Information Systems 60 (2016) 85–94

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Declarative semantics of transactions in ORM

E.O. de Brock [1]

University of Groningen, Faculty of Economics and Business, P.O. Box 800, 9700 AV Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 6 March 2016
Accepted 10 March 2016
Available online 19 March 2016

Keywords:
Transaction modeling
Transaction language design
Semantics
Rollback
ORM-method
Transaction verbalization

ABSTRACT

In order to specify databases completely at the conceptual level, conceptual database specification languages should contain a data *definition* (sub)language (DDL), for specifying *data structures* (+*constraints*), a data *retrieval* (sub)language (DRL), for specifying *queries*, as well as a (declarative) data *manipulation* (sub)language (DML), for specifying *transactions*.

Object Role Modeling (ORM) is a powerful method for *designing* and *querying* database models at the conceptual level. By means of *verbalization* the application is also described in natural language as used by domain experts, for communication and validation purposes. ORM currently comprises a DDL and a DRL (ConQuer). However, the ORM-method does not yet contain an expressive DML for specifying transactions at the conceptual level.

In an earlier paper we designed a syntactic extension of the ORM-method with a DML for specifying transactions at the conceptual level in a purely declarative way. For all transactions we proposed syntaxes, verbalizations, and diagrams. However, we did not give a formal semantics then.

The purpose of this paper is to add a clear, formal and purely declarative semantics to the proposed ORM-transactions. The paper also formally defines rollbacks and illustrates everything with examples (including a solution to a well-known transaction specification problem). The extension of ORM with an expressive set of completely declaratively specified transactions makes ORM complete as a database specification method at the conceptual level.

1. Introduction

In order to specify databases completely at the conceptual level, conceptual database specification languages should contain a data *definition* (sub)language (DDL), i.e., a part for specifying *data structures* (+*constraints*), a data *retrieval* (sub) language (DRL), i.e., a part for specifying *queries*, as well as a (declarative) data *manipulation* (sub)language (DML), i.e. a part for specifying *transactions*.

In the well-known database language SQL for example, the DDL typically contains CREATE-, DROP-, and ALTER-statements, the DRL typically contains SELECT-statements, and the DML typically contains INSERT-, DELETE-, and UPDATE-statements (see e.g. [1]).

Object Role Modeling (ORM) is a powerful method for *designing* and *querying* database models at the conceptual level. By means of *verbalization* the application can also be described in natural language which is easily understood by non-technical users (e.g., domain experts). Verbalization supports communication and validation with domain experts and future users of the system to be developed. Refs. [2] and [3] contain a lot of background information about ORM. ORM is also extensively described in [4]. ORM

E-mail address: E.O.de.Brock@rug.nl
[1] Tel. +31 50 3637315.

http://dx.doi.org/10.1016/j.is.2016.03.005
0306-4379/

started as a DDL. Later on the language was extended with a DRL, called ConQuer ([5,6]).

However, the ORM-method is incomplete in the sense that it does not yet contain a good and sufficiently expressive DML, for specifying and verbalizing transactions at the conceptual level, e.g., in order to discuss (standard, canned and complex) transactions with non-technical users. In this paper we will solve this incompleteness problem. One of our contributions is proposing the right set of primitive transactions, including a clear semantics.

When we look at the situation in comparable languages such as ER and UML, we first note that the Entity-Relationship (ER) conceptual modeling ([7]) does not have any notion of transaction at all. The Unified Modeling Language (UML, see [8]) does not have the notion of database nor does it have the notion of database transaction. The paper [18] tries to add transactions to UML-diagrams, but it leads to a restricted and cumbersome theory. UML is not very well suited for database modeling anyway. ORM provides a simpler, more accurate and more powerful approach to information modeling at the conceptual level than UML ([3,9,10]). Ref. [2] contains various articles comparing UML and ORM.

The basic notion of a *fact type* in ORM is more or less 'comparable' to the notion of a class in UML, an entity type (or relationship) in ER, and a table type in relational database theory.

The operations **add** and **del** in [4] only apply to one fact (instance) at a time. The operation **add** in [4] corresponds to our addition of an instance, treated in Section 3. The operation **del** in [4], removal of an instance, is a very special case of the removal of a subset treated in Section 5. However, only adding or deleting one particular fact at a time is not enough (e.g., deleting all order lines belonging to a given order is a natural counterexample). Moreover, the operation **del** in [4] requires the complete fact to be mentioned; e.g., with a fact type such as *Employee earns Salary* (see the example in Section 6) a removal would look like "**del:** Employee 123 earns Salary 4857". But maybe you only want to say that the salary fact on Employee 123 has to be removed; maybe you don't know the exact salary (or maybe are not even allowed to know the salary).

Balsters et al. mention transactions in [11] and [12], but actually they concentrate on dynamic rules, and do not give a syntax for data manipulation operations as such. In [11], adding actual operations to the ORM-language that explicitly model transactions is mentioned as future work, which it still is up to now. Our paper addresses this open problem.

The ORM-situation sketched above is schematically summarized below:

	SQL	ORM-approach
DDL:	CREATE, DROP, ALTER	+(ORM's origin)
DRL:	SELECT	+(ConQuer)
DML:	INSERT, DELETE, UPDATE	–

In [13] we proposed a syntactic extension of ORM with a DML for specifying transactions at the conceptual level in a purely declarative way, to be easily validated by domain experts. By a transaction we informally mean an *attempt* to update the contents of the database; the attempt fails when any specified constraint will be violated (also known as a *rollback*). We will call our DML *ConTrans* (for Conceptual Transaction), analogous to *ConQuer* (which stands for Conceptual Query).

In [13] we introduced a collection of four basic classes of specifiable transactions: add a fact, add a query result, remove a subset, and change a subset. We also introduced compound transactions in ORM. Together they constitute an expressive collection of transactions. Other conceptual constructs are not needed.

In principle, a database treats only one transaction at a time (except for compound transactions). We note however that the order of application of two transactions can be semantically relevant: the transaction sequence T1**;** T2 (where the semicolon means 'followed by') can have another end result than the transaction sequence T2**;** T1. For instance, let T1 be the transaction to increase the salary of all employees living in London with 10% and T2 be the transaction to increase the salary of all employees with a salary less than 5000 with 100. Then the transaction sequence T1**;** T2 implies that all Londoners with a salary originally less than 4545 will earn 10% more plus an additional 100 (i.e., 1.10*Old_Salary+100), while the transaction sequence T2**;** T1 implies that all Londoners with a salary originally less than 5000 will earn 10% more plus an additional 110, namely 1.10*(Old_Salary+100).

Although one table (resp. one tuple) in a relational database is directly associated to *a set of* fact types (resp. facts) in ORM, it is a crucial decision in our theory that the smallest *unit* of transaction is a fact, whereas in SQL it is a tuple. Nevertheless a nice feature of the syntax (and semantics) of transactions we propose for ORM at the conceptual level is that it is similar to that of SQL, and that other conceptual constructs were not necessary. The basic transactions attempt to populate (add), de-populate (remove) or re-populate (change) a fact type (or an independent object type). An independent object type can be considered as a fact type having only one role associated to it. Recall that the attempt fails when any specified constraint will be violated. Each basic transaction will apply to only one fact type (or one independent object type) at a time. The proposal is inspired by (the expressiveness of) the DML of SQL.

Because the ORM-tradition distinguishes several kinds of specifications, in [13] we proposed for each transaction a syntax (in a formal language), a verbalization (in natural language, "fully communication oriented" [14]), and a diagram (in a graphical language). A verbalization of a transaction (i.e., in natural language) is intended for communication and validation with domain experts and future users.