A Semantics-Driven Query Optimizer for OODBs

Jean Paul Ballerini*, Domenico Beneventano◇,
Sonia Bergamaschi◇, Claudio Sartori°, Maurizio Vincini◇

Int. Workshop on Description Logics, Rome, Italy. june 1995

1 DLs techniques for OODB query optimization

Semantic query optimization uses problem-specific knowledge (e.g. integrity constraints) for transforming a query into an *equivalent* one (i.e., with the same answer set) that may be answered more efficiently. The optimizer is applicable to the class conjunctive queries is based on two fundamental ingredients. The first one is the *ODL* description logics proposed as a common formalism to express: class descriptions, a relevant set of *integrity constraints* rules (IC rules), queries as ODL types. The second one are DLs (Description Logics) inference techniques exploited to evaluate the logical implications expressed by IC rules and thus to produce the *semantic expansion* of a given query. The optimizer tentatively applies all the possible transformations and delays the choice of beneficial transformation till the end. Some preliminar ideas on filtering activities on the *semantically expanded query* are reported. A prototype semantic query optimizer (*ODB-QOptimizer*) for object-oriented database systems (OODBs) is described.
Let us briefly explain the main ingredients of our approach for semantic query optimization, firstly proposed in [4, 3].

- ODL *description logics for generalized database schema*

ODL (Object Description Logics) was proposed in [5] and extends the expressiveness of implemented description logics languages in order to represent the semantics of complex object data models (*CODMs*), recently proposed in the areas of deductive databases [1] and object oriented databases [6]. For instance,

*Centro Interdipartimentale di Ricerca in Filosofia del Diritto e Informatica Giuridica, Università di Bologna, Via Galliera 3, I-40121 Bologna, Italy.

° Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy.

◇ Dipartimento di Scienze dell'ingegneria, Università di Modena, Via G. Campi 213/B, I-41100 Modena, Italy.

e_mail: {dbeneventano, sbergamaschi, csartori}@deis.unibo.it

a distinction between *values* and *objects* with identity and, thus, between *value types* and *class types* (briefly called classes) has been introduced (i.e. concepts are partitioned into the ones representing objects sets and the one representing values sets); type constructors, such as *tuple*, *set* and *sequence* often supported by CODMs are directly expressible. Most importantly, the representation and management of *cyclic classes*, i.e., classes which directly or indirectly make reference to themselves, as CODMs usually require, are supported.

The extension to ODL introduced in the present work allows the declarative formulation of a relevant set of database integrity constraints. Actual database schemata are, in fact, given in terms of base classes (i.e. primitive concepts) while further knowledge is expressed with integrity constraints, which should guarantee data consistency. In general, constraints go beyond DLs expressivity and are expressed in various fashions, depending on the database data model: e.g. subsets of first order logic, inclusion dependencies and predicates on row values, procedural methods in OO environments. In particular, we extend ODL by: *quantified path types* and *integrity constraints* rules. The former extension has been introduced to deal easily and powerfully with nested structures. Paths, which are essentially sequences of attributes, represent the central ingredient of OODB query languages to navigate through the aggregation hierarchies of classes and types of a schema. In particular, we provide *quantified* paths to navigate through set types. The allowed quantifications are existential and universal and they can appear more than once in the same path. By means of path types (a path type is a type associating a path to an ODL type) we can express a relevant set of integrity constraints.

Viewing a database schema as a set of ODL *inclusion statements* [7] allows the declarative formulation of another relevant set of integrity constraints, expressing *if then rules* whose antecedent and consequent are ODL *virtual* types (i.e. defined concepts). For example, it is possible to express correlations between structural properties of the same class or sufficient conditions for populating subclasses of a given class. A *generalized database schema* can be thus defined as a set of inclusion statements between general ODL types. Inclusion statements constitute a generalization of the description logics perspective which perfectly fits the usual database viewpoint.

In order to illustrate our approach, let us introduce the Company domain example which describes part of the organizational structure of a company (see table 1 for its ODL description).

Classes descriptions are given with a syntax richer than usual DLs (see type constructors and paths). As an example, the class Storage of table 1 describes departments with a category property and stocking materials all of quantity between 10 - 300. Let us briefly describe in words IC rules of table 1:

R_1 says that, having a risk greater equal to 10, is a sufficient condition for a material to be an smaterial. Note that it is different from giving for smaterial the following description: $\texttt{Material} \sqcap \triangle[\texttt{risk}\colon 10 \div \infty]$, as this last description expresses a necessary condition whereas R_2 expresses a sufficient condition). R_2,

$$
Classes \quad : \quad \begin{cases}
\texttt{Manager} & \sqsubseteq \quad \triangle[\texttt{name}: \texttt{String}, \texttt{salary}: 40 \div \infty, \texttt{level}: 5 \div 15] \\
\texttt{Department} & \sqsubseteq \quad \triangle[\texttt{name}: \texttt{String}, \texttt{managed-by}: \texttt{Manager}] \\
\texttt{TManager} & \sqsubseteq \quad \texttt{Manager} \sqcap \triangle[\texttt{level}: 8 \div 12] \\
\texttt{Material} & \sqsubseteq \quad \triangle[\texttt{name}: \texttt{String}, \texttt{risk}: \texttt{Int}, \texttt{feature}: \forall\{\texttt{String}\}] \\
\texttt{SMaterial} & \sqsubseteq \quad \texttt{Material} \\
\texttt{Storage} & \sqsubseteq \quad \texttt{Department} \sqcap \triangle[\ \texttt{category}: \texttt{String}, \\
 & \qquad\qquad \texttt{stock}: \forall\{[\texttt{item}: \texttt{Material}, \texttt{qty}: 10 \div 300]\}] \\
\texttt{SStorage} & \sqsubseteq \quad \texttt{Storage}
\end{cases}
$$

$$
IC\ rules \quad : \quad \begin{cases}
R_1 \quad : & \texttt{Material} \sqcap (\triangle.\texttt{risk}: 10 \div \infty) \sqsubseteq \texttt{SMaterial} \\
R_2 \quad : & \texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}: \texttt{SMaterial}) \sqsubseteq \texttt{SStorage} \\
R_3 \quad : & \texttt{Storage} \sqcap (\triangle.\texttt{managed-by}.\ \triangle\ .\texttt{level}: 6 \div 12) \sqsubseteq (\triangle.\texttt{category}: \text{"}A2\text{"}) \\
R_4 \quad : & \texttt{Storage} \sqcap (\triangle.\texttt{category}: \text{"}A2\text{"}) \\
 & \sqcap(\triangle.\texttt{stock}.\exists.\texttt{item}.\ \triangle\ .\texttt{risk}: 10 \div \infty)) \sqsubseteq (\triangle.\texttt{managed-by}: \texttt{TManager})
\end{cases}
$$

Table 1: The Company domain schema

R_3 and R_4 are rules on storages: R_2 says that, stocking at least an smaterial is a sufficient condition for a storage to be an sstorage; R_3 says that, if a storage is managed by a manager with a level from 6 to 12, it is of category "A2". R_4 constrains storages of category "A2", stocking at least a material having a risk greater equal to 15, to be managed by a tmanager.

Let us give as an example on paths and of the relevance of multiple quantifications to increase expressivity. The type $S = (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{feature}.\exists\text{: }\text{"}F1\text{"})$ individuates all the domain objects (objects are denoted by the $\triangle$ operator) such that *at least* one of the stocked materials has a feature *"F1"*. On the other hand, the type $S' = (\triangle.\texttt{stock}.\forall.\texttt{item}.\ \triangle\ .\texttt{feature}.\exists\text{: }\text{"}F1\text{"})$ individuates *all* the domains objects such that *all* the stocked materials has a feature *"F1"*.

- *Queries and DLs inference techniques*

A relevant set of queries, corresponding to the so called single-operand queries [8] , can be expressed as *virtual* ODL types. Note that, this assumption is an important restriction as it excludes operations comparable to relational joins. However, it is important to realize that join is a crucial operation in relational databases largely because of the limitation of the relational model of data. In OODB environment we have an *implicit join* of the classes on a class-composition hierarchy rooted at the target class of the query. Subsumption computation, incoherence detection and canonical form generation can be used to produce the *semantic expansion* $EXP(Q)$ of a query Q. It is a transformed query which incorporates any possible restriction which is not present in the original query but is *logically implied* by the query and by the overall schema (classes + IC rules). Following the approach of [9, 10] for semantic query optimization, we perform the semantic expansion of the types included at each nesting level in the query description. Type expansion is based on the iteration of this simple transformation: if a type implies the antecedent of an IC rule then the consequent of that rule can be added. Logical implications between

these types (the type to be expanded and the antecedent of a rule) are evaluated by means of *subsumption computation* [5]. Following this approach, the semantic expansion of a query coincides with the *most specialized query* among all the semantically equivalent ones[4]. Moreover, during the transformation, we compute and substitute in the query, at each step, the *most specialized generalization classes* satisfying the query. Let us give two simple query optimization examples related to Company schema.
Q: "Select storages storing *at least* a material having a risk ≥ 15"

$$Q = \texttt{Storage} \sqcap (\texttt{stock}.\exists.\texttt{item}\colon \texttt{Material} \sqcap (\triangle.\texttt{risk}\colon 15 \div \infty))$$

The applicable rules are R_1, R_2, leading to the semantic expansion of Q:

$$EXP(Q) = \texttt{SStorage} \sqcap (\texttt{stock}.\exists.\texttt{item}\colon \texttt{SMaterial} \sqcap (\triangle.\texttt{risk}\colon 15 \div \infty))$$

In this way, we can optimize the query by obtaining the *most specialized generalization* of the classes involved in the query `SStorage` and `SMaterial`.

2 Detection of eliminable factors

The above example show an effective query optimization, independent from any specific cost model and due to the substitution of the classes mentioned in the query with their specializations. In fact, by substituting `Storage` with `SStorage` and `Material` with `SMaterial`, we reduced the number of classes to be accessed to process the query as we implicitely eliminate the need to access the classes `Material Storage`.

A same general query optimization can be obtained if we reduce the number of classes on a class-composition hierarchy rooted at the query. In the literature [9, 10] this problem is generally addressed as *constraint removal*, i.e., removal of implied restrictions from the query.

Our approach is to detect the *eliminable* factors of a query, that is, the factors logically implied by the query, by exploiting subsumption. Furthermore, as we will show in the following this activity must be performed on the semantic expansion of a query to be conducted to a maximum degree.

As an example, let us consider the query:
Q_1: "Select storages of category *"A2"* managed by a `TManager`":

$$Q_1 = \underbrace{\texttt{Storage} \sqcap (\triangle.\texttt{managed-by}\colon \texttt{TManager})}_{S_1} \sqcap \underbrace{(\triangle.\texttt{category}\colon \textit{"A2"})}_{S_2}$$

By rule R_3 and by the description of `TManager`, we have, with respect to the overall schema, S_2 subsumes S_1 and thus, S_2 can be eliminated from Q_1.

If we test the redundancy of a factor against the query, we exploit only sufficient conditions to eliminate a factor. We can have necessary and sufficient conditions if we test the redundancy of a factor against the semantic expansion of the query. This is shown by the following example:

$$Q_2 = \underbrace{\texttt{Storage} \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}.\triangle.\texttt{risk}: 10 \div \infty)}_{S_3} \sqcap \underbrace{(\triangle.\texttt{managed-by}: \texttt{TManager})}_{S_4}$$

In this case, S_4 does not subsume S_3, thus, S_4 is not redundant w.r.t. the query Q_2. However, if we consider the semantic expansion of the query Q_2:

$$EXP(Q_2) = \underbrace{(\triangle.\texttt{managed-by}: \texttt{TManager})}_{S_4} \sqcap$$
$$\underbrace{\texttt{SStorage} \sqcap (\triangle.\texttt{category}: \text{"}A2\text{"}) \sqcap (\triangle.\texttt{stock}.\exists.\texttt{item}: \texttt{SMaterial} \sqcap (\triangle.\texttt{risk}: 10 \div \infty))}_{S_5}$$

S_4 is redundant w.r.t. $EXP(Q_2)$, as, S_4 subsumes S_5. The query Q_2 can thus be reduced to S_5, avoiding the access to the class `TManager`.

3 ODB-QOPTIMIZER : a semantics driven query optimizer

ODB-QOPTIMIZER is composed of two modules: the ODL-DESIGNER prototype [2] and the GES prototype.

ODL-DESIGNER was developed in Sicstus Prolog at CIOC-CNR, Bologna, as part of the LOGIDATA$^+$ project [2] and implements DLs reasoning techniques for advanced database management systems handling complex objects data models. It is an active tool which supports automatic building of type taxonomies for complex object database systems, preserving coherence and minimality with respect to inheritance. It implements the theoretical framework of [5]. Due to the generality of the ODL formalism, this tool can be used as a kernel component for schema acquisition for object-oriented database system.

GES was developed at the Department of Engineering Sciences of the University of Modena to produce the semantic expansion of a query Q. It has been developed in C, to achieve better performance.

The current version of ODB-QOPTIMIZER is a quite rudimentary quick prototype, including extensions to ODL-DESIGNER to support if then rules. Its main limits include a keyboard-based user interface and a loose coupling between the different programming environments Prolog and C.

As you can see in figure 1, having a pre-existing ODL database schema *previously pre-processed* (see below) by ODL-DESIGNER we can insert a query Q and begin its expansion by activating ODB-QOPTIMIZER first. ODL-DESIGNER is activated and applies DLs techniques to the query Q (virtually added to the ODL schema) giving as output the detection of the query Q as incoherent or, if it is coherent, the list of the subsumers types (including the rules antecedents) of all the types contained at each nesting level in the query (i.e.

the file GES-IN.TXT). If the query is coherent, ODB-QOPTIMIZER activates the GES component which receives GES-IN.TXT as input file; GES expands the query ($EXP^i(Q)$) by adding all the consequents of the applicable rules as conjunction factors and sends a message to ODB-QOPTIMIZER about its execution. Two cases are distinguished: at least one rule was applied (the semantic expansion is probably not completed) or no rule was applied (the semantic expansion has been completed). In the former case, ODB-QOPTIMIZER activates ODL-DESIGNER again sending to it the GES ouput file GES-OUT.PL which contains the performed expansion $EXP^i(Q)$ as a new query, say Q'. In the latter case the iteration stops, having obtained the semantic expansion of the query $EXP(Q)$.

Performance issues are very critical for the optimization activity, since every query must be optimized at run time before its execution. We found two solutions to this problem (to be implemented in the ODL-DESIGNER C version):

- pre-processing of schema, in order to materialize the transitive closure of the classes and types hierarchies and to tag each rule with respect to the contents of its antecedent
- fast access, via indexing, to the only rules which are *relevant* to a given query.

References

[1] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[2] J. P. Ballerini, S. Bergamaschi, and C. Sartori. The ODL-DESIGNER prototype. In P. Atzeni, editor, *LOGIDATA+: Deductive Databases with complex objects*. Springer-Verlag, 1993.

[3] D. Beneventano, S. Bergamaschi and C. Sartori. Using Subsumption for Semantic query optimization in OODB. *International Workshop on Description Logics*, DFKI Technical Report D-94-10.

[4] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Using subsumption in semantic query optimization. In A. Napoli, editor, *IJCAI Workshop on Object-Based Representation Systems*, Chambery - France, August 1993.

[5] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 4,185-203 (1994).

F5

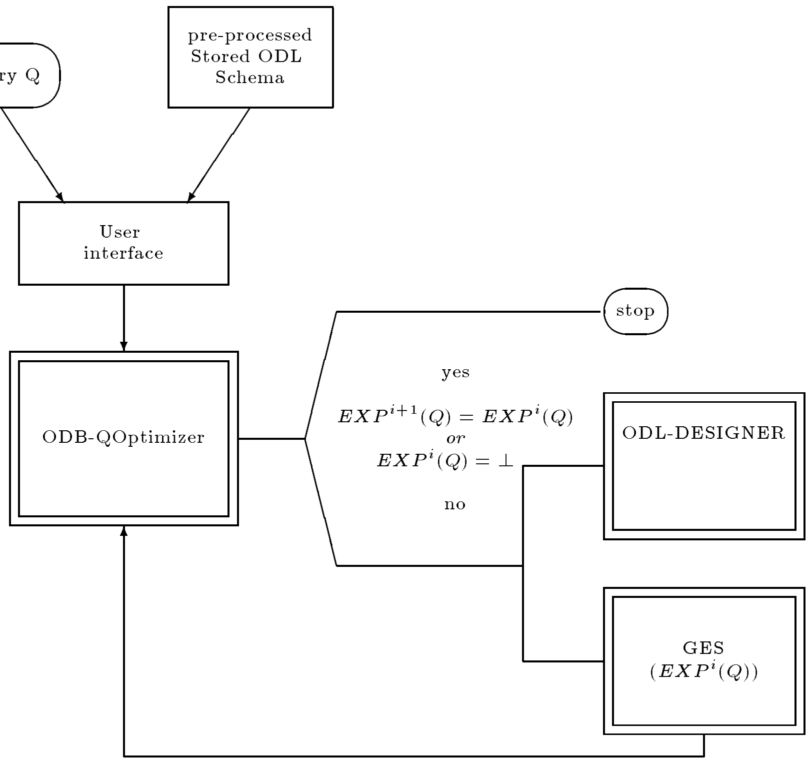

Figure 1: ODB-QOptimizerFunctional Architecture

[6] C. Lécluse and P. Richard. Modeling complex structures in object-oriented databases. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database-Systems*, pages 360–367, Philadelphia, PA, 1989.

[7] M. Buchheit, F.M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *13th. Int. Joint Conf. on Artificial Intelligence*, 1:704–709, 1993.

[8] W. Kim. A model of queries for object-oriented database systems. In *Int. Conf. on Very Large Databases*. Amsterdam, Holland, August 1989.

[9] S. Shenoy and M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. and Data Engineering*, 1(3):344–361, September 1989.

[10] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.