ORM Logic-Based English (OLE) and the ORM ReDesigner Tool: Fact-Based Reengineering and Migration of Relational Databases

Herman Balsters

University of Groningen
Faculty of Economics and Business
P.O. Box 800 9700 AV Groningen,
The Netherlands
h.balsters@rug.nl

Abstract. The problem of database reengineering stems from (legacy) databases that are hard to understand (incorrect-, incomplete- or missing semantics) or that perform inefficiently. Reengineering is often cumbersome due to lack of semantics of the original database, and often the data migration target is also unclear. This paper addresses those two problems. We shall show how fact-based modeling, in particular ORM and its representation in (sugared) Sorted Logic, can help in reengineering (relational) databases. We reconstruct the semantics of the source database by offering a set of natural-language sentences capturing conceptual structure and constraints of the source. These sentences are written in a structured natural language format, coined as OLE: *ORM Logic-based English*. OLE is then used to define the mappings from the original source to a reengineered and restructured target database. We shall also discuss the ORMReDesigner: a semi-automatic tool, based on OLE and NORMA, available as a research prototype, used for reengineering and migrating relational databases.

Keywords: Reengineering, data migration, fact-based modeling, structured English, derivation rules.

0 Introduction

Object-Role Modeling (ORM) is a fact-oriented approach for modeling information in terms of the underlying facts, where facts and rules may be verbalized in a language easily understandable by non-technical domain experts. In contrast to Entity-Relationship (ER) modeling [8] and Unified Modeling Language (UML) class diagrams [27], ORM models are attribute-free, treating all facts as relationships (unary, binary, ternary etc.). ORM does, however, include procedures (e.g. the RMap [18]) for mapping to attribute-based structures, such as those of ER or UML. For a basic introduction to ORM see [17], for a thorough treatment see [18]. We will use the term Fact-based modelling (FBM [14]) as the general name of several fact-based conceptual data modelling dialects, such as ORM, Natural language Information

P. Herrero et al. (Eds.): OTM 2012 Workshops, LNCS 7567, pp. 358–367, 2012.

Analysis Method (NIAM) [17], and Fully-Communication Oriented Information Modeling (FCO-IM) [1].

Database reengineering of a source database, and subsequent data migration to some target database, is an important activity in both academia and current database practice [2,3,4,5,12,21,22,24,25,28,29]. Reengineering is often cumbersome due to lack of semantics of the original database, and often the data migration target is also unclear. In practice, it may be the case that the target schema is fixed, in which case there is similarity with the data exchange problem [13]. In our case, however, we deal with the situation where we are free in choosing a suitable target schema.

Reengineering involves the reconstruction of the semantics of some source database. Subsequent data migration involves mapping a source database schema to a target database schema. We shall show how Fact-based modeling, in particular ORM and its representation in (sugared) Sorted Logic [7], can help in reengineering (relational) databases. We reconstruct the semantics of each table from the source database by offering a set of natural-language sentences capturing the intended semantics of that table, as well as its conceptual structure and constraints. These sentences are written in a structured natural language format, coined as OLE: *ORM Logic-based English*. OLE closely follows the representation of ORM models in terms of Sorted Logic as offered in [15]. OLE is used to specify elementary facts and constraints in ORM. The OLE-syntax is consistent with a subset of Common Logic Controlled English (CLCE, [11]). CLCE has been adopted by the OMG [26], and is accepted as an ISO standard [23]. A related specification language used to describe the contents (facts and rules) of ORM models is CQL [20]. Texts in CQL can be seen as an alternative for ORM diagrams to model the Universe of Discourse (UoD). FORML2 [19], a highly expressive natural-language based specification language for derivation rules in ORM, but FORML2 (unlike OLE) is not directly based on logic.

OLE expressions are used as input for the NORMA tool [9,10], to generate associated ORM models. OLE will be used to read and write derivation rules for derived fact types. Since the derivation rules are written in a structured natural language format, they can be easily validated by non-technical domain experts. Since NORMA does not support view definitions in SQL, we will use OLE, and the so-called CoRef-version of ORM [18], to eventually obtain fully-defined view-version DDL. Our method for reengineering and migrating is supported by a semi-automatic tool [28] called ORMReDesigner. In section 1, we describe the migration problem using a simple case, and offer an introduction to the OLE language. Section 2 shows how to apply OLE to reengineer the semantics of a source database. Section 3 shows how OLE offers derivation rules that eventually lead to a view-version DDL of the target database. Section 4 offers some notes on OLE and the ORMReDesigner tool. Finally, in Section 5, we offer some conclusions. This paper uses basic ORM notations, cf. [16,18].

1 Structured English for Reengineering

The main reasons for reengineering an existing database system usually deal with that system being incorrect, incomplete, or inefficient. Reengineering, or restructuring, such a system entails that the redesigner has to construct a model of the existing

system, as well as construct a model of the improved database. Designers want to specify facts and rules to describe their business context, and how that context relates to the database that has to be redesigned. For that purpose, they don't want to use SQL, e.g., since SQL -for this purpose- will be too technical, too implementation directed, and too difficult to validate directly. What they would like, is a language that is easy to read and write by both designers and domain experts: hence, the facts and rules should be able to be validated by a non-technical domain expert. At the same time, the language has to be expressive enough to capture typical business rules and constraints, and also be precise enough to eventually translate to a technical platform.

We will reconstruct the semantics of each database table from a given source database by offering a corresponding natural language sentence capturing the intended meaning of the table heading. The first offering of this sentence can be in informal semantics/syntax. This soft-semantics sentence is then rewritten to a structured format, coined as **OLE**: *ORM Logic-based English*. This format is mandatory and fixed for the user. OLE is a sugared form of Sorted Logic, and is easy to read and write by non-technical domain experts. OLE expressions are expressed as elementary facts. Objects are categorized as entities and values, and entities are offered reference modes. All elementary facts in OLE may be added with the following relevant constraints: uniqueness, mandatory constraints , external uniqueness, and subtyping. Entities may also be identified by using compound reference schemes.

Let's start by offering an example of a very simple database in need of redesign. Consider the following source table

STUDENT(nr, project, mentor, projectDescription)

where this table has the key (nr, project). The semantics of this table could be offered by: *"Student is identified by number and has mentor for the project described by projectDescription"*. Note that this semantics is offered as an informal English sentence, hence providing a soft semantics definition of the table meaning. Also note that this table is a candidate for reengineering, because the semantics definition reveals that we are dealing with a composite fact type. In general, we could say that a table is in *acceptable format*, when the table is normalized (no redundancy), and has a correct and complete semantics (including all relevant constraints and derivation rules). Our example table is therefore not acceptable, because it contains a non-key dependency *project* → *projectDescription*, telling us that the STUDENT table is not normalized.

The following section introduces the OLE language and ORMReDesigner method, and shows how to apply this method to our example source table STUDENT in order to transform it into an acceptable database.

2 OLE and ORM ReDesigner Method

In order to reengineer our example table, we (i.e, the redesigner) try to discover (together with the domain expert) all of the elementary facts within our soft-semantics description of the source table. In our case, we find the following fact types

- Student has Mentor for Project
- Project is described by ProjectDescription

The elementary fact types are the basic building blocks for our target model (avoiding eventual redundancy in the resulting databases).

Subsequently, for each elementary fact, we list all entities and values, along with reference modes and associated basic types. In our case we find

- Student **is Entity** (**and is referred to by** nr (**of type** integer))
- Mentor **is Entity** (**and is referred to by** name (**of type** varchar(25)))
- Project **is Entity** (**and is referred to by** code (**of type** char(4)))
- Description **is Value** (**with Datatype** varchar(20))

The third step involves listing all uniqueness constraints for each elementary fact

- **for each** Student **and** Project, **there is at most one** Mentor, **such that** Student has Mentor for Project
- **for each** Project, **there is exactly one** Description, **such that** Project has Description

Note that all of the expressions listed above are examples of valid OLE-expressions (where bold-case syntax indicates a reserved word). Also note that any **such that**-part always binds to the **there is**-part that is nearest to it; in that way no ambiguity can arise from its reading. In Sorted Logic the last two OLE-expressions would read as

- ∀x:Student ∀y:Project $\exists^{0..1}$z:Mentor Student has Mentor for Project
- ∀x:Project $\exists^{1}$y:Description Project has Description

Our ORM ReDesigner tool will assist in offering suggestions for the second and third steps listed above. Inputting the fact types, entities, values, and constraints into the *NORMA tool* will yield the following proper ORM model (intended to replace the schema of the original STUDENT base table)

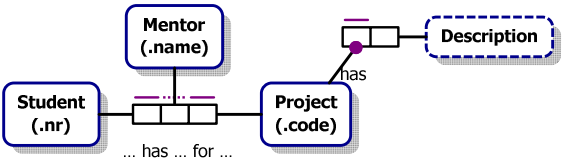


Fig. 1. Target Model M1

Our aim is to eventually construct a set of views in SQL that serve as a (virtual) target database: the original source database STUDENT is used to populate this set of views (which we shall call the Target View). In order to do so, we will perform a *binary breakdown* of the Target model M1 by taking the *co-referenced* version of model M1. In CoRef-ORM, only binary fact types with functional uniqueness constraints have to be considered, along with subtyping relations and all constraints. The CoRef-version of model M1 results in the following model M2

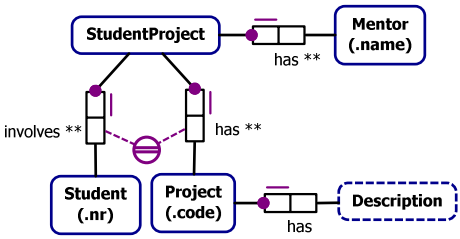


Fig. 2. Target Model M2

Model M2 is the direct conceptual counterpart of a relational view used as the starting point for an eventual implementation of our target view. M2 has the following advantages over model M1

- it is the model format that is *as close as possible* to the desired view-table format
- it is, however, still possible to reason *completely on the conceptual level* about M2 (semantics are given by fully validated elementary fact statements), this in contrast to the corresponding relational view DB-Target (table headings do **not** offer the underlying fact types!)
- it offers fully re-engineered, re-structured and validated semantics of the original source database

For all of the above-mentioned reasons, we use this binary model M2 as the basis for offering the derivation rules that will eventually calculate the population of our Target View. These derivation rules (fully defined in OLE) offer what we call the *binary build-up* of the restructured database (our Target View). We note that very basic CoRef-versions of ORM models can also be *generated* by the NORMA tool (the modeler will, however, often have to redo the layout of the generated model). In deriving the population of the target model M2 from an assumed population of the source model M1, we employ the following rules (specified in OLE):

(1) **for each** Student, Project, **and** Mentor,
if Student has Mentor for Project **then**
there is some StudentProject , **such that**
StudentProject involves Student **and**
StudentProject involves Project **and**
StudentProject has Mentor

(2) **for each** StudentProject, Student, Project, **and** Mentor,
if StudentProject involves Student **and**
StudentProject has Project **and**
StudentProject has Mentor **then**
Student has Mentor for Project

Rule (1) tells us that entity type StudentProject is ***at least*** populated by taking instances from fact type Student has Mentor for Project from Model M1. Rule (2) tells us that

entity type StudentProject is ***at most*** populated by instances taken from fact type Student has Mentor for Project from Model M1. We can now prove the following properties, written in OLE (proof is based on previous rules (1, 2) and the uniqueness constraints taken from M1):

(i) **for each** Student **and** Project,
there is at most one StudentProject, **such that**
StudentProject involves Student **and**
StudentProject has Project

(ii) **for each** StudentProject,
there is exactly one Student, **such that**
StudentProject involves Student

(iii) **for each** StudentProject,
there is exactly one Project, **such that**
StudentProject has Project

We shall refrain from offering the actual proofs (that would be out of the scope of this paper), but these could be offered by coding OLE-expressions (1, 2) as well as the OLE-expressions (i, ii, iii) as formulas in first-order logic, and then provide the proofs by using, e.g., natural deduction trees [15] .
Properties (i), (ii), and (iii) now allow us to introduce the following (derived) *compound reference scheme* for StudentProject written in OLE:

StudentProject **is Entity** (**and is referred to by** Student **and** Project, **where**
StudentProject involves Student **and** StudentProject has Project)

Rules (1, 2) are sufficient for target model M2 to have all of the desired properties of the CoRef-version of model M1. Equipped with this knowledge, we shall proceed, in the subsequent section, by defining derivation rules for the target view.

3 Derivation Rules for the Target View

In this section, we aim to define a virtual target database (Target View) with the following schema where all of the attributes and all of the constraint properties (PK, FK) are derived.

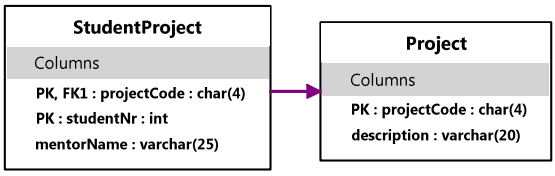
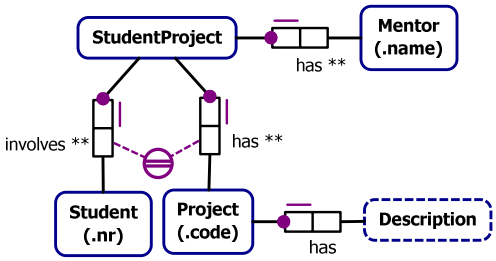


Fig. 3. Target View

We note that this target view has model M2 as its (full) conceptual counterpa



where all of the (binary and functional) fact types and all of the constraint properties (uniqueness constraints, mandatories) are derived (cf. previous section).

We now offer a list of derivation rules -specified in OLE- that takes the source base relation STUDENT(nr,mentor,project,projectDescription) as the input schema, and offers definitions of the derived attributes in the target view:

(1) **for each** Nr, Name, **and** Code,
there is some StudentProject, **such that**
the studentNr **of** StudentProject **is that** Nr **and**
the mentorName **of** StudentProject **is that** Name **and**
the projectCode **of** StudentProject **is that** Code
iff
there is some STUDENT, **such that**
STUDENT.nr = Nr **and**
STUDENT.project = Code **and**
STUDENT.mentor = Nam

(2) **for each** Nr, Name, String **and** Code,
there is some Project, **such that**
the projectCode **of** Project **is that** Code **and**
the Description **of** Project **is that** Name
iff
there is some STUDENT, **such that**
STUDENT.nr = Nr **and**
STUDENT.project = Code **and**
STUDENT.mentor = Name **and**
STUDENT.projectDescription = **that** String

We note that definition (1) is based on the (derived) compound reference scheme for StudentProject, proved in the previous section.

Here, we have used a general scheme for defining views. Say that we have some view V, with attributes x_1, .., x_n, (of domain type X_1, .., X_n respectively), and that the view is to be defined in terms of a set of base relations B_1, .., B_m. If the predicate describing the definition of V is denoted by Ψ, then the general formula (in Sorted Logic) stating that V is defined in terms of Ψ can be stated as

$\forall x_1{:}X_1, .., \forall x_n{:}X_n\ [\exists v{:}V\ v(x_1, .., x_n) \Leftrightarrow \exists b_1{:}B_1, .., \exists b_m{:}B_m\ \Psi(b1, .., bm, x_1, .., x_n)]$

Note that our OLE-specifications (1,2) offered directly above, refer to the base relation STUDENT. The general idea is that we take the original base relations of the (to be reengineered) source database, which subsequently offer the input population for the derivation rules of a target view. In this manner we migrate the data from the source database to a target view. Once this process has been completed, we can change the status of the target view into that of the actual target database. OLE-definitions offer an alternative for SQL-syntax; an OLE-text easy to read and write – and, hence, validated- by non-technical domain experts.

The ORMReDesigner tool assists in writing OLE-specifications like (1, 2) listed above; in this case the tool will generate the following specifications

Define the target table StudentProject
with column names StudentNr, projectCode, mentorName **as:**
rows consisting of STUDENT.nr, STUDENT.project, STUDENT.mentor
for each STUDENT

Define the target table Project
with column names projectCode, description **as:**
rows consisting of STUDENT.project, STUDENT.projectDescription
for each STUDENT

and subsequently the tool generates the following view definitions in SQL

```
CREATE VIEW `Project` (`projectCode`, `descriptionName`)
AS SELECT `STUDENT`.`Project`, `STUDENT`.`ProjectDescription`
FROM `STUDENT`;

CREATE VIEW `StudentProject` (`studentNr`, `projectCode`, `mentorName`)
AS SELECT `STUDENT`.`nr`, `STUDENT`.`Project`, `STUDENT`.`Mentor`
FROM `STUDENT`;
```

4 Some Notes on OLE and the ORMReDesigner

OLE has been offered a full BNF language specification [6], which was implemented in the ORMReDesigner [28]. In [6] we have made a comparison between OLE, ORM, and first-order predicate logic (in particular Sorted Logic), and we demonstrated that

- OLE contains at least the expressive power of Sorted Logic
- OLE can handle most of ORM's fact type constructions and constraints

Space limitations, however, do not allow for more details on the above-mentioned aspects of OLE. In this paper, we have taken only a simple base relation as a case study; we have, however, successfully applied OLE and the ORMReDesigner to larger case studies (e.g. migrating patient medical records from two different hospitals

to a common EPD (Electronic Patient Dossier) database). Both OLE and the ORMReDesigner proved in practice to be reasonably simple to use, both by the non-technical domain expert (involved in the validation of the OLE-specifications) and professional designer (using the tool to migrate databases).

5 Conclusions

The problem of database reengineering stems from (legacy) databases that are hard to understand (incorrect and/or incomplete semantics) or that perform inefficiently. Database reengineering is often cumbersome due to lack of semantics of the original database, and often the data migration target is also unclear. We have shown how fact-based modeling, in particular ORM and its representation in (sugared) Sorted Logic, can help in reengineering (relational) databases. We reconstruct the semantics of the source database by offering a set of natural-language sentences capturing the conceptual structure and constraints of the source. These sentences are written in a structured natural language format, coined as OLE: *ORM Logic-based English*. OLE is used to define the mappings from the original source to a reengineered target database. We have also discussed the ORM ReDesigner: a semi-automatic tool, based on OLE and NORMA, available as a research prototype, used for reengineering and migrating relational databases.

References

1. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication Oriented Information Modelling. Ten Hagen Stam (2000)
2. Balsters, H., de Brock, E.O.: Integration of Integrity Constraints in Federated Schemata Based on Tight Constraining. In: Meersman, R. (ed.) OTM 2004. LNCS, vol. 3290, pp. 748–767. Springer, Heidelberg (2004)
3. Balsters, H., Halpin, T.: Modeling data federations in ORM. In: Meersman, R., Tari, Z. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805, pp. 657–666. Springer, Heidelberg (2007)
4. Balsters, H., Huitema, G.B.: Semantics of Outsourced and Interoperable Information Systems. In: Morel, M. (ed.) Interoperability for Enterprize Software and Applications. Springer, Berlin (2007)
5. Balsters, H., Haarsma, B.: An ORM-Driven Implementation Framework for Database Federations. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 659–670. Springer, Heidelberg (2009)
6. Balsters, H.: ORM Logic-based English (OLE) and the ORM ReDesigner tool: Fact-based Re-engineering and Migration of Relational Databases, Technical Report, Faculty of Economics and Business (May 2012)
7. Chang, C.C., Keisler, H.J.: Model Theory. Studies in Logic and the Foundations of Mathematics, vol. 73. North Holland Publishing Company (1977)
8. Chen, P.P.: The entity-relationship model—towards a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
9. Curland, M., Halpin, T.: Model Driven Development with NORMA. In: Proc. 40th Int. Conf. on System Sciences (HICSS-40). IEEE Computer Society (January 2007)

10. Curland, M., Halpin, T.: The Norma Tool for Orm 2. In: Pernici, B. (ed.) Advanced Information Systems Engineering. LNCS, vol. 6051. Springer, Heidelberg (2010)
11. Common Logic, http://www.Common-Logic.org
12. Drumm, C., Schmitt, M., Do, H.H., Rahm, E.: Quickmig: Automatic schema matching for data migration projects (2007)
13. Embley, D.W., Xu, L., Ding, Y.: Automatic direct and indirect schema mapping: experiences and lessons learned. SIGMOD Record 33(4), 14–19 (2004)
14. FBM working group: Fact-based modeling exchange schema. Version 20111021c (2011), http://www.factbasedmodeling.org/
15. Halpin, T.: A Logical Analysis of Information Systems: static aspects of the data-oriented perspective. Doctoral dissertation, University of Queensland (1989), http://www.orm.net/Halpin_PhDthesis.pdf
16. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
17. Halpin, T.: ORM/NIAM Object-Role Modeling. In: Bernus, P., Mertins, K., Schmidt, G. (eds.) Handbook on Information Systems Architectures, 2nd edn., pp. 81–103. Springer, Heidelberg (2006)
18. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
19. Halpin, T., Wijbenga, J.P.: FORML 2. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 247–260. Springer, Heidelberg (2010)
20. Heath, C.: The constellation query language. In: OTM 2009: ORM Workshop, OTM 2009, pp. 1–10 (2009)
21. Henrard, J., Roland, D., Cleve, A., Hainaut, J.-L.: An Industrial Experience Report on Legacy Data-Intensive System Migration. In: IEEE International Conference on Software Maintenance, pp. 473–476 (2007)
22. Hull, R.: Managing Semantic Heterogeneity in Databases. In: ACM PODS 1997. ACM Press (1997)
23. ISO, http://standards.iso.org/ittf/licence.html
24. Lenzerini, M.: Data integration: a theoretical perspective. In: ACM PODS 2002. ACM Press (2002)
25. Lin, C.Y.: Migrating to relational systems: Problems, methods, and strategies. Contemporary Management Research 4(4), 369–380 (2008)
26. OMG/FUML, http://www.omg.org/spec/FUML
27. OMG/UML: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.3 (May 2010)
28. Pastoor, J.J.: Database-migratie en -normalisatie in ORM Logic-based English (in Dutch), Bachelor's thesis, University of Groningen (2012)
29. Sluis, T.C.: The ORM Infusion Migration Method, Master's Thesis, University of Groningen (2011)
30. Wu, L., Sahraoui, H., Valtchev, P.: Coping with legacy system migration complexity. In: 10th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2005, pp. 600–609 (2005)