and constrained by an associative membership condition. Answers are then regarded as their instances. The interpretation is based on a deductive database view of queries. Generic query classes are introduced with a simple parameter substitution construct. The syntactic separation of structural and associative conditions opens the way to semantic query optimization: subsumption between the structural parts of queries can be decided efficiently.

1 Introduction

The query language of a database determines which and how information can be retrieved from the database. Users generally demand a query language to be declarative and efficient (i.e. easy to optimize). In the case of (deductive) object-oriented databases the query language must be closely related to the data model. Ideally, queries would be just class descriptions where instances are not inserted by the user but computed by the query evaluator.

For this purpose, we propose an amalgamation of paradigms from object-oriented databases (*query by class*), deductive databases (*query by rule*), and knowledge representation languages (*query by concept*). The next section reviews related work on each of the aspects. Section 3 contains the exposition of our query language, discusses the interaction between the paradigms and reports shortly on the influence of each of the three faces on the implementation of query classes in the deductive object base management system ConceptBase. Finally, section 4 briefly presents fields of applications where our approach has been successfully used.

* This work was supported in part by the Commission of the European Communities under ESPRIT Basic Research Action 6810 (Compulog 2) and by the Ministry of Science and Research of Nordrhein-Westfalen.

2 Three Views of Queries

The basic understanding of what constitutes a query strongly influences the design of a query language and a query processing concept, and thus which of the desirable features it can offer. In this section, we review query models followed in the areas of object-oriented databases, deductive databases, and AI concept languages which understand queries, respectively, as

- classes of answer objects manipulated by methods,
- deduction rules with parameterized answer attributes, or
- concepts with automatic classification through subsumption relationships.

It turns out that each of these approaches basically follows the idea to make queries the same kind of thing it also handles otherwise. Each is particularly suitable for certain application areas of query handling. Moreover, there are specific reasons for this suitability which lie in the very nature of the approaches. In the last subsection, we draw together these observations to come up with a list of requirements our query model has to satisfy.

Query by Class: The Object-Oriented Approach. Object-oriented databases (OODB) combine the paradigms of object identity, class membership, inheritance, and methods from object-oriented programming languages with the database functionalities of persistence, concurrency, security, and declarative querying.

Since classes are the mechanism for managing sets of objects it is natural to use them also for describing query results, seen as sets of answer objects. A general advantage of queries as classes is that answers can be managed the same way as the objects in the OODB. Methods for processing answer objects can be attached to query classes. The answer objects can be organized in generalization/aggregation hierarchies and reuse methods of OODB classes. Moreover, the integration of methods in OODB suggests an implementation approach to query evaluation/view maintenance based on triggers attached to specific object/query class subtypes.

Query by Rule: The Deductive Approach. A deductive database consists of a finite set of facts (extensional database), and a finite set of deductive rules (intensional database). We restrict ourselves to rules where negation in the conclusion literal is disallowed and negated literals in the rule body must obey the stratification condition (Datalog$^\neg$). With these assumptions the preferred interpretation is a certain minimal Herbrand model, the so-called perfect model. It can be computed using a fixpoint operator [10]. Queries in deductive databases look rather simple compared to object-oriented databases but they have some advantages. The membership condition is defined by the rules which have a matching conclusion literal. Thus, it is very easy to formulate parameterized versions of a query just by replacing one or more variables by a constant.

Implementation techniques for query optimization have been thoroughly investigated yielding algorithms for recursion optimization (magic sets [3], query/subquery approach [27]), and deductive integrity enforcement (e.g. [8], [21]).

View maintenance algorithms (storing the result of a query and keeping it up-to-date with the database) have been less studied but are rather simple generalizations of integrity enforcement. Some work has been done on updating views in deductive databases, esp. by abductive reasoning [17].

Query by Concept: A Type Inferencing Approach. Starting with KL-One [7], concept languages in artificial intelligence have pursued the idea to define knowledge bases as type lattices of so-called concepts. Because of the restricted usage of logical connectives a dedicated syntax for concept expressions has been developed: the axioms (schema) are written as $C \sqsubseteq E$ (necessary condition) or $C = E$ (necessary and sufficient condition) where E is constructed from other defined concepts and binary predicates expressing relationships between concepts.

The main purpose is to relate concepts to each other, e.g. to derive the *subsumption* between two concepts C and D. One should note that concept languages make statements for all possible models of the theory whereas database languages usually only consider the current database state as their model.

Systems such as CANDIDE [2] and CLASSIC [5] use concept descriptions as a flexible query language for databases. Queries are just considered as a new concept which is positioned in the concept lattice by subsumption algorithms. Answers to the query are all instances of all subconcepts and some instances of the direct super concepts (satisfying the additional constraints of the new concept).

The benefit of concept lattices is that the search space of a query (i.e., a new concept) is greatly reduced. The exact placement of a query into the lattice opens the ability for intensional query answering: instead of enumerating all instances of the query the system answers with the names of the subconcepts of the query (slightly more complicated for the contribution of the direct super concepts). On the other hand, if the extension of a super concept is materialized then a query subsumed by such a concept may reuse the extensions as a fine range.

Summary of Requirements. Our short review has shown that each approach has advantages supporting certain functionalities of a desired query system. The question addressed in our approach is how to integrate the three approaches. The following basic observations underly our solution, the query class:

- Queries as classes and queries as concepts offer the same object-oriented structure for questions and answers; therefore, this will be our approach.
- To integrate objects and rules, we need an object model with a logic-based semantics that allows a two-way transition between both. Our model of choice, Telos (see section 3), goes even further by offering an object model with a standard deductive database semantics, it has a precise translation into facts, rules, and integrity constraints of a deductive database.
- Concept language subsumption algorithms can be expected to work safely and effectively only for a sublanguage of the query language offered, e.g., by a full deductive database. A way has therefore to be found to offer both aspects separately. Complete subsumption is only done on the sublanguage

(subsequently called type system). As in programming languages and in constraint logic programming, the communication between both formalisms is one-way, that is, the type system influences query processing but not vice versa.

3 Query Classes in O-Telos

The O-Telos data model [14] is a variant of the knowledge representation language Telos [20]. In the following subsections we introduce the basic properties of the data model and show how queries in each of their roles fit into this framework. With each role we explain briefly its influence on the implementation of query classes in ConceptBase [13], a deductive object base management system. A detailed description of the implementation is given in [25].

The syntax of query classes is a class definition with superclasses, attributes, and a membership condition. The interpretation is based on a deductive rule containing predicates for each of the clauses in the class definition. Integration with concept languages is viewed as the extraction of the structurel part of a query class, i.e., the portion of the query which is representable as a concept expression.

3.1 Representing Objects and Formulas

We define deductive object bases as deductive databases with builtin axioms formalizing the three object-oriented abstraction dimensions of classification, aggregation, and generalization. More specifically, a deductive object base is a triple $DOB = (OB, R, IC)$ where OB is the extensional object base, R and IC contain deduction rules and integrity constraints. The axioms are represented as predefined formulas in R and IC. Let ID and LAB be sets of identifiers, and labels resp. An **extensional O-Telos object base** is a finite set

$$OB \subseteq \{P(o, x, l, y) \mid o, x, y \in ID, l \in LAB\}.$$

The elements of OB are called **objects** with identifier o, source and destination components x and y and name l. Objects of the form $P(o, o, l, o)$ are called *individuals*, $P(o, x, in, c)$ describes an instantiation relationship whereas specializations are of the form $P(o, c, isa, d)$. All other objects $P(o, x, l, y)$ are called attributes. This arrangement of an extensional object base permits an intuitive representation as a semantic net: individuals as nodes, all other objects as links. A third view on deductive object bases results in a frame-based notation of objects which only relies on object labels. Around the name l of an object o we group the names of all other objects which have o as source component.

The running example in this paper is based on the following scenario from a medical database: *Patients are persons, suffer from diseases and take drugs which have an effect on diseases.* The frame notation of the objects `Patient` and `Drug` is e.g. the following:

```
Patient isA Person with          Drug with
  attribute                        attribute
    suffers:Disease;                 against:Disease
    takes:Drug                     end
end
```

We use the following literals when defining the complete axiomatization but also when specifying deductive rules and integrity constraints: `(x in y)` denotes an instantiation relationship between `x` and `y`, `(c isA d)` an specialization relationship between `c` and `d`, and `(x m y)` indicates that there exists an attribute of category `m` with source `x` and destination `y`.

O-Telos requires a set of 35 axioms and axiom schemata implemented as builtin deductive rules or integrity constraints of the deductive object base [14]. Here, we refer to a few important ones. Like most OODB and semantic data models, O-Telos requires that instances of an object are also instances of its superclasses. Another important property of the language is realized by the attribute typing axiom: instantiation relationships between objects imply instantiation relationships between their source resp. destination components. A similar restriction holds for specialization links and allows a consistent way of refining attributes of classes by their subclasses. Thus, although attributes of an object are not explicitly inherited by subclasses they can nevertheless be instantiated by their instances.

Instantiation links between objects in OB constitute different layers within an object base. O-Telos does not restrict the number of layers; classes can be instances of other classes (metaclasses), these can be instances of metameta-classes, and so on. In addition to the O-Telos axioms the sets R and IC contain application specific deduction rules and integrity constraints which are specified in a many-sorted first order language. Variable quantifications range over classes and are interpreted as instantiation relationships.

As an example a deduction rule can be defined that deduces for a given patient doctors who are suited to give him medical treatment. The constant `this` refers to the instances of the class which carries the rule. Similarly one may impose an integrity constraint on `Patient` that all instances actually must have a filler for the `suffers` attribute. As a consequence of the attribute typing axiom, the variable ranges can be used to determine type errors inside a formula.

```
Patient isA Person with
  attribute
    suited_doc:Doctor
  rule
    suited_rule:$ forall p/Doctor,d/Disease
                  (this suffers d) and (p specialist d)
                  ==> (this suited_doc p) $
  constraint
    mustsuffer:$ exists d/Disease (this suffers d) $
end
```

3.2 Queries as Classes

The query language CBQL [24] represents queries as classes whose instances are the answer objects to the query. These query classes are themselves instances of the predefined object `QueryClass` and contain necessary and sufficient membership conditions for their instances (=answers). These conditions can be used to check whether a given object is an instance of a query class or not. On the other hand, they can be used to compute the set of answer objects. Query classes have superclasses to which they are connected by an isa-link. These superclasses restrict the set of possible answers to their common instances.

Two different kinds of query class attributes can be distinguished. *Retrieved attributes* are already defined for one of the superclasses of the query class. An explicit specification of such an attribute in a query class description means that answer instances are given back with values for this attribute, similar to relational projection. In addition a necessary condition for the instantiation by an admissible value is included. The attributes of superclasses can also be refined, i.e. the target class is substituted by a subclass which results in an additional value restriction. If we assume a subclass `Antibiotics` of `Drug` the query class

```
QueryClass MaleOldAntibioticsPatient isA MalePatient,OldPatient with
  attribute
    takes:Antibiotics
end
```

has those male and old patients as instances who take antibiotics. In addition the concrete drugs are included in the answer description.

Computed attributes have values derived in the query evaluation process. Neither the extensional object base contains this relationship between the answer instance and the attribute value, nor is it inferable by deduction rules. For prescribing how to deduce these new relationships by analogy to deduction rules and integrity constraints, many-sorted first order logic expressions are admissible as building elements for query classes.

```
QueryClass WrongDrugPatient isA Patient with
  attribute, parameter
    wrong:Drug
  constraint
    wrongconstr:$ (this takes wrong) and
                  not exists d/Disease ( (this suffers d) and
                                         (wrong against d) ) $
end
```

As in section 3.1 `this` refers to the answer instances of `WrongDrugPatient`, namely all patients who take a drug which is against a disease they don't suffer from. The computed attribute `wrong` is identified with the variable of the same name within the formula. All deduced wrong drugs are part of the answer.

The logical expression in query classes descriptions can also contain arbitrary other constraints for the answer instances which would then work like the selection operation in relational databases.

In order to avoid the frequent reformulation of similar more specialized queries, attributes of query classes can be declared as *parameters*. Substitution of a concrete value for such an attribute or specialization of its target class by a subclass leads to a subclass of the original query which implies a subset relationship of the answer sets. For the parameter `wrong` the expressions

```
WrongDrugPatient(Aspirin/wrong)
WrongDrugPatient(wrong:Antibiotics)
```

denote two derived query classes which restrict the answer instances of `WrongDrugPatient` to those patients who take `Aspirin` resp. a drug of the class `Antibiotics` as wrong drug. For example, `WrongDrugPatient(Aspirin/wrong)` is a shorthand for the (non-parameterized) query class

```
QueryClass WrongAspirinPatient isA Patient with
  constraint
    Aspiconstr:$ (this takes Aspirin) and
                 not exists d/Disease ( (this suffers d) and
                                      (Aspirin against d) ) $
end
```

Expressions denoting (derived) query classes are allowed to occur within the definitions of other query classes and objects and are managed as full-fledged objects, too. Most of the compilation steps applied to normal classes are also applied to them, e.g. type checking of attributes. ConceptBase stores queries as any other class. When methods are defined for schema classes then all subclasses including query classes inherit these methods.

3.3 Queries as Rules

In deductive databases queries are represented as intensional relations derived by a set of rules. We have shown above that O-Telos object bases are just special cases of deductive databases. Obviously it should be possible to extend this view to query classes.

The definition of a query class `Q` induces a so called *query literal* $Q(x, x_1, ..., x_n)$ whose arity depends on the number of attributes and parameters of `Q`. The first argument x of Q stands for the answer object identifiers. Query classes are transformed to rules[2] concluding their corresponding query literals. By convention the object identifier of an individual with name `i` is written as `#i`.

- For each superclass `C` of `Q` the body of this rule contains a literal $In(i, \#C)$ where $\#C$ is the object identifier of the class `C` and i is a new variable replacing `this` in the query class description.
- Attributes `a:C` of the first type (defined for a superclass or refined) result in a conjunction $In(v, \#C) \wedge A(i, a, v)$ where v is a new variable.
- Attributes `a:C` of the second type just require $In(v, \#C)$ where v is a new variable.
- The logical formula describing the derivation of attribute values and other additional restrictions is transformed straightforward by substituting the newly introduced variables v and i for their symbolic counterparts `a` and `this`, replacing all occuring object names by object identifiers and resolving typed quantifications.

[2] We use a predicate-like notation of the literals to formulate deduction rules: In(x,y) for (x in y) and A(x,m,y) for (x m y).

– As a last step all generated expressions have to be linked by conjunctions. The new variables become arguments of the query literal.

Following these steps the example query class `WrongDrugPatient` is transformed to the following rule:

$$\begin{aligned}
&\forall i, v\ In(i, \#Patient) \wedge In(v, \#Drug) \wedge \\
&\quad \wedge A(i, takes, v) \wedge \neg\exists d\ In(d, \#Disease) \\
&\quad \wedge A(i, suffers, d) \wedge A(v, against, d) \\
&\quad \Rightarrow WrongDrugPatient(i, v)
\end{aligned}$$

Each query class yields exactly one rule concluding its corresponding query literal. Hence with closed world assumption the rule body provides a necessary and sufficient characterization of class membership.

The deduction rule form of queries not only provides a clean semantics but also a whole array of algorithms for evaluation and optimization known from the deductive database area [10]. In ConceptBase, the deductive rules are rewritten with the magic set method [3]. The main advantage of our query language is that it completely falls into Datalog$^{\neg}$. Thus, virtually any evaluation method from this area is applicable. Optimization of rules and integrity constraints benefits from the predefined axioms of the object model by elimination of redundant predicates [15]. This also enhances the maintenance of materialized views, i.e. stored answers to a query [16]. Recently, compilation of deductive rules into algebra expressions has been added to ConceptBase. It profits from the fact that the object model provides an access predicate for each class attribute.

3.4 Queries as Concepts

In programming languages objects (variables, functions, etc.) usually have types. Type constructors for tuples ([]), sets ({ }), lists etc. allow to handle objects of arbitrarily complex structure. In the same way most object-oriented databases distinguish between objects and values on the one hand and classes and types on the other hand. Objects have an unique identity and an assigned value of a certain type. Classes are object containers which collect objects of the same type (sometimes called member type).

O-Telos only provides the diction of classes where membership is constrained by the builtin axioms and by user-defined constraints. Nevertheless we can extract type information from classes. A class definition is divided into a 'clean' part, i.e. its type, and a 'dirty' part containing all aspects for which the type system is not powerful enough. The example class

```
Patient isA Person with
  attribute
    suffers:Disease;
    takes:Drug
end
```

carries the information that patients have attributes **takes** and **suffers**, each with certain value restrictions. In addition, patients have attributes (e.g. **name** and **address**) defined for the superclass **Person**. Under consideration that **takes** and **suffers** can have multiple fillers, a (member) type of **Patient** could be the following:

$$[suffers : \{DiseaseT\}, takes : \{DrugT\}, name : string, address : AddressT]$$

where AT is a shorthand for the type of a class **A**. The relationship between types in OODB's and concept languages are discussed in [4]. It turns out that the latter allow more restrictions on objects than the common plain type languages.

Following the syntax used in the concept language literature for our patient example, we get a member type $PatientT$ for **Patient**.

$$PatientT \sqsubseteq \forall suffers.DiseaseT \sqcap \forall takes.DrugT \sqcap PersonT$$

The type information can be regarded as being a layer orthogonal to the object base definition presented in 3.1.

The 'type view' of deductive object bases introduced so far only used the $\forall$- and $\sqcap$- constructor for types and represented necessary conditions in the sense that e.g. every patient is also a person and that all attribute fillers for the **takes** attribute are drugs. Both conditions are guaranteed by O-Telos axioms. The type expression above doesn't cover neither the constraint nor the deductive rule specified for **Patient** in 3.1. These parts of the class definiton are referred to as the dirty part of a class definition.

On the other hand type restrictions may also be interpreted as sufficient membership conditions for their instances which in concept languages leads to the distinction between (defined) *concepts* and *primitive concepts* (the latter with only necessary conditions) [6]. Thus we can say that ordinary classes in an O-Telos object base have primitive concepts as their types. Primitive concepts (as e.g. $PatientT$) are directly subordinated under their corresponding type expression in the lattice which exactly represents the interpretation as necessary conditions. Defined concepts are related to their defining expression by equality.

There is an obvious correspondence between query classes and defined concepts since both state sufficient membership conditions. As with ordinary classes query classes often have 'dirty' parts not expressible by the type language. It can be distinguished between query classes that actually use only type language conform constructs – with a defined concept as their type – and those who have additional parts and hence have just a primitive concept as their type.

The goal of a separation between a type layer and the usual object layer for a deductive object base is to shift several kinds of inferences to the type layer which promises more efficient computations due to the lower complexity of the underlying language. The concept language community has examined a broad palette of type languages and offers efficient algorithms for several of them [12]. If one of these languages is chosen as type language for a deductive object base with corresponding syntactic object counterparts for type language constructs (as e.g., **atleastn** and **atmostn** above) we can apply these well understood inference algorithms at the type level to infer new information at the object level. The more complete the type language is, the more conditions of classes and query classes can be expressed in concept expressions and used for these inferences. A simple example is to support a type language extension with cardinality restrictions $\geq n\, r$ and $\leq n\, r$ for fillers of an attribute r by additional new built-in attribute categories `atmostn` and `atleastn`.

One example how inferences at the type layer may affect operations on the object layer is the computation of subsumption relationships between queries in order to avoid superfluous recomputations and reuse queries with stored answers. The determination of such relationships between queries and views that are materialized promises increasing efficiency. An efficient calculus for deciding subsumption in this case is presented in [9]. The following simple query classes shall demonstrate this.

```
QueryClass DrugPatient isA Patient with
  attribute
    takes:Drug
end

QueryClass AntibioticsPatient isA Patient with
  attribute
    takes:Antibiotics
  constraint
    appconstr:$ (this suffers Appendicitis) $
end
```

`DrugPatient` contains all those patients that take atleast one drug, `AntibioticsPatient` requires a value restriction for `takes` to drugs of the class `Antibiotics`, at least one filler for `takes` and `Appendicitis` as filler for `suffers`. Note that the specified value filler[3] `Appendicitis` belongs to the 'dirty' part of `AntibioticsPatient` and hence this query class has only a primitive concept as its type. It can be deduced that

$$\begin{aligned} AntibioticsPatientT &\sqsubseteq PatientT \sqcap \forall takes.AntibioticsT \sqcap \geq 1\, takes \\ &\sqsubseteq PatientT \sqcap \geq 1\, takes = DrugPatientT \end{aligned}$$

provided that $AntibioticsT \sqsubseteq DrugT$.

Whenever the query `AntibioticsPatient` has to be evaluated and the object base already contains the answers to `DrugPatient` since it is designed as materialized view, it is not necessary to scan all patients in the object base but only the precomputed answer set of drug taking patients.

4 Applications

The implementation of query classes in ConceptBase has turned out to be a useful basis for several application areas that can exploit the different facets of queries as classes, as rules, and as concepts. While some of these applications

[3] Several concept languages offer an enumeration construct which would allow to express such a condition in the type language.

used the query language concept as is, others base extensions such as access to external data stores on it.

Software databases and repositories. *Dynamic clusters* are introduced in [22] as collections of objects where the membership frequently changes. Instead of procedurally assigning objects to such classes, a membership condition expressed as a query class automatically classifies objects into the correct clusters. Due to the deductive nature of query classes, the dynamic aspect of re-assigning objects is completely covered by view maintenance. The application mainly profits from queries as classes: the dynamic cluster is a query class persistently stored on the object base. It also profits from the representation of classes as objects in O-Telos. In fact, there are many queries which actually have class objects as anwers. This feature of query classes is sometimes called *schema querying* or *meta querying*.

Security management. Security is becoming an increasing concern in many databases. The Group Security Model GSM [26] uses generalization and aggregation to define task-based access to parts of the object base (identified by classes). The question what information a certain task may access is encoded as a query class. Thus, the deductive view is applied for evaluation while the object-oriented view and the availability of parameterized query classes are exploited for the precise and concise characterization of access right patterns.

Integration of heterogeneous databases. In many applications, the information to be processed is distributed in more or less autonomous information bases. Data may be replicated or even contradictory, the languages may be heterogeneous. We have made a first attempt with query classes to schema integration of distributed relational databases [19]. The system represents the base relations and their abstractions to the integrated schema uniformly as query classes. The deductive interpretation is used for accessing the base relations from queries formulated at the level of the integrated schema. Object-oriented principles come into play when organizing the communication between the external databases: each database can be seen as a class with a small interface of methods offered to the integrated system.

5 Conclusions

This paper presented a way to integrate the view of queries as classes, rules, and concepts. In one sentence the idea is the following: queries are expressed as classes, evaluated as deductive rules, and semantically optimized as concepts. The first view offers easy formulation since relationship of queries to schema classes is expressed by subclass and attribute statements in a simple frame notation. The second view, queries as rules, provides the efficient evaluation algorithms like magic sets and mapping to algebra expressions. Finally, we proposed to extract the structural part from a query (queries as concepts) and make it subject to reasoning on subsumption relationship between the answer sets of two queries.

Previous integration efforts only concentrated on two of the three aspects of queries. Similar to our approach, CoCoon [23] integrates concepts with classes but with a fixed and very simple type system (only lattice of attribute names). [1] presents a query language combining deduction with object-orientation. An integration of deductive rules and concept languages is investigated in [11]. Views in the object-oriented query language XSQL [18] are quite similar to query classes by seperating the signature of the answer objects from the membership condition. However, XSQL is too expressive (wrt. object model and generation of OID's inside a query) for allowing the optimization techniques discussed here.

Several applications of query classes in ConceptBase have validated the usefulness of the idea but also pointed out the need for various extensions, both from the user side (imprecise and intelligent question answering) and from the system side (integration of multiple formalisms). Finally, the symmetric evaluation of updates on answers to queries (view update) is a major challenge: a declarative language for both updates and retrieve methods.

References

1. S. Abiteboul, "Towards a deductive object-oriented database language", *Data & Knowledge Engineering*, 5, 1990, pp. 263–287.
2. H.W. Beck, S.K. Gala, and S.B. Navathe, "Classification as a query processing technique in the CANDIDE semantic data model", in *Proc. 5th Int. Conf. on Data Engineering*, 1989, pp. 572–581.
3. C. Beeri and R. Ramakrishnan, "On the power of magic", in *Proc. 6th ACM SIGMOD-SIGACT Symp. on Principles of Database Systems*, 1987.
4. A. Borgida, "From type systems to knowledge representation: natural semantics specifications for description logics", *Int. Journal of Intelligent and Cooperative Information Systems* **1**(1), pp. 93–126, 1992.
5. A. Borgida, R.J. Brachman, D. McGuiness, and L.A. Resnick, "CLASSIC: A structural data model for Objects", in *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1989, pp. 58–67.
6. R.J. Brachman, D.L. McGuinness, P.F. Patel-Schneider, L.A. Resnick, and A. Borgida, "Living with CLASSIC: When and how to use a KL-ONE-like language", in *Principles of Semantic Networks*(Sowa J.,ed.), Morgan Kaufmann, 1991.
7. R.J. Brachman and J.G. Schmolze, "An overview of the KL-ONE knowledge representation system", *Cognitive Science* **9**(2), pp. 171–216, 1985.
8. F. Bry, H. Decker, and R. Manthey, "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases", in *Int. Conf. on Extending Database Technology*, 1988, pp. 488–505.
9. M. Buchheit, M.A. Jeusfeld, W. Nutt, and M. Staudt, "Subsumption between queries to object-oriented databases", appears in *Proc. EDBT'94*, Cambridge, UK, March 1994.
10. S. Ceri, G. Gottlob, and L. Tanca, *Logic programming and databases*, Springer-Verlag, 1990.
11. F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf, "A hybrid system with datalog and concept languages", in *Trends in Artificial Intelligence*, (Ardizzone E., Gaglio S., Sorbello F., eds.), LNAI 549, Springer Verlag, pp. 88-97, 1991.

12. B. Hollunder, W. Nutt, M. Schmidt-Schauss, "Subsumption algorithms for concept description languages", in *Proc. 9th European Conf. on Artificial Intelligence*, pp. 348–353, 1990.
13. M. Jarke (ed.), *ConceptBase V3.1 user manual*, Report Aachener Informatik-Berichte Nr. 92-17, RWTH Aachen, Germany, 1992.
14. M.A. Jeusfeld, *Update control in deductive object bases* (in German). Infix-Verlag, St.Augustin, Germany, 1992.
15. M.A. Jeusfeld and M. Jarke, "From relational to object-oriented integrity simplification", in *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases, LNCS 566*, Springer-Verlag, pp. 460–477, 1991.
16. M.A. Jeusfeld and M. Staudt, "Query optimization in deductive object bases", in *Query Processing for Advanced Database Applications*, (Freytag et al., eds.), Morgan-Kaufmann, 1993.
17. A.C. Kakas and P. Mancarella, "Database updates through abduction", in *Proc. 16th Int. Conf. on Very Large Databases*, 1990, pp. 650–661.
18. M. Kifer, W. Kim, Y. Sagiv, "Querying object-oriented databases", in *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, San Diego, Ca., 1992, pp. 393–402.
19. A. Klemann, *Schema integration of relational databases* (in German), Diploma thesis, Universität Passau, Germany, 1991.
20. J. Mylopoulos, A. Borgida,M. Jarke, and M. Koubarakis, "Telos: a language for representing knowledge about information systems", in *ACM Trans. Information Systems* **8**(4), pp. 325–362, 1990.
21. A. Olivé, "Integrity constraints checking in deductive databases", in *Proc. 17th Int. Conf. on Very Large Databases*,1991, pp. 513–524.
22. T. Rose, M. Jarke, J. Mylopoulos, "Organizing software repositories - modeling requirements and implementation experiences", in *Proc. 16th Int. Computer Software & Applications Conf.*, Chicago, Ill., 1992.
23. M.H. Scholl, C. Laasch, and M. Tresch, "Updatable views in object oriented databases", in *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases*, Munich, Germany, 1991.
24. M. Staudt, *Query representation and evaluation in deductive object bases* (in German), Diploma thesis, Universität Passau, Germany, 1990.
25. M. Staudt, H.W. Nissen, M.A. Jeusfeld, "Query by class, rule, and concept", in *Applied Intelligence*, Special issue on Knowledge Base Management, (Mylopoulos L., ed.), 1993.
26. G. Steinke, "Design aspects of access control in a knowledge base system", in *Computers & Security*, 10, 7, 1991, pp. 612–625.
27. L. Vieille, "Recursive axioms in deductive databases: The query-subquery approach", In *Proc. 1st Int. Conf. on Expert Database Systems*, 1986.