A FORMAL DEFINITION FOR OBJECT-RELATIONAL DATABASE METRICS

Aline Lúcia Baroni[1], Coral Calero[2], Mario Piattini[2], Fernando Brito e Abreu[1],
[1]*QUASAR Research Group. Faculty of Sciences and Technology. Universidade Nova de Lisboa. Portugal*

[2]*ALARCOS Research Group. Department of Compuer Science. Universirt of Castilla-La Mancha. Spain*

Keywords: Databases, metrics, SQL:2003.

Abstract: Relational databases are the most important in the database world and are evolving to object-relational databases in order to allow the possibility of working with new and complex data and applications. One widely accepted mechanism for assuring the quality of an object-relational database is the use of metrics formally and empirically validated. Also it is important to formalize the metrics for having a better understanding of their definitions. Metrics formalization assures the reliable repetition of their computation and facilitates the automation of metrics collection. In this paper we present the formalization of a set of metrics defined for object-relational databases described using SQL:2003. For doing the formalization we have produced the ontology of the SQL:2003 as a framework for representing the SQL schema definitions. The ontology has been represented using UML and the definition of the metrics has been done using OCL (Object-Constraint Language) which is part of the UML 2.0 standard.

1 INTRODUCTION

The history of databases last since mid-sixties and it has been characterised by its extraordinary productivity and its impressive economic impact. This is because databases have become a strategic product, being the basis of all information systems and supporting organizational decisions.

Relational databases are the most important ones in the database world. This success can be explained because they are not too difficult to understand and also because there is a widespread standard (SQL) for them. Another key success factor is that the relational industry has reacted and has evolved to object-relational databases in order to allow the possibility of working with new and complex data and applications without a revolutionary change in the market. So, these databases have all the elements of the relational model (relations connected by referential integrity relationships) but with the particularity that the columns of a relation can be defined over a UDT (User Defined Type).

Some studies predict that object-relational databases will substitute the relational ones (Stonebraker and Brown, 1999, Leavitt, 2000) and, very recently, the new SQL:2003 standard (ISO/IEC 9075, 2003) that integrates additional OR features, has been published.

Taking into account the brilliant predicted diffusion of object-relational databases it is essential to assure their quality. One widely accepted mechanism for assuring the quality of a software product in general and of object-relational database designs in particular, is the use of metrics.

However, it is also important to formalize the 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.

In this paper we present the formalization we have performed upon a set of metrics defined for assessing the complexity of OR database schemata. For performing this formalization we have produced an ontology of the SQL:2003 (the last version of the object-relational database standard), as a framework for representing the SQL schema definitions. The ontology was represented using UML and the metrics have been defined with OCL -Object-Constraint Language (OMG, 2003). OCL allows express invariants, pre and post conditions, as well as operations semantics and is part of the UML 2.0 standard.

Lúcia Baroni A., Calero C., Piattini M. and Brito e Abreu F. (2005).
A FORMAL DEFINITION FOR OBJECT-RELATIONAL DATABASE METRICS.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 334-339
DOI: 10.5220/0002510503340339

Section two presents the ontology we have defined for the new SQL:2003. Section three presents an example of a database definition using the new SQL:2003 which is represented using the ontology. In section four the metrics definition and formalization using OCL is shown. Finally, the last section presents the conclusions and future work.

2 SQL:2003 ONTOLOGY

Among the different languages present in the earliest DBMS, SQL has imposed itself as a “de iure” and “de facto” database standard.

Recently, the last version of the standard has been published, SQL:2003 (ISO/IEC 9075, 2003) which makes revisions to all parts of SQL:1999 and includes some new issues (Eisenberg et al., 2004).

The fact of having a standard is fundamental. However, sometimes standards are hard to understand and it is difficult to extract all the information contained. It usually happens that standards are not free of inconsistencies due to the big amount of information that they try to cover. In that case, most of the advantages derived of the disposal of a standard disappear.

For avoiding most of these problems, the standard can be complemented by its ontology. In such a manner the ontology helps in finding the information and detecting inconsistencies which is essential in order to define the metrics based on the concepts of the standard.

An ontology is a specification of a conceptualization. This means that, through the definition of an ontology, we try to formalize and recover the knowledge of a given domain.

So, we developed an ontology for the SQL:2003. For doing it, we have used several parts of the standard. We have worked mainly with the information of the part 1 (Framework) but basically with the one of part 2 (Foundation) of the standard. In the other hand, we also reengineered the Part 11 (Information and Definition Schema) considering those schemata as metamodels of the SQL:2003 which represent in their tables all the concepts of the language.

The ontology was thought for the object-relational aspects of a database schema, discarding elements such as triggers and stored procedures, as they are not needed for the metrics we consider in this work. The inclusion of these elements can be easily done because the main components are included in this version and the discarded ones are related to them.

The ontology has been divided into two. One contains all the aspects related to data types (figure 1) and the other all the information about the SQL schema objects (figure 2).

Figure 1 shows three different kinds of *Data Types*: *Predefined*, *Constructed* and *User Defined Types*. *Constructed Types* can be *Composite* or *Reference Types*. *Composite Types* can be *Collections* (*Arrays* or *Multiset* – a new type of the SQL:2003 standard) composed by *Elements,* or *Row Types*, which in turn are composed by *Fields*. Each *Element* or *Field* has one *Data Type*.

The *User Defined Types* can be either *Distinct Types* (which are defined over one *Predefined Data Type*) or *Structured Types*[1]. *Structured Types* are composed by *Attributes* and by one or more *Method Specifications*. Inheritance is allowed among *Structured Types*, *Row Types* and *Reference Types*.

Figure 2 illustrates four different SQL schema objects: *Constraints*, *Domains*, *User Defined Types* and *Tables*.

Constraints can be *Assertions*, *Domain Constraints* or *Table Constraints* (*Unique Constraints* including *Primary Keys*, *Table Check Constraints* and *Referential Constraints* – the latter for representing the foreign keys).

Domains are used by *Columns* and can include a *Domain Constraint*.

Tables can be *Derived Tables* – and particularly *Views*, *Transient Tables* or *Base Tables*. They are composed by *Columns* that can be defined as *Identity Columns* or *Generated Columns*. *Columns* can be defined over a *Domain* and they can have *Referential Constraints* or *Unique Constraints* (or *Primary Keys*) defined.

Base Tables can also be part of an inheritance hierarchy. They are defined over a *Data Type* (through a *Reference Type*) and they can have *Candidate Keys*.

[1] Structured types are entities corresponding to classes in object-oriented notations. Thus, when we use the word “class” in this document, we refer to the ontology entity *StructuredType*.

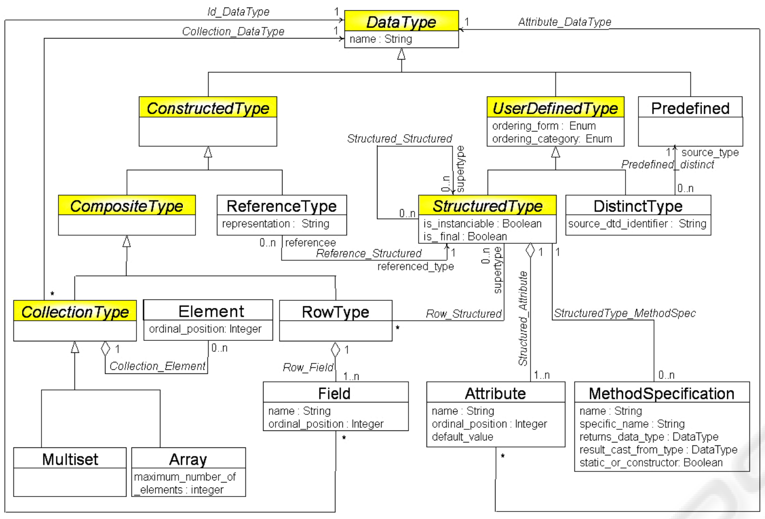

Figure 1: SQL:2003 Data types sub-ontology

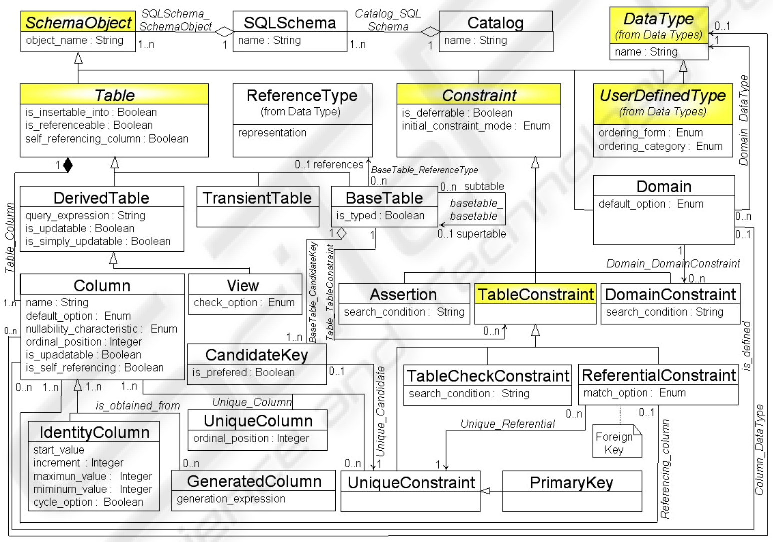

Figure 2: SQL:2003 Schema objects sub-ontology

3 METRICS FOR OBJECT-RELATIONAL DATABASES

One widely accepted mechanism for evaluating the quality of a software product in general and of object-relational database designs in particular, is the use of metrics (Briand et al. 1996; Pfleeger, 1997; Fenton y Pfleeger, 1997; Chidamber y Kemerer, 1994; Zuse, 1998; Sneed and Foshag, 1998; Basili et al, 1996)

Metrics must be defined for capturing a specific characteristic of a product (in our case object-relational databases). One of the most important characteristic to be captured is complexity. With a set of metrics for measuring the complexity, we will be able to estimate understandability and maintainability (Li and Henry, 1993; Briand et al. 1995; Briand et al. 1999), two important dimensions in software product quality (ISO9126, 2001).

In the work of Piattini (Piattini et al., 2001) a set of metrics for object-relational database complexity are defined. These metrics have been formalized using the approach presented in (Baroni, 2002; Baroni and Brito e Abreu, 2002). In the next sub-sections, each informal definition is presented together with its formal one (some auxiliary functions, were used in the formalization process but are not presented due to space restrictions)

3.1 Metrics concerning table properties

3.1.1 Table Size Metric

The size of a table (TS) is defined as the sum of the size of the simple columns (TSSC) and the size of the complex columns (TSCC). The TSCC is calculated as the sum of the size of each complex column (CCS).

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

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

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

The size of a complex column (CCS) is defined as the size of the class hierarchy above which the column is defined (SHC) divided by the number of complex columns that are defined over this hierarchy (NHC). This expression is due to the fact that the size of the hierarchy must be considered only once independently of the number of columns defined above it.

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

   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()
```

The size of a class (SC) is calculated as the sum of its attributes size (SAC) and its methods size (SMC). It is necessary to take into account that a class can have simple attributes (SAS), that we consider with a size equal to one, and complex attributes (CAS), which are attributes related to other classes by an aggregation relationship. In that case the size of a complex attribute is calculated as the size of the aggregation hierarchy. Again in that case, as a class can belong to more than one hierarchy, it is necessary to divide its size into the number of hierarchies that use the class (NHC).

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

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

   StructuredType:: SAS(): Integer
   =    self.allSimpleAttributes()    ->
size()

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

(StructuredType).SC())
        -> sum

   StructuredType:: SMC(): Integer
   = self.NMC()

   StructuredType:: NMC(): Integer
   = self.allMethods() -> size()

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

3.1.2 Coupling Metrics

NIC (Number of Involved Classes): Number of classes needed for defining all the columns of a table.

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

NSC (Number of Shared Classes): Number of classes used by a table, for defining its complex columns, which are also used by other tables of the schema.

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

3.1.3 Complexity Metrics

PCC (Percentage of Complex Columns): Number of the complex columns of a table (NCC) divided by the total number of columns of the same table.

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

BaseTable:: NCC(): Integer
= self.allComplexColumns() -> size()
```

3.1.4 Referential Integrity Metrics

NFK (Number of Foreign Keys): Number of foreign keys defined in a table.

```
BaseTable:: NFK(): Integer
= self.foreignKeyNumber()
```

RD (Referential Degree): Number of foreign keys in a table divided by the number of attributes of the same table.

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

DRT (Depth of Referential Tree): The longest path between a table and the remaining tables in the schema database, considering the schema as a graph where nodes are tables and arcs are referential integrity relations between tables (Foreign key to Primary key links).

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

3.2 Metrics concerning schema properties

All the metrics applied over tables can also be applied at the schema level, iterating over the *BaseTables* in the *SQLSchema*. These are the coupling, complexity and referential integrity metrics.

Additionaly, a new size metric for the schema (SS) can also be defined as the sum of the sizes of each table in the schema.

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

4 CONCLUSIONS AND FUTURE WORK

Our current work direction addresses the solution of two main problems: the lack of metrics for evaluating the quality of databases and the lack of formalization of the existing metrics definitions.

The first problem was treated with the proposal of some metrics for object-relational databases (Piattini, 2001), in some of our previous work. However, metrics for other aspects, not covered by our work, are still necessary.

This paper presented an approach to solve the second problem, using UML and OCL (OMG, 2003). The original approach was proposed in (Baroni, 2002; Baroni and Brito e Abreu, 2002), and it was successfully applied here.

Besides formalizing some metrics definitions, we created an ontology for the new SQL:2003 standard (ISO/IEC 9075, 2003), which tries not only to reduce the inconsistencies in the textual version of the standard, but also to make the standard easier to grasp.

The ontology definition is an on-going work, and it can be seen more as a proposal than as a complete version. It requires some extensions for treating of other aspects in the standard, such as triggers, stored procedures, parameters in methods, etc. Notwithstanding, the ontology as shown in this paper has enough features for the formalization we addressed.

The formalized metrics definitions and a database representation mapped to ontology meta-objects served as input to an OCL evaluator tool (there are several tools able to work with OCL, and more are emerging to work with its newer version, i.e., OCL 2). With these two inputs, and also with the ontology as background, we could extract real metric values from database representations. One simple example was illustrated here.

As future work, there are many possible directions to explore varying from the proposition of new metrics and their validation, including their formal definitions, until the use of these metrics to perform refactorings on database schemata.

REFERENCES

Baroni A. L., 2002. *Formal Definition of Object-Oriented Design Metrics*. Master Thesis. Vrije Universiteit Brussel - Belgium, in collaboration with Ecole des Mines de Nantes - France and New University of Lisbon - Portugal. August.

Baroni A. L., Brito e Abreu F., 2002. Formalizing Object-Oriented Design Metrics upon the UML Meta-Model. In Proceedings of the 16th Brazilian Symposium on Software Engineering, Gramado - RS, Brazil. October.

Basili V., Briand L., Melo W. L., 1996. A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering*, vol. 22, pp. 751-760.

Briand L., Arisholm S., Counsell F., Houdek F., Thévenod-Fosse P., 1999. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. *Empirical Software Engineering*, 4 (4), 387-404.

Briand L., El Emam K., Morasca, S., 1995. Theoretical and Empirical Validation of Software Product Measures. *International Software Engineering Research Network,* Technical Report ISERN-95-03.

Briand L., Morasca S., Basili, V., 1996. Property-Based Software Engineering Measurement. *IEEE Transactions on Software Engineering*. 22 (1). pp.68-85.

Chidamber S., Kemerer C., 1994. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*. 20 (6). pp.476-493. June.

Fenton N., Pfleeger S. L., 1997. *Software Metrics: A Rigorous Approach* 2nd. edition. London. Chapman & Hall.

ISO/IEC 9075 Standard, 2003. Information Technology - Database Languages - SQL, *International Organization for Standardization*.

ISO/IEC 9126 Standard, 2001. Software Product Evaluation-Quality Characteristics and Guidelines for Their Use, *International Organization for Standardization.*

Leavitt N., 2000. Whatever Happened to Object-Oriented Databases?. *IEEE Computer Society*. pp. 16-19.

Li W., Henry S., 1993. Object-Oriented Metrics that Predicts Maintainability, *Journal of Systems and Software*, 23, 111-122.

Melton, J., 2003. Advanced SQL:2003. Understanding object-relational and other features. Morgan Kauffman Publishers. Elsevier Science. USA

OMG, 2003. UML 2 Object Constraint Language Specification (version 2.0)", *Object Management Group*, October.

OMG, 2003. Unified Modeling Language Specification (version 1.5), *Object Management Group*, March.

Pfleeger, S. L., 1997. Assessing Software Measurement. *IEEE Software*. March/April. pp. 25-26.

Piattini M., Calero C., Sahraoui H., Lounis H., 2001. Object-Relational Database Metrics. *L'Object*, vol. March.

Sneed H. M., Foshag O., 1998. Measuring Legacy Database Structures*. Proceedings of the European Software Measurement Conference (FESMA 98)*. Antwerp. May 6-8. Coombes. Van Huysduynen and Peeters (eds.). pp.199-211.

Stonebraker M., Brown P., 1999. *Object-Relational DBMSs Tracking the Next Great Wave*, California, Morgan Kauffman Publishers.

Zuse H., 1998. *A Framework of Software Measurement*. Berlin. Walter de Gruyter.