ODB-Tools: a description logics based tool for schema validation and semantic query optimization in Object Oriented Databases

Sonia Bergamaschi[1,2], Claudio Sartori[2], Domenico Beneventano[1], Maurizio Vincini[1]
e_mail: {dbeneventano, sbergamaschi, csartori}@deis.unibo.it
vincini@dsi.unimo.it

[1] Dipartimento di Scienze dell'Ingegneria, Università di Modena, Via G. Campi 213/B, I-41100 Modena
[2] Dipartimento di Elettronica, Informatica e Sistemistica CSITE - CNR Viale Risorgimento 2, I-40136 Bologna

Abstract.
ODB-Tools is a integrated environment for the object oriented database (OODB) validation, preserving taxonomy coherence and performing taxonomic inferences, and semantic 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 approach of the tool is based on two fundamental ingredients. The first one is the *OCDL* description logics proposed as a common formalism to express class descriptions, a relevant set of *integrity constraints* rules (IC rules) and queries. The second one are 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. ODB-Tools is a ODMG 93 [1] compliant tool, both for the schema definition (ODL language) and for the query language (OQL); The tool is available in internet at http://sparc20.dsi.unimo.it and supports an on-line graphical interface developed in Java language.

1 The approach

Let us briefly explain the main ingredients of our approach [2, 3].

OCDL*: a description logic (DL) for database schema with integrity constraints*

OCDL (Object Constraints Description Language) is a new description logics [4], extending the expressiveness of traditional description logics languages (derived from the KL-ONE model [5]) in order to represent the semantics of complex object data models. Its main characteristics are: a distinction between *values* and *objects* with identity and, thus, between *value types* and *class types* (briefly called classes); type constructors, such as *tuple*, *set* and *sequence* recursively used to define complex objects. In particular, *quantified path types* and *integrity constraints*

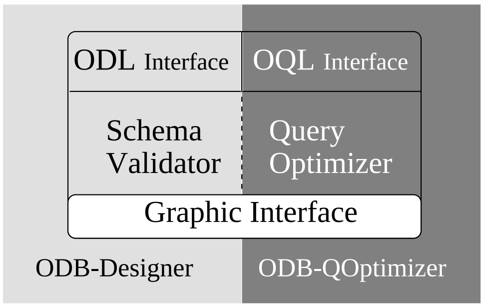

Fig. 1. ODB-Tools

rules have been introduced. Paths, which are essentially sequences of attributes, represent the central ingredient of OODB query languages to navigate through the aggregation hierarchies of classes of a schema. Quantified paths are paths existentially and universally quantified. Integrity constraints (IC) rules are *if then rules* whose antecedent and consequent can be expressed as *OCDL virtual* types (i.e. *defined* type descriptions expressing a set of sufficient and necessary conditions) and allow the declarative formulation of a relevant set of integrity constraints. A *generalized database schema* definition can be thus introduced which perfectly fits the usual database viewpoint.

Query Optimization by DLs inference techniques

A relevant set of queries, that is the ones referred to a target class and to the navigation through its composition hierarchy, can be expressed as *virtual OCDL* types. Description Logics inference techniques such as subsumption computation, incoherence detection and canonical form generation can be used to produce the *semantic expansion* of an *OCDL* query. 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 + value types + IC rules). Following the approach of [6] for semantic query optimization, but exploiting subsumption computation to evaluate logical implication, we perform the semantic expansion of the types included at each nesting level in the query description.

2 ODB-Tools Architecture

ODB-Tools, whose architecture is shown in Fig. 1, provide a *user-friendly* integrated environment based on the ODMG-93 standard, with the following features:

Schema validation and classification (ODB-Designer): The user inserts a DB schema, using ODL language, and the system performs the coherence validation and the classification, i.e., for each class, the system determines the right place of the class in the inheritance hierarchy between its most specific generalizations and its most generalized specializations. The result is shown by a graphic representation of the schema inheritance and aggregation hierarchies.

Semantic Query Optimization (ODB-QOPTIMIZER): now the user can insert a query, using ODL language, related to the given schema and the system executes the semantic query optimization. The result is the semantic expansion of the query shown by a new OQL description and by a graphic representation of the classification of the query with respect the schema.
ODB-Tools is composed of five main modules:

- **ODL Interface**: the schema acquisition module that accepts a schema description in ODL language and translate it into a *OCDL* schema. ODL syntax has been extended to provide the IC Rules descriptive capability.
- **OQL Interface**: the query input (output) interface module that receives a query in OQL language and translates it into *OCDL* syntax (and vice-versa).
- **Schema Validator**: the schema validation module which automatically builds the class taxonomy and preserves the coherence with respect to the inheritance and aggregation hierarchies.
- **Query Optimizer**: the module which executes the semantic expansion of the query (by using the Schema Validator).
- **Graphic Interface**: the module which visualizes the schema inheritance and aggregation hierarchies.

ODB-Tools is available on Internet at http://sparc20.dsi.unimo.it. The interfaces, validation and optimization modules are realized using C language (gcc 2.7.2 compiler, flex 2.5 and bison 1.24 generator), while the graphic module is developed by Java language (JDK 1.1 compiler).

3 The Storage domain example

In order to illustrate our approach, let us introduce the Storage domain example which describes part of the organizational structure of a company in an extended (integrity rules have been added) ODMG-93 syntax (see table 1). The involved classes are: Material and its subclass SMaterial, Storage and its subclass SStorage, Manager and its subclass TManager. Note that the class Storage description is quite complex: departments stocking materials all of quantity between 10 - 300 of a given category.
Let us briefly describe the IC rules of table 1:
Rule R_1 says that managers with a level from 5 to 10 must have a salary from 40K to 60K.
R_2 says that, having a risk greater equal to 10, is a sufficient condition for a material to be an smaterial.
R_3 constrains storages, with a "B4" category, to be managed by a tmanager.

Let us give a simple query optimization example using OQL syntax.

Q: "Select storages storing *all* materials having a risk ≥ 15"
select $*$ **from** `Storage`
where for all x **in** `stock` : (x.`item.risk` ≥ 15)

The applicable rules are R_2, R_3, leading to the semantic expansion of Q:
select $*$ **from** `SStorage`
where for all x **in** `stock` : (x.`item` **in select** y
from `SMaterial` **as** y
where `risk` ≥ 15)

```
interface Material () {                          interface SMaterial:Material()
   attribute string name;                        { };
   attribute int risk;
   attribute set<string> feature;  };

interface Manager ()  {                          interface TManager:Manager(){
   attribute string                name;         attribute range{8, 12} level;};
   attribute range {40000, 100000} salary;
   attribute range{1, 15}          level;  };

interface  Storage ()  {                         typedef struct t_stock
   attribute string         name;                  { Material item;
   attribute string         category;                range {10, 300} qty;
   attribute Manager        managed_by;            } t_stock;
   attribute set<t_stock>   stock; };
interface SStorage:Storage ()  { };

rule R1 forall X in Manager: ( X.level >= 5 and X.level <= 10 )
  then X.salary >= 40000 and X.salary <= 60000 ;
rule R2 forall X in Material: ( X.risk >=  10 ) then X in SMaterial ;
rule R3 forall X in Storage: ( forall X1 in X.stock: ( X1.item in SMaterial ))
  then X in SStorage ;
```

Table 1. The Storage Domain Schema

In this way, the query is optimized as we obtained the *most specialized generalization* of the classes involved in the query `SStorage` and `SMaterial`.

References

1. R.G.G. Cattell. *The Object Database Standard: ODMG-93, Release 1.1.* Morgan Kaufmann Publishers, Inc., 1994.
2. 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.
3. S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.
4. D. Beneventano, S. Bergamaschi, and C. Sartori. Semantic query optimization by subsumption in oodb. In *Int. Workshop on Flexible Query Answering Systems*, Roskilde, Denmark, May 1996.
5. R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
6. 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.

This article was processed using the LaTeX macro package with LLNCS style