

Formalizing Object-Relational Structural Metrics

Aline Lúcia Baroni¹, Coral Calero², Francisco Ruiz², Fernando Brito e Abreu¹

1 QUASAR Group

Faculty of Sciences and Technology - New University of Lisbon (Portugal)

{alinebaroni, fba}@di.fct.unl.pt

2 ALARCOS Research Group

Department of Computer Science – University of Castilla-La Mancha (Spain)

{Coral.Calero, Francisco. RuizG}@uclm.es

Abstract

This paper describes a meta-modeling approach to formalize a set of metrics suited for object-relational databases. Those metrics are expected to help database designers in their activity, offering a mechanism for comparing and selecting, among possible schema designs, the one with more quality. Formalization of metrics is important for avoiding misinterpretation of definitions, while producing metrics collection tools. The metamodel used for formalization purposes is represented as a UML class diagram and is based on the newest version of the ISO SQL:2003 standard. The formalizing technique uses OCL for expressing the metrics, which in turn allows automating metrics collection. An extract of an object-relational database schema is used to illustrate the metrics collection process using the proposed approach.

Key words: metrics; object-relational databases; ontology; OCL; SQL:2003

1 INTRODUCTION

Information systems (IS) are usually built on top of database management systems (DBMS) which, in turn, support a given database model. The object-relational (OR) database model gives additional modeling power to database designers, when compared with its relational predecessor, due to the incorporation of new data types, rules, generalizations or complex objects. Since the end of the nineties OR products [Stonebraker, 1999] such as Informix/Illustra, Oracle 8 and DB2/2 have appeared in the market. Very recently, the new SQL:2003 standard [ISO, 2003] that integrates additional OR features, has been published.

The increased OR modeling power has a price – more complex database schemata. This leads to increased maintenance effort on the information systems developed with these OR-based DBMSs. To assess and keep the complexity under control, metrics are required. These can be used for building prediction systems for database projects [MacDonell et al., 1997], for understanding and improving database development and maintenance projects [Briand et al., 1996], for maintaining the quality of the systems, for zooming in on problematic areas [De Champeaux, 1997], or for determining the best ways to help practitioners and researchers [Pfleger, 1997]. Considering the big picture, it is very important to evaluate OR database complexity to understand its contribution to the overall IS complexity.

Although there are a few published works [Calero et al. (a, b), 2001; Genero et al., 2000] on OR database metrics, to the best of our knowledge, no attempt was yet performed to formally define a set of such metrics. Formality allows clear understanding of metrics definitions which, in turn, assures that their computation can be repeated in a reliable fashion. Furthermore, the formalization itself may facilitate the automation of metrics collection, as it described in our previous work [Baroni, 2002; Baroni et al. (a, b, c), 2002; Brito e Abreu, 1994; Brito e Abreu e Carapuça, 1994].

In this paper we present the formalization of a set of metrics defined for assessing the complexity of OR database schemata. For performing this formalization we have produced an ontology for the SQL:2003, as a framework for representing the SQL schema definitions. The ontology was represented with UML and the metrics were defined with OCL (Object-Constraint Language) [OMG (a), 2003], a textual constraint language that is part of the UML 2.0 standard [OMG (b), 2003].

OCL was conceived to construct expressions above UML diagrams, in particular, above class diagrams like the ontology introduced in the next sections. It is a language that provides the power of formalization without sacrificing understandability to software practitioners. Additionally, it can be used in the analysis and design phases of the software life cycle. Its expressions can be executed upon a certain instance of a class diagram (in this case, the instantiated ontology, with concrete schemata).

In the following section the ontology for the new SQL:2003 standard is presented. In section 3, a set of metrics for OR schemata complexity evaluation and their corresponding OCL formalization are illustrated. Section 4 acquaints with a short example of a SQL:2003 schema, with its instantiation into the ontology. Finally, on the last section, conclusions and future work arise.

2 AN ONTOLOGY FOR OBJECT-RELATIONAL DATABASES

An ontology is a specification of a conceptualization. Its aim is to formalize the knowledge on a given problem domain, allowing the understanding and communication among peers through the establishment of a common repository of concepts and their inter relationships.

We propose herein an OR ontology expressed in UML that was developed based upon the information contained in the new SQL:2003 standard [ISO, 2003], namely: part 1 (Framework), part 2 (Foundation) and Part 11 (Information and Definition Schema), along with two well-known books on the topic [Melton, 2002; Melton and Simon, 2002].

For the sake of clarity, this ontology was divided in two parts. One contains the aspects related to data types (Figure 1) and the other the information about the SQL schema objects (Figure 2).

An OR database schema is composed of a number of tables related by referential integrity, which have columns that can be defined over simple or complex (user-defined) data types.

2.1 Data Types

OR databases combine the characteristics of relational databases with object-oriented principles. These database products provide the capability to define classes or abstract data types in addition to relational theory.

There are three different kinds of Data Types: *Predefined*, *Constructed Types* and *User Defined Types*. Predefined or simple data types are the classic data types such as integer, number or character. Complex or user-defined data types can be related with other data types by generalization associations. Constructed types can be *Composite* or a *Reference*. Composite types can be either *Collections* (*Arrays* or *Multisets* - a new type of the SQL:2003 standard) composed by elements, or *Row Types* which in turn are composed by *Fields*. Each Field element has a Data Type. The User Defined Types can be either *Distinct* (which are defined over a predefined Data Type) or *Structured*¹. The latter is composed by *Attributes* and *Methods*. Structured, Row and Reference Types can be defined using inheritance relationships.

¹ Structured Types are schema objects corresponding to *classes* in object-orientation. In this work, the word “class” refers to a Structured Type.

2.2 Schema Entities

The *Catalog* (as a set of *SQL Schemas*) concept is included for having an additional way of qualifying the names of the elements (*catalog_name*, *schema_element*) making namespace management easier. A SQL environment can contain zero or more Catalogs and, similarly, a Catalog can contain one or more schemata. On each Catalog there exists a *DEFINITION_SCHEMA* which contains the tables over a set of views denominated *INFORMATION_SCHEMA*.

There are four different kinds of *Schema Objects* in a Schema: *Constraints*, *Domains*, *User Defined Types* and *Tables*. Constraints can be *Assertions*, *Table Constraints* (*Unique Constraints* including *Primary Keys*, *Table Check Constraints* and *Referential Constraints* for representing the foreign keys) or *Domain Constraints*. *Domains* are used by *Columns* and can include a Domain Constraint. *Tables* can be *Derived Tables* (and particularly *Views*), *Transient Tables* or *Base Tables*. Tables are composed by *Columns* that can be defined as *Identity Columns* or *Generated Columns*. Columns can be defined over a Domain and can have Referential Constraints or Unique Constraints (or Primary Keys). Base Tables can be part of an inheritance hierarchy. They can have *Candidate Keys* and they are defined over a Data Type (through a Reference Type).

A more detailed description of this ontology, including a set of its well-formedness rules (expressed as OCL invariants) can be found in [*].

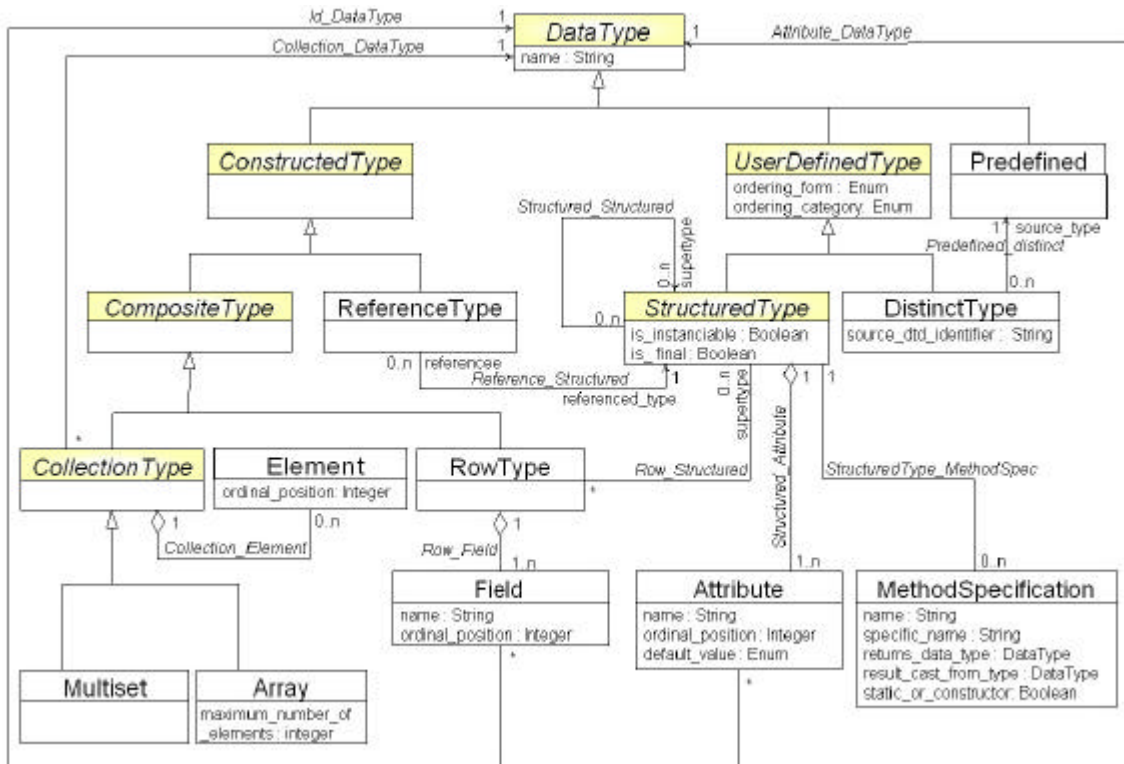


Figure 1- Object-Relational Databases Ontology (Data Types)

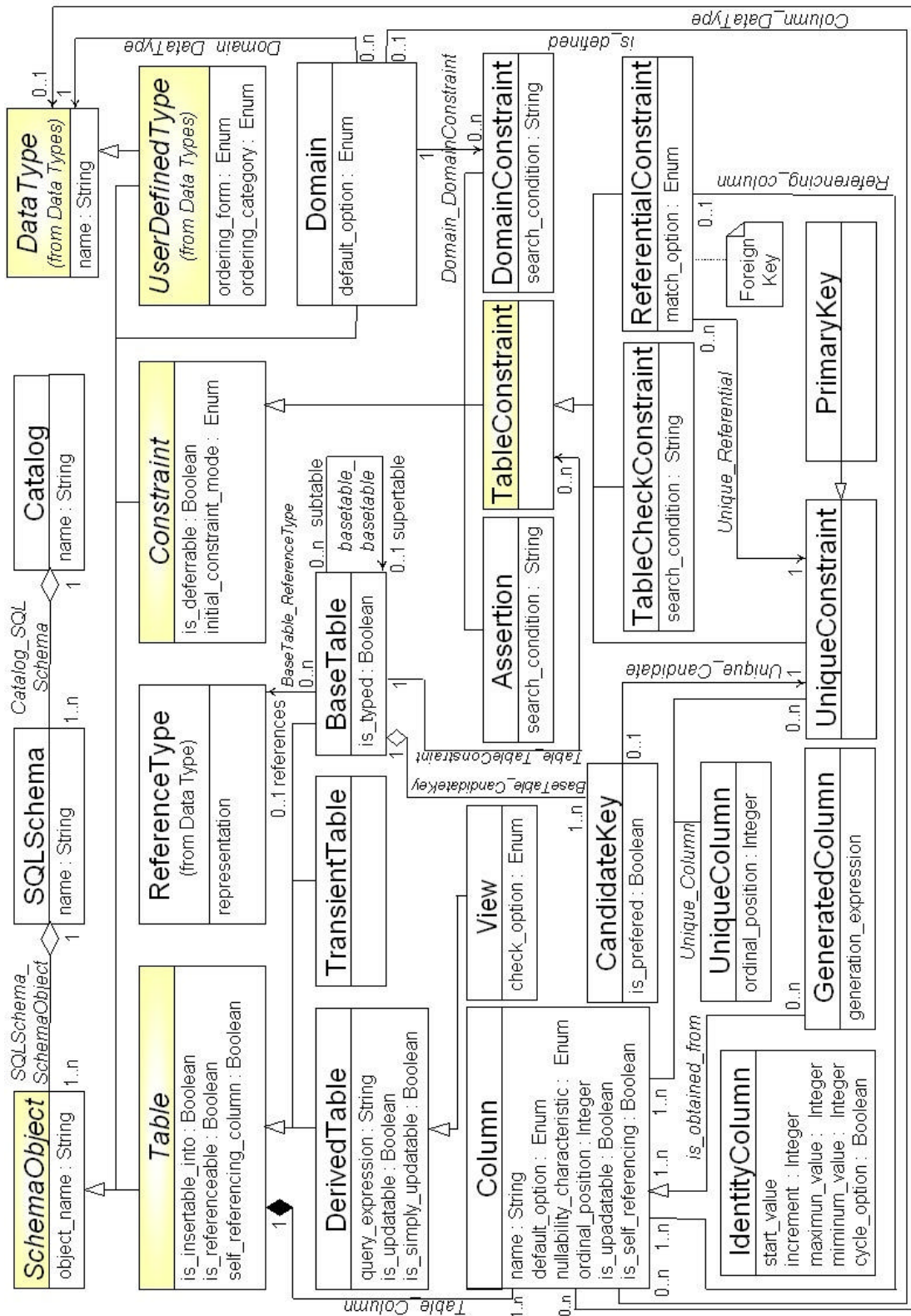


Figure 2 - Object-Relational Databases Ontology (Schema Objects)

3 METRICS FORMALIZATION FOR OBJECT-RELATIONAL DATABASES

In this section we formalize a set of metrics for OR databases, which was informally proposed in [Calero, 2001; Calero et al. (c), 2001], using the approach described in [Baroni, 2002]. For each metric we present its informal definition (alike the original proposal) and its formalization expressed as a OCL function, whose context and interface is described in the first line (underlined) and whose semantics is described in the following lines. The definitions of some intermediate functions are not included in this paper due to space restrictions, but will be fully available in a future paper. The reader is invited to follow the navigation links in the ontology (represented in Figures 1 and 2), as they are used in the metrics definitions that follow. The metrics set includes two kinds of metrics: size and referential integrity ones.

3.1 Size Metrics

3.1.1 Size of a System (SS)

SS is the sum of the size of every table in the schema:

```
SQLSchema:: SS(): Real  
= self.allBaseTables()  
  -> collect (elem: BaseTable | elem.TS()) -> sum
```

where the function SQLSchema:: allBaseTables(): Set(BaseTable) returns the set of all BaseTables existing in the current SQLSchema.

3.1.2 Table Size (TS)

TS is defined as the sum of the Total Size of the Simple Columns (TSSC) and the Total Size of the Complex Columns (TSCC) in the table.

```
BaseTable:: TS(): Real  
= if self.is_typed then  
  self.references.referenced_type.hierarchySize()  
else  
  self.TSCC() + self.TSSC()  
endif
```

We consider that all simple columns have a size equal to one, so that the TSSC metric is equal to the Number of Simple Columns in the table (NSC):

```
BaseTable:: TSSC(): Integer  
= self.allSimpleColumns() -> size()
```

where the function BaseTable:: allSimpleColumns(): Set(Column) returns the set of simple Columns belonging to the current BaseTable. A simple Column is a Column whose type is neither StructuredType nor ReferencedType.

TSCC is defined as the sum of each Complex Column Size (CCS):

```
BaseTable:: TSCC(): Real  
= self.allComplexColumns()  
  -> collect(elem: Column | elem.CCS()) -> sum
```

where the function `BaseTable:: allComplexColumns() : Set(Column)` returns the set of complex Columns belonging to the current BaseTable. A complex Column is a Column whose type is either StructuredType or ReferencedType.

3.1.3 Complex Column Size (CCS)

The value for CCS is obtained by:

```
Column:: CCS(): Real
= self.SHC() / self.NCU()
```

SHC is the Size of the Class Hierarchy (formed by the user-defined data types and their ascendants) upon which the column is defined and NCU is the number of columns defined upon this hierarchy. This expression reflects the fact that the effort to understand two or more complex columns decreases if both are defined upon the same class.

```
Column:: SHC(): Real
= self.dataType.oclAsType(StructuredType).SC() +
  self.dataType.oclAsType(StructuredType).ascendants()
  -> collect (elem: DataType | elem.oclAsType(StructuredType).SC())
  -> sum
```

```
Column:: NCU(): Integer
= self.dataType.oclAsType(StructuredType).columnsNumberUsingThis()
```

where `oclAsType(Type)` is a type casting function (OCL is strongly type checked), and the function `StructuredType:: columnsNumberUsingThis(): Integer` returns the number of Columns using the current StructuredType.

3.1.4 Data Type Size (DTS)

DTS is defined as:

```
StructuredType:: DTS(): Real
= (self.SAC + self.SMC) / self.NHC
```

where the Size of Attributes in a Class (SAC) is the sum of the size of all the Attributes in the current class; the Size of Methods in a Class (SMC) is the sum of the size of all the MethodSpecifications in the current class and NHC is the Number of Hierarchies to which the current Class pertains.

The attributes of a data type may be simple or complex. SAC is defined as the sum of the Simple Attributes Size (SAS, which have size equal to one) and the Complex Attributes Size (CAS) in the data type.

```
StructuredType:: SAC(): Real
= self.SAS() + self.CAS()
```

```
StructuredType:: SAS(): Real
```

```
= self.allSimpleAttributes() -> size()
```

where the function `StructuredType:: allSimpleAttributes(): Set(Attribute)` returns the set of simple Attributes declared in the current class. Simple Attributes are the ones whose type is neither a StructuredType nor a ReferenceType.

```
StructuredType:: CAS(): Real
= self.allComplexAttributes()
  -> collect(elem: Attribute |
            elem.dataType.oclAsType(StructuredType).SC())
  -> sum
```

where the function `StructuredType:: allComplexAttributes(): Set(Attribute)` returns the set of complex Attributes declared in the current class. Complex Attributes are the ones whose type is either a StructuredType or a ReferenceType.

The SMC function (`MethodSpecification:: SMC(): Real`) can be obtained in different ways. Possibilities include the use of the fan-in and fan-out metrics proposed by Li and Henry [Li and Henry, 1993], or to consider the method size to be unitary. Our ontology would have to be extended for using the metrics of Li and Henry, since the class MethodSpecification does not hold all the required information to obtain such metrics.

NHC is defined as:

```
StructuredType:: NHC(): Integer
= if self.hasChildren() then
  self.childrenNumber()
else
  1
endif
```

where `StructuredType:: hasChildren(): Boolean` indicates whether the StructuredType has children or not. A true value indicates it has children and a false value indicates the contrary. `StructuredType:: childrenNumber(): Integer` returns the number of directly derived classes.

3.1.5 Percentage of Complex Columns (PCC)

PCC is the Number of Columns defined over a Class hierarchy (NCC) divided by the total number of columns.

```
BaseTable:: PCC(): Percentage
= self.NCC() / (self.allColumns() -> size())
```

```
SQLSchema:: PCCSchema(): Percentage
= (self.allBaseTables()
  -> collect(elem: BaseTable | elem.NCC())
  -> sum()) / self.baseTablesSize()
```

where `BaseTable:: allColumns(): Set(Column)` returns the set including all Columns belonging to the current BaseTable and `BaseTable:: NCC(): Integer` is the number of complex Columns in the current BaseTable.

3.1.6 Number of InvolvedClasses (NIC)

NIC measures the number of all the classes that compose the types of the complex columns of a table using the generalization and the aggregation relationships.

```
BaseTable:: NIC(): Integer  
= self.involvedClasses() -> size
```

```
SQLSchema:: NICSchema(): Integer  
= self.involvedClassesInSchema() -> size()
```

3.1.7 Number of Shared Classes (NSC)

The NSC (Number of Shared Classes) measures the number of involved classes for a table that are used by other tables.

```
BaseTable:: NSC(): Integer  
= self.involvedClasses()  
  -> select(elem: StructuredType | elem.isShared())  
  -> size
```

```
SQLSchema:: NSCSchema(): Integer  
= self.sharedClassesInSchema() -> size ()
```

3.2 Referential integrity metrics

In OR databases, characteristics of relational databases are preserved. Consequently, some other metrics related with the referential integrity can be considered.

3.2.1 Number of Foreign Keys (NFK)

```
BaseTable:: NFK(): Integer  
= self.tableForeignKeys() -> size()
```

```
SQLSchema:: NFKSchema(): Integer  
= self.allBaseTables()  
  -> select(elem: BaseTable | elem.NFK() > 0)  
  -> collect(elem: BaseTable | elem.NFK()) -> sum
```

3.2.2 Referentiability Degree (RD)

RD is the number of foreign keys divided by the number of Attributes.

```
BaseTable:: RD(): Real  
= self.tableForeignKeyNumber() / (self.allColumns() -> size())
```

```

SQLSchema:: RDSchema():Real
= (self.allBaseTables()
  -> collect(elem: BaseTable |
            elem.foreignKeyNumber() -> sum()) /
  self.baseTablesSize()

```

3.2.3 Depth of the Referential Tree (DRT)

DRT is defined as the length of the longest referential path in the database. Note that cycles are only considered once.

```

BaseTable:: DRT(): Integer
= self.longestPath() -> size

```

```

SQLSchema:: DRTSchema(): Integer
= self.allBaseTables()
  -> iterate(elem: BaseTable; acc: Integer = 0 |
            if elem.DRT() > acc then
              elem.DRT()
            else
              acc
            endif)

```

4 ONTOLOGY INSTANTIATION

In this section we show how the ontology is instantiated for a given SQL:2003 expression. For illustration purposes, consider the following DDL clause:

```

CREATE SCHEMA video_and_music
CREATE TABLE movie_stars (
  movie_title CHARACTER (30)
  CONSTRAINT movies_stars_movie_title_not_null NOT NULL,
  movie_year_released DATE,
  movie_number CHARACTER (10),
  actor_last_name CHARACTER (35)
  CONSTRAINT movies_stars_actor_last_name_not_null NOT NULL,
  actor_first_name CHARACTER (25)
  CONSTRAINT movies_stars_actor_first_name_not_null NOT NULL,
  actor_middle_name CHARACTER (25),
  CONSTRAINT movies_stars_unique
  UNIQUE (movie_title, actor_last_name, actor_first_name,
          actor_middle_name)
  CONSTRAINT movies_stars_fk_movie FOREIGN KEY (movie_number)
  REFERENCES movies (stock_number));

```

The corresponding ontology instantiation is represented in Figure 3. We omitted the Predefined types to simplify the figure.

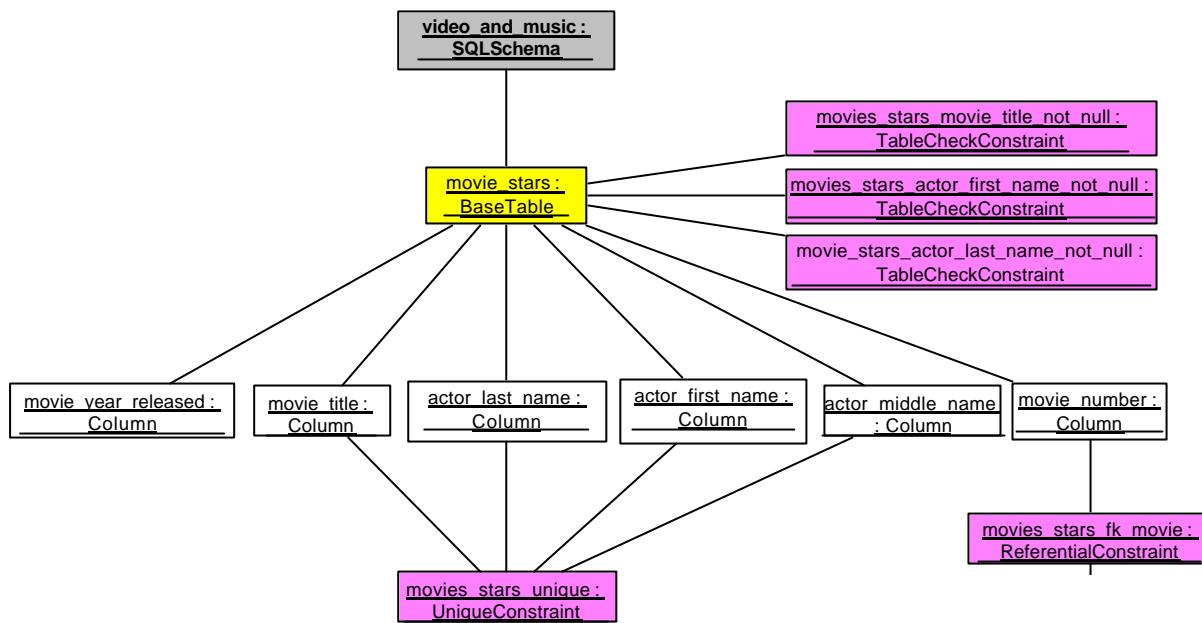


Figure 3 – Ontology Instantiation

5 CONCLUSIONS AND FUTURE WORK

The increased adoption of OR DBMS as the backbone s of newly built information systems incurs on additional effort to understand and maintain database schemata, when compared to their relational counterparts. To evaluate such complexity, OR metrics are required. This paper is the first known attempt to formalize such a metrics proposal. Formalization allows clear understanding of metrics definitions, so that their computation can be repeated independently. This formalization also facilitates metrics collection automation, but that is beyond the scope of this paper.

The context to define the metrics is granted by the proposed ontology, that describes the basic concepts of OR schemata. The metrics are calculated by navigating through the UML class diagram used to represent the ontology. An illustrative example of the ontology instantiation is presented. In this example an OR DDL expression is converted in an ontology instantiation (expressed as a UML object diagram). It is upon the objects and links contained in this diagram that an OCL evaluator is able to compute the results of the defined metrics (OCL functions).

We are currently developing the computational support to conduct experiments where large OR database schemata can be evaluated. This evaluation will encompass the following steps:

- a) the SQL:2003 ontology, represented as a UML class diagram, is loaded in a tool that supports UML class instantiation (objects creation) and OCL expressions evaluation;
- b) from a SQL:2003 OR DDL specification a parser generates a textual description of the objects and links to instantiate the ontology classes and associations, as we have done manually in the example presented in section 4;
- c) the objects and links generated in the previous step are loaded in the tool mentioned in a) to instantiate the ontology;
- d) the required metrics values can be obtained by selectively querying ontology instances; the OCL expressions will then be evaluated and a value is returned;

e) the values calculated in the previous step are fed to a metrics visualization component which embodies heuristics that help database designers and maintainers.

We are already able to perform steps a), c) and d). Our future work in this area will concentrate on the support to the missing steps, b) and e).

REFERENCES

- Baroni A. L., Brito e Abreu F. (a), "Formalizing Object-Oriented Design Metrics upon the UML Meta-Model". Proceedings of the 16th Brazilian Symposium on Software Engineering, Gramado - RS, Brazil. October, 2002.
- Baroni A. L., Goulão M., Brito e Abreu F. (b), "Avoiding the Ambiguity of Quantitative Data Extraction: An Approach to Improve the Quality of Metrics Results". Proceedings of Work in Progress Session at the 28th EUROMICRO Conference, Dortmund, Germany. September, 2002.
- Baroni A. L., "Formal Definition of Object-Oriented Design Metrics", MSc Thesis, Vrije Universiteit Brussel - Belgium, in collaboration with Ecole des Mines de Nantes - France and Universidade Nova de Lisboa - Portugal. August, 2002.
- Baroni A. L., Braz S., Brito e Abreu F. (c), "Using OCL to Formalize Object-Oriented Design Metrics Definitions". Proceedings of QUAOOSE'2002, Malaga, Spain. June, 2002.
- Briand, L.C., Morasca, S., Basili, V., "Property-Based Software Engineering Measurement", *IEEE Transactions on Software Engineering*, Vol. 22 (1), 68-85, 1996.
- Calero, C. "Definición de un Conjunto de Métricas para la Mantenibilidad de Bases de Datos Relacionales, Activas y Objeto-Relacionales", PhD Thesis, Universidad de Castilla-La Mancha, Spain. March, 2001.
- Calero C., Piattini M., Genero M. (a), "A Case Study with Relational Database Metrics". ACS / IEEE International Conference on Computer Systems and Applications (AICCSA 2001), 26-29 June 2001, Beirut, Lebanon. IEEE Computer Society 2001, ISBN 0-7695-1165-1, pp. 485-487.
- Calero C., Piattini M., Genero M. (b), "Empirical validation of referential integrity metrics". *Information & Software Technology*, volume 43, numer 15, pp. 949-957, 2001
- Calero, C., Sahraoui H. A., Piattini M., Lounis H. (c), "Estimating Object-Relational Database Understandability Using Structural Metrics". Proceedings of the Database and Expert Systems Applications, 12th International Conference (DEXA 2001), Munich, Germany, September 35, ISBN 3-540-42527-6, pp. 909-922
- Champeaux, D. de, *Object-Oriented Development Process and Metrics*, Upper Saddle River, NJ, Prentice Hall, 1997.
- Brito e Abreu F., "Object-Oriented Software Design Metrics", Pragmatic and Theoretical Directions in Object-Oriented Software Metrics Workshop, OOPSLA'94, Portland, Oregon, USA, October 1994.
- Brito e Abreu F., Carapuça R., "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", proceedings of the 4th International Conference on Software Quality, McLean, Virginia, USA, October 1994.
- Genero M., Piattini M., Calero C., Serrano M., "Measures to Get Better Quality Databases". Proceedings of the Second International Conference on Enterprise Information Systems (ICEIS 2000), pp.: 49-55 Stafford, UK, July 4-7, 2000.
- MacDonell, S.G., Shepperd, M.J., Sallis, P.J., "Metrics for Database Systems: An Empirical Study", *Proc. Fourth International Software Metrics Symposium - Metrics'97*, Albuquerque. IEEE Computer Society, 99-107, 1997.

- OMG (a), "UML 2 Object Constraint Language Specification (version 2.0)", Object Management Group, October, 2003.
- OMG (b), "Unified Modeling Language Specification (version 1.5)", Object Management Group, March, 2003.
- ISO, "ISO/IEC 9075 standard: Information technology - Database languages - SQL", International Organization for Standardization, 2003.
- Li, W., Henry, S., "Object-Oriented Metrics that Predicts Maintainability", *J. Systems Software*, 23, 111-122, 1993.
- Melton, J., "Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features", *Morgan Kaufmann Publishers*, 2002.
- Melton, J., Simon, A. R., "SQL: 1999 - Understanding Relational Language Components", *Academic Press*, 2002.
- Pfleeger, S. L., "Assessing Software Measurement", *IEEE Software*. March/April, 25-26, 1997.
- Stonebraker, M., Brown, P., "Object-Relational DBMSs Tracking the Next Great Wave", *Morgan Kauffman Publishers*, California, 1999.