Query Optimization in Deductive Object Bases

Manfred Jeusfeld†, **Martin Staudt**‡

† Universität Passau, 8390 Passau, Germany, *jeusfeld@uni-passau.de*
‡ RWTH Aachen, 5100 Aachen, Germany, *staudt@informatik.rwth-aachen.de*

Abstract[1]**.** *Deductive object bases* are extended database systems which amalgamate structural object-orientation with logical specification. Queries in such a system are regarded both as classes and as deduction rules. Besides a general architecture for query processing in deductive object bases, two specific query optimization techniques are presented: semantic query optimization with structural axioms of the object base, and view maintenance optimization. The approach has been formalized in the language Telos and implemented in the system ConceptBase.

[1] This work has been supported in part by ESPRIT BRA 3012 Compulog. A version of this paper also appears in Freytag, Maier, Vossen (eds.): Query processing in object-oriented, complex-object, and nested relation databases, Morgan Kaufmann, 1992.

1. Introduction

Traditionally, databases are systems for storing and accessing large amounts of shared persistent data in a secure way. Database research has always been concerned with providing efficient methods for these tasks. One major conjecture was that data should be independent from specific application programs. This point of view has been materialized by the relational-style databases. They provide a simple and very attractive data model, and *declarative* query languages. Compared to complete programming languages, a query language is usually rather limited – a limitation that has an important benefit: it is easier to prove properties of queries. One such property is guaranteed termination. Another is the ability to decide whether two expressions are equivalent. The latter is a prerequisite for query optimization. While relational (and deductive) databases were very successful partly because of their query optimizers, they can fail for complex applications like design [MAIE86]. The reason for this failure is the too simple data model which is unable to describe adequately the data strutures manipulated by such applications. Another drawback is the total negligence of operational properties of data, aka abstract data types [GUTT75]. Object-oriented databases address these problems by their richer type (or class) system and - at least a few of them - by their ability to store operations or *methods* together with the data. At the first glance, the added complexity implies some obstacles for query optimization. For instance, a complete programming language for methods introduces undecidability when they are used within queries (like e.g. in O_2 [BCD89]). The notion of complex objects offers a multitude of nested structures that can all hold the same kind of data. This makes it a more difficult task to combine such data. Shaw and Zdonik [SZ90] propose a whole array of different equality operators dependent on the nesting depth of objects. The problem arises from the distinction between objects given by their identifier and objects given by its (composite) value (see also [BEER90]).

This paper pleads for a deductive style of object bases achieved in a two layer architecture (see fig. 1). The top layer of such an object base system consists of declarative expressions, i.e., frame-like object specifications, views (esp. for complex objects), deductive rules, general integrity constraints, and a query language. All of these items are firmly based on a first-order theory of object bases. This layer corresponds to a modified view of databases as systems for *managing a model of knowledge about the world in an accessible and accurate manner for users.* Some basic optimization techniques for the logical layer are simplification methods [NICO82,BCL89], recursion optimization [BR86], and semantic query optimization [JARK84,CGM90]. They have in common that optimized formulas are obtained by partially evaluating either integrity constraints or update specifications with queries. The bottom layer contains the implementations of the logical expressions. Object names are mapped to object identifiers which behave similar to virtual memory addresses. The basic optimization techniques at this level are rewritings of algebraic expressions (e.g. [FREY87,GD87]), and redundant data, esp. indices (e.g. [KM90a]). An integrated trigger mechanism at the implementation layer meets the requirements for *active database systems* whose active components enable the creation and execution of data manipulation operations in a production rule like way. In our view and also in our implementation these action rules are compiled automatically from the specifications at the logical layer [JK90]. Other approaches propose semi-automatical derivation methods with user interaction for purposes of integrity checking and view maintenance [CW90,CW91].

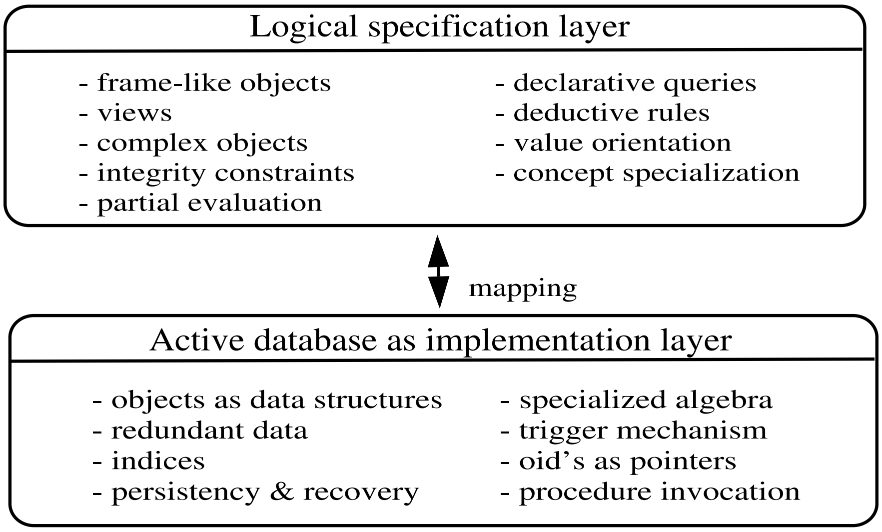

Fig. 1: Architecture of a deductive object base

We claim that such a two layer architecture is able to combine both deductive-relational query optimization and the advantages of an object-oriented data model. This claim is justified as follows:

- ▷ Section 2 defines *deductive object bases* as a special case of a deductive databases with integrity constraints. Object-oriented abstraction principles like object identity, classification, and specialization become axioms of a first-order database theory.
- ▷ Queries can be defined as classes whose instances are the answer to the query (section 3). Such a view allows to *classify queries* into the class hierarchy of the object base. A mapping of queries to deductive rules precisely defines the semantics of such queries.
- ▷ The *increased structure* of object bases can be used to perform semantic query optimization based on the structural axioms of the object model. Section 4.1 shows, as an example, the exploitation of the attribute typing axiom in order to eliminate class membership predicates.
- ▷ Trigger mechanisms from deductive integrity checking methods are taken to optimize views, i.e., queries whose answers are maintained consistent with the object base (section 4.2). The mechanism works in combination with (recursive) deductive rules. An additional benefit of *maintained views* is that they can help to optimize ad hoc queries which are subclasses to the view.

The interplay of these logic-based optimization techniques with the algebraic and indexing techniques is sketched in the introduction to section 4. Experiences gained from an implementation within the deductive object base ConceptBase [JARK91] are reported in section 5.

2. Object Bases as Deductive Databases

Most query optimization techniques have been developed for relational and deductive databases. On the other hand, the relational model has intrinsic weaknesses [JACK90] and performance comparisons [DD88] indicate that object-oriented databases can beat their relational counterparts, esp. when following references. This section presents object bases to be special cases of deductive databases (EDB,IDB,IC). EDB is the extensional database of base relations, IDB is a set of deductive rules, and IC is a set of integrity constraints. The formulas in $IDB \cup IC$ have to be range-restricted [NICO82] which is a widely accepted sufficient condition for domain independence (see [BRY88,ML90] for more details). To ensure unique perfect models, we also assume that the set IDB is stratified [CGT90]. The next subsection defines the extensional database for deductive object bases. Then, the object-oriented principles are defined by axioms based on the extensional object base. The uniform representation of classes and instances requires an adaption of the notion of stratification for defining the semantics of deductive rules.

2.1. The Extensional Object Base

The definition of the structure of an object base has considerable impact on the possible operations on it. For relational-style databases these operations are restricted to tuple updates on the base relation of the EDB. More demanding applications like software design, computer supported cooperative work, and others are characterized by two requirements that are hard to fulfill by relational-style databases:

- ▷ Updates on the database schema are frequent as the knowledge of the application domain is evolving. A data model that makes schema updates a cumbersome task is not appropriate.
- ▷ The data structures to be processed by the application tend to be dynamic in size rather than fixed as for first normal form tuples. Delivering exactly the required data structures for the application program is a precondition for a seamless integration of a database with the application.

For the reasons mentioned before we adopt an object model originally proposed for the requirements engineering of information systems: the knowledge representation language Telos [MBJK90]. In Telos classes, instances, attributes, instantiation and specialization relationships are uniformly represented as quadruples:

Definition 1

Let ID, LAB, $LAB \subseteq ID$, be countable sets of identifiers or labels, resp. Then a finite set

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

is called an (extensional) **object base**. The elements of OB are referred to as **objects**.

In fact, this definition of an object base is like a total decomposition of a relational database into binary relations [ABRI74] with two important differences:

1) Each binary "tuple" gets an object identifier which enables references from and to it.

2) The "relation name" is an argument of the quadruple. That allows quantification over such labels without leaving first order logic.

The intuition behind an object $P(o, x, l, y)$ is that there is a relation l between the objects x and y. This relationship is itself an object with identifier o. In a graphical representation, *individual* objects, $P(o, o, l, o)$, are drawn as nodes. The other objects are links (attributes). Two special labels are reserved: $P(o, x, in, c)$ denotes that the object x is an instance of the (class) object c, and $P(o, c, isa, d)$ defines c to be a subclass of d. The semantics are defined by axioms of the object base theory (see next subsection).

In contrast to types in programming languages [CW85] classes do not express sufficient conditions for class membership but only necessary conditions. This is especially true for the attributes of a class: an instance of a class may instantiate these attributes but is not obliged to do so (provided there are no additional integrity constraints demanding that). Figure 2 shows an example Telos object base on patients and drugs.

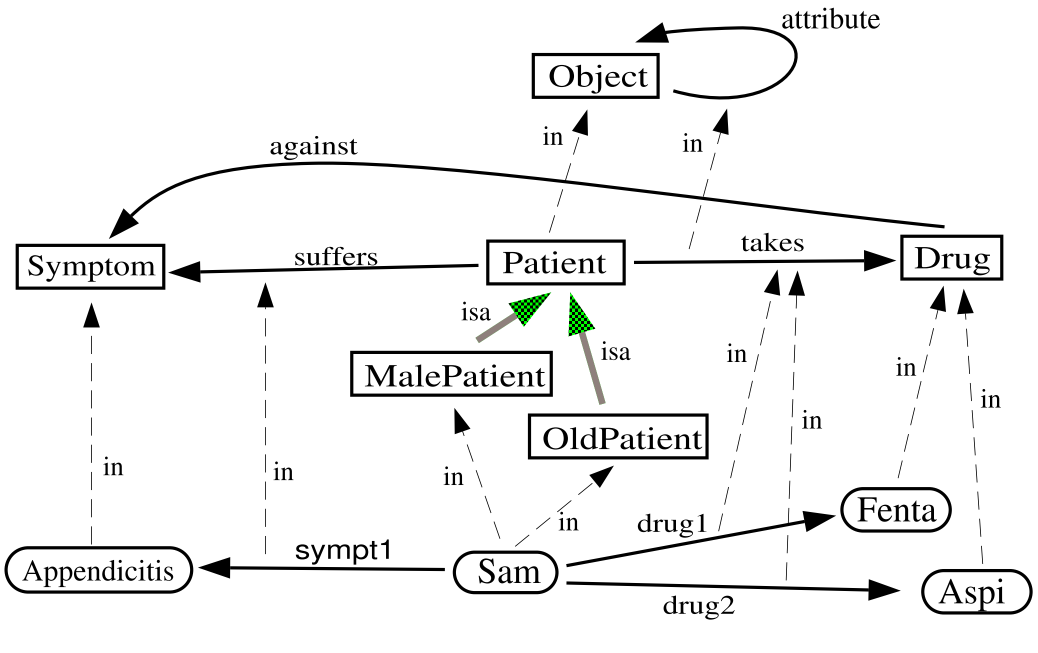

Fig. 2: Object base for the drug example

The class `Patient` has an attribute labelled `takes` to the class `Drug`. Both `MalePatient` and `OldPatient` are subclasses of `Patient` where the former has an instance called `Sam`. `Sam` has two attributes `drug1` and `drug2` which are both instances of the `takes` attribute class. The destinations `Fenta` and `Aspi` are instances of the class `Drug`. It should be noted that the graph does not contain oid's but only labels. An object base for fig. 2 would look like

$$OB = \{ \; P(\#Pat, \#Pat, Patient, \#Pat),\; P(\#MP, \#MP, MalePatient, \#MP),$$
$$P(\#isa1, \#MP, isa, \#Pat),\; P(\#OP, \#OP, OldPatient, \#OP),$$
$$P(\#isa2, \#OP, isa, \#Pat),\; P(\#Drug, \#Drug\; Drug, \#Drug),$$

$P(\#Sympt, \#Sympt, Symptom, \#Sympt)$, $P(\#tak, \#Pat, takes, \#Drug)$,
$P(\#suff, \#Pat, suffers, \#Sympt)$, $P(\#ag, \#Drug, against, \#Sympt)$,
$P(\#Sam, \#Sam, Sam, \#Sam)$, $P(\#in1, \#Sam, in, \#MP)$,
$P(\#Fen, \#Fen, Fenta, \#Fen)$, $P(\#Asp, \#Asp, Aspi, \#Asp)$,
$P(\#in2, \#Fen, in, \#Drug)$, $P(\#in3, \#Asp, in, \#Drug)$,
$P(\#dr1, \#Sam, drug1, \#Fen)$, $P(\#dr2, \#Sam, drug2, \#Asp)$,
$P(\#in4, \#dr1, in, \#tak)$, $P(\#in5, \#dr2, in, \#tak)$, $P(\#App, \#App, Appendicitis, \#App)$,
$P(\#in6, \#App, in, \#Sympt)$, $P(\#sy1, \#Sam, sympt1, \#App)$ $\}$

A frame-like representation of objects and their properties is solely based on object labels (the third component of an object quadruple). The objects of fig. 2 are mapped from the following frames:

```
Object Patient with
  attribute
    takes: Drug;
    suffers: Symptom
end

Object MalePatient isA Patient
Object OldPatient isA Patient
Object Fenta in Drug
Object Aspi in Drug

MalePatient,OldPatient Sam with
  takes
    drug1: Fenta;
    drug2: Aspi
  suffers
     sympt1: Appendicitis
end
```

Labels for individual objects have to be globally unique. Attribute labels have to be unique within the same frame. The classes of an object are written before the object, superclasses are preceded by the key word `isA`. The attributes of an object are instantiated from to the attributes of the class by grouping them under the class attribute label. We omit the formal definition, esp. the conflict resolution on multiple inherited attributes with identical labels. The object `Sam` shows that the attribute `takes` may be instantiated several times.

There are some objects whose oid has a meaning outside the object base. These objects are called *values* [BEER90]. Examples for (atomic) values are numbers and strings. We write values as objects of the form $P(v, v, v, v)$, i.e., individual objects whose label and oid are the same [KMSB89].

2.2. Deductive Object Base Theory

From the deductive database standpoint an object base (def. 1) is an extensional database with a single relation. That situation is undesirable since each update deductive rule and each integrity constraint would be affected by any update. This section presents an axiomatic definition of object-oriented abstraction mechanisms (partially taken from [MBJK90,KMSB89]). Deductive rules and integrity constraints are expressed with three literals for instantiation, specialization, and attribute relationships. Stratification is then applied to a rewriting of the formulas where class object identifiers are used as predicate names.

The first axiom defines object identity. No two objects in the extensional object base may have the same identifiers.

$$\begin{aligned}&\forall\ o, x_1, l_1, y_1, x_2, l_2, y_2\ P(o, x_1, l_1, y_1) \wedge P(o, x_2, l_2, y_2) \Rightarrow \\ &\quad (x_1 = x_2) \wedge (l_1 = l_2) \wedge (y_1 = y_2)\end{aligned} \tag{A_1}$$

For individual objects the label (third component) must be unique within the object base, too. The next three axioms induce base solutions for the three literals for instantiation, specialization, and aggregation. These three literals are later used to formulate deductive rules and integrity constraints.

$$\forall\ o, x, c\ P(o, x, in, c) \Rightarrow In(x, c) \tag{A_2}$$

$$\forall\ o, c, d\ P(o, c, isa, d) \Rightarrow Isa(c, d) \tag{A_3}$$

$$\forall\ o, x, l, y, p, c, m, d\ P(o, x, l, y) \wedge P(p, c, m, d) \wedge In(o, p) \Rightarrow A(x, m, y) \tag{A_4}$$

Inheritance of class membership is a deductive rule. The *Isa* literal is defined to form a partial order on the set objects identifiers of an object base (another 3 axioms not shown here).

$$\forall\ x, c, d\ In(x, c) \wedge Isa(c, d) \Rightarrow In(x, d) \tag{A_5}$$

The next axiom declares the "weak" attribute typing in a Telos object base. Objects may instantiate the attributes of their classes only if the destination of the attribute belongs to the right class.

$$\begin{aligned}&\forall\ o, x, l, y, p\ P(o, x, l, y) \wedge In(o, p) \Rightarrow \\ &\quad \exists\ c, m, d\ P(p, c, m, d) \wedge In(x, c) \wedge In(y, d)\end{aligned} \tag{A_6}$$

Axiom A_4 provides a single literal $A(x, m, y)$ for all attribute accesses from an object x to its attribute value y (sometimes written as $x.m = y$). In object-oriented languages it is common to allow the use of the same attribute label m for different classes. The next axiom demands that for any object x the literal $A(x, m, y)$ is uniquely assignable to an attribute class with label m. If there are two different attribute classes that x instantiates, then there must be a common subclass of them:

$$\begin{aligned}&\forall\ x, m, y, c, d, a_1, a_2, u, v\ In(x, c) \wedge In(x, d) \wedge A(x, m, y) \wedge P(a_1, c, m, u) \wedge P(a_2, d, m, v) \\ &\quad \Rightarrow \exists\ e, a_3, w\ In(x, e) \wedge P(a_3, e, m, w) \wedge Isa(e, c) \wedge Isa(d, e) \wedge Isa(a_3, a_1) \wedge Isa(a_3, a_2)\end{aligned} \tag{A_7}$$

The rest of the axioms (unique labeling, refinement of attributes for subclasses, membership to predefined classes) are omitted for sake of readability [JEUS91].

2.3. Deduction and Integrity

Deductive rules and integrity constraints are range-restricted first order formulae over the three literals *In, Isa,* and *A*. Range-restrictedness can be guaranteed by assigning the quantified variables to classes: $\forall\ x/C\ \varphi$ stands for $\forall x\ In(x, C) \Rightarrow \varphi$, and $\exists\ x/C\ \varphi$ stands for $\exists x\ In(x, C) \wedge \varphi$. This syntax is not a real restriction since variables in our model always refer to elements in the object base.

Efficiency and stratification in deductive databases depend on the number of base and deduced relations. The above definition offers one base relation P and three deduced relations *In, Isa, A*. That number is surely not satisfactory. Therefore, a restricted interpretation of deductive rules and integrity constraints is adopted. Firstly, an object base with only $A_2 - A_4$ as deductive rules delivers ground facts for the three predicates. Note that the axioms are range-restricted and stratifiable. For a ground fact $A(x, m, y)$, the closure axiom [REIT84] applied to A_4 guarantees the existence of an object $P(p, c, m, d)$ to which an attribute of x was instantiated to. We extend the deduction machine by a rule that delivers a fact $A.p(x, y)$ for each such p. Similarily, a fact $In.c(x)$ for each ground fact $In(x, c)$ is derived. The Isa-literal remains unchanged. These modifications extend the number of literals by all class and attribute identifiers.

Now, deductive rules and integrity constraints are rewritten such that they only contain $A.p$, $In.c$ and *Isa* literals. For the *In* literal, this rewriting is trivial provided the second component is a constant. Otherwise the formula is rejected (see [JJ91] for handling meta formulas which range over classes). The interesting case is the literal A. Let $A(x, l, y)$ be a literal occurrence in a formula φ where x is bound to class c. Then c' is defined to be the lowest superclass of c that has an attribute labelled l. If there is such a c' at all, then it is uniquely defined due to axiom A_7. The following example shows that there are cases where no such class exists ($\#Obj$ is the identifier of the system class `Object` which has all objects as instances):

$$\forall\ x/\#Obj\ \ A(x, takes, 1000)$$

Such formulas are forbidden as deductive rules or integrity constraints. The refinement of attributes creates no further problems since axiom A_7 demands a specialization relation between those attributes. Thereby, instances of refined attributes will also be visible at their superclasses.

The text representations of deductive rules and integrity constraints uses labels instead of object identifiers. The mapping from labels is a compilation task and not subject of this paper. The administration of the formula strings is done within the object base: they are stored as instances of the predefined classes `DeductiveRule` and `IntegrityConstraint`. Both are attributes of `Object`. As an example, a constraint demanding a patient to have at least one symptom, is assigned to the class `Patient`:

```
Object Patient with
  attribute
    takes: Drug;
    suffers: Symptom
  constraint
    sick: $ forall p/Patient exists s/Symptom A(p,suffers,s) $
end
```

The string represents the formula

$$\forall\ p\ \ In.\#Pat(p) \Rightarrow \exists\ s\ \ In.\#Sympt(s) \wedge A.\#suff(p, s)$$

which becomes part of the set IC in the deductive object base (OB,R,IC). The concerned attribute of $A(p, suffers, s)$ is *#suff*. Note that the constraint may only become part of the deductive object base if the rewriting of the literals succeeds. Further examples with recursive deduction rules are in [JJ91].

3. Queries as Classes

Many proposals exist in the literature for query representation formats in the context of object-oriented databases and knowledge representation systems. One group uses query languages adapted from conventional databases e.g. SQL like languages augmented by some typical object-oriented constructs (ORION [BKK88], Iris [LHW90], O_2 [BCD89]). Another group of query languages is based on first order logics with a link to deductive databases, e.g. [AG91,KLW90]. They investigate the relations of complex objects and operations (given by their signature) with logical statements about the object base. Among the derivatives of the knowledge representation language KL-ONE the systems CLASSIC [BBMR89] and CANDIDE [BGN89] offer a frame format for queries. Queries are described as any other concept stored in the knowledge base (KB). Kernel of these systems is a subsumption algorithm which is used to store new objects and to evaluate queries by temporary placement in the KB. Candidates for answer objects are those objects positioned underneath the query concept within the concept and individual hierarchy of the KB. An important advantage of these classification based query languages is their nearness to the underlying data description language. The main concepts of the data model occur in the query language too which means clearness and user friendliness. In addition queries are seen as objects which can be stored in the KB and manipulated in any other way. Answers are always objects already existing in the KB.

In [STAU90] this latter idea of query representation has been adopted and used to build up a query language for the above outlined notion of an object base. Using for objects the frame notation of the last section a metaclass `QueryClass`[2] shall contain all possible queries which themselves are classes. Following the instantiation and classification principle instances of a query class are answers to the represented query. To express necessary and sufficient membership conditions for answer instances in query classes by which membership can be tested and answers be computed we have on the one hand the possibility to specify structural constraints like in the above mentioned classification based query languages.

First, query classes can have super classes to which they are connected by an isa link. These super classes restrict the set of possible answer instances to the common instances of the super classes of the query class.

```
QueryClass Q1 isA MalePatient,OldPatient
```

`MalePatient` and `OldPatient` are here assumed to be themselves subclasses of `Patient`. Since the two specialization relationships represent the only membership condition for `Q1` the answer consists of the intersection of the instances of both subclasses of `Patient`.

Second, query classes may have attributes of two different types. The first kind of attributes are inherited of one of the super classes . If such an attribute is specified explicitly in a query class description this means that answer instances are given back with value instantiations of this attribute, similar to projection in relational algebra. In addition, this explicit specification includes a necessary condition for the instantiation of the attribute with an admissible attribute value by the answer instances. This necessary condition can

[2] An approach with similar properties has independently been proposed in [AB91].

be enforced by specializing the target class of the inherited attribute. Then an answer instance must instantiate the attribute with a value which is instance of this more special class.

```
QueryClass Q2 isA MalePatient,OldPatient with
  attribute
    takes: ADrug
end
```

`ADrug` is assumed to be a specialization of class `Drug`. `Q2` inherits attribute `takes` from class `Patient` via direct superclass `MalePatient` or `OldPatient`. These answer instances are the common instances of both super classes which actually take drugs of the class `ADrug`.

Attributes for query classes of the second type are attributes whose instantiation value by an answer instance is computed during the query evaluation process. That means that a relation between the answer instance and the computed attribute value is not necessarily stored explicitly in the KB or deducible by a stored deduction rule but nevertheless part of the answer. The prescription how to deduce these attributes must be included in the definition of the query class and can obviously not be done in a structural way without loss of generality. So there is need for a supplementing formalism. As assertions (integrity constraints and deductive rules) are used in the data description language of the object base, typed first order logic expressions can be introduced as possible building elements for query classes.

```
QueryClass Q3 isA MalePatient,OldPatient with
  attribute
    wrong:Drug
  constraint
    wrongRule: $ A(this,takes,wrong) and
               not exists s/Symptom A(this,suffers,s)
               and A(wrong,against,s) $
end
```

Within the formula `this` is a shorthand reference to the answer instances of `Q3`. The variable `wrong` is identified with the value of the corresponding attribute. So here the first order logic expression denotes a prescription for the deduction of the `wrong` attribute. `Q3` computes all old and male patients who take drugs against symptoms they do not suffer from. The deduced values for the `wrong` attribute are part of the answer. In addition it is possible to include any other membership conditions for instances of a query class using the logical representation. The integration of structural and logical representation formalisms leads to a hybrid query language. KRYPTON [BGL85], a derivative of KL-ONE and ancestor of Telos, uses both formalisms for its query language too but as in its data description language they are not combined to one closed format.

In order to avoid the reformulation of similar more specialized queries, attributes of query classes can be parameterized. Substitution of a concrete value for an attribute or specialization of its target class by a subclass leads to a subclass of the original query class which implies a subset relationship of the answer sets. In the frame syntax, specialized classes are written as terms that define the substitution. For instance, `Q3` can be specialized by substituting the `wrong` attribute:

```
Q3(Fenta/wrong)
```

That query is a shorthand for a query class which has only those patients as answer instances which are instances of `Q3` and have the concrete drug `Fenta` as their `wrong` attribute. Another kind of substitution refines the attribute value class: `Q3(wrong:ADrug)` considers only those answer instances of `Q3` whose "wrong" drugs belong to the subclass `ADrug`. The semantics of query classes is twofold: First, a query class is just a class object that has some attributes. Second, there is a mapping from the query class frame to a deductive rule that defines which objects are the answer to the query. Consider the frame

```
QueryClass Q isA C1,...,Ck with
  attribute
    a1: S1;
    ...
    am: Sm;
    b1: T1;
    ...
    bn: Tn
  constraint
    c: $ <formula text> $
end
```

We assume that `a1,...,am` are attributes which are refined from existing attributes with the same labels in classes `C1,...,Cm`. The attributes `b1,...,bn` are additional properties of `Q`. Let φ be the first-order formula represented by `<formula text>`. Then the query rule corresponding to the frame is:

$$\begin{aligned} &\forall\ x, y_1, ..., y_m, z_1, ..., z_n\ In.\#C1(x) \wedge ... \wedge In.\#Ck(x) \wedge \\ &\quad In.\#S1(y_1) \wedge A.\#a1(x, y_1) \wedge ... \wedge In.\#Sm(y_m) \wedge A.\#am(x, y_m) \wedge \\ &\quad In.\#T1(z_1) \wedge ... \wedge In.\#Tn(z_n) \wedge \varphi \Rightarrow Q(x, y_1, ..., y_m, z_1, ..., z_m) \end{aligned} \tag{1}$$

The first argument x of Q is called the **answer variable** of the query rule. The other arguments are called **query attributes**. The constants *#C1,#S1,#T1* etc. are the object identifiers of the labels `C1`, `S1`, `T1` etc., and *#a1,...,#am* are the identifiers of the attributes labelled by `a1,...,am` (see section 2). As formula (1) shows the variables in $Q(x, y_1, ..., y_m, z_1, ..., z_m)$ are bound to objects in the object base. There is no "invention" of object identifiers for answers to queries as for example in [HY90].

Since query classes may be superclasses, or attribute value classes of other queries, class membership to queries is defined by a second rule.

$$\forall\ x, y_1, ..., y_m, z_1, ..., z_n\ Q(x, y_1, ..., y_m, z_1, ..., z_m) \Rightarrow In.\#Q(x)) \tag{2}$$

The evaluation of this rule leads to a set of ground instances of the literal Q which can be used to build up the answer instances of the query class `Q` in frame format. The two deductive rules for query class `Q3` are

$$\begin{aligned} &\forall\ x, y_1\ In.\#MP(x) \wedge In.\#OP(x) \wedge In.\#Drug(y_1) \wedge A.\#tak(x, y_1) \wedge \\ &\quad \neg\exists\ s\ In.\#Sympt(s) \wedge A.\#suff(x, s) \wedge A.\#ag(y_1, s) \Rightarrow Q_3(x, y_1) \end{aligned} \tag{3}$$

$$\forall\ x, y_1\ Q_3(x, y_1) \Rightarrow In.\#Q3(x) \tag{4}$$

The parameterization of query classes has a strong logical counterpart, the simplification of formulas [NICO82]. For instance, the deductive rule for the specialized query class `Q3(Fenta/wrong)` is obtained by simplifying formula (3) with $In.\#Drug(\#Fen)$. This step instantiates the universally quantified variable y_1. The result is

$$\forall\ x\ In.\#MP(x) \wedge In.\#OP(x) \wedge A.\#tak(x, \#Fen)\ \wedge$$
$$\neg\exists\ s\ In.\#Sympt(s) \wedge A.\#suff(x,s) \wedge A.\#ag(\#Fen, s) \Rightarrow Q_3'(x, \#Fen)$$

$$\forall\ x, y_1\ Q_3'(x, y_1) \Rightarrow In.\#Q3'(x)$$

The simplification of the query (3) conforms well with the specialization axiom A_5 since $In.\#Q3'(x)$ implies $In.\#Q3(x)$. The same property holds for the attribute value class refinement, e.g. `Q3(wrong:Adrug)`.

4. Query Optimization Methods

With the given object model and query language as prerequisite in the following two opportunities of query optimization in deductive object bases are presented. Figure 3 shows the general query optimizer architecture of a deductive object base [JK89].

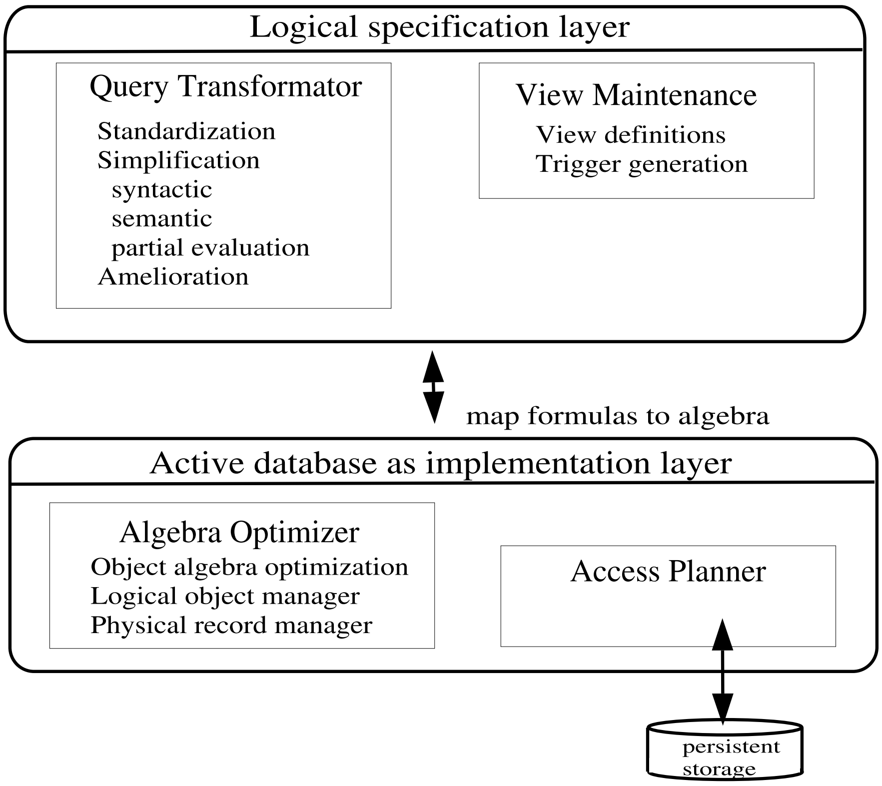

Fig. 3: Architecture of a query optimization component for deductive object bases

The optimizer is subdivided into components for logical and algebraic query representations. The first one maps query declarations to algebra expressions which are then evaluated on the physically stored records. This mapping is accompanied by using different techniques of optimization to obtain necessary efficiency.

The query transformation component at the logical layer applies general rules, laws and heuristics to optimize and map a user query. Three different aspects have to be distinguished. Standardization means mainly the introduction of a normal form used as a starting point for further transformations. Simplification can be understood in a syntactic (e.g. elimination of tautologies and redundancies) or semantic way (consideration of rules and integriy constraints) and includes partial evaluation methods. Amelioration covers issues such as recursion elimination and optimization, reuse of common subexpressions and application of general heuristics.

At the implemetation layer besides the mapping to logical and physical storage structures optimizations for the special selected object algebra are performed. The access planning component, takes quantitative statistical information about the current objectbase into account when optimizing the algebra expression resulting from the transformation step.

Finally, a view maintenance mechanism allows the use of materialized views which in fact are queries with stored answers and prevents unnecessary computations. Here we have an additional aspect of optimization which comprises the efficiency of the view maintenance algorithm. This last topic is discussed in paragraph 4.2. The following paragraph concentrates on a semantic simplification technique using structural axioms of the underlying object model.

4.1. Structural Query Optimization

The deductive object base theory presented in section 2 is characterized by an increased number of axioms when compared with deductive relational databases. The axioms are theorems, i.e. they are true in any consistent deductive object base (OB,R,IC). Therefore, they can always be used to simplify queries. In this section the exploitation of the attribute typing axiom A_6 is investigated.

Recalling the query rule formula (3), one observes a considerable number of class membership literals $In.c(x)$. They are introduced by the assignment of variables to classes, by the attribute value classes, and by the interpretation of superclasses as classes of the answer variable. Consider now an expression $A.p(x, y) \wedge In.c(x)$, and suppose that p is the oid of an object $P(p, c, m, d)$ in the extensional object base. Then, two cases can be distinguished:

(I) $A.p(x, y)$ is derived from axiom A_4 (section 2). Then there must be an object $P(o, x, l, y)$ with $In(o, p)$. The attribute typing axiom A_7 then establishes the truth of $In(x, c)$.

(II) $A.p(x, y)$ is derived from a deductive rule $\forall\ x/c', y/d'\ \varphi \Rightarrow A(x, m, y)$ where $A(x, m, y)$ is rewritten to $A.p(x, y)$ (see section 2). Then, c must be the lowest superclass of c' that has an attribute with label m. Then, $In.c'(x)$ is true, and as a consequence of axiom A_5 $In.c(x)$ follows.

In both cases, $In.c(x)$ is already guaranteed by $A.p(x, y)$, $P(p, c, m, d)$, and the axioms of the object base. A similar argument holds for $In.d(x)$ in a conjunction $Ap(x, y) \wedge In.d(y)$.

Thus, any conjunction $A.p(x,y) \wedge In.c(x)$ or $A.p(x,y) \wedge In.d(y)$, resp., can be replaced with $A.p(x,y)$ provided p is the identifier of an object $P(p,c,m,d)$.

The query rule (3) for query `Q3` serves as an example. Firstly, there is a conjunction $In.\#Drug(y_1) \wedge A.\#tak(x,y_1)$ where $P(\#tak, \#Pat, takes, \#Drug)$ is in the object base. Thus, $In.\#Drug(y_1)$ can be cancelled. Analagously, $In.\#Sympt(s) \wedge A.\#suff(x,s)$ reduces to $A.\#suff(x,s)$. The result is

$$\begin{array}{c} \forall\ x, y_1\ In.\#MP(x) \wedge In.\#OP(x) \wedge A.\#tak(x,y_1) \wedge \\ \neg\exists\ s\ A.\#suff(x,s) \wedge A.\#ag(y_1,s) \Rightarrow Q_3(x,y_1) \end{array} \tag{5}$$

The first two instantiation literals are not redundant since they are subclasses of the class $\#Pat$, and membership to $\#Pat$ does not imply membership to $\#MP$ or $\#OP$. Whether the elimination of the instantiation literals yields a performance gain depends on the representation and evaluation algorithms. The point is that we now have the **choice** to use either the original or the reduced formula.

4.2. Complex Object View Optimization

A view is a query whose answer is used over a longer period of time. The view update problem is twofold. First, an update to the base data has to be efficiently propagated to the view in order to keep the answer consistent. Second, an update on the view has to be translated into updates on the base data. For *deductive* databases, views are seen as (materialized) deductive rules. In that framework, the first subproblem could be solved by re-evaluating the whole deductive rule on each update (this is much too costly). For the second subproblem, abduction has been proposed [KM90b]. In this section, we propose a deductive definition for views that takes the complex object principle into account. Simplification [NICO82] is proposed as an optimization technique for the first direction of view updates (*view maintenance*).

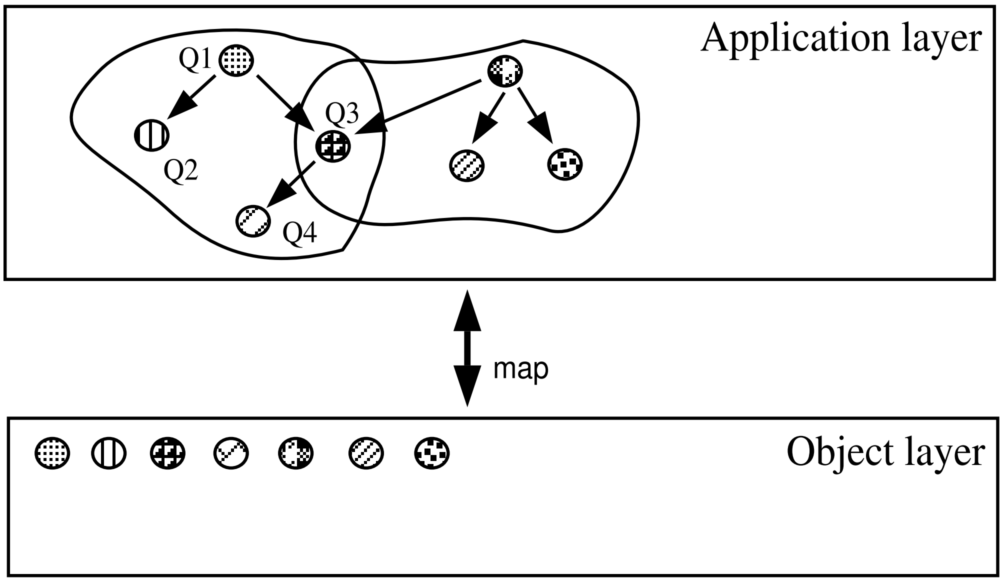

Fig. 4: Molecules and atoms in PRIMA

While views in relational databases are just (derived) relations, the definition for object bases seems not that clear. For instance, the PRIMA object base system [HM88] introduces the concept of *molecules*. These complex data structures are declared by means of SQL-like query statements on top of an object base of *atoms*. The definition of molecules depends on the application that manipulates them. Figure 4 shows two molecules which share a common atom. It is important to note that the molecule boundaries is prescribed by the application, not by the object base.

In a logic-based framework for an object base the usual representation for complex objects are complex terms [CCT90]. However, such representation forbids the direct use of deductive database methods that are based on a function-free horn logic. An alternative is shown in definition 2.

Definition 2

Let $Q_1, ..., Q_k$ be the conclusion predicates of query rules. Then, a rule

$$\forall\ v_1, ..., v_m\ Q_1(x_1, t_{11}, ..., t_{n_1 1}) \wedge Q_2(x_2, t_{12}, ..., t_{n_2 2}) \wedge ...$$
$$\wedge\ Q_k(x_k, t_{1k}, ..., t_{n_k k}) \Rightarrow Q(x_1)$$

is called a **complex object view** iff all $x_i, i > 1$, appear at least once as a t_{rj} of a Q_j.

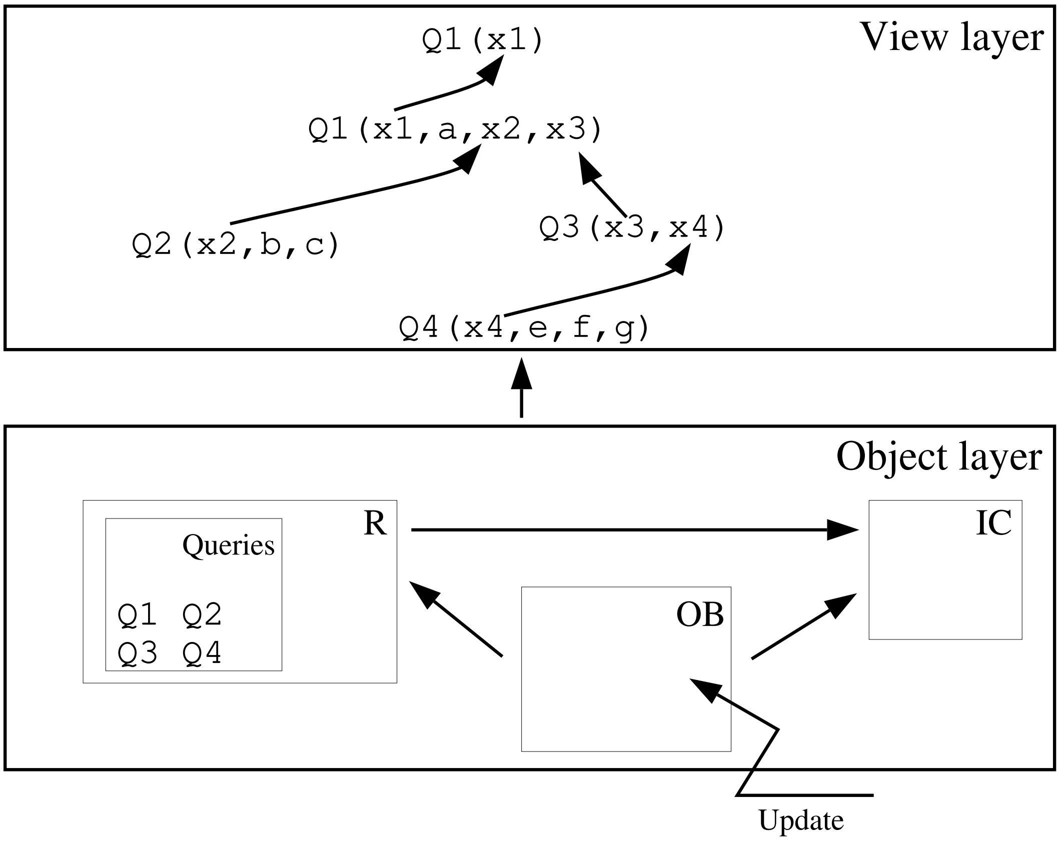

Fig. 5: Complex object and update propagation

Figure 5 presents a typical example. The complex object identified by x_1 has an attribute a, and two part complex objects identified by x_2 and x_3. The complex object x_2 is not further decomposed, whereas x_3 has another part complex object x_4. The object layer of fig. 5 shows the principal propagation of an update in the object base. The update is propagated to the simplified forms of deductive rules and integrity constraints. The idea is to use the same technique for the maintenance of the view.

The complex object rule in definition 2 has two interpretations. The first is just the logical statement, i.e. all the ground facts $Q(x_1)$ that are derivable from the deductive object base. The set of these object identifiers can be seen as the set of pointers to the complex objects. The second interpretation is more algorithmic. The complex object rule is a statement which of the ground facts Q_i shall be stored and maintained in the view layer. All items contributing to a view are either ground facts of the object base or deductive rules, esp. queries are mapped to deductive rules (sec. 3). Therefore, methods from deductive database theory appear to be most appropriate to perform view maintenance optimization. The problem is as follows:

> Given a sequence of updates
>
> $$< op_1(P(o_1, x_1, l_1, y_1)), op_2(P(o_2, x_2, l_2, y_2)), ... > \quad (op_i \in \{Insert, Delete\})$$
>
> find at lowest costs the induced update for a complex object $Q(x)$.

A subproblem is to find – starting from the object base update – the set of deductive rules that are affected by the update. The simplification method [NICO82,BDM88] gives an answer to this question for integrity constraints:

> Let $op(L)$ be an update and L' be a matching literal occuring in an integrity constraint φ. Then it is sufficient to check the simplified form $\varphi_{op(L)}$. If no matching literal occurs then the constraint is not affected by this update.

The simplification idea can be extended to deductive rules as well, see e.g. [OLIV91]. For each rule r and each update $op(L)$ that matches a literal L' occuring in the rule body a simplified rule $r_{op(L)}$ is generated. If L' is positive and op=*Insert* then the evaluation of $r_{op(L)}$ yields a superset of the derived conclusions that are true due to the truth of L. Similar cases hold for negative L's and for delete operations. The original methods were proposed for deductive integrity checking for efficient computation of the derived facts that affect an integrity constraint.

A simple trick extends the upward propagation to complex object views: Include a formal constraint

$$\forall\ x/\#Obj\ Q(x) \vee \neg Q(x)$$

into the set IC of the deductive object base (OB,R,IC). Then, each insertion and deletion of a subobject $Q_i(x_i, ...)$ will be efficiently propagated from updates to the object base. To store the solutions in the view is a matter of implementation, see e.g. [SLR89].

As an example, consider the following complex object view (see also fig. 5). The query Q_3 is taken from section 3.

$$\begin{aligned} &\forall\ x_1, x_2, x_3, x_4, a, b, c, e, f, g\ \ Q_1(x_1, a, x_2, x_3) \wedge Q_2(x_2, b, c) \wedge \\ &Q_3(x_3, x_4) \wedge Q_4(x_4, e, f, g) \Rightarrow Q(x_1) \end{aligned} \tag{6}$$

We assume that the view already contains the complex object identified by $\#drSm$:

$$
\begin{array}{l}
Q_1(\#drSm, "Dr.med.Smith", \#KlinAc, \#Sam) \\
Q_2(\#KlinAc, 5100, "KlinikumAachen") \\
Q_3(\#Sam, \#Aspi) \\
Q_3(\#Sam, \#Fen) \\
Q_4(\#Aspi, 17, 3) \\
Q_4(\#Fen, 7, 21)
\end{array}
$$

With the example of section 2, the update is

$$
< Delete(P(\#dr2, \#Sam, drug2, \#Asp)), Delete(P(\#in5, \#dr2, in, \#tak)) >
$$

deletes the literal $A.\#tak(\#Sam, \#Asp)$ due to axiom A_4. That derived update affects rule (5) for query `Q3` (see subsection 4.1). The simplified form for this update is

$$
\begin{array}{l}
In.\#MP(\#Sam) \wedge In.\#OP(\#Sam) \wedge \\
\neg\exists\ s\ A.\#suff(\#Sam, s) \wedge A.\#ag(\#Asp, s) \Rightarrow Q_3(\#Sam, \#Asp)
\end{array}
\tag{7}
$$

Assuming that the condition holds, we derive the deletion of $Q_3(\#Sam, \#Asp)$ which triggers the complex object view. The fact $Q_3(\#Sam, \#Aspi)$ is removed from the view. In an implementation, facts with identical first components can be aggregated by using a set data structure,. e.g.

$$
CO_3(\#Sam, \{\#Aspi, \#Fen\})
$$

That gives us a restricted simulation of the set constructor found in other object bases. The update above would then effectively remove the element $\#Aspi$ from the list. The dangling fact $Q_4(\#Aspi, 17, 3)$ should then also be removed since it not longer contributes to the complex object. A small problem arises if the last fact of a predicate symbol is removed, e.g. $Q_3(\#Sam, \#Fen)$. Then, the condition of the view (6) fails, and $Q(\#Sam)$ is no longer true. In a strict interpretation, the complex object $\#Sam$ has to be totally removed from the view. Alternatively, one could interpret this situation by an empty set in the data structure:

$$
CO_1(\#drSm, \{"Dr.med.Smith"\}, \{\#KlinAc\}, \{\})
$$

Another optimization opportunity with views is to use their content as ranges for ad hoc queries. The general idea is to exploit subset relationships between answers to queries. If the query `QA` is a superclass of `QB`, then each answer to `QB` is also an answer to `QA`:

$$
\forall x\ Q_B(x, ...) \Rightarrow Q_A(x, ...)
$$

This result follows immediately from the transformation of superclasses in formula (1). If `QB` is queried frequently compared to `QA` then it makes sense to materialize the answer to `QA` for getting a finer range for the computation of the answer to `QB`. This materialization can be achieved by simply defining a view $\forall x\ Q_A(x, ...) \Rightarrow Q(x)$. An important application are the parameterized queries (section 3). They are superclasses of all their specializations. If the query class `Q3` is materialized by a view

$$
\forall\ x, y_1\ Q_3(x, y_1) \Rightarrow Q(x)
$$

then ad hoc specializations of `Q3`, e.g. `Q3(Fenta/wrong)` can scan the solutions of `Q3` as a range and just check the additional condition $y_1 = \#Fen$.

5. State of Implementation

The object model described in section 2 with deductive rules and integrity constraints and the presented optimization techniques are implemented in the deductive object base system ConceptBase [3] [JARK91] and its query language CBQL [STAU90]. ConceptBase has been developed since 1987 at the Universities of Passau and Aachen. It is being used in a number of projects in Europe, the US, and Canada as a knowledge-based repository for design process information.

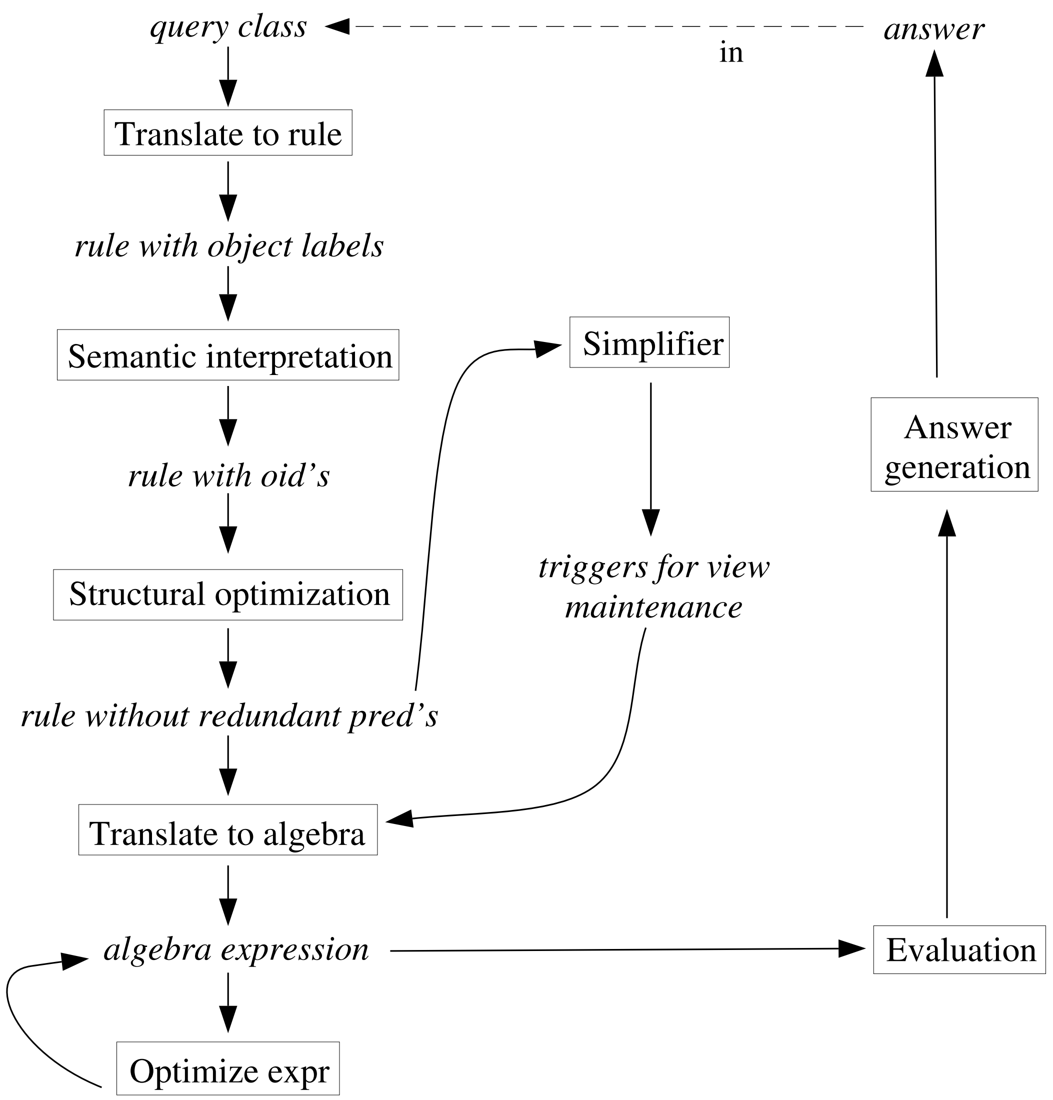

Fig. 6: Query optimization in ConceptBase

The system consists of two parts: the kernel which realizes the object base administration mostly implemented in Prolog on SUN machines, and a X11-based usage environment for

[3] In addition to the concepts mentioned in the previous sections, ConceptBase contains a temporal component for representing validity and system time for each object.

browsing, querying and editing of the object base. The communication between both parts and between the kernel and other application programs follows a client/server architecture.

The query optimizer of ConceptBase (see fig. 6) accepts as input query classes which are transformed to a query rule according to section 3. The constants which represent object labels are replaced by object identifiers, and the literals *In*,*A* are rewritten with the specialized literals. The resulting rule is then transformed by exploiting the structural axioms of the object base, esp. the attribute typing axiom. For queries which contribute to views triggers for the view update are generated (sec. 4.2). The last transformation step is the translation to the object algebra. The algebra expression can then be evaluated over the object base. The algebra optimizer is currently simulated by a direct implementation of the base literals (P, In, A, Isa) on an extensible main memory-oriented object store [GALL90].

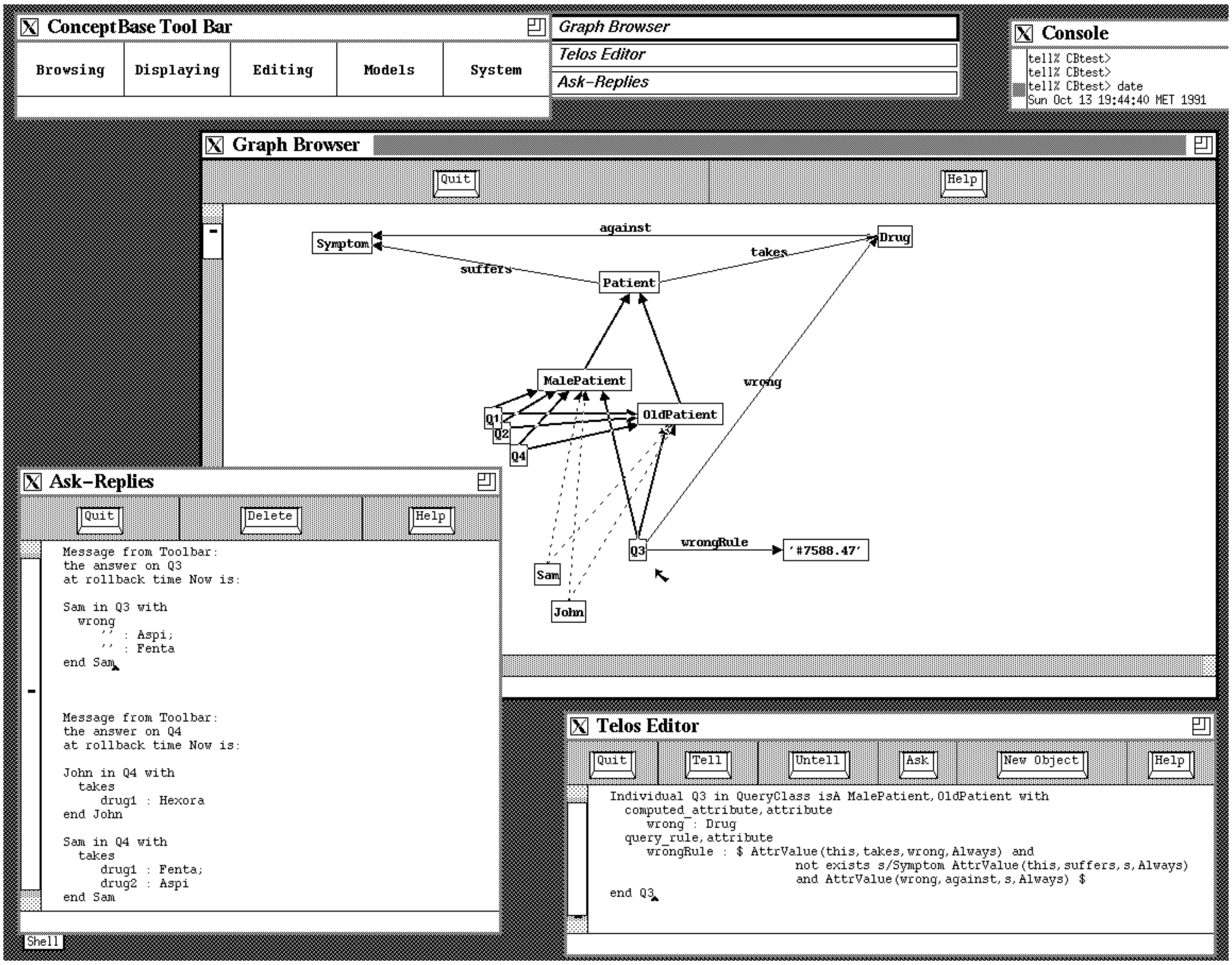

Fig. 7: Screendump of a session with ConceptBase

Figure 7 shows the X11-based user interface of ConceptBase with the graphical browser and the object editor. The browser displays the object network of the drug example in section

2 together with the query classes introduced in section 3. Individuals and attributes are represented as nodes and labelled links, specialization and instantiation relationships as dotted and normal links. The query classes are stored as objects (and therefore browsable like any other object). The frame representation of `Q3` shown in the editor slightly differs from the format in section 3. Deduced attributes of query classes belong to a special category `computed_attribute`. The literal *AttrValue* corresponds to the former *A* literal. The displayed situation is as follows: After selecting `Q3` in the graph with menu item "Ask" `Q3` is called to be evaluated i.e. the instances of `Q3` together with values for the `wrong` attribute are to be computed. This task is transmitted to the ConceptBase kernel via an inter-process message. The answers are sent back to the usage environment and are displayed in the *Ask-Replies* window in their textual frame representation. In this example, `Q3` has only a single instance of `Patient` as answer: `Sam`. `Sam` takes drugs `Aspi` and `Fenta` although he does not suffer from a symptom these drugs are against. A second result is shown for a query `Q4`.

For queries, rules and constraints the actual implementation of ConceptBase contains different evaluable internals formats which can be chosen. Forward evaluation of deductive rules and integrity constraints triggered by class instantiations is done by a meta program interpreting an internal format of the formulas. Current implementation work concentrates on improving the evaluation of deductive rules (and queries) with the magic set format. Further efforts are directed towards the application of the view maintenance mechanisms via query classes to schema integration for heterogenous databases [KLEM91].

6. Conclusions

The aim of this paper was to demonstrate that a deductive kind of object bases offers optimization opportunities that fit well together with object-orientation. The Telos object model introduced in section 2 is characterized by a total decomposition of information into quadruples forming the extensional object base. Instances, attributes, classes, instantiation and specialization links are uniformly treated as objects. As a consequence, updates on either of them are principally undistinguished. Based on this extensional object base, a first-order language for deductive rules and integrity constraints is defined. The interpretation is restricted by a stratification that is not applied to the original formulas but to a rewriting on them that replaces the fixed set of literals by literals based on object identifiers at the class level.

Our frame-like query language describes queries as classes with superclasses, attributes, and a logical statement that constrains the instances of the query class. The advantage of this notation – which is familiar with certain knowledge representation languages – is that users do not have to learn a special query language different from the data definition language. A mapping of the frame-like query classes to deductive rules defines the semantics of queries, and integrates well with (recursive) deductive rules. We only presented a small segment of query optimization at the level of logical formulae:

- ▷ Instantiation literals introduced by the class ranges of variables can be eliminated by applying the attribute typing axiom. Experiments indicate that the efficiency increase is about factor 2 - 5. But, the axioms of the object model hold in any object base. Thereby, this optimization can be applied to virtually any query, deductive rule, and integrity constraint inserted into the system.

▹ If complex objects are defined as function-free deductive rules, then well-known update propagation methods become applicable. We presented a definition that is able to simulate tuple and set constructors. Views can also be used to materialize queries that serve as ranges for ad hoc queries specialized from them.

During the upward propagation of updates to views, a lot of intermediate updates on derived facts can be generated, each triggering another simplified rule. Surely, the proposed propagation method is not efficient enough if the amount of rules or their complexity exceeds a certain level. A possible solution is to restrict views and rules in a way that allows further optimization.

A major application of the deductive definition of complex objects is the software configuration problem [RJG*91]. Software systems are composed from modules that are further decomposed. Updates to components have to be propagated downwards to submodules (this is a special abduction problem [KM90b]), and upwards to the modules (complex objects) that depend on the updated object (this is a special case of the view update problem in section 4.2). Currently, we are investigating the special properties of complex objects in software configuration. Such properties are for example, that integrity checking can be drastically simplified if the constraints only relate configured objects. Another research issue is to find a good mapping from the logical layer into the structural layer. We expect that the optimization methods based on rewritings of algebraic expressions can then be transfered to our storage system.

Acknowledgements. The authors would like express their thanks to Matthias Jarke for his constant advice. Many thanks also to the participants of the Dagstuhl seminar for the vivid discussions and important hints.

7. References

[ABRI74] Abrial,J.R. (1974). Data semantics. In Klimbie, Koffeman (eds.): Data Base Management, North-Holland Publ.

[AB91] Abiteboul,S., Bonner,A. (1991). Objects and views. *Proc. ACM-SIGMOD 1991.*

[AG91] Abiteboul,S., Grumbach,S. (1991). A rule-based language with functions and sets. *ACM Trans. on Database Systems* 16(1), March 1991.

[AK89] Abiteboul,S., Kanellakis,P.C. (1989). Object identity as a query language primitive. *Proc. ACM-SIGMOD 1989.*

[BBMR89] Borgida,A., Brachman,R.J., McGuiness,D., Resnick,L.A. (1989). CLASSIC: A structural data model for Objects. *Proc. ACM-SIGMOD 1989.*

[BCD89] Bancilhon,F., Cluet,S., Delobel,C. (1989). A query language for the O2 object-oriented database system. Rapport Technique Altaïr 35-89.

[BCL89] Blakely,J.A., Coburn,N., Larson,P.-A. (1989). Updating derived relations: detecting irrelevant and autonomously computable updates. *ACM Trans. on Database Systems* 14(3), Sept. 1989.

[BEER90] Beeri,C. (1990). A formal approach to object-oriented databases. *Data & Knowledge Engineering* 5.

[BGL85] Brachman,R.J., Gilbert,V.P., Levesque,H.J. (1985). An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. *Proc. Int. Joint. Conf. on Artificial Intelligence*

[BGN89] Beck,H.W., Gala,S.K., Navathe,S.B. (1989). Classification as a query processing technique in the CANDIDE semantic data model. *Proc. 5th Int. Conf. on Data Engineering*

[BKK88] Banerjee,J., Kim,W., Kim,K.C. (1988). Queries in object-oriented databases. *Proc. 4th Int. Conf. on Data Engineering*

[BR86] Bancilhon,F., Ramakrishnan,R. (1986). An amateur's introduction to recursive query processing strategies. *Proc. ACM-SIGMOD 1986.*

[CCT90] Ceri,S., Cacace,F., Tanca,L. (1990). Object orientation and logic programming for databases: a season's flirt or a long-term marriage? In Schmidt, Stogny (eds.): Next Generation Information System Technology, *LNCS* 504, Springer-Verlag, 1990.

[CGM90] Chakravarthy,U.S., Grant,J., Minker,J. (1990). Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems* 15(2), Juni 1990.

[CW85] Cardelli.L., Wegner,P. (1985). On understanding types, data abstraction, and polymorphism. *Computing Surveys* 17(4), Dec. 1985.

[CW90] Ceri,S., Widom,J. (1990). Deriving production rules for constraint maintenance. *Proc. 16th VLDB Conf.*, Brisbane, Australia.

[CW91] Ceri,S., Widom,J. (1991). Deriving production rules for incremental view maintenance. *Proc. 17th VLDB Conf.*, Barcelona, Spain.

[DD88] Duhl,J., Damon,C. (1988). A performance comparison of object and relational databases using the Sun benchmark. *OOPSLA'88 Conference Proceedings.*

[FMV91] Freytag,J.C., Maier,D., Vossen,G. (eds.,1991). Query processing in object-oriented, complex-object and nested relation databases. Seminar-Report 15, Schloß Dagstuhl, Germany.

[FREY87] Freytag,J.C. (1987). A ruled-based view of query optimization. *Proc. ACM-SIGMOD 1987.*

[GALL90] Gallersdörfer,R. (1990). Realisierung einer deduktiven Objektbank durch abstrakte Datentypen. Diploma thesis, Universität Passau, Germany.

[GD87] Graefe,G., DeWitt,D.J. (1987). The EXODUS optimizer generator. *Proc. ACM-SIGMOD 1987.*

[GUTT75] Guttag,J.V. (1975). The specification and application to programming of abstract data types. Ph.D. thesis, University of Toronto, Dept. of Computer Science, Technical Report CSRG-59.

[HM88] Hübel,C., Mitschang,B. (1988). Object-Orientation within the PRIMA-NDBS. In Dittrich (ed.): Advances in Object-Oriented Database Systems, *Proc. 2nd Int. Workshop on Object-Oriented Database Systems*, Bad Münster, Germany, Sept. 1988, Springer Verlag.

[HY90] Hull,R., Yoshikawa,M. (1990). ILOG: declarative creation and manipulation of object identifiers. *Proc. 16th VLDB Conf.*, Brisbane, Australia.

[JACK90] Jackson, M.S. (1990). Beyond relational databases. *Information and Software Technology* 32(4), May 1990.

[JARK84] Jarke,M. (1984). External semantic query simplification: a graph-theoretic appoach and its implementation in Prolog. *Proc. 1st Int. Workshop Expert Database Systems*, Kiawah Island, South Carolina, 1984.

[JARK91] Jarke,M. (ed.,1991). ConceptBase V3.0 user manual. Report MIP-9106, Universität Passau, Germany.

[JEUS91] Jeusfeld,M. (1991). Axiome des Telos-Datenmodells. Working paper, Universität Passau, Germany, Sept. 1991.

[JJ91] Jeusfeld,M., Jarke,M. (1991). From relational to object-oriented integrity simplification. *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases*, München, Germany, Dec. 1991; also in: Aachener Informatik-Berichte 91-19, RWTH Aachen, Germany.

[JJR89] Jarke,M., Jeusfeld,M., Rose,T. (1989). Software process modeling as a strategy for KBMS implementation. In Kim, Nicolas, Nishio (eds.): Deductive and Object-Oriented Databases, North Holland, 1990.

[JK89] Jarke,M., Koubarakis,M. (1989). Query optimization in KBMS: overview, research issues, and concepts for a Telos implementation. Technical report KRR-TR-89-6, University of Toronto, Ontario.

[JK90] Jeusfeld,M., Krüger,E. (1990). Deductive integrity maintenance in an object-oriented setting. Report MIP-9013, Universität Passau, Germany.

[KLEM91] Klemann,A. (1991). Schemaintegration relationaler Datenbanken. Diploma thesis, Universität Passau, Germany.

[KLW90] Kifer,M., Lausen,G., Wu,J. (1990). Logical foundations of object-oriented and frame-based languages. *Reihe Informatik* 3/1990, Universität Mannheim, Germany.

[KM90a] Kemper,A., Moerkotte,G. (1990). Access support in object bases. *Proc. ACM-SIGMOD 1990.*

[KM90b] Kakas,A,C., Mancarella,P. (1990). Database updates through abduction. *Proc. 16th VLDB Conf.*, Brisbane, Australia.

[KMSB89] Koubarakis,M., Mylopoulos,J., Stanley, M., Borgida,A. (1989). Telos: features and formalization. Technical Report KR-89-04, University of Toronto, Ontario.

[LHW90] Lyngbæk,P., Wilkinson,K., Hasan,W. (1990). The Iris kernel architecture. *Proc. EDBT'90*, Venice, Italy.

[MAIE86] Maier,D. (1986). Why object-oriented databases can succeed where others have failed. *Proc. 1986 Int. Workshop on Object-Oriented Database Systems.*

[MBJK90] Mylopoulos,J., Borgida,A., Jarke,M., Koubarakis,M. (1990). Telos: a language for representing knowledge about information systems. *ACM Trans. Information Systems* 8(4), October 1990.

[NICO82] Nicolas,J.-M. (1982). Logic for improving integrity checking in relational databases. *Acta Informatika* 18.

[RJG*91] Rose,T., Jarke,M., Gocek,M., Maltzahn,C., Nissen,H. (1991). A decision-based configuration process environment. Appears in: *Software Engineering Journal*, Special Issue on Software Process and its Support.

[SLR89] Sellis,T., Lin,C.-C., Raschid,L. (1989). Data intensive production systems: the DIPS approach. *SIGMOD Record* 18(3), Sept. 1989.

[REIT84] Reiter,R. (1984). Towards a logical reconstruction of relational database theory. In Brodie et al. (eds.): On Conceptual Modelling, Springer-Verlag.

[STAU90] Staudt,M. (1990). Anfragerepräsentation und -auswertung in deduktiven Objektbanken. Diploma thesis, Universität Passau, Germany.

[SZ90] Shaw,G.M., Zdonik,S.B. (1990). Object-oriented queries: equivalence and optimization. In Kim, Nicolas, Nishio (eds.): Deductive and Object-Oriented Databases, North Holland, 1990.