TRANSFORMATION OF RELATIONAL DATABASE MODELS INTO OBJECT-ORIENTED DATABASE MODELS

V. Holub

Czech University of Life Sciences, Faculty of Economics and Management, Department of Information Engineering, Prague, Czech Republic

This paper presents a method to transform relation database models into object ones. Unlike other papers and works, we use standard technologies and tools and show that even poorly designed relational data model may be automatically adjusted and then transformed. We also present an UML extension to support the transformation. Available outputs also include an entity-relation model.

MDA; UML; OCL; transformation rules; mapping; entity-relationship; semantic enrichment

INTRODUCTION

Database Systems

Database systems (DS) generally serve one common purpose: information storage and retrieval. From the information-system viewpoint, database systems provide these services to other subsystems through a defined *interface*, which (to certain extent) decouples means of DS integration from an internal *database management system* (DBMS) implementation.

A crucial factor that determines capabilities (in respect to quality of provided services) of a DS is a *data metamodel* (DMM). Since first DS appeared, many different DBMSs were developed. However, within a given DMM only relatively minor distinctions between DBMSs exist. On the other hand, when aiming for a different DMM, DBMS must be completely re-engineered. This principle also applies for interfaces.

In respect to a data model, we commonly distinguish two kinds of database systems:
- relational DSs (RDSs),
- object DSs (ODSs).

Formally, interfaces are represented by a language, each having its syntax and semantics. For RDSs and ODSs, the interface languages are SQL (data-modification DML part) and a subset of an object-oriented language (preferably the one that other, integrated subsystems, are written in), respectively.

In order to overcome one of many drawbacks that limits RDSs usage (i.e. discrepancy between reality-close object-oriented paradigm in present programming languages and outdated relational DMM, known as a "semantic gap"), some relational DSs are equipped with a mapping layer that performs real-time conversion between a DBMS-native relational interface and an outer interface that complies with object-oriented paradigm. These DSs, however, are still to be considered relational.

DMMs that preceded *relational DMM* (RDMM) and *object DMM* (ODMM), namely hierarchical and network DMMs, are beyond scope of this paper.

Database Systems Transformation

Presently available works (A l h a j j, 2003; A n d e r s s o n, 1994; H a i n a u t, 1998; P r e m e r l a n i, B l a h a, 1993, among others) that deal with DS or *data model* (DM) transformation address partial issues only, not the problem domain as a whole. In general, to successfully transform a data model, following these basic steps is inevitable:
1. Obtain source *relational DM* (RDM).
2. Enrich source RDM in order to increase its semantics.
3. Transform RDM to *object DM* (ODM), using information gathered in semantic enrichment.

Some of techniques applicable to fulfilling step 2 are explored by a discipline of *reverse engineering* (RE). For ODM creation, loss-free mapping shall be used. Steps 2 and 3 put together form a process that is a logical opposite to a forward-wise relational database design (semantic enrichment produces information once lost, mapping outputs become inputs and v.v.).

Relational Database Systems in Real Life

Relational data models that are commonly employed in enterprise environment are rarely available in form suitable for transformation. However, any of works mentioned above simply puts well-designed data model as a prerequisite. Such a requirement turns out to be unrealistic in real world.

Common violations of design principles may include (according to B l a h a, P r e m e r l a n i, 1995; H a i n a u t, 1998; P e t i t et al., 1994; P r e m e r l a n i, B l a h a, 1993):
- massive denormalization done in favor of performance,
- poor design resulting from frequent data model updates that are performed by unqualified personnel or under time pressures,
- implicit data structures that are modeled outside data model (e.g. in application layer),

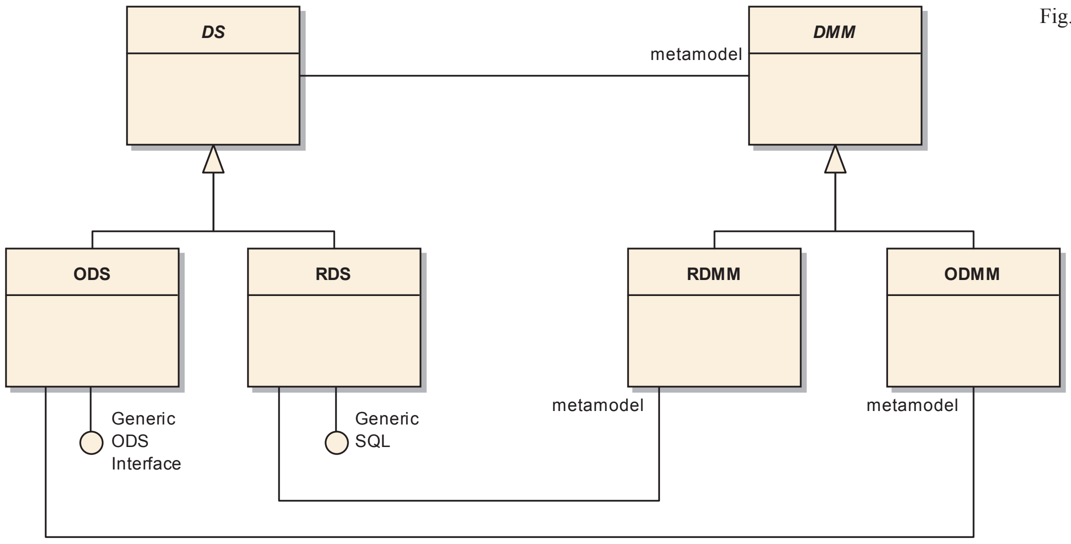

Fig. 1. Database systems

- presence of data structures that are unrelated to modeled domain (e.g. irrelevant technology constructs),
- etc. …

Applying some of existing methods (sources above) leads to unexpected results and design flaws of source DM propagate to DM.

MATERIAL AND METHODS

Key requirements of the transformation method were set as follows:

- use standard and proven tools and techniques with widespread knowledge and usage of,
- make transformation as transparent as possible (i.e. define steps with clear inputs, transformations and outputs; put in a flow, these steps shall constitute the transformation process),
- avoid excessive user interaction,
- if possible, create a technology-independent domain model that may be subsequently used for further maintenance and development of the system that is being transformed.

Tools and techniques that take part in the proposed transformation process include: modeling language that is not limited to one particular metamodel or is extendible to fit needed metamodels, model transformation framework (either self-developed or generally available), modeling environment that supports user-defined transformations (i.e. suitable CASE tool), and a language to express transformation rules. To avoid possible incompatibility threat, most of these we picked from a portfolio provided by OMG (Object Management Group) – namely transformation framework, modeling and transformation language.

MDA Framework

MDA (Model Driven Architecture) is mostly used in enterprise segment to deal with large and complex IS infrastructure that needs to be updated frequently. The general idea is that a vast effort is inadequately spent projecting business needs into program code. During the development process, many intermediate artifacts (models) are created, which are later difficult and expensive to maintain. To speed up and ease the initial development and further maintenance, MDA introduces a framework that allows automated transitions between models on different *levels of abstraction* (LOA). Three different LOAs are defined – *computation independent model* (CIM), *platform independent model* (PIM) and *platform specific model* (PSM). In UP (Unified Process) terminology, the models correspond to business, analysis and design models, respectively.

Mapping of cross-level elements is specified at a *metamodel* (MM) level. Hence, the transformation is *meta-*

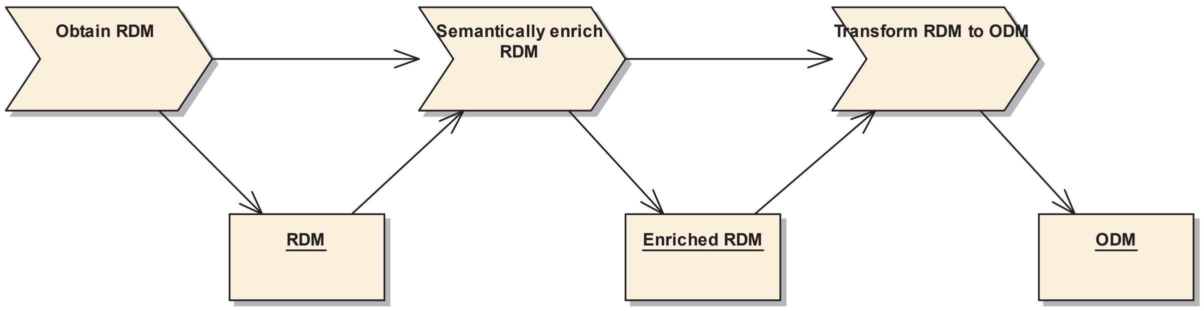

Fig. 2. Transformation process (simplified)

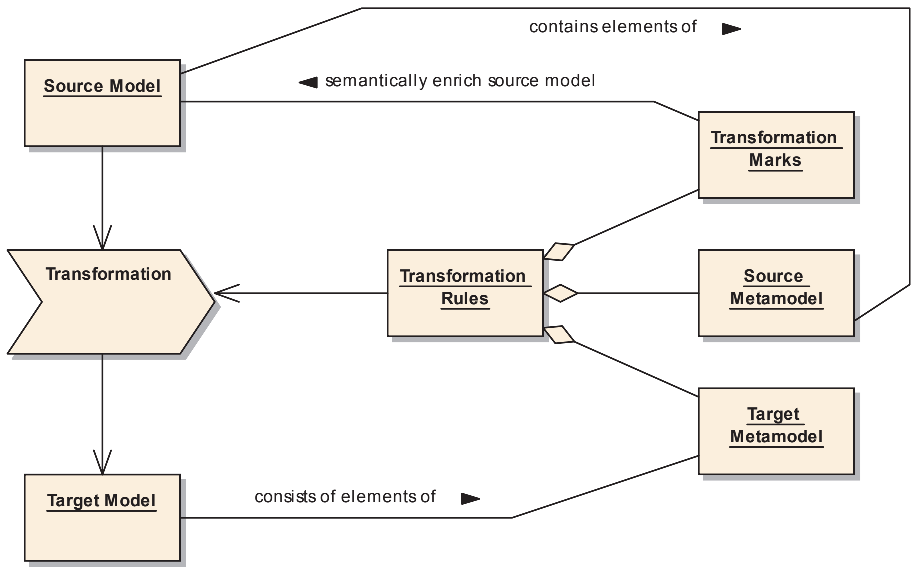

Fig. 3. MDA transformation

model based. However, moving from each LOA to a more concrete one, additional information must be provided in order to allow refinement of the output model. Such information is called *mark*. Using mapping defined on metamodel level and the source model enriched by marks, unambiguous and clear automatic transformation is achieved.

Developing a DM requires LOAs no less than developing any other software applications. First, an *entity-relation model* (ERM) is created, followed by a *logical DM* (LDM) and finally completed by a *physical DM* (PDM). We state, there is a parallel between ERM and CIM, LDM and PIM, and PDM and PSM. LOAs definitions show that our statement fits MDA perfectly.

Note: In database DM transformation, we will traverse from a lower LOA (RDM) to higher LOA (ODM). Still, we will need to semantically enrich the RDM. This is because RDMM contains very little information (it barely evolved over time, while programming languages along with ODS did).

UML Models

Although mainly used as a mere diagramming notation, UML's (Unified Modeling Language) semantic power is best visible when used along with MDA architecture. As a software modeling language, it contains elements that may fully describe a software structure and behavior. Having adopted use cases and business modeling elements, it can also be used to model at the CIM level. Elements of (meta)models of different LOAs may relate via dependencies, tracking CIM elements down to PSM elements.

Primary goal of UML is to provide means of modeling an application layer of a system. It contains packages of

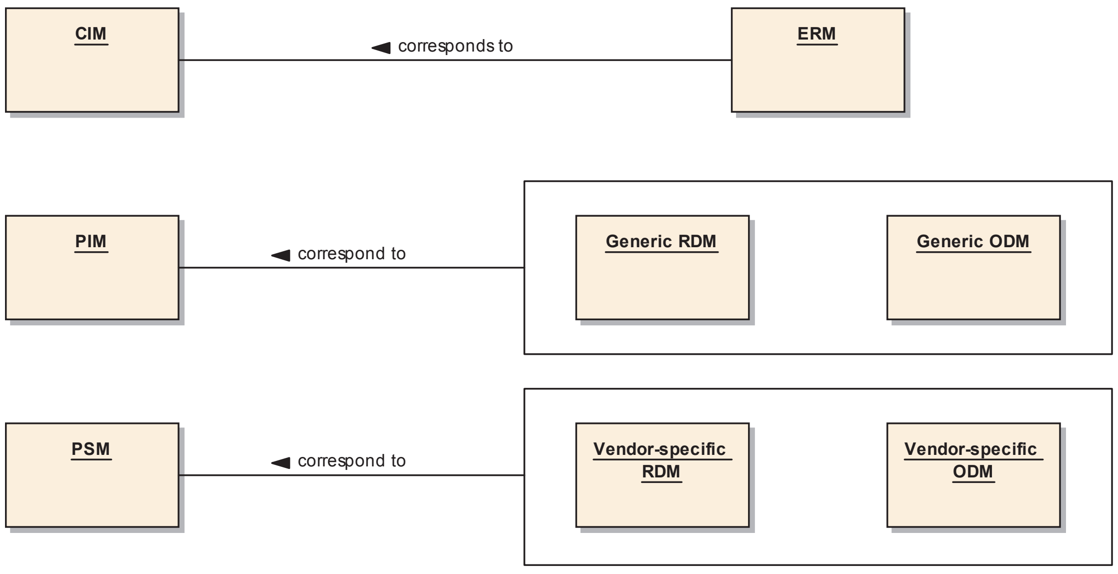

Fig. 4. MDA abstraction levels

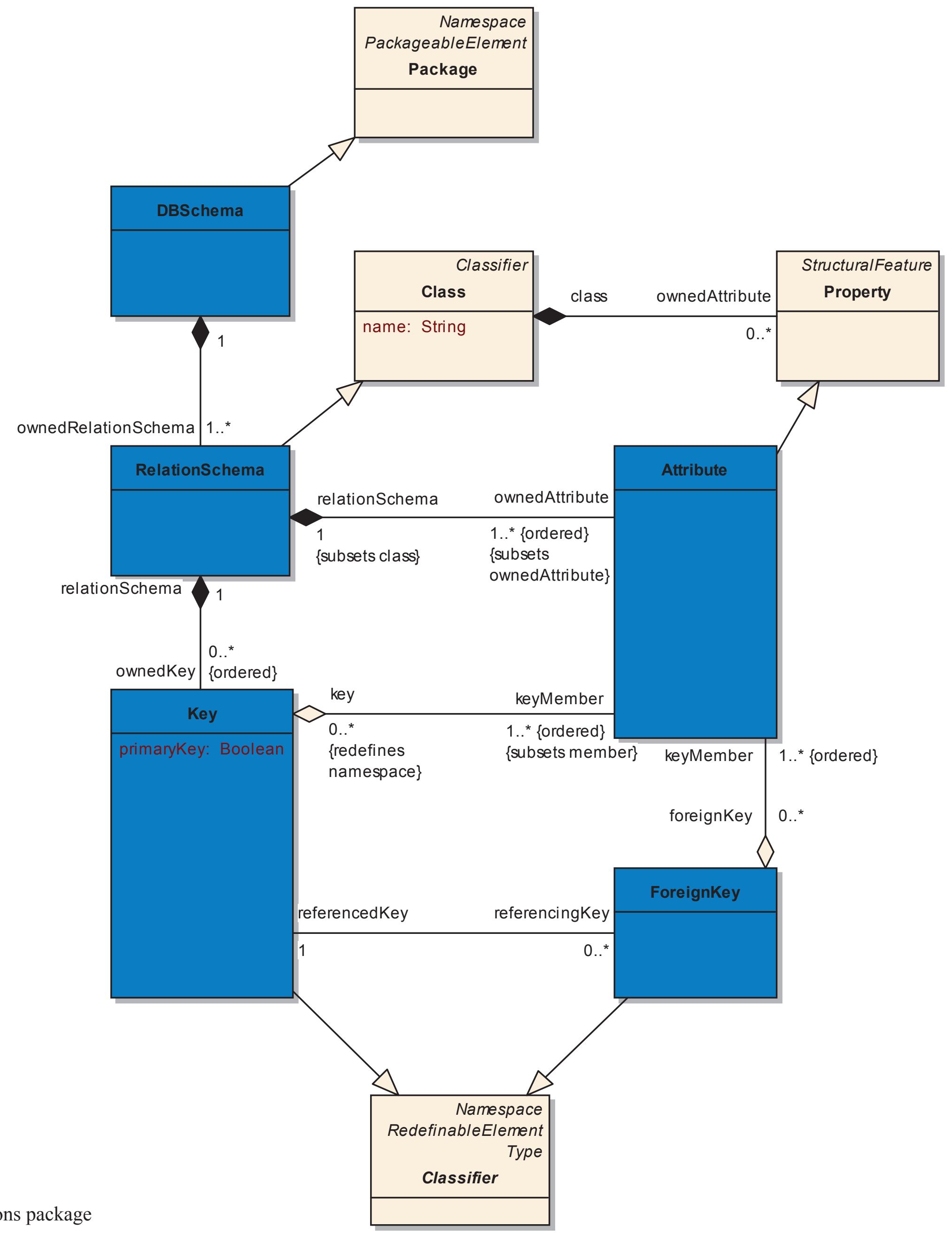

Fig. 5. Relations package

elements, each dedicated to a different aspect of a software product. However, UML core may be easily extended by different mechanisms (importing user-defined packages, which we use, is one of them). Thus, support of modeling DM structure may be added by a package import. We define a custom package *Relations* which holds elements corresponding to RDMM elements.

Also, for functional dependencies modeling (see below) we added package FunctionalDependencies, which appends to UML core the same way.

As object databases (e.g. O2, Jasmine, Caché) reuse the object model, known from object-oriented programming languages, for its data model, no additional extension to support ODMM is necessary. In fact, only a very limited subset of UML Kernel package elements is used to specify an ODMM.

As for the marks that semantically enrich models, we use *UML tags* to store such additional information into. Each mark has a name and value. Each piece of semantic enrichment type (e.g. generalization type, see later) has a dedicated tag name. Corresponding value further specifies semantic enrichment value (e.g. tag GenParentName states that a given *RelationSchema* takes part in a generalization relationship with its parent (more general *RelationSchema*) name stored as a tag value).

OCL Transformations

OCL (Object Constraint Language) is an extension to UML that provides means to further specify required software structure and behavior by adding expressions to UML elements (e.g. methods). By definition, OCL expres-

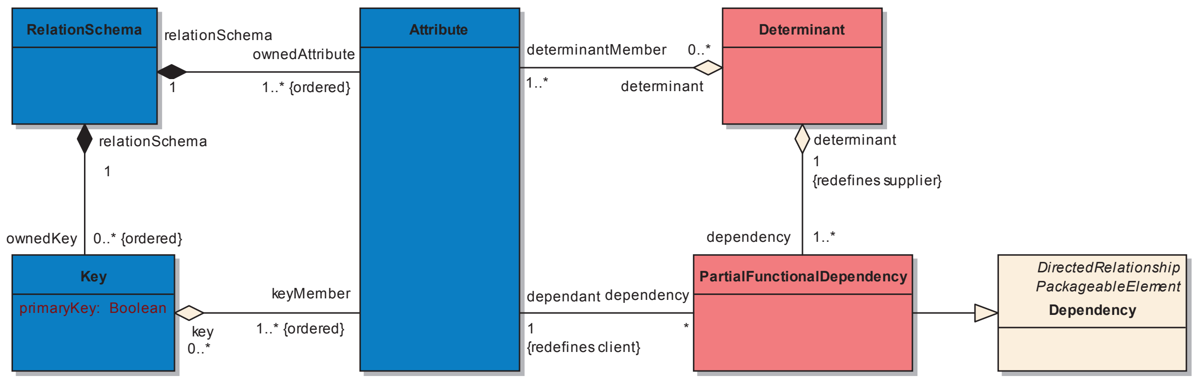

Fig. 6. Functional Dependencies package

sions are limited to query operations. Yet, any object method (including non-query) may be defined by its pre-conditions and post-conditions. For MDA usage, various tools exist to transform OCL expressions present in PIM model to a language-specific code.

Although OCL is not a transformation language, it may well be used as a platform-independent specification of transformation routines. OCL precondition states what condition must be fulfilled to start the transformation step, while OCL post-condition defines what element is to be created in that case. Following example OCL code specifies that each relation (referred to as relation schema) in source PIM model becomes an entity of the same name in target CIM model:

```
pre:
ERModel.Entity.allInstances()–>isEmpty() and
RelationModel.RelationSchema.allInstances()–>notEmpty()

post:
RelationModel.RelationSchema.allInstances()–>iterate(rs |
Let newEntity = Entity.oclIsNew() in
newEntity.owningElement = ERModel and
newEntity.name = rs.name)
```

OCL expressions shall be translated to a language native to an MDA transformation tool.

Transformation Process

The transformation process comprises of two subsequent flows, each split into steps.

The initial flow normalizes and consolidates the source RDM so that it is well-designed and ready for transformation into ODM. It may be omitted if a well-designed RDM is available. It consists of following steps:

1. Remove irrelevant structures
   Such structures are irrelevant to the universe of discourse and shall be excluded from transformation input. We use tag Abandoned of a Boolean type – when set true on a RelationSchema, its holder is excluded from further transformations.
2. Recover implicit keys
   In a real-life RDM, candidate and/or foreign keys might be omitted in RelationSchema declaration – data consistency is then maintained outside DS (e.g. in application layer). However, candidate keys are crucial for functional dependencies analysis and therefore shall be recovered. As we aim for as much automation as possible, we propose to use all three possible enrichment information sources that are known to us: application-layer database queries, DDL declarations that may be mined for more queries (e.g. views definitions) and data itself. The first two turn out to be valuable sources of both candidate and foreign keys, while the latter serves to verify hypotheses of attributes being constituents of candidate keys. Generally, attributes receive (either automatically or by user interaction) various tags that suggest their membership in a key, and the order within one.
3. Define functional dependencies
   Due to the fact that data analysis that would lead to a finite set of functional dependencies is enormously resource-intensive, we propose that user marks dependencies in the extent that his knowledge of a particular DM allows. The routines performed on data basis then serve to verify whether hypotheses stated by user are true or not. FDDependant tag identifies an attribute that functionally depends on a set (possibly of one member only) of attributes, which are in turn tagged FDDeterminant (each having its order as a value).
4. Normalize
   While steps 1–3 only added information to a particular RDM, normalizing changes the RDM structure by means of creating new relation schemata and moving attributes from one to another (that is newly created). We use the algorithm of decomposition to produce new schemata. Both checks for actual highest normal form achieved and transformation rules are expressed in OCL.

The next and final flow performs the transformation of RDM into ER model and then further into ODM. Steps are:

1. Create ERM
   Since ERMM contains a very limited set of elements compared to conceptual ERM (or the ultimate ODM),

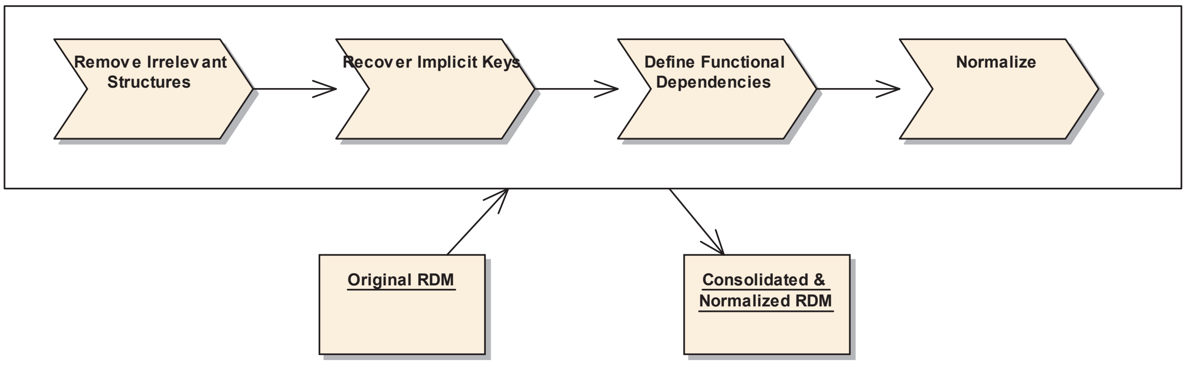
Remove Irrelevant Structures
Recover Implicit Keys
Define Functional Dependencies
Normalize
Original RDM
Consolidated & Normalized RDM

Fig. 7. 1st process flow

simple element-to-element mapping alone would not produce a well-designed ERM (nor ODM). Transformation therefore splits into four sub-steps:

i) Map source elements to equivalent target elements
Relation schemata map to entities and attributes to entity attributes (features of both are preserved). Moreover, all foreign keys need to be transformed into binary association. The association end of the entity, which originated from the relation schema "owning" the foreign key, is assigned cardinality of many, while the opposite end has cardinality of one instead. Finally, the attributes that once appeared in foreign keys, are removed from a target model.

ii) Replace relation schemata that act as many-to-many "links" with associations
Newly created entities that possess no entity attribute are necessarily "remnants" of link relation schema. The reason is that no relation schema would pass the 1NF (first normal form) check if there was no attribute to set a primary key on. For these entities, mapping to associations applies so that any adjacent associations disappear in favor of a new one. Cardinality of both association ends is set to many.
The same principle is applicable to entities with no identifier (an element equivalent to primary key in RDMM) but at least one attribute. The forthcoming association obtains association attributes derived from attributes of an extinct entity.

iii) Detect and create generalizations
RDMs may implement generalizations in three different ways depending on number of successor types and number of instances (tuples) of each successor.
We define an *isomorphic* generalization when all tuples reside a single relation schema which contains a union of all successors' (and predecessor's) attributes. User is to mark attributes with dedicated tag IsoChildName whose value specifies the intended name of a new child entity, and a tag IsoParentName whose value specifies its predecessor. For each distinct IsoChildName, a new entity with corresponding attributes is created. The generalization relationship is also created, having a new entity at its *specific* end, and the existing entity (or even another new one) at its *general* end. No automation is proposed for detecting such a generalization (might be based on data analysis) since too many false generalizations would be suggested.
We define a *structured* generalization when a relation schema, which fits the most general "entity" (not to be confused with an entity element in ERMM), is integrated into inter-relation relationships (foreign keys). Children to such an "entity" occupy the same relation schema, but attributes specific to each child "entity" are stored in separate relation schemata, which are only linked (via foreign keys) to the "master" schema. User shall mark each entity with a tag GenParentName whose value specifies its parent entity. Based on this tag, generalizations are created, replacing existing associations. Proposed automation benefits from the fact that even that the foreign key represents one-to-many relationship, the actual cardinality observable on data is one-to-one (in RDM, foreign key attributes are primary key attributes at once).
We define an *independent* generalization when all distinct entities reside in different relation schemata with no relationship between (in fact, relationships are possible, but none represent generalization). Each relation schema just fits an entity within. Child entities (in an ERM) are tagged IndParentName with a name of its parent in the tag's value. Based on this tag, generalizations are produced, but not at the expense of existing associations. Detection automation makes use of the fact that all relation schemata that take part in the generalization tree have a set of common attributes (i.e. common name and type) that form their primary keys. Moving the tree downwards (to "leaf" schemata), the number of non-key attributes that are common to several schemata increases. Following this principle, the whole tree may be restored.

2. Create ODM
Since class-based object databases (in contrary to those with more or less adjusted metamodel, e.g. Gemstone or various XML-based databases) reuse a mere portion of UML class metamodel, so do we for the target

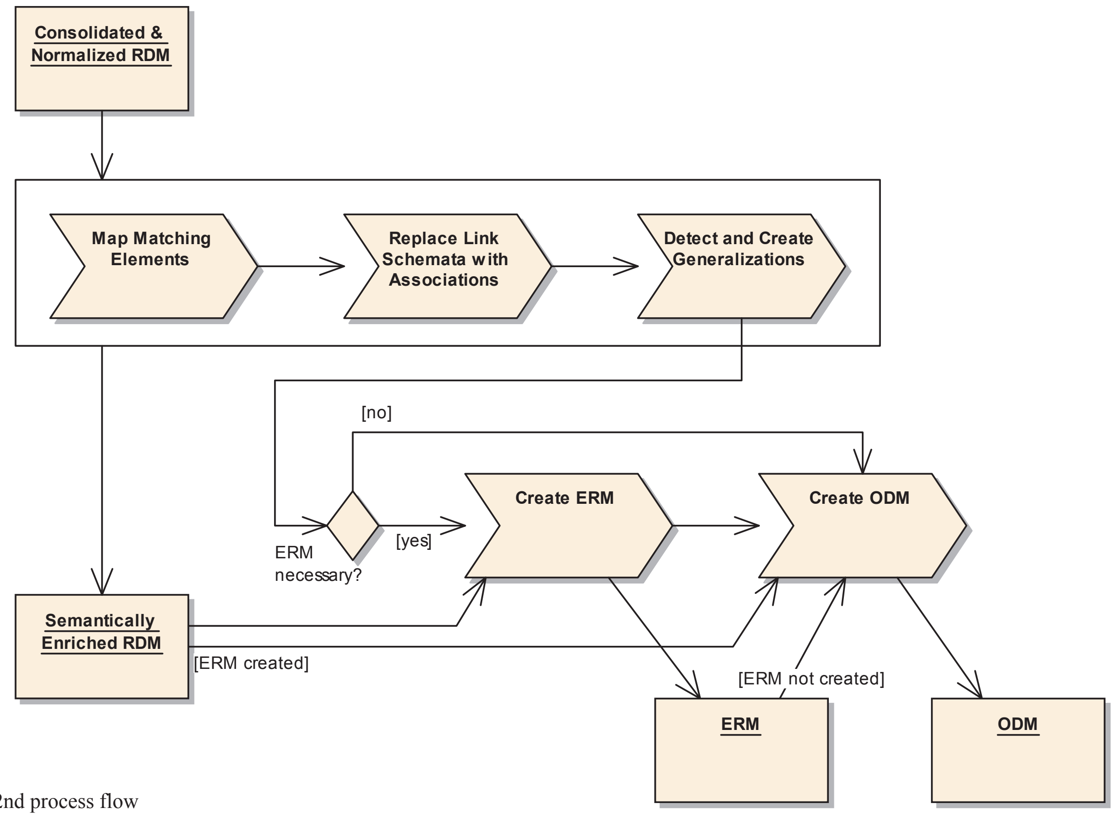
Consolidated & Normalized RDM
Map Matching Elements
Replace Link Schemata with Associations
Detect and Create Generalizations
[no]
Create ERM
Create ODM
[yes]
ERM necessary?
Semantically Enriched RDM
[ERM created]
[ERM not created]
ERM
ODM

Fig. 8. 2nd process flow

ODMM. From the database perspective, the semantic power of ODMM is nearly the same as of the ERMM. Having the ERM finished, we take its elements as inputs into plain mapping to ODM. Considering that the user is expected to further refine the resulting ODM on his own, for his own purposes, we recommend doing alterations to ODM after being generated. The idea is that since ODS persistence is transparent, the user should refine the ODM so that it is usable even for the application-layer data model. Developing methods and defining precise constraint is recommended.

The ERM however seems to become outdated soon, if it is not already. Our experience suggests that the majority of business and system analyses today use a UML Classes package for static conceptual modeling. The benefit is that the analysis model is compliant with the design model (which may be created by refining the former) and also the project team members are not required to master two different paradigms.

If the user chooses to skip the ER model for whatever reason, the straight PIM to PIM (i.e. RDM to ODM) transformation may be carried out applying adjusted transformation rules and leaving out the intermediate CIM model (i.e. ERM).

Example Transformation

Here in a simple example we show a source RDM, describe application of many transformation steps, and present the resulting ODM (no intermediate ERM model will be created). The RDM is a part of a IS used to report document assignment to workers in order to control their performance. The structure is as follows:

Apparently, the DOCUMENTS_LOAD relation is irrelevant from the transformation viewpoint. Thus, user marks it Abandoned so that it is excluded from further transformations.

Secondly, application code is mined for equi-joins to verify that all foreign keys are explicit. There is an unmatched join found: ... FROM PERSON P, DOCUMENT D WHERE P.ALIAS = D.RESPONSIBLE AND P.IS_ACTIVE = 1. A possible foreign key is presented to user for approval, which is granted. Appropriate transformation marks are placed.

Next, user marks groups of attributes that he suspects to be candidate keys. In this case, user marks one group: PERS_DOC.{ALIAS; DATE_STARTED; DATE_FINISHED}. An SQL statement counting records with distinct candidate key members is build to be executed by user. If such a statement evaluated to the total count of records in table, there would be no two distinct records having the group of tested attributes equal, and the hypothesis would pass. However, the expression evaluates to a lower number meaning that the group of attributes is not a candidate key. No transformation marks are placed in this step.

Another user hypothesis testing is involved in search for functional dependencies. User's knowledge suggests that in DOCUMENT relation, TEMPLATE_NAME and ISSUED_BY might be dependent on TYPE. A set of SQL statements is constructed to test partial functional dependencies (i.e. each possible dependant, one by one) by counting records with equal determinant members and distinct dependant. If the statement evaluated to zero for all de-

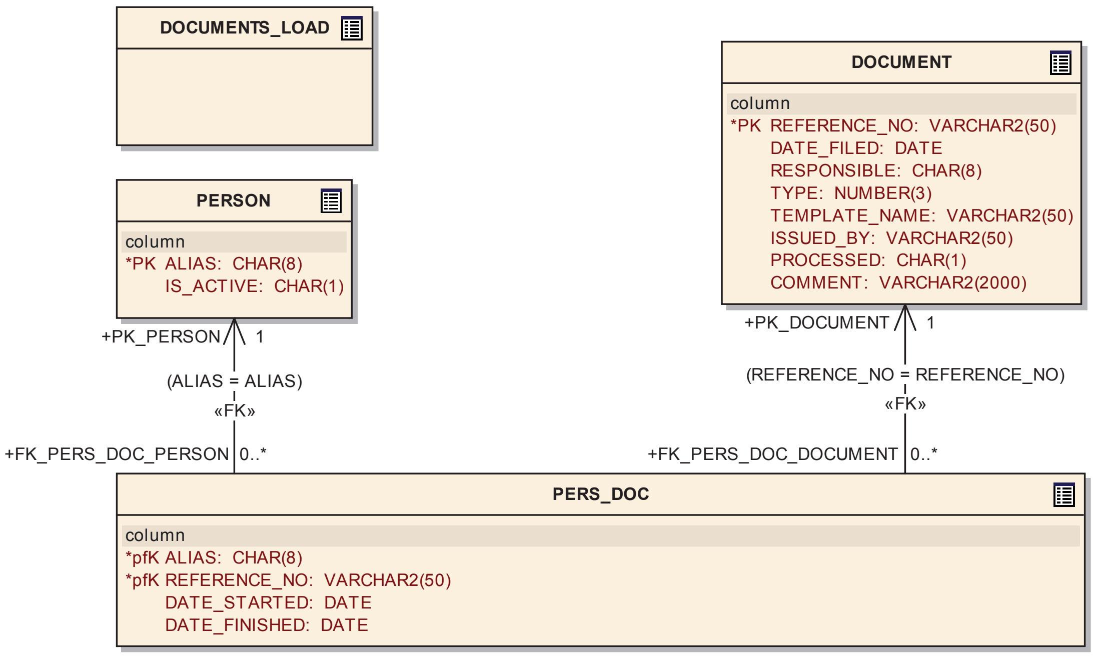

Fig. 9. Source RDM

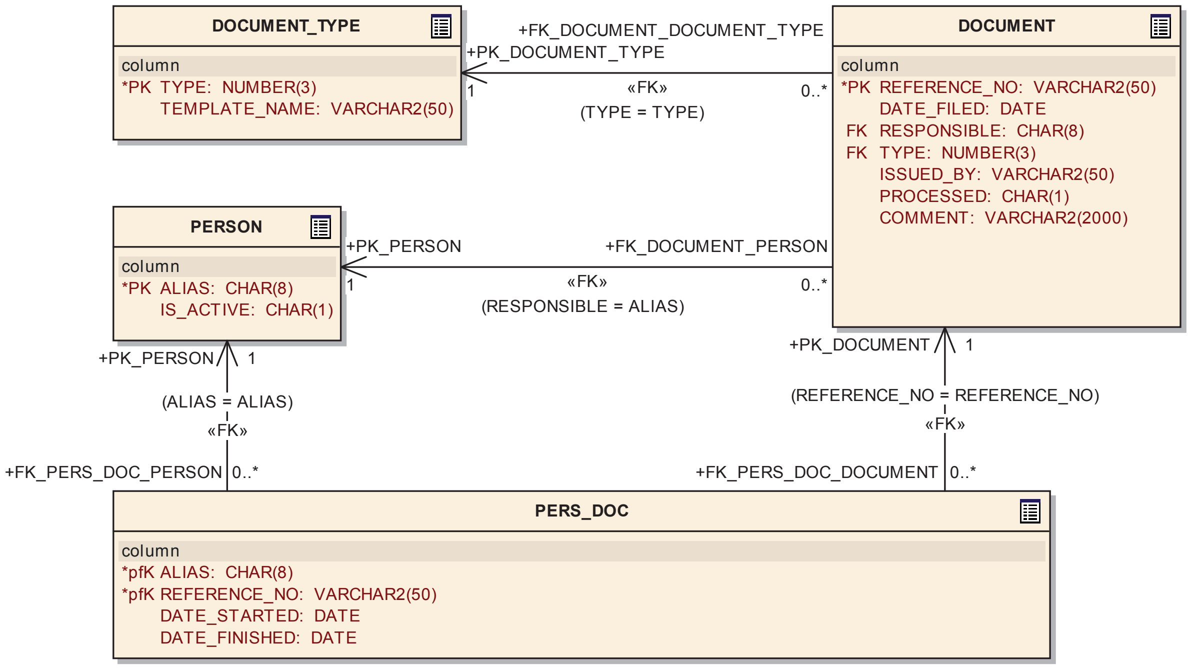

Fig. 10. Updated RDM

pendant members, then the hypothesis would be verified. In this case, only TEMPLATE_NAME was found dependant on TYPE, rendering user hypothesis false as a whole, but right for one dependant member. For this member, transformation marks were placed.

Using added transformation marks, the RDM is transformed into normalized and consolidated one: all relation schemata that are not to be "abandoned" are transported into new model, implicit keys are reconstructed and the schema is normalized up to BCNF. Normalizing produces 1 new schema in this case, as DOCUMENT relation failed the 2NFCompliant() OCL test.

Having the RDM normalized and consolidated, we proceed to ODM derivation. First, target elements are created as described in the previous chapter: relation schemata map to classes (attributes are preserved) and foreign keys map to associations. There is one class in the ODM (PersDoc) that served as a "link" in the RDM (no ex-primary key attributes other than those being part of foreign keys) which must be further transformed into an associa-

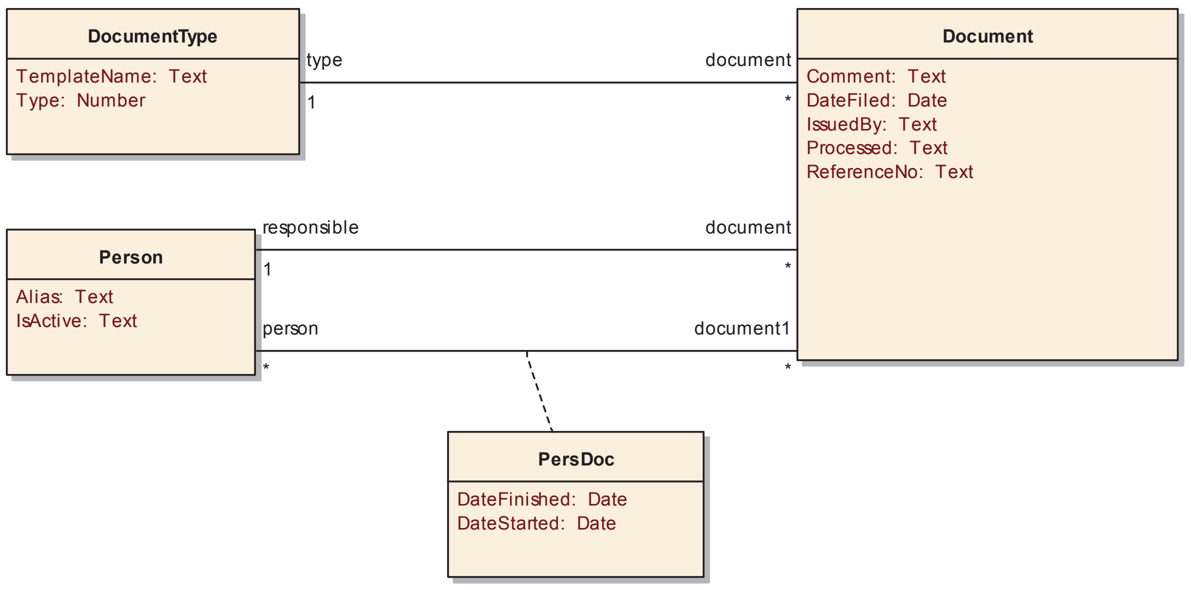

Fig. 11. Early ODM

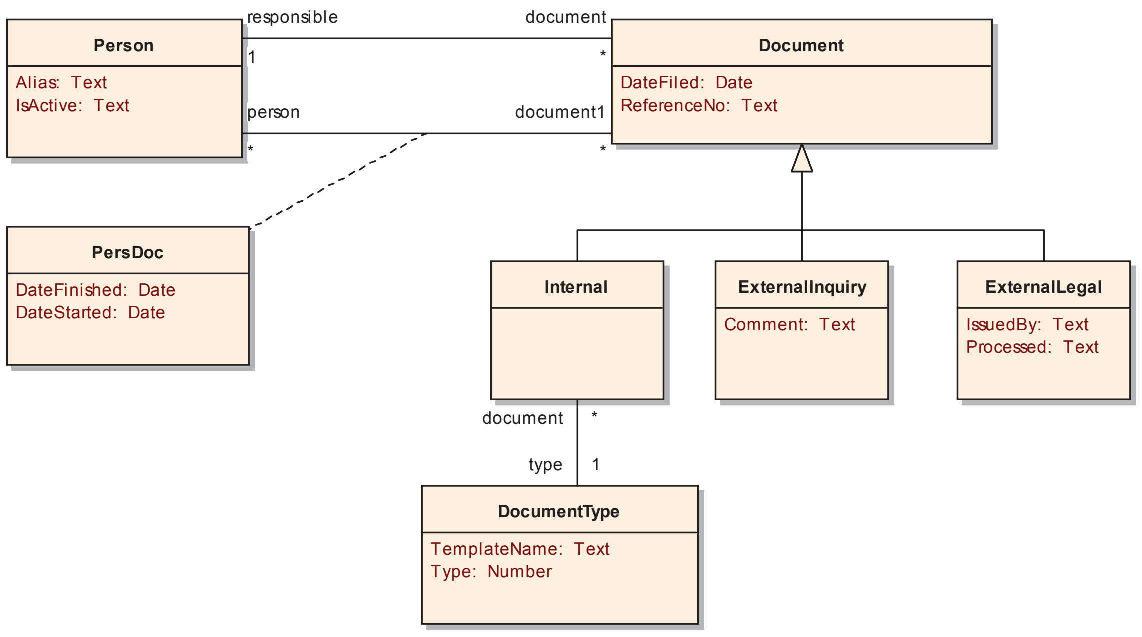

Fig. 12. Target ODM

tion class. This part of transformation is fully automated and as such it does not require user intervention.

The last step requires user to mark generalizations. In this case, user identifies an isomorphic generalization in class Document. He learned that only external inquiries might be commented, also that issuer and "processed" status is tracked in non-inquiring external legal documents only and that type and template are only defined for internal documents. Thus he suggests 3 new classes to become subtypes of a Document class by placing appropriate marks. Then, the automatic transformation is carried out, which produces the final ODM. Further fine-tuning is possible.

During the example transformation process, following enrichment marks have been placed:

RESULTS

Using the proposed method, one is able to obtain an ODM with its semantic power near exploited. The intermediate entity-relation model may, or may not be created. Also, the method is applicable in real-life conditions, with no unrealistic assumptions put on it. User is required to possess only basic knowledge of the universe of discourse.

Model	Element	Mark	Value	Purpose
RDM	Document. Responsible	FKSequence	1	distinguishes newly created FKs
		ReferencedCKOwner	Person	specifies target relation schema
		Referenced-CKSequence	1	specifies target key
		Referenced-CKMemberSequence	1	specifies target key member
	Document load	Abandoned	1	exclude from transformation?
	Document. Type	FDDeterminant	1	determinant number (for referencing)
	Document. Template name	FDDependant	1	specifies determinant
ODM	Document. Comment	IsoParentName	Document	specifies parent
		IsoSiblingName	ExternalInquiry	specifies owner
	Document. IssuedBy	IsoParentName	Document	specifies parent
		IsoSiblingName	ExternalLegal	specifies owner
	Document. Processed	IsoParentName	Document	specifies parent
		IsoSiblingName	ExternalLegal	specifies owner
	Document. Type	IsoParentName	Document	specifies parent
		IsoSiblingName	Internal	specifies owner

REFERENCES

ALHAJJ, R.: Extracting the EER model from a legacy relational database. Information Systems, Volume 28, Issue 6 (September), 2003, pp. 597–618.

ANDERSSON, M.: Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In: Proc. Int. Conf. on the Entity-Relationship Approach (ERA), Manchester, 1994, pp. 403–419.

BLAHA, M. – PREMERLANI, W.: Observed Idiosyncracies of Relational Database designs. In: Proc. 2nd IEEE Working Conf. on Reverse Engineering, Toronto, July 1995, IEEE Computer Society Press.

HAINAUT, J. L.: Database Reverse Engineering. In: Proc. 10th Conf. on ER Approach, San Mateo (CA), 1998.

PETIT, J. M. – KOULOUMDJIAN, J. – BOULICAUT, J. F. – TOUMANI, F.: Using Queries to Improve Database Reverse Engineering. In: Proc. Int. Conf. on the Entity-Relationship Approach (ERA), Manchester, 1994, pp. 369–386.

PREMERLANI, W. – BLAHA, M.: An Approach for Reverse Engineering of Relational Databases. In: Proc. IEEE Working Conf. on Reverse Engineering, 1993, IEEE Computer Society Press.

Received for publication on January 14, 2008
Accepted for publication on March 3, 2008

HOLUB, V. (Česká zemědělská univerzita, Fakulta provozně ekonomická, katedra informačního inženýrství, Praha, Česká republika):

Transformace relačních databázových modelů do modelů objektových.

Scientia Agric. Bohem., *39*, 2008: 67–76.

Tento článek popisuje metodu transformace relačních databázových modelů do objektových pomocí architektury řízené modelem (MDA). Je popsán způsob, jak docílit požadovaných výsledků i za situace, kdy zdrojový relační model je z nějakého důvodu nevhodně navržen. Pro úpravu relačního modelu do stavu vhodného k transformaci jsou navrženy postupy vedoucí k částečné či plné automatizaci. Popsána je transformace používající entitně-relační model jako mezistupeň, nastíněna je i transformace přímá. Podrobně je prezentován způsob sémantického obohacení.

V článku jsou dále diskutovány datové metamodely a jejich specifika pro relační i objektové paradigma, je demonstrováno rozšíření jazyka UML právě o elementy relačního metamodelu. UML je mj. dále rozšířeno o balíček sloužící k modelování funkčních závislostí potřebných k normalizaci. Popsána je forma zápisu transformačních pravidel i potřebného sémantického obohacení, tzn. vytváření uživatelských značek, které tvoří jeden ze vstupů do MDA transformace.

MDA; UML; OCL; transformační pravidla; zápis; entitně-relační model; sémantické obohacení

Contact Address:

Ing. Vít H o l u b , Česká zemědělská univerzita v Praze, Fakulta provozně ekonomická, katedra informačního inženýrství, Kamýcká 1076, 165 21 Praha 6-Suchdol, Česká republika, tel.: +420 224 382 039, e-mail: holub@pef.czu.cz