Using Metamodeling in order to Evaluate Data Models

ERKI EESSAAR
Department of Informatics
Tallinn University of Technology
Raja 15, 12618 Tallinn
ESTONIA
http://staff.ttu.ee/~eessaar

Abstract: - A data model (for example, relational or object-relational) specifies data types, types of data structures, types of operations with data and types of integrity constraints. A database system (DBMS) that implements a particular data model allows us to create databases to different enterprises. In this paper we explain, why it is useful to create metamodels of data models. We illustrate the advantages with concrete examples based on the metamodels of two object-relational data model approaches. The results that are revealed through the examples are also novel contributions of our work.

Key-Words: - Object-relational data model, SQL, Metamodel, Data types, Language design, Metrics

1 Introduction

Definition: "A data model is an abstract, self-contained, implementation-independent definition of elements of a 4-tuple of sets (T, S, O, C) that together make up the abstract machine with which database users interact. In this case: T is a set of data types; S is a set of data structure types; O is a set of data operation types; C is a set of integrity constraint types."

Examples of the data models are hierarchical, network, relational, object-oriented and object-relational. The names of these models are actually general names because there are different proposals about their exact nature.

In this paper we are interested in the object-relational (OR) data model. There is no common object-relational data model yet. In this paper we consider two OR data model approaches:

- The underlying data model of SQL:2003 standard ("OR_{SQL}") [1].
- The data model ("OR_{TTM}") that is described in The Third Manifesto [2, 3].

The Third Manifesto can be seen as a compilation of principles of Object-Relational DBMS that is free from the problems and limitations of SQL. "Accordingly, we also believe that a true object/relational system would be nothing more nor less than a true relational system – which is to say, a system that supports the relational model, with all that such support entails." [3]

We call the DBMSs that support OR_{SQL} or OR_{TTM} as $ORDBMS_{SQL}$ or $ORDBMS_{TTM}$, respectively.

Data model (as specified at the beginning of this section) is a kind of abstract *language* [4]. One possibility to describe abstract syntax of a language is to create metamodel of a language [5]. For example, abstract syntax of UML is presented as metamodel [6], which is created by using subset of UML – class diagrams.

The *goal* of this paper is to demonstrate that it is advantageous to create metamodels of data models. We have created metamodels of OR_{SQL} and OR_{TTM} data models. We present some of the results that we have achieved by analyzing these metamodels.

The rest of the paper is organized as follows. Section 2 lists some of the advantages of creation of metamodels of data models. Section 3 illustrates *metamodel-based comparison* of data models. We compare parts of OR_{SQL} and OR_{TTM} metamodels that specify constructed/generated data types. Section 4 presents *metrics values* that are calculated based on the metamodels of these two data models. Section 5 describes violations of *orthogonality principle* in language design that we discovered by observing OR_{SQL} metamodel. Section 6 summarizes this article.

2 The use of Metamodeling in case of Data Models

Metamodels are widely used in various software engineering processes. Metamodel is "a model of a model" that provides "the rules/grammar for the modelling language (ML) itself." [7]

Some advantages of metamodels of data models:

1. Creation of a metamodel may cause actual specification of a data model. For example, there is no *clear* and *compact* specification of "OR_{SQL} data model". Instead, there is huge

textual specification of SQL database language. A foundation part of SQL:2003 standard [1] is 1332 pages long. On the other hand, The Third Manifesto [2, 3] specifies the data model (OR_{TTM}) in the form of prescriptions, proscriptions and suggestions. However, it does not provide visual specification.

2. A metamodel of a data model visualises underlying concepts of a data model. It is possible to get overview about a data model with the help of much more compact document compared to purely textual specification.
3. If we create a metamodel by using some visual language (like UML) that is well known to the software engineering community, then it facilitates understanding of data models among many professionals. Maybe it also helps to improve understanding of data models by the DBMS vendors and improve current DBMSs. For example, Eessaar [8] describes some shortcomings of $ORDBMS_{SQL}$s that make more difficult to implement whole-part relationships in a database.
4. A metamodel can be used for teaching purposes, in order to give visual overview of the model constructs and their relationships.
5. It is possible to compare data models:

- by finding mappings and discrepancies between elements of their metamodels (see section 3).
- by calculating metrics values based on their metamodels (see section 4) and comparing these values. It is possible to use existing special tools like UML Model Measurement Tool [9] in order to calculate metrics values.

6. A metamodel of a data model could help to improve a data model and its specification:

- Inspection of visual structures in a metamodel helps to find violations of the orthogonality principle by a language (see section 5).
- Creation of a metamodel requires thorough study of existing specifications and therefore can help to find incompletenesses, inconsistencies and other mistakes in them.

3 Metamodel-based Comparison of Data Models

In this section, we demonstrate that the creation of metamodels of data models helps to compare the data models in order to find their similarities and differences. One possible method for evaluating information-modeling methods is metamodel-based comparison [10]. If we have metamodels of data models, then we can compare data models in the same way.

Next, we present example of this kind of comparison. Specification of a data model consists of specification of *data structures*, *data operators*, *data integrity,* and *data types*. We present *parts* of metamodels that specify constructed/ generated data types. We selected this part because existing research about the object-relational data models considers possibility to create these types as an important advantage of the object-relational data model compared to the relational data model (here "relational model" is the underlying model of SQL:1992 or earlier standards). It is possible to use these types in order to implement whole-part relationships in a database [8].

Fig. 1 presents part of OR_{TTM} metamodel and Fig. 2 presents part of OR_{SQL} metamodel. Table 1 contains mappings between the metaclasses that are shown in these models. There is a mapping between two metaclasses that belong to the different metamodels if the underlying constructs of these metaclasses have semantic equvelance or are at least semantically quite similar.

Table 1Mapping of OR_{SQL} and OR_{TTM} metaclasses that belong to package "Data type" and describe constructed/generated data types

OR_{SQL} metaclass	*OR_{TTM} metaclass*
Collection type	Collection type
Collection type constructor	Collection type generator
Constructed data type	Generated type
Data type	Type (data type, domain)
Data type constructor ("value constructor")	Type generator
ROW Con	TUPLE Gen
Row type	Tuple type
Table type	Relation type

Not all the metaclasses participate in this mapping. It shows that there are *discrepancies* between OR_{SQL} and OR_{TTM}. One type of discrepancy is *construct deficit*. In this case a metamodel element of one metamodel does not have a corresponding metamodel element in another metamodel.

Construct deficit in OR_{TTM}: ARRAY Con[(1)], Array element[(1)], Array type[(1)], MULTISET Con[(1)], Multiset element[(1)], Multiset type[(1)], REF Con[(3)], Reference type[(3)].

[(1)]- Date and Darwen [3] permitted ARRAY and SET type generators but more lately they have come to the conclusion that these type generators and corresponding types are unnecessary [2]. [(3)] - Authors of OR_{TTM} argue explicitly against pointers.

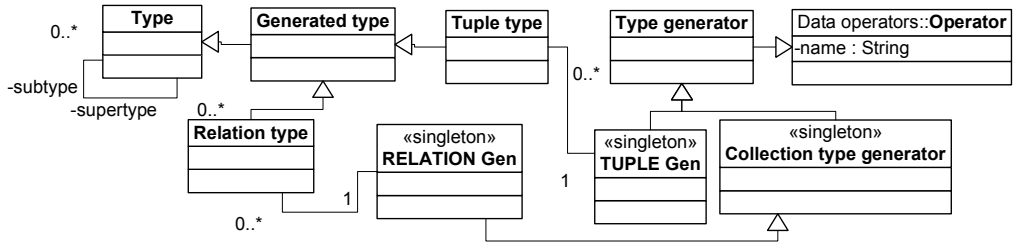

Fig. 1 Data type generators in OR_{TTM}

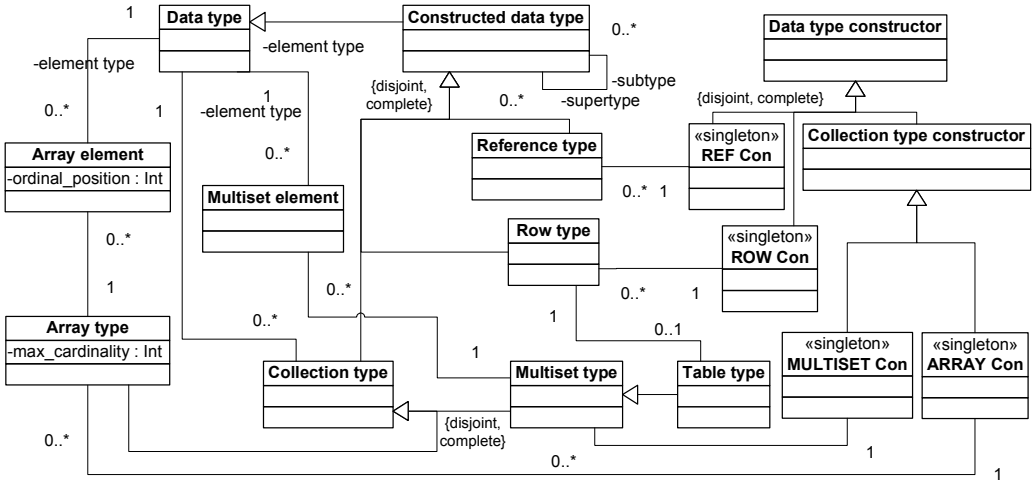

Fig. 2 Data type constructors in OR_{SQL}

Construct deficit in OR_{SQL}: RELATION Gen.

OR_{TTM} requires that an $ORDBMS_{TTM}$ must support two type generators that allow creation of non-scalar types – TUPLE and RELATION (see Fig. 1). OR_{SQL} specifies four type constructors – REF, ROW, ARRAY and MULTISET (see Fig. 2).

Fields of a constructer row type are left-to-right ordered in OR_{SQL}. If we change the order of fields in the declaration of a row type, then this declaration specifies a new type. On the other hand, attributes of a generated tuple type are not left-to-right ordered in OR_{TTM}. A value of a constructed array type in OR_{SQL} is an *ordered* collection of elements. A value of a constructed multiset type in OR_{SQL} is an *unordered* collection of elements. All the elements in a collection must have the same type (see Fig. 2). These collections can contain repeating elements.

A value of a generated relation type in OR_{TTM} is an unordered set of tuples each of which has the same tuple type. This set cannot contain repeating elements (tuples). OR_{SQL} uses the concept "table type" in the context of table functions. The wording "<returns type>::= <returns data type> [<result cast>] | <returns table type>" [1, p. 676] gives an impression that a table type is not a data type. However, an invoked table function returns a value that has a type ROW (...) MULTISET (multiset of rows). Mapping of "table type" and "relation type" (see Table 1) is disputable. We cannot use a "table type" (as described by [1, p. 676]) as declared type of a column in OR_{SQL}. However, we can create a column with a type ROW(...) MULTISET.

Reference type, that is a kind of constructed data type, is used together with the typed tables in OR_{SQL}. OR_{SQL} allows us to create typed tables based on the user-defined structured types. The row type of a typed table is derived from a structured type. A typed table is a referenceable table. "A REF value is a value that references a row in a referenceable table." [1, p. 43]. A reference type is a set of REF values that reference rows in the typed tables that are defined based on a structured type. These values are like Object ID-s in object systems that "are addresses – at least conceptually – and are hidden from the user" [4, p. 826]. Such state of the affairs is caused by the view of SQL creators that the object-oriented concepts "class" (or type) and "instance" are the counterparts of the database concepts "table" and "row", respectively. OR_{TTM}, on the other hand, advocates that the counterpart of the concept "class" is concept "data type".

As you can see, metamodel-based comparison provides the *framework* that allows us to discuss the similarities and differences of data models.

4 Metrics Values

Rossi and Brinkkemper [11] propose a set of metrics. Their values are calculated based on the metamodels and they help to compare complexity of system development methods and techniques. It is also possible to use these metrics in case of the data models if their metamodels are available.

Table 2 presents values of three types of metrics – (1) number of metaclasses, (2) number of attributes of the metaclasses, and (3) sum of these values. These metrics values are calculated for OR_{SQL} and OR_{TTM} in general (row $\sum$) and also for the subsections of these data models. For the comparison purposes we also present the metrics values for the underlying data model of SQL:1992 (*without* the extensions that were added by *Persistent Stored Modules - 96* specification).

In case of these metrics, bigger values mean bigger complexity. However, Rossi and Brinkemper [11] write about them: "the metrics by themselves cannot be used to judge the "goodness" or the appropriateness for the task of the method" and should be used together with other comparison methods (like for example the one that is presented in section 3).

The underlying data model of SQL:1992 has smaller metrics values as compared to OR_{SQL} and OR_{TTM}. In this case, smaller metrics values (and complexity) are caused by the lack of many important features. It actually makes creation of applications that use a database more difficult. It means *repositioning* complexity within the system because more work has to be done by the application. The work of Eessaar [8] contains literature-based overview of problems of the underlying data model of SQL:1992 or earlier SQL standards. OR_{SQL} and OR_{TTM} data models try to solve many of the referenced problems.

Metrics values of OR_{SQL} are bigger than metrics values of OR_{TTM}. The amount of metaclasses in OR_{SQL} and OR_{TTM} data model metamodels is quite similar. However, the metaclasses of OR_{SQL} metamodel have much more attributes compared to OR_{TTM} metamodel. Bigger amount of attributes indicates that a database designer who designs database based on OR_{SQL} has more opportunities to "tune" the database objects, compared to OR_{TTM}. It also points to the bigger complexity of OR_{SQL} compared to OR_{TTM}.

In this case, smaller metrics values of OR_{TTM} compared to OR_{SQL} do not mean that OR_{SQL} is "better".

Firstly, analysis of similarities and discrepancies of OR_{SQL} and OR_{TTM} (part of it is presented in section 3) shows that despite the lack of some constructs in OR_{TTM} (for example, typed tables, reference types, triggers) it is still possible to use an $ORDBMS_{TTM}$ in the cases that require the use of these constructs in an $ORDBMS_{SQL}$. We just have to use some OR_{TTM} construct (that may have corresponding construct in OR_{SQL}) in a way that is not possible in an $ORDBMS_{SQL}$. For example, in $ORDBMS_{SQL}$s we often have to use triggers in order to enforce complex data integrity rules that refer to more than one table. In an $ORDBMS_{TTM}$ we can use declarative database constraints for the same purpose.

In addition, OR_{SQL} violates orthogonality principle (see section 5).

5 Orthogonality Principle in Language Design

Date and Darwen [3, p. 505] explain that a programming language that displays orthogonality provides "(a) a comparatively small set of primitive constructs together with (b) a consistent rules for putting those constructs together, and (c) every possible combination of those constructs is both legal and meaningful (*in other words, a deliberate attempt has been made to avoid arbitrary restrictions*)." [3, p. 505] [*Italics* added by author] It is also true in case of abstract programming languages like data models.

An advantage of OR_{TTM} compared to OR_{SQL} is that OR_{TTM} is based on the small set of core concepts that makes the model much easier to understand (see requirement (a) of orthogonality). Unlike OR_{SQL}, OR_{TTM} uses the concepts "variable" and "operator" as a basis of specification of its data structures and data operators, respectively. Some of the concepts are metaphors that help to make a data model easier to understand to people with a programming background. Examples of such concepts are "variable" and "assignment operator". Rittgen [12] recommends to use *metaphors* in the software engineering in order to make a particular topic more understandable because "they resort to knowledge that is rooted in common sense and therefore shared by everybody." [12, p. 434]

Date and Darwen [3] illustrate SQL violations of orthogonality principle with the non-exhaustive list of examples. Their examples are about the requirement (c) of orthogonality.

We found additional examples. We present problem in OR_{SQL} as well as comments about the state of affairs in OR_{TTM}.

Table 2 Metrics values - number of metaclasses, number of their attributes and sum of these values

Subsection of a data model metamodel	*SQL:1992*	*OR_{SQL}*	*OR_{TTM}*
Data types	10+10=20	38+21=59	27+4=31
Data structures	18+12=30	26+17=43	17+4=21
Data integrity	13+11=24	16+21=37	9+5=14
Data operators	8+2=10	27+32=59	42+5=47
Metaclasses that we cannot classify and their attributes	3+3=6	3+3=6	-
Metrics values for a data model in general ($\sum$)	52+38=90	110+94=204	95+18=113

1. Attributes, fields and columns are *structural components* but only a column can be associated with a domain.
   OR_{TTM}: OR_{TTM} does not use the constructs *field*, *column* and *domain*. An attribute of a relational variable (relvar) can have a type that is either a built-in or user-defined scalar type or a generated type.
2. It is not possible to declare a default value to a field of a row type but it is possible in case of other structural components – attributes and columns.
3. Both base tables and viewed tables (views) have columns. However, it is not possible to declare a default value to a column of a view. Together with updateable views, it could allow us to record different default values in a column of a base table in the different situations.
   OR_{TTM} (problems 2-3): A relvar (base or virtual) attribute can have a default value [3, p. 202].
4. It is possible to use generated columns but not generated attributes or fields.
5. Attributes, fields and columns are structural components. However, it is possible to use generated columns but not generated attributes or fields.
   OR_{TTM} (problems 4-5): A default value of a relvar attribute can be found by using some expression. It can refer to system functions.
6. A domain can be associated with a predefined data type but not with a user-defined or constructed type.
   OR_{TTM}: OR_{TTM} uses the concepts *domain* and *type* as synonyms. Attributes that are in the heading of a relation or a tuple type or components of a possible representation of a scalar type can have any type.
7. A base table or a view cannot contain two or more columns with the same name in OR_{SQL}. However, a derived table that is derived directly or indirectly from one or more other tables by the evaluation of a query expression can contain more than one column with the same name.
   OR_{TTM}: It does not allow two or more attributes with the same name in a relvar, in a relation or in the heading of a relation or tuple type.
8. Table constraints can only be explicitly associated with base tables but not with views. Here explicit association means that constraint specification is part of a table specification.
   OR_{TTM}: OR_{TTM} does not distinguish base and virtual relvars in this regard. For example, candidate key and foreign key specifications could be part of specification of a base or a virtual relvar.
9. It is possible to create temporary base tables but not temporary views.
   OR_{TTM}: It specifies private application relvars that correspond to declared local temporary tables in OR_{SQL} and public application relvars that are kind of virtual relvars.
10. It is possible to create a typed table based on a user-defined structured type but not based on a distinct type.
11. Each typed table must have exactly one self-referencing column. If this typed table is a typed base table, then this column has an implicit uniqueness constraint. On the other hand, OR_{SQL} permits not-typed base tables, which do not have any associated (explicitly or implicitly defined) uniqueness constraint.
12. A self-referencing column in a typed table cannot be updated.
    OR_{TTM} (problems 10-12): OR_{TTM} does not support typed tables. However, each base relvar must have at least one explicitly defined candidate key. All attributes of relvars are updatable.
13. A subject table of a trigger can only be a persistent base table. It cannot be a view or a temporary base table.
14. We can use triggers and declarative constraints in order to implement integrity rules. It is possible to defer checking of a declarative constraint (*but not* execution of a trigger procedure) until the end of a transaction.

OR_{TTM} (problems 13-14): OR_{TTM} does not specify triggers. $ORDBMS_{TTM}$ performs constraint checking at the end of each update (assignment) operation. However, OR_{TTM} permits multiple assignment operations that allow us to assign a value to more than one relvar as an atomic operation.

A data model evolves over time, some orthogonality violations disappear but others come into existence. For example, Date and Darwen [3, p. 436] note based on SQL:1999 that only the surrogate column of a *typed* base table can use "VALUES ARE SYSTEM GENERATED" option. However, SQL:2003 allows us to use identity columns in the base tables that are not typed.

How is this topic associated with metamodels? If a metamodel contains a *generalization* relationship between metaclasses (see Fig. 3) so that some attributes and/or relationships are at the superclass level and some are at the subclass level, then it could be a sign of a possible violation of requirement (c). For example, in case of problem (10) we could replace the letters in the figure in the following way: A – User-defined type, B – Structured type, C – Distinct type, D – Typed table.

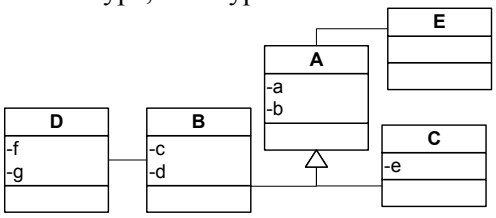

Fig. 3 Constructs in a metamodel that identify possible violation of the orthogonality principle

6 Conclusions

In this paper we explained why it is advantageous to create metamodel of a data model. We demonstrated that a metamodel could be used in order to find similarities and differences with other data models. It could also be used in order to calculate metrics values. These values are useful if we know metrics values of other data models as well. In this case they show relative complexity of each model. Inspection of a metamodel helps to find possible violations of orthogonality principle in language design.

We evaluated two approaches of object-relational data model based on their metamodels. We found that the underlying data model of SQL:2003 (OR_{SQL}) is more complex than the underlying data model of The Third Manifesto but it does not mean that the former is "better". For example, OR_{SQL} has many violations of orthogonality principle. We presented 14 violations.

Participation in the conference was supported by the Estonian Information Technology Foundation (by the Nations Support Program for the ICT in Higher Education "Tiger University").

References:

[1] Melton J, ISO/IEC 9075-2:2003 (E) Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation), August, 2003. Retrieved December 26, 2004, from http://www.wiscorp.com/SQLStandards.html

[2] Date CJ, Darwen H, *Databases, Types and the Relational Model*, 3rd edn., Addison Wesley, 2006

[3] Date CJ, Darwen H, *Foundation for Future Database Systems: The Third Manifesto*, 2nd edn., Addison-Wesley, 2000

[4] Date CJ, *An Introduction to Database Systems*, 8th edn., Pearson/Addison Wesley, 2003

[5] Greenfield J, Short K, Cook S, Kent S, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley Publishing, Inc, 2004.

[6] OMG UML 2.0 Superstructure Specification, formal/05-07-04

[7] Henderson-Sellers B, Atkinson C, Kühne T, Gonzalez-Perez C, Understanding Meta-modelling, Tutorial in 22nd International Conference on Conceptual Modeling ER2003, 15 October 2003. Retrieved November 20, 2004 from http://www.er.byu.edu/er2003/slides/ER2003T1HendersonSellers.pdf

[8] Eessaar E, *Relational and Object-Relational Database Management Systems as Platforms for Managing Software Engineering Artifacts*, Ph.D. thesis, Tallinn University of Technology, 2006. Available at http://digi.lib.ttu.ee/i/?85

[9] Lavazza L, Agostini A, Automated Measurement of UML Models: an open toolset approach, *Journal of Object Technology*, Vol. 4, No. 4, May-June 2005

[10] Siau K, Rossi M, Evaluation of Information Modeling Methods -- A Review, In: Proc. of the 31st Annual Hawaii Int. Conf. on System Sciences, Vol. 5, 1998, p. 314.

[11] Rossi M, Brinkkemper S, Complexity Metrics for Systems Development Methods and Techniques, *Information Systems*, Vol. 21, No. 2, 1996, pp. 209-227.

[12] Rittgen P, Translating Metaphors into Design Patterns, Advances in Information Systems Development, Vol. 1, Springer, 2006 pp. 425-436.