

A Design Environment for Migrating Relational to Object Oriented Database Systems

Jens Jahnke, Wilhelm Schäfer, and Albert Zündorf

AG-Softwaretechnik, Fachbereich 17, GH-Paderborn,
Warburger Str. 10, D-33098 Paderborn, Germany;
e-mail: [jahnke|wilhelm|zuendorf]@uni-paderborn.de

WWW: http://www.uni-paderborn.de/fachbereich/AG/schaefer/index_dt.html

Abstract

Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing (CIM), or software engineering (SE). However, a competitive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. In addition, complex dependencies between those data stored in the different systems might exist and should be maintained. One approach for seamless integration of object-oriented and relational systems is to migrate the data (and the corresponding schema) from a relational to an object-oriented system. In this paper we describe an integrated design environment that supports the migration process and overcomes major drawbacks of comparable approaches.

1 Motivation

Object-oriented technology has become mature enough to satisfy many new requirements coming from areas like computer-aided design (CAD), computer-integrated manufacturing (CIM), or software engineering (SE). Those new requirements are not fulfilled by relational technology [LS88, Mai89]. However, a competitive information management infrastructure often demands to merge data from CAD-, CIM-, or SE-systems with business data stored in a relational system. In addition, complex dependencies between those data stored in the different systems might exist and should be maintained. One approach for seamless integration of object-oriented and relational systems is to migrate the data (and the corresponding schema) from a relational to an object-oriented system. The main advantage of this approach is that it is always feasible because the

object-oriented data model is a superset of the relational model.

In more detail, such a migration process consists of three steps which are (1) the *schema migration process* which maps a relational schema to an equivalent¹ object oriented schema, (2) the *data migration process* which converts extensions of the relational schema to equivalent² extensions of the object oriented schema and (3) the *application migration process* which creates a new application program using the object oriented database for every application program that uses the legacy database such that the input/output behaviours of the corresponding application programs are identical for equivalent database extensions.

Existing approaches supporting schema migration basically argue for a waterfall-like transformation process, in many cases even without any user interaction [NA87, DA87, SK90, JK90, MM90, And94, FV95]. These approaches suffer from one of the following two drawbacks. Either the structure of the resulting object oriented schema still looks rather "relational" or non-realistic assumptions about the existing relational schema were made. Examples for such non-realistic assumptions are that the relational schema has been developed rigorously using a certain database design method or the relational schema includes no optimization structures. Other examples for non-realistic assumptions can be found in [HEH⁺93]. In any case, the result of such an antisemantic transformation is a bad object-oriented database schema.

Other approaches like [PB94] overcome these shortcomings by proposing a more robust interactive reengineering

-
1. In this paper a relational database schema and an object oriented database schema are considered to be equivalent iff they represent exactly the same universe of facts. A formal description of this notion of schema equivalence is beyond the scope of this paper.
 2. Two database extensions are considered to be equivalent iff they represent exactly the same facts.

process using a set of loosely coupled tools like AWK scripts, simple programs and editor macros.

However, loosely coupled tools do not offer enough support for data migration after the schema transformation has been finished. We argue that the usually complex and thus error prone task of data migration should be carried out automatically. Indeed, our approach in contrast to loosely coupled tools makes this feasible by establishing and maintaining the necessary dependencies between a relational and an object-oriented schema.

The next section will sketch a simple user scenario with our migration environment to illustrate the reasons for and benefits of an interactive schema transformation process. Section 3 explains how the dependencies between the relational and object-oriented schema respectively which define corresponding syntactical constructs in the two schemes are formally specified, established and maintained during editing of an object-oriented schema. Our approach even supports to edit an object-oriented schema in such a way that established dependencies get lost (for a while) and are re-established later on again (semi-)automatically. Section 4 gives an overview about the current status of the migration environment and sketches some further plans for its utilization.

2 The Migration Environment: An Example Scenario

Figure 1 shows an example screen dump of the migration environment. The left window shows an editor for the relational schema supporting SQL (with additional semantic annotations) while the right window shows an editor for the object-oriented schema according to the ODMG-93 standard³ [Cat93].

In migrating a database system the first step is to analyse the legacy relational system. For this purpose, the *schema extraction* and *analysis tool* offers commands within the "Extract" submenu of the SQL-schema window that inspect existing relational database systems and extract as much schema information as possible in order to build up an (initial) SQL-schema.

The schema extraction tool first will query the relational database system for its schema. This will reveal at least the existing tables together with their attributes. In more sophisticated database management systems we may gain information on primary keys, foreign keys, referential integrity constraints, and additional dependencies. By inspecting the code of database system applications, stored procedures, and event handling procedures and by inspecting the data

itself we may get even more information about e.g. the cardinality of relationships and higher level abstraction mechanisms like aggregation and inheritance structures.

A detailed discription on how these informations can be extracted is beyond the scope of this paper and can be found in other contributions [And94, FV95, PB94, PKBT94, SLGC94]. As stated in the introduction, a fully automatic schema extraction normally will fail to detect all relevant information and to recover the higher level design concepts of a schema. Thus, the user, e.g. a database system administrator or an application programmer, may add his specific design and application knowledge to the relational schema initially constructed by the schema extraction tool.

Let us assume that in Figure 1 the automatic schema extraction tool was not able to identify the `pc_chair` attribute of table `Conf_Info_Addr` as foreign-key attribute referencing the `name` attribute of table `Person` and/or the uniqueness of this attribute. The user could easily add this information within the SQL-schema window. Another example is the "HINT:" comment that table `Conf_Info_Addr` is a kind of subrelation of table `Conference` which suggests an aggregation relationship between these two tables.

Since the user is allowed to annotate the SQL schema manually, he/she may erroneously introduce inconsistencies with the given schema of the described relational database system. In order to gain maximal consistency, the schema extraction tool offers additional commands that are used to compare the edited schema with the schema of the legacy system (cf. command "Compare with database" in the popup menu in Figure 1) and e.g. to validate the correctness of cardinality constraints by inspecting the actual data of the legacy system (cf. command "Validate Constraints").

Such schema enhancements should be performed by a database system administrator or application programmer who is familiar with the legacy relational system in order to exploit their application knowledge within this step. In order to facilitate this task for these peoples it is important that they can do this work within the SQL notation they use in their every day work. So these people not necessarily have to become experts in the ODMG data definition language and object oriented design concepts first.

A further step in schema migration is the transformation of the SQL schema reached so far into an equivalent ODMG schema. This transformation must exploit all semantic information contained in the annotated SQL schema and has to employ the additional expressive power of the object oriented data model as far as possible. Due to the semantic gap between the two data models this schema transformation can not be done fully-automatically but again user interaction is necessary in order to yield a high quality resulting ODMG schema.

The schema transformation or mapping tool of the proposed migration environment is based on an adaptable set of

3. Due to layout reasons we use a simplified syntax for the definition of the object-oriented database schema.

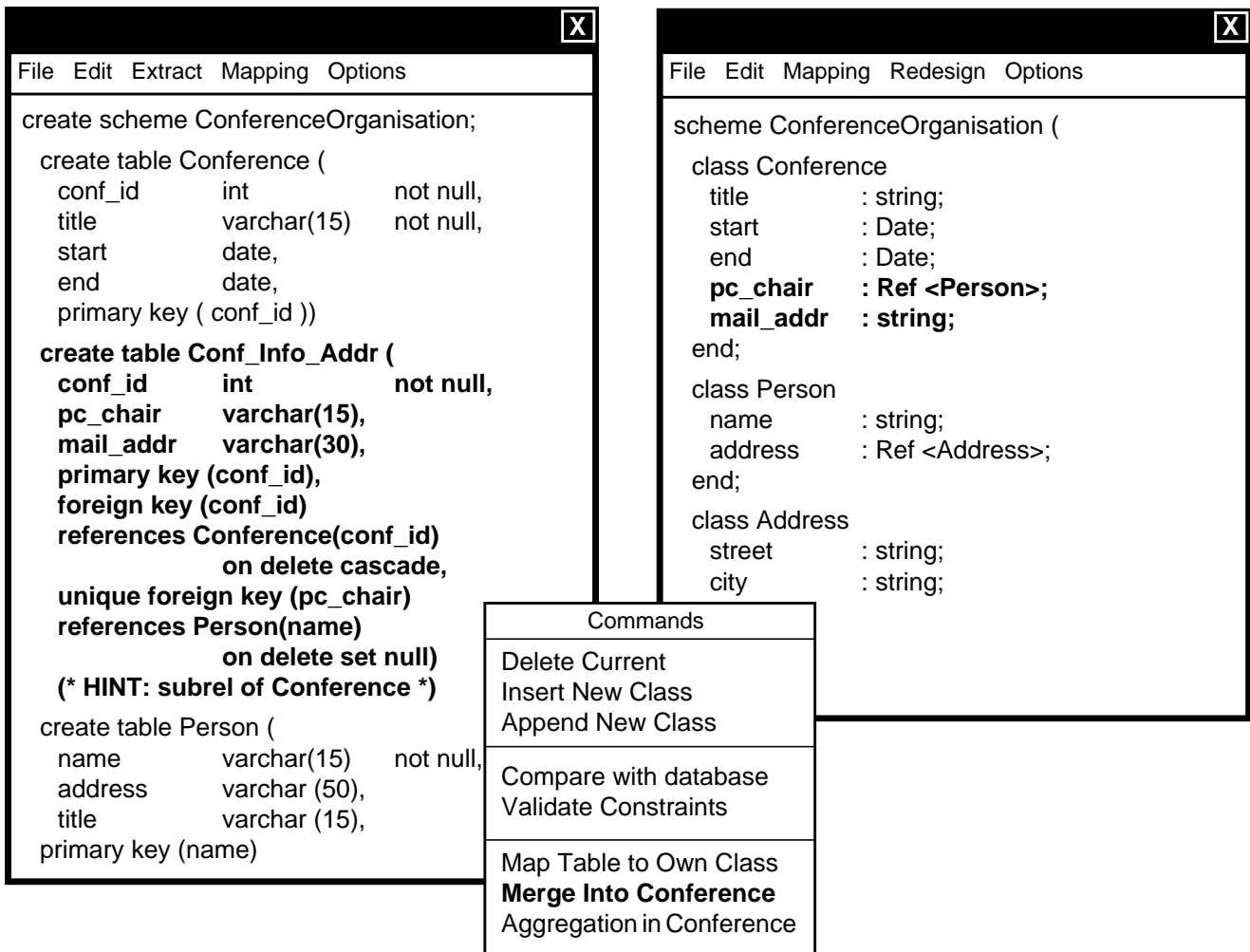


Figure 1: User Interface of the SQL/ODMG Editor/Mapper

schema mapping rules. Each mapping rule defines how a specific construct or situation in a SQL schema is mapped to an equivalent construct or situation within the ODMG schema. However, in general there are several alternatives to define the schema mapping. For example, within Figure 1 the currently selected table `Conf_Info_Addr` has been mapped to a group of attributes of class `Conference` using the mapping rule "Merge into class of father relation". This was possible since the additional schema information indicates that for every `Conference` tuple/object there exists exactly one `Conf_Info_Addr` tuple. Thus, the attributes of `Conf_Info_Addr` can be incorporated into class `Conference` without losing any information. Alternatively, the table `Conf_Info_Addr` could have been mapped to a complex (aggregation/tuple) attribute of class `Conference` using the mapping rule "Aggregation in class of father relation". Third, the table `Conf_Info_Addr` could have been mapped to its own class `Conf_Info_Addr` in combination with a refer-

ence Attribute within class `Conference` using the rule "Map Table to Own Class".

Within our migration environment first an initial mapping from the SQL schema to the ODMG schema is computed based on (user defined) priorities for alternative mapping rules. Since these general priorities will not fit for all actual situations, the user may inspect the resulting mappings and interactively select more appropriate mapping alternatives, cf. Figure 1. To provide maximum flexibility the rule set of our migration environment is designed to be easily extended, enhanced or exchanged. This allows to improve the mapping tool later on and to deal with SQL schemata belonging to different "design styles/schools".

The mapping process described so far is based on the information contained in the SQL schema definition. But, due to the simplicity of the relational data model it contains no or only little information about the higher level design concepts like aggregation and inheritance structures. Thus

the ODMG schema derived so far can still be enhanced by employing these higher level concepts. Therefore, the migration environment offers a set of equivalence preserving design transformations within its "Redesign" menu. Examples for such redesign operations are

- renaming of any element of the ODMG schema, or
- the combination of a group of attributes of a class to a new complex aggregation attribute, or
- the replacement of a group of attributes by a reference to a newly introduced class containing these attributes, or
- the extraction of a group of common attributes from several classes and their reorganisation as a new common super class.

All these design transformations will not violate the dependency between the SQL schema and the ODMG schema, respectively.

In addition the database designer has the possibility to edit (any parts of) the ODMG schema freely. He may add new attributes and/or classes and drop existing schema parts. Thereby, the designer is allowed to define parts of (or even the whole) ODMG schema anew according to different design rationals or to new system requirements. Clearly, during free editing dependencies with the SQL schema is lost. Within the migration environment such unmapped schema elements will be marked using a special colour in both windows, the SQL schema window and the ODMG schema window. Using the mapping rules dependencies are reestablished automatically or semi-automatically, i.e. in many cases no user interaction is necessary to reestablish the correspondence between the two schemas, in some cases the environment may need advice to identify corresponding syntactical constructs, and in some cases the user may have to confirm explicitly the insertion or deletion of syntactical constructs in the ODMG-schema.

During the enhancement of the ODMG schema we occasionally will face the situation that the mapping engineer or the ODMG schema designer will find parts of the schema where some semantic information is missing in the SQL schema. At this point further investigation within the relational database system may reveal additional informations that now can be added to the SQL schema description. This additional schema information may enable new mapping alternatives (perhaps of higher priority) for this schema parts or may even falsificate old mappings. In this situation again the dependencies of the two schemata are reestablished semi-automatically using the mapping tool. It is even possible to start with a part of the schema of the legacy relational system and to map this part first and to extend this partial schema migration by the remaining parts later on.

3 Implementation Concepts

Within our migration environment we employ structure oriented editors for the representation and manipulation of the SQL and the ODMG schema. These structure oriented editors internally store an abstract syntax tree representation of the edited schemas.

Figure 2 shows a simplified cutout of the internal data structures of our migration environment for the example szenario shown in Figure 1. The left side of Figure 2 shows the representation of the SQL schema and the right side shows the representation of the ODMG schema. The root of the SQL schema is represented by a node/an object of type `SQLSchema` carrying the name "Conf.Org."⁴ of the example schema as (one of its) attribute(s). The SQL schema node has three outgoing edges/references of type `c` leading to the contained table definitions. A table definition is represented by a node of type `table` with outgoing `c` edges leading to the contained attribute definitions.

The editors offer structure oriented editing commands that directly manipulate the underlying abstract syntax tree.⁵ Additional semantic information like identifier bindings, type information and results of the schema extraction tool is represented by additional attributes and edges. For example, the edge labeled `foreign_key` leading from attr node "conf_id" of table "C.Inf.Adr"⁶ to the corresponding attr node of table "Confer." represents a foreign key condition. Note also the `subrel` edge relating table "Confer." and table "C.Inf.Adr", cf. hint in Figure 1. The structure oriented editors of our migration environments are easily generated from a specification of the syntactic structure of the corresponding schema language (cf. [Emm95]).

Besides supporting structure-oriented editing this graph-based approach has an additional advantage in the context of database reengineering. The graph-structure is extended by additional nodes and edges which establish the dependencies between the relational and the object oriented schema respectively. The formal definition of these dependencies as well as the preservation of these dependencies during editing the ODMG or the SQL definition is given by a triple graph grammar [Lef94, Lef95, Sch94] which we explain now.

The top of Figure 3 shows as an example the tripple graph grammar rule `MapTableToClass`. Basically, a tripple graph grammar rule, in the following called a mapping rule,

4. Abbreviated due to layout reasons. Stands for "ConferenceOrganisation".

5. The editors are hybrid structure oriented/textual editors. The user may edit any syntactic part of the schema in normal text oriented mode. Then a multiple entry parser analysis the edited text and (re)builds the corresponding abstract syntax subtree.

6. Abbreviation for "Conf_Info_Adr".

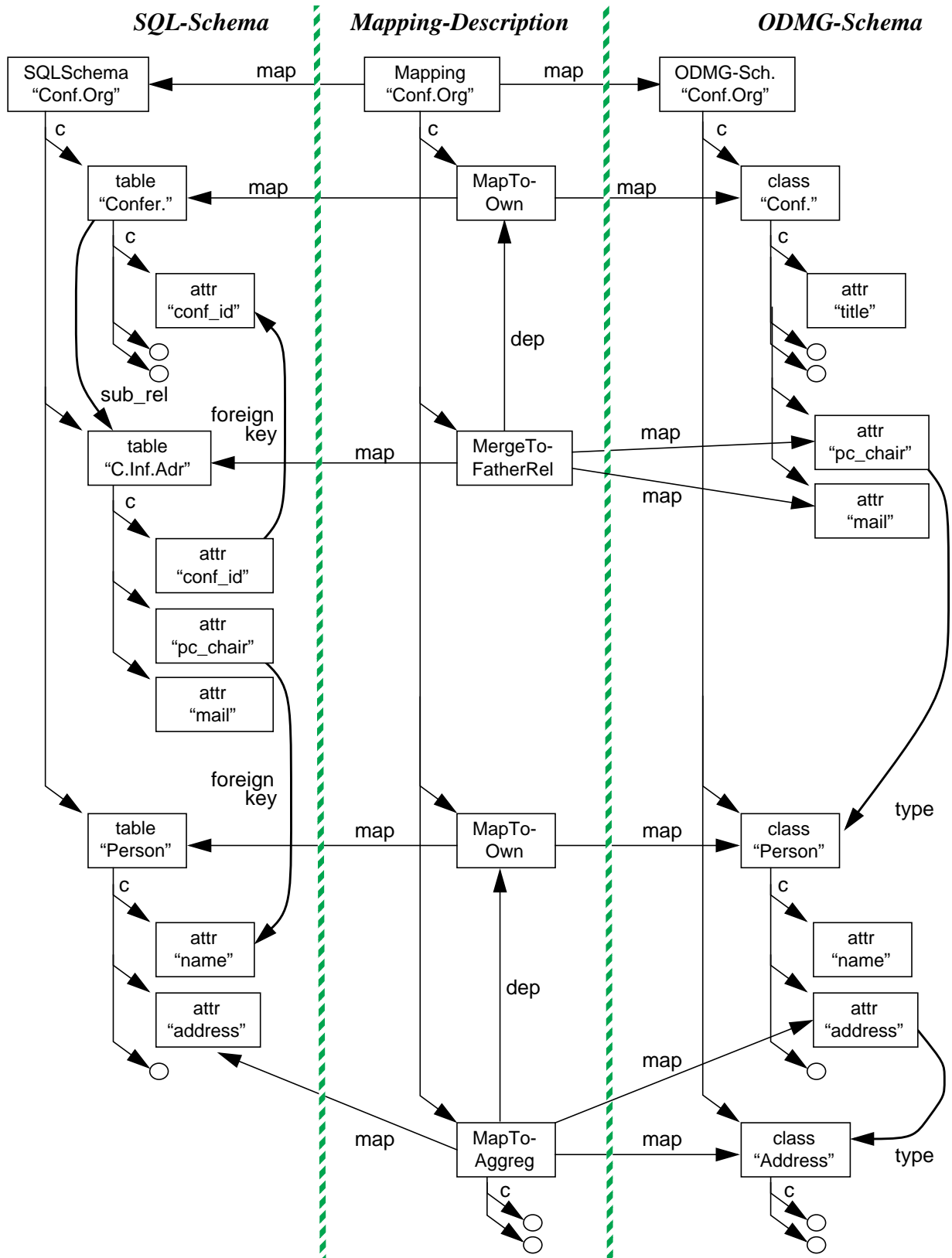


Figure 2: Cut-Out of the Internal Representaion of Documents

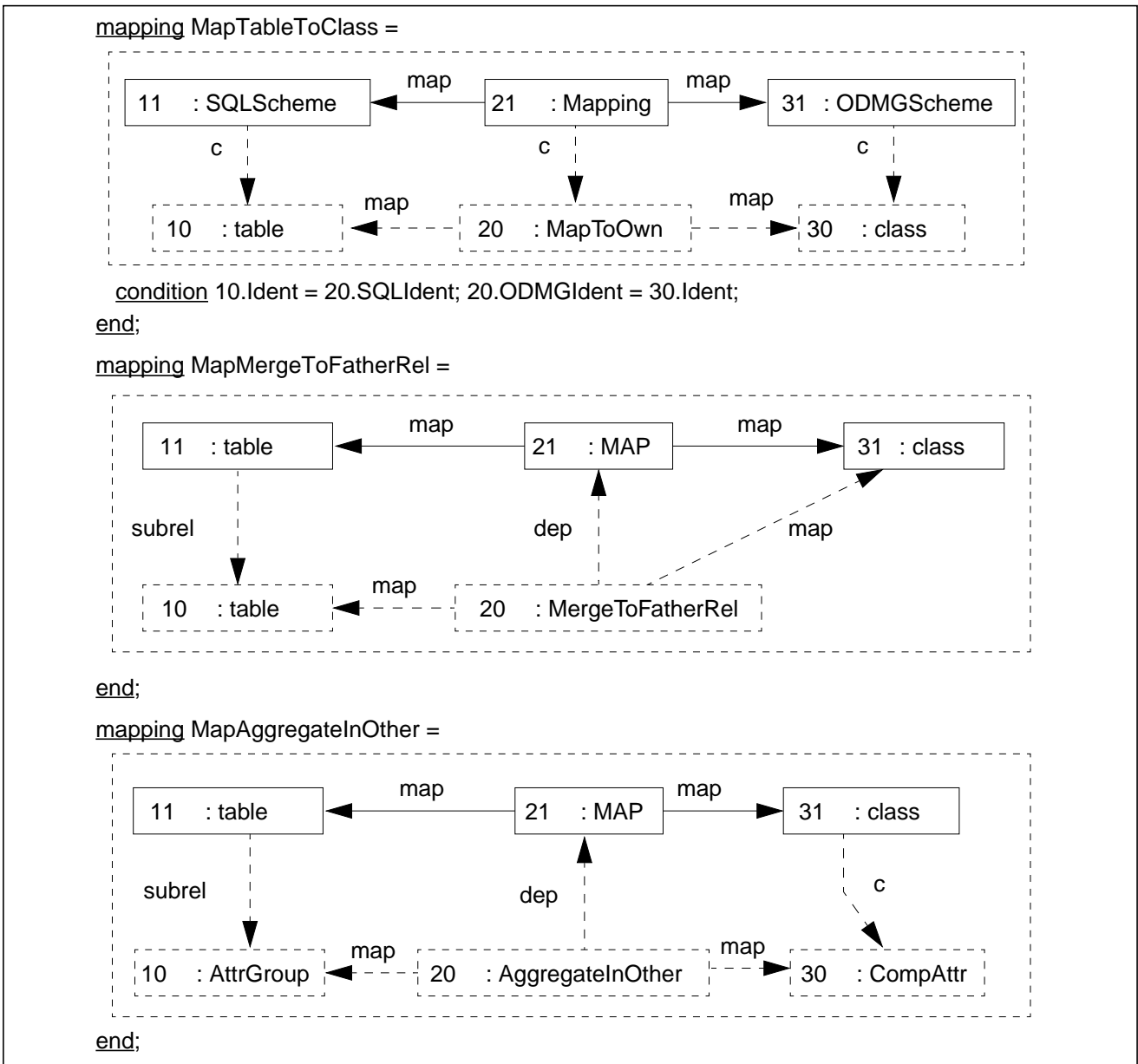


Figure 3: Trippel Graph Grammar Rules for tables

is a specially notated graph rewriting rule (see [HE95] for an overview on graph grammars). Within a mapping rule the solid parts (like the nodes 11, 21, and 31 and the connecting map edges) describe the so called *left hand side*. The dashed parts (like node 10, 20, and 30 and the attached map edges) describe the so called *addition part*. A mapping rule is executed by searching for an isomorphic image of its left hand side and extending the found image by an isomorphic image of its addition part.

The example mapping rule MapTableToClass searches for a node 11 of type SQLScheme (the root of the SQL schema) and a node 31 of type ODMG Schema (the root of

the ODMG schema) that are connected to a node 21 of type Mapping (the indicator of a correspondency between the two schemata) by two map edges. The rule adds a new table 10 to the SQL schema and simultaneously a new class 30 to the ODMG schema and a mapping indicator 20 of type MapToOwn to the current data structure.

Figure 3 shows two additional rules for SQL tables. Rule MapMergeToFatherRel describes that a table 10 that is known to be a subrel of another table 11 that already has been mapped⁷ to class 31 can be mapped to the same class 31 as its father relation. Rule MapAggregateInOther describes that in the same situation table 10 also can be

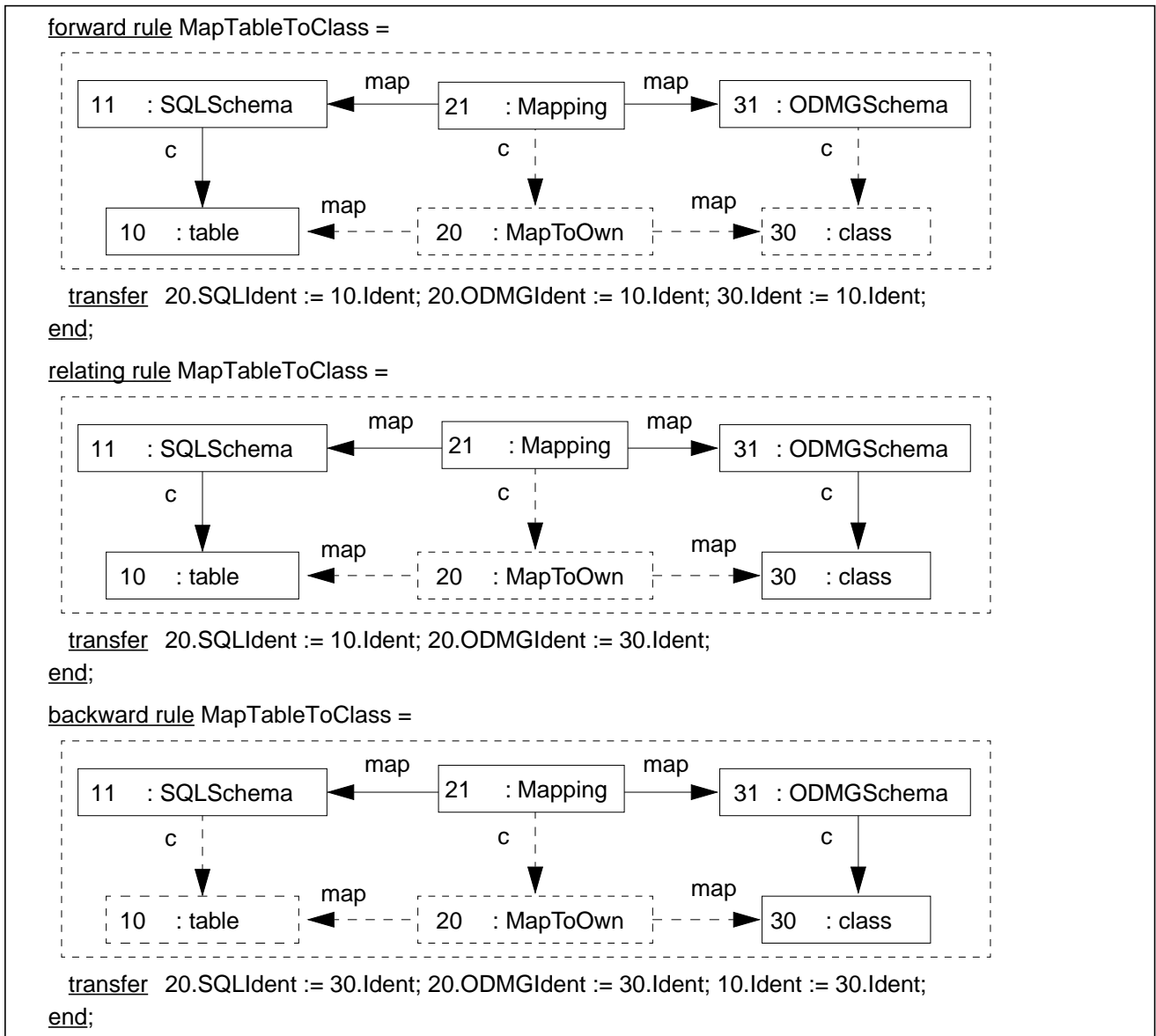


Figure 4: Graph Grammar Rules derived from MapTableToClass

mapped to a complex attribute (of record or tuple type) of the class 31 that corresponds to its father relation/table 11. Additional mapping rules are used to define the mapping of attributes and attribute types.

Formally, a set of such mapping rules (together with an appropriate start graph) builds a (triple graph) grammar with the set of all pairs of SQL and ODMG schemata as its language that we consider to be equivalent or to depend on each other.⁸

7. The mapping indicator 21 of super class MAP can be matched to mapping indicators of any of its subclasses, like MapTableToClass, MapMergeToFatherRel, or MapAggregateInOther in our example.

8. The prove that exactly the set of all equivalent schema pairs is defined by a set of mapping rules is still a topic of current research.

From the mapping engineers point of view, each mapping rule defines for a given element/substructure of a SQL schema one possible translation into an element/a substructure of an ODMG schema that he/she considers to be equivalent.

The mapping tool utilizes this equivalence definition by deriving three different operations/graph rewriting rules from every mapping rule, a forward transformation rule, a relating rule and a backward transformation rule.

Figure 4 shows the forward transformation, relating, and backward transformation rules derived from the mapping rule MapTableToClass shown in Figure 3.

A forward transformation rule is derived from the corresponding mapping rule by changing the dashed SQL parts

of the mapping rule to become solid parts. For rule MapTableToClass these parts are node 10 and the attached c edge coming from node 11. The resulting forward transformation rule searches for a table⁹ in the SQLSchema and creates an appropriate class within the corresponding ODMG-Schema (and a mapping indicator MapToOwn describing the correspondency of the translated elements). The relating rule is derived from the mapping rule by changing the dashed SQL and the dashed ODMG parts to become solid. The resulting rule is used to reestablish correspondency information that was lost due to editing of the ODMG schema or that had become invalid due to new schema information added to the SQL schema, e.g. by the schema extraction tool. The backward transformation rule is derived by making the ODMG parts solid. Such rules can be used to translate an object oriented conceptual database schema into a logical SQL database schema.

4 Current and Future Work

A prototype of the migration environment is available and demonstrates the feasibility of the approach. This prototype is now developed further by a one-year project-including 15 graduate students and a number of MSc-theses.

In cooperation with local industry and the Paderborn C-lab (former CADLAB) the resulting environment shall become part of a design environment for federated data base systems. Through its OpenDM-system (Open Data Base Middleware) C-lab provides the possibility to execute queries to a federated system whose schema is defined as an ODMG schema, in such a way that the appropriate local data base(s) of the federation are queried. The necessary information to translate the result of such a query back into the ODMG-model, if the local system is a relational one, is provided by the migration environment as sketched in this paper.

Based on our approach to schema migration our current research focuses on the remaining two migration phases: data migration and application migration.

References

- [And94] M. Andersson. Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In *Proc. of the 13th Int. Conference of the Entity Relationship Approach, Manchester*, pages 403–419. Springer, 1994.
- [Cat93] R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufman, 1993.

- [CEER96] J. Cuny, H. Ehrig, G. Engels, and G. Rozenberg (Eds.). *Proc. 5th Int. Workshop on Graph Grammars and Their Application to Computer Science*. LNCS 1073. Springer-Verlag, 1996.
- [DA87] K. H. Davis and A. K. Arora. Converting a Relational Database Model into an Entity-Relationship Model. In *Proc. of the 6th Int. Conference of the Entity Relationship Approach, New York*, pages 271–285. North-Holland, November 1987.
- [Emm95] W. Emmerich. *Tool Construction for Process-Centered Software Development Environments based on Object Database Systems*. PhD thesis, University of Paderborn, 1995.
- [FV95] C. Fahrner and G. Vossen. Transforming Relational Database Schemas into Object-Oriented Schemas according to ODMG-93. In *Proc. of the 4th Int. Conf. of on Deductive and Object-Oriented Databases 1995*, 1995.
- [HEH⁺93] J-L. Hainaut, V. Englebert, J. Henrard, J-M. Hick, and D. Roland. Requirements for information system reverse engineering support. Technical Report RP-95-13, University of Namur, Belgium, 1993.
- [JK90] P. Johannesson and K. Kalman. A method for translating relational schemas into conceptual schemas. In F. H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*. ERI, 1990.
- [Lef94] Martin Lefering. Development of incremental integration tools using formal specifications. Technical report, RWTH Aachen, 1994.
- [Lef95] Martin Lefering. *Integrationswerkzeuge in einer Softwareentwicklungsumgebung*. Informatik. Verlag Shaker, 1995.
- [LS88] C. Lewerentz and A. Schürr. GRAS, a management system for graph-like documents. In *Proc. of the 3rd Int. Conf. on Data and Knowledge Bases*. Morgan Kaufmann, 1988.
- [Mai89] D. Maier. Making database systems fast enough for CAD applications. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 573–582. Addison-Wesley, 1989.
- [MM90] V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering*, 16(8):777–790, August 1990.
- [NA87] S. B. Navathe and A. M. Awong. Abstracting Relational and Hierarchical Data with a Semantic Data Model. In *Proc. of the 6th Int. Conference of the Entity Relationship Approach, New York*, pages 305–333. North-Holland, November 1987.
- [PB94] W. J. Premerlani and M. R. Blaha. An approach for reverse engineering of relational databases. *Communications of the ACM*, 37(5):42–49, May 1994.
- [PKBT94] J-M. Petit, J. Kouloumdjian, J-F. Boulicaut, and F. Toumani. Using queries to improve database reverse engineering. In *Proc. of 13th Int. Conference of ERA, Manchester*, pages 369–386. Springer, 1994.
- [Sch94] Andreas Schürr. Specification of graph translators with triple graph grammars. Technical report, RWTH Aachen, 1994.
- [SK90] F. N. Springsteel and C. Kou. Reverse Data Engineering of E-R Designed Relational Schemas. In *Proc. of Databases, Parallel Architectures and their Applications*, pages 438–440. Springer, March 1990.
- [SLGC94] O. Signore, M. Loffredo, M. Gregori, and M. Cima. Reconstruction of er schema from database applications: a cognitive approach. In *Proc. of 13th Int. Conference of ERA, Manchester*, pages 387–402. Springer, 1994.

9. We employ additional conditions to ensure that the regarded table is not yet transformed.