Algebraic Database Migration to Object Technology

Andreas Behm, Andreas Geppert, Klaus R. Dittrich

Database Technology Research Group
Department of Information Technology, University of Zurich
Winterthurerstr. 190, CH-8057 Zurich, Switzerland
[abehm|geppert|dittrich]@ifi.unizh.ch

Abstract

Relational database systems represent the current standard technology for implementing database applications. Now that the object-oriented paradigm becomes more and more mature in all phases of the software engineering process, object-oriented DBMS are seriously considered for the seamless integration of object-oriented applications and data persistence. When reengineering existing applications or constructing new ones on top of relational databases, a large semantic gap between the new object model and the legacy database's model must still be bridged. We propose database migration to resolve this mismatch: the relational schema is transformed into an object-oriented one and the relational data is migrated to an object-oriented database. Existing approaches for migration do not exploit the full potential of the object-oriented paradigm so that the resulting object-oriented schema still "looks rather relational" and retains the drawbacks and weaknesses of the relational schema. We propose a redesign environment which allows to transform relational schemas into adequate object-oriented ones. Schemas and transformation rules are expressed in terms of a new data model, called semi object types (SOT). We also propose a formal foundation for SOT and transformation rules. This formalization makes it possible to automatically generate the input of the data migration process.

1 Introduction

The presence of new technologies like the World Wide Web, E-commerce or data warehousing is a key argument for many companies to reengineer legacy information systems [34]. Now that the object-oriented method is prevailing in modern software development, almost all components of new information systems are developed within an object-oriented software engineering life cycle. The notable exception is in many cases the (relational or even hierarchical and CODASYL-) database component. Many organizations still principally refrain from using object-oriented database management systems (OODBMS), for reasons which are beyond the the scope of this paper. Other organizations are willing to give OODBMS a try, but then require the existing relational data to be available in the object-oriented database system (OODBS). This requirement raises the research problem of how to convert schemas and databases stored in an existing (relational) database system into those of an OODBS. To that end, mainly hybrid approaches such as object-oriented views over relational schemas have been proposed, but these do not resolve the data model mismatch, and the required conversions at runtime lead to performance degradations. We therefore propose

database migration, i.e. the transformation of a relational schema into an object-oriented one and the subsequent migration of the data to the object-oriented DBMS.

Several database vendors now offer so-called connectivity tools to relational databases, some of them providing also load facilities. However, these and additional approaches proposed in research [1, 19, 8, 36] are not flexible enough, especially in supporting extensive schema restructuring operations. They primarily map each relation to a class and replace inclusion dependencies by references or inheritance relationships. All approaches have in common that they provide a *one-step mapping*, i.e., every element of the target object schema is directly derived from an element of the relational source schema. These approaches are adequate for small and well-designed schemas which, e.g., are derived from an entity-relationship schema and/or are in third Normal Form (3NF). However, connectivity tools and other proposals exhibit two major inadequacies:

- **Weaknesses and drawbacks of relational schemas:** Relational legacy databases typically contain "obscure" optimizations or are denormalized in order to avoid expensive join operations in queries. For example, multiple tables conceptually related by aggregation or inheritance relationships are often collapsed into a single table. A direct mapping of such structures into object-oriented ones preserves inadequacies; consequently, schema transformation must be flexible (e.g., not in all cases semantics-preserving), as overcoming such drawbacks enhances the semantic expressiveness of the schema.
- **Different design strategies:** Relational and object-oriented database design are principally different and follow different design strategies. Besides additional semantic features in object-oriented schemas, such as aggregation or inheritance, object-oriented models comprise the concepts of methods and encapsulation, objects as an abstraction level, but usually do not provide a view mechanism. As a consequence, the straightforward transformation of a well-designed relational schema does not necessarily result in a well-designed object schema. These differences are extensively discussed in [10, 26, 31]. Again, flexible transformations need to be supported in order to obtain a well-designed object-oriented schema.

Approaches supporting both, flexible schema transformation and an automatically generated data migration process, do not yet exist. The main contributions of our approach therefore consist of:

- **Flexible schema transformation:** This allows transforming the relational schema into (any) "adequate" object-oriented schema as obtained by forward engineering, rigorously using an object-oriented design method like OOD [11]. We therefore introduce a *redesign environment*.
- **Automatically generated data mapping:** The *formal foundation* of our approach is given by an algebra. This makes it possible to successively define and rewrite data mappings during schema redesign. In

consequence, the data migration process can be generated automatically as soon as the schema transformation process is completed.

Recently, object-relational DBMS (ORDBMS) have started to offer some object-oriented features defined in the SQL3 standard such as inheritance, user-defined types, etc. Other features are likely to be addressed in SQL4 (e.g., collection-valued attributes). ORDBMS will have the same problem as discussed here for OODBMS, namely to convert an existing relational schema into one exploiting object-oriented features, and to adapt existing databases. Thus, the results presented here for OODBS will also be important for (future) ORDBS.

The remainder of this paper is structured as follows. An overview of all migration activities is presented in section 2. The basis of our approach is the *redesign environment* which is illustrated in section 3. We introduce a new data model, called semi object types (SOT), and an algebra, in which the activity of schema transformation is embedded. The migration process, embedded in this redesign environment, is presented in section 4. The essential element of the migration process is the concept of *transformation rules* which supports both, schema transformation and data mapping. The application of transformation rules is illustrated by an example in section 5. Section 6 contains concluding remarks and open issues.

2 The Migration Framework

The framework of our view of the migration process is illustrated in Fig. 1. The part of the approach presented in this paper resides in the grey box in the lower part of the figure.

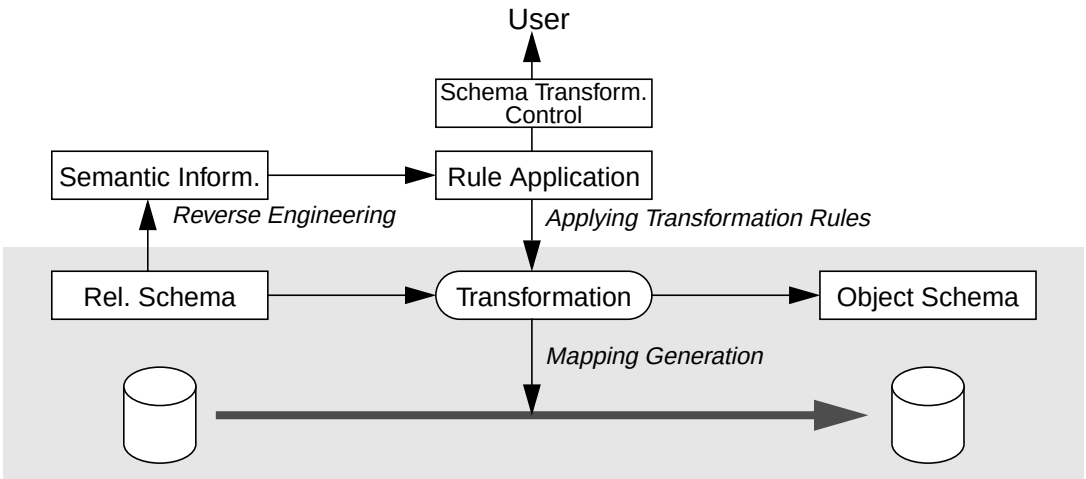


Figure 1: The Migration Framework

The first step of database migration is a reverse engineering process of the relational schema. This step is necessary since no specific assumptions are made about the source schema (e.g., it might not be in 3NF). Reverse engineering has found considerable attention in research in this decade, not only in the context of

database migration but also for redocumentation, restructuring, maintenance or extension of existing databases and applications [2, 14, 17, 12, 21, 22, 25, 30]. Its objective is to extract semantic information like primary keys or foreign keys from the database, which might not be explicitly available, e.g., as part of the DDL code (note that older versions of relational DBMS did, e.g., not support foreign key constraints). Data models capturing semantic information in the context of database migration have been proposed in [18, 36]. Semantic information can also be gathered by analysing queries from application programs [24, 29] or data. The activity of adding semantic information to an existing schema is called *semantic enrichment*.

Even though various approaches for semantic enrichment exist, the extracted information is not sufficient for an *automatic* schema transformation process (recall the limitations of one-step, automatic mappings discussed above). Hence, similar to other recently devised approaches, we propose an *interactive* schema transformation process. The user is thereby supported by (i) semantic information and (ii) an additional layer, called *schema transformation control*, which supports the user through schema transformation strategies. The latter can be implemented by extracting common transformation patterns, as will be shown in subsequent examples. A detailed presentation of this layer is outside the scope of this paper.

The result of the schema transformation process is (i) an object-oriented schema and (ii) an algebraic data mapping definition which enables an automatic migration of the database.

3 Redesign Environment

The redesign environment contains an intermediary model for the transformation of a relational schema into an object-oriented one. It consists of two major parts: a data model in which schema transformations are expressed, and an algebra which supports to formulate data mapping expressions.

The central modelling construct of the data model are *semi object types* (SOT), which are comparable to relations or classes. The purpose of the SOT model is to express all (static) properties of object-oriented schemas in a way that makes restructuring operations as easy as possible.

Neither the relational nor the object-oriented data model fulfils the requirements for schema transformation and data migration. The relational data model lacks (unique) identifiers and the support of complex structures. Identity is simulated through key attributes. Since key attributes may also represent contents of the universe of discourse, transformation rules relaxing the uniqueness requirement of these attributes cause problems. As regards complex structures, aggregates and sets may be simulated through additional relations, however list or array structures cannot be expressed directly.

On the other hand, the object-oriented model is too restrictive with respect to object identity and inheritance. Moreover, schema restructuring cannot easily be propagated to the data level. The interface of an object or its class membership (usually) cannot be changed once it is created. No influence can be taken on object identifiers.

An SOT schema consists of a set of SOTs $S = \{s_1, \ldots, s_n\}$. Every SOT s_i consists of a set of attributes $s_i.A = \{a_1, \ldots, a_m\}$. SOTs and attributes are identified by unique (artificial) identifiers. In this way we avoided the problem of name conflicts and the need for defining default names for attributes and SOTs being created during the transformation process. However, in subsequent examples we use names instead of identifiers for better readability. Attributes can be divided into basic attributes (of type integer, string, etc.), collection attributes (of set, list or array type) and reference attributes. Binary relationships between SOTs are expressed through reference attributes, which also define the cardinalities for both SOTs participating in the relationship. A reference attribute of an SOT s_i is a triple $r = (s_j, C_1, C_2)$, where s_j is the referenced SOT, C_1 denotes the cardinality of s_i-references to instances of s_j, and C_2 denotes the inverse cardinality. The cardinalities C_1 and C_2 are pairs $C_i = (c_{i_1}, c_{i_2})$ where $c_{i_1} \in \{0, 1\}$ and $c_{i_2} \in \{1, n\}$. In other words, the cardinalities denote whether the relationship is injective, total, surjective or functional. Inheritance is expressed as a special kind of one-to-one relationship between SOTs.

Data is expressed in form of objects which are elements of extensions. An object *o* is an instance of an SOT and consists of an identifier and a tuple value: $o = (id, [a_1{:}v_1, \ldots, a_n{:}v_n])$. An extension consists of a set of objects $\{o_1, \ldots, o_n\}$.

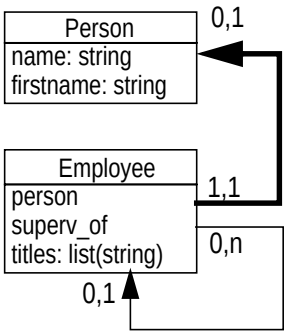
The SOT algebra defines a number of operators for both metadata and data manipulation. Various algebras for object-oriented data models exist [4, 16, 33, 32], most of them with the purpose of formalizing a query language. The SOT algebra was derived from NO_2 [20] and extended with restructuring and metadata operators. Operators are distinguished in schema operators and data operators. The application of schema operators produces side effects on the SOT schema. Data operators are free of side effects and provide various data restructuring operations. In the following, we briefly introduce a subset of those operators of the SOT algebra, which are used in subsequent examples. The complete and formal definition of the algebra is presented in [6].

- *Image operator*: The image operator $\iota[\lambda x\,.\,f(x)]$, as known from [4], applies the function *f* to every object within an extension argument: $\iota[\lambda x\,.\,f(x)]\{o_1, \ldots, o_n\} = \{f(o_1), \ldots, f(o_n)\}$.
- *Projection*: The projection operators has the same purpose as in the relational algebra. We distinguish *identifier projection* π_I and *tuple projection* π_T, which can be applied to objects, and *object projection* π_O, which can be applied to both, objects and extensions:

- $\pi_I(id, [a_1{:}v_1, \ldots, a_n{:}v_n]) = id$
- $\pi_{T\{a_{\pi_1}, \ldots, a_{\pi_m}\}}(id, [a_1{:}v_1, \ldots, a_n{:}v_n]) = [a_{\pi_1}{:}v_{\pi_1}, \ldots, a_{\pi_m}{:}v_{\pi_m}]$ where $\{\pi_1, \ldots, \pi_m\} \subseteq \{1, \ldots, n\}$
- $\pi_{O\{a_{\pi_1}, \ldots, a_{\pi_m}\}}(id, [a_1{:}v_1, \ldots, a_n{:}v_n]) = (id, [a_{\pi_1}{:}v_{\pi_1}, \ldots, a_{\pi_m}{:}v_{\pi_m}])$, and
- $\pi_{O\{a_{\pi_1}, \ldots, a_{\pi_m}\}}\{o_1, \ldots, o_n\} = \iota[\lambda x \,.\, \pi_{O\{a_{\pi_1}, \ldots, a_{\pi_m}\}}x]\{o_1, \ldots, o_n\}$

- *Selection:* This operator can be applied to all kinds of collection values, for example to extensions:
  - $\sigma[\lambda x \,.\, f(x)]\{o_1, \ldots, o_n\} = \{o_i \mid o_i \in \{o_1, \ldots, o_n\}, p(o_i)\}$
- *Map (extend)*: This variation of the map operator $\chi_{a:\lambda x \,.\, f(x)}$ extends an object with an attribute *a* whose value is computed by a function *f*:
  - $\chi_{a:\lambda x \,.\, f(x)}(id, [a_1{:}v_1, \ldots, a_n{:}v_n]) = (id, [a_1{:}v_1, \ldots, a_n{:}v_n, a : f((id, [a_1{:}v_1, \ldots, a_n{:}v_n]))])$
  - $\chi_{a:\lambda x \,.\, f(x)}\{o_1, \ldots, o_n\} = \iota[\lambda y \,.\, \chi_{a:\lambda x \,.\, f(x)}y]\{o_1, \ldots, o_n\}$
- *Map (rename)*: This variation of the map operator $\chi_{\langle a_{1_1}, \ldots, a_{1_m}\rangle : \langle a_{2_1}, \ldots, a_{2_m}\rangle}$ replaces attributes in an object without changing their values such that attribute a_{1_1} is replaced by a_{2_1} and so on:
  - $\chi_{\langle a_{1_1}, \ldots, a_{1_m}\rangle : \langle a_{2_1}, \ldots, a_{2_m}\rangle}(id, [a_{1_1}{:}v_{1_1}, \ldots, a_{1_m}{:}v_{1_m}, a_2{:}v_2, \ldots, a_n{:}v_n]) =$ $(id, [a_{2_1}{:}v_{1_1}, \ldots, a_{2_m}{:}v_{1_m}, a_2{:}v_2, \ldots, a_n{:}v_n])$
  - $\chi_{\langle a_{1_1}, \ldots, a_{1_m}\rangle : \langle a_{2_1}, \ldots, a_{2_m}\rangle}\{o_1, \ldots, o_n\} = \iota[\lambda x \,.\, \chi_{\langle a_{1_1}, \ldots, a_{1_m}\rangle : \langle a_{2_1}, \ldots, a_{2_m}\rangle}x]\{o_1, \ldots, o_n\}$
- *Concatenation*: Objects can be constructed through concatenation of an identifier and a tuple value:
  - $id \oplus [a_1{:}v_1,\ldots,a_n{:}v_n] = (id, [a_1{:}v_1,\ldots,a_n{:}v_n])$
- *Union*: The union operator unifies two extensions: $\{o_{1_1},\ldots,o_{1_n}\} \cup \{o_{2_1},\ldots,o_{2_m}\} = \{o_{1_1},\ldots,o_{1_n},o_{2_1},\ldots,o_{2_m}\}$
- *New identifier*: The operator *newid* creates a new identifier, derived from one (or more) existing identifier and one SOT identifier, e.g: $id_2 = newid(id_1, s_1)$. The novelty of this operator definition lies in the avoidance of side effects, that is, applying the operator for a certain identifier id_1 and a certain SOT s_1 always results in the same identifier id_2. In contrast, traditional (object creating) algebras always yield a new identifier value when invoking operators creating a new object. Details as well as variations of this operator can be found in [6].

An example of an SOT schema with two SOTs, Person and Employee, is shown on the left side of Fig. 2. The attributes name and firstname are basic attributes of type string. The type of the collection attribute titles is a list of strings. The reference attribute superv_of denotes a recursive relationship between employees. It is represented as an arrow with cardinalities at both ends. An employee can supervise zero or more employees whereas every employee is supervised by at most one other employee. The reference attribute person denotes an inheritance relationship, represented as a thick arrow. Instances of both SOTs are presented on the right side.



Person

id	name	firstname
id_1	Benn	Chris
id_2	Berge	Glenn
id_3	Felli	Paul
id_4	Kumar	Irina

Employee

id	person	superv_of	titles
id_5	id_1	{id_6, id_7}	<Prof,Dr>
id_6	id_2	{}	<Dr>
id_7	id_3	{}	<>

Figure 2: SOT schema and instances

4 The Migration Process

In this section, the complete migration process from relational schemas and data to object-oriented schemas and data is presented. An overview of the migration process within the SOT redesign environment is shown in Fig. 3. The schema transformation process is subdivided into three sequential tasks:

1. Transformation of the relational schema into an SOT schema
2. Redesign of the SOT schema
3. Transformation of the SOT schema into an object-oriented schema

The data migration process is generated automatically after completing these three steps of schema transformation.

4.1 Transformation of the Relational Schema into an SOT Schema

The transformation of the relational schema into an SOT schema is a straightforward process. For each relation in the relational schema, composed of relations $\{r_1, \ldots, r_n\}$, one SOT is created containing the same attributes. The initial SOT schema thus consists of a set of SOTs $S_I = \{s_1, \ldots, s_n\}$. Then, a set of *initial extensions* $\{e_{s_1}, \ldots, e_{s_n}\}$ is computed, which contains all the instances of the initial SOTs. These instances are composed by concatenating tuples of the corresponding relation and (unique) object identifiers. The initial extensions remain constant throughout the entire process.

During the schema transformation process, every SOT s_i within the schema has its own *current extension* E_{s_i} which consists of an algebraic expression determining the current instances of s_i. Consequently, the current extensions of the initial SOT schema are initialised as $E_{s_i} = e_{s_i}$ for $s_i \in S_I$.

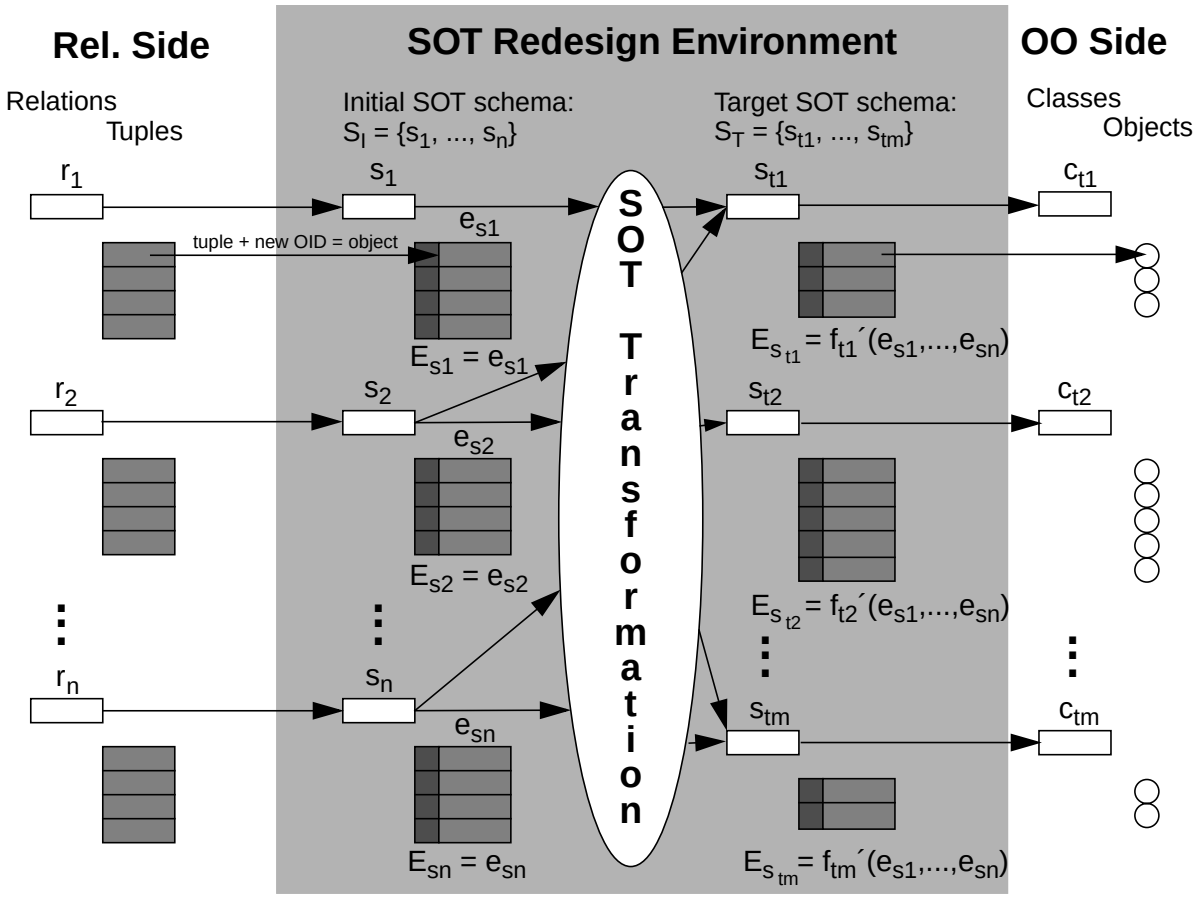


Figure 3: Overview of the migration process

4.2 Redesign of the SOT Schema

Redesign of the SOT schema means transforming the initial SOT schema into a schema having a more object-oriented flavour, i.e., an object-oriented schema as created by forward engineering using rigorously an object-oriented design method like OOD [11]. We now introduce the concept of *transformation rules*, which supports a consistent modification of schema and data.

Schema transformations are well-known in both, the relational [23, 27, 3] and the object-oriented [9] context. On the relational side, schema transformations have been used for reverse engineering [23] or quality improvement [3], in order to reduce deficiencies of the relational schema like denormalization or optimization. On the object-oriented side, schema transformations have been used for the detailed design phase [9]. Some of those transformation rules have been adopted. We currently propose 22 transformation rules, which are presented in detail in [7]; new transformation rules can easily be added. All transformation rules have a common structure which consists of five parts: a pattern, definitions, preconditions, schema operations and data operations:

- **Pattern:** The pattern defines to which elements of the SOT schema the rule can be applied. These elements also serve as input of the subsequent operations parts.

- **Definitions:** In this part, new SOTs or attributes are defined which result from applying the transformation rule.
- **Preconditions:** This part contains a number of conditions which must hold when the transformation rule is applied for a certain pattern. All predicates are first order logic expressions of the SOT algebra and contain pattern elements as variables.
- **Schema Operations:** The way a transformation rule is executed is defined in the schema and data operations parts. This part contains a set of SOT schema operations which add, modify or remove SOTs or attributes.
- **Data Operations:** The last part of a transformation rule contains a set of data operations modifying the extension expressions E_{s_i} of modified or added SOTs s_i. Suppose a transformation rule creates a new SOT s_3 derived from two SOTs s_1 and s_2. The extension E_{s_3} is then expressed as $E_{s_3} = f(E_{s_1}, E_{s_2})$ where *f* is a sequence of algebraic operators.

In the following we introduce two concrete transformation rules, both of which are used in the example in section 5. The first transformation rule vertically splits an SOT, as shown in Fig. 4. The formalism of this transformation rule is presented in Fig. 5. Vertically splitting an SOT s_1 means creating a new SOT s_2 which contains a proper subset of the attributes of s_1. In turn, these attributes are removed from s_1. The pattern consists of the SOT s_1 which has to be split, and a set of attributes A_1 which are moved to the new SOT. The *definitions* part creates the new SOT s_2 which contains the attributes A_1 and a new attribute *a*, referencing s_2 in an exactly-one-to-exactly-one relationship. The precondition states that A_1 must be a proper subset of the attributes of s_1. The effects of schema operations are modifying the attributes of s_1 and including s_2 in the SOT schema, denoted by a global variable *SOT*. The data operators compute the extensions of s_1 and s_2. The extension E_{s_1} results from a projection on those attributes remaining in s_1 and a mapping of the reference attribute *a*. The map operator $\chi_{a:\lambda x\,.\,f(x)}$ adds the attribute *a* to each object in the argument extension. The value of the attribute is computed by applying the function *f(x)* captured in the lambda expression to the object. The extension E_{s_2} is computed by creating one new object for every existing object within E_{s_1}. Each new instance of E_{s_2} is composed of a new object identifier and the attributes A_1.

Two variations of this transformation rule exist which are not presented here. The first variation does not create an instance of s_2 for those instances of s_1 where all attributes of A_1 contain null values. Thus, the reference attribute a_2 denotes a zero-or-one-to-exactly-one relationship. The second variation eliminates duplicates such that a_2 denotes a one-to-many relationship.

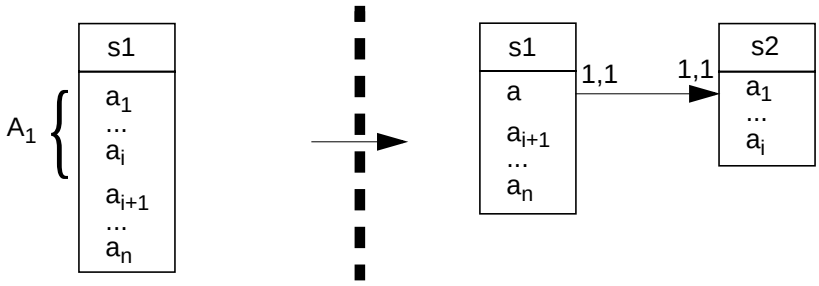


Figure 4: Vertical split of an SOT

Pattern	$s_1 : S$ $A_1 : set(A)$
Definitions	$s_2 = new_{SOT}(A_1)$ $a = new_{Ref}(s_2, (1,1), (1,1))$
Preconditions	$A_1 \subset s_1.A$
Schema Operations	$s_1.set\_attributes(s_1.A - A_1 + set\{a\})$ $SOT = SOT + set\{s_2\}$
Data Operations	$E_{s_1} = \chi_{a : \lambda x\,.\, newid(\pi_I x,\, s_2)} \pi_{O_{s_1.A - A_1}} E_{s_1}$ $E_{s_2} = \iota[\lambda x\,.\, newid(\pi_I x, s_2) \oplus \pi_{T_{A_1}} x] E_{s_1}$

Figure 5: Transformation rule: vertical split of SOT

The second transformation rule merges two SOTs s_1 and s_2 into a single one, as shown in Fig. 6. The formalism of this transformation rule is presented in Fig. 7. The original rule is rather complex and thus has been simplified in this context. In this case, all attributes of s_1 are mapped to the attributes of s_2 and both SOTs do not contain reference attributes. The pattern consists of two SOTs s_1 and s_2, denoting s_2 is merged into s_1. Two attribute lists $al_1 = \langle a_{1_1}, \ldots, a_{1_n} \rangle$ and $al_2 = \langle a_{2_1}, \ldots, a_{2_n} \rangle$ denote the order in which the attributes are mapped, i.e., a_{2_1} is mapped to a_{1_1} and so on. The definitions part is empty. The preconditions state that the participating SOTs must be different, all attributes of both SOTs s_1 and s_2 appear in the pattern's attribute lists, and the types of corresponding attributes must be equal, i.e., the type of a_{1_1} must be the same as the type of a_{2_1} and so on. Schema operators remove s_2 from the SOT schema and modify all reference attributes referencing either s_1 or s_2. The new extension of s_1 is computed by unifying E_{s_1} and E_{s_2} whereby the instances of E_{s_2} undergo the attribute mapping.

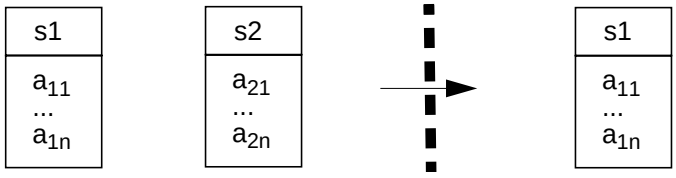


Figure 6: Merging two SOTs

Pattern	$s_1 : S$ $s_2 : S$ $al_1 : list(A)$ $al_2 : list(A)$
Definitions	
Preconditions	$s_1 \neq s_2$ $members(al_1) = s_1.A$ $members(al_2) = s_2.A$ $\forall(i \in indexlist(al_1))(type(al_1[i]) = type(al_2[i]))$

Schema Operations	$SOT = SOT - set\{s_2\}$
	for ($s \in SOT$)
	  for ($a \in s.R$)
	  if ($a.S = s_2$)
	  $a.set\_S(s_1)$
	  if ($a.S = s_1$)
	  $a.set\_c21(0)$

Data Operations	$E_{s_1} = E_{s_1} \cup \chi_{al_2 : al_1} E_{s_2}$

Figure 7: Transformation rule: merge SOTs

4.3 Transformation of the SOT Schema into an Object-Oriented Schema

The last step of an SOT-schema redesign is the computation of target extensions $\{E_{s_{t_1}}, \ldots, E_{s_{t_m}}\}$. Once this step has been completed, the target SOT schema $S_T = \{s_{t_1}, \ldots, s_{t_m}\}$ is transformed into an object-oriented schema. Extensions can be computed either by successive computation of the incurred data operations (without the possibility of optimization) or by recursively rewriting the extension expressions such that all target extensions are expressed as functions defined over the initial extensions: $E_{s_{t_i}} = f_{t_i}'(e_{s_1}, \ldots, e_{s_n})$. These expressions can be optimized, which is outside the scope of this paper. Single terms can be optimized by applying predefined algebraic rewriting rules. Common subexpressions of different extensions have to be computed only once. All resulting target extensions are independent of each other, therefore parallel computation is possible. Optimization for relational and object-oriented algebras has been extensively studied in [5, 15, 16, 28, 33, 35]. The resulting expressions are then compiled into queries against the relational database.

The target SOT schema can then, again in a straightforward process, be transformed into an object-oriented schema. The object-oriented database is populated by creating objects from the SOT target extensions. This can be performed by creating code in the ODMG object interchange format (OIF) [13], or by generating a migration program. An example of OIF code for the instances presented in Fig. 2 is shown in Fig. 8. One employee object is composed of one SOT Employee object and one SOT Person object. This step concludes the migration process.

```
id4 Person {
    name "Kumar",
    firstname "Irina"}

id5 Employee {
    name "Benn",
    firstname "Chris",
    superv_of {id6, id7},
    titles {"Prof", "Dr"}}

id6 Employee {
    name "Berge",
    firstname "Glenn",
    superv_of {},
    titles {"Dr"}}

id7 Employee {
    name "Felli",
    firstname "Paul",
    superv_of {},
    titles {}}
```

Figure 8: Example of OIF code

5 A Migration Example

The examples presented in this section illustrate most issues introduced in this paper. In particular, we present an example for schema transformation, and afterwards consider a subset of this example for the demonstration of two transformation rules and data migration.

5.1 Schema Transformation

The relational schema in Fig. 9 is composed of seven relations and contains information about institutes, employees, students and hardware. For simplicity we omitted the type of each attribute. Most of the relations are self-explanatory. The relation Institute contains five description attributes representing a textual field of at most five lines. The relation Employee contains information about name, office and address of employees. Students are characterized by a name, a local address and a home address. The relation Hardware contains information about workstations and monitors, which are distinguished by the attribute type. An IP can be applied to workstation entities and a screen size can be applied to monitors. The relation InstEmp defines a many-to-many relationship between Institute and Employee.

The initial SOT schema is presented in Fig. 10. After applying various transformation rules, the resulting target SOT schema is shown in Fig. 11. Finally the class structure of the object-oriented schema is presented in Fig. 12, expressed by the interface notation of the ODMG object definition language (ODL) [13].

Two aspects are worth mentioning in this example. First, traditional migration tools are only able to replace the relation InstEmp by a many-to-many relationship. The remaining relations will be mapped to classes having the same attributes. Second, the schema contains three typical examples of transformation patterns. The first one is the multiple occurrence of common attribute groups like the address attributes street, zip and city, or the office attributes office_floor and office_nr. The second example is the implemen-

Employee (emp_id, name, office_floor, office_nr, adr_street, adr_zip, adr_city)
Institute (name, street, zip, city, descr1, descr2, descr3, descr4, descr5)
Student (stud_id, name, adr_street, adr_zip, adr_city, home_street, home_zip, home_city)
Hardware (serial_nr, office_floor, office_nr, type, ip, screen_size)
InstEmp (emp_id, inst_name)

Primary Keys: Employee (emp_id)
Institute (name)
Student (stud_id)
Hardware (serial_nr)
InstEmp (emp_id, inst_name)

Foreign Keys: InstEmp (inst_name) -> Institute (name)
InstEmp (emp_id) -> Employee (emp_id)

Figure 9: Relational schema

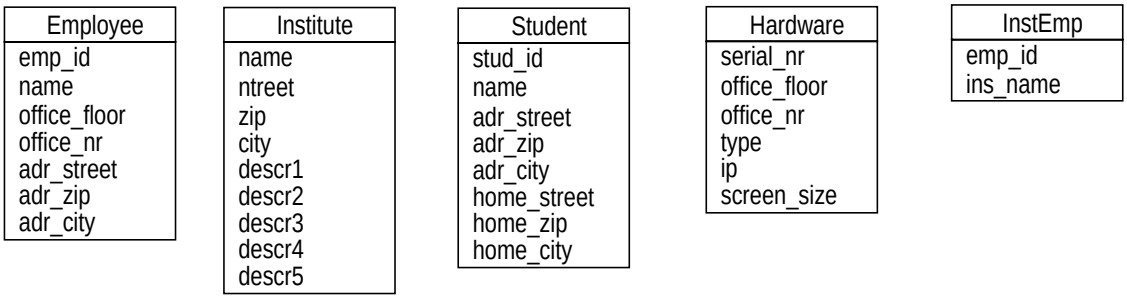


Figure 10: Initial SOT schema

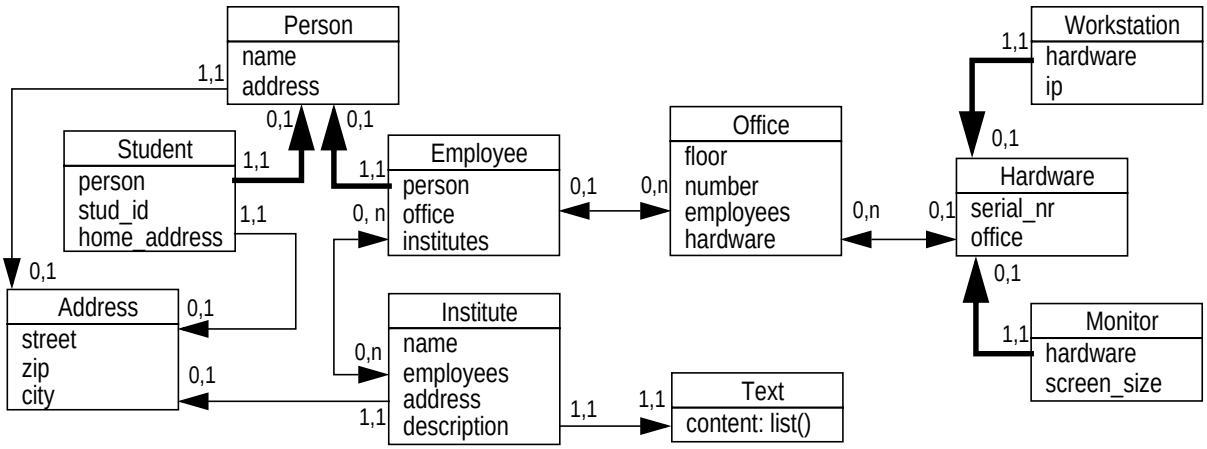


Figure 11: Target SOT schema

tation of a multivalued attribute through multiple separated attributes as in the description of an institute. Finally, the last example consists of the variant attribute type of the relation Hardware, whose purpose it is, to distinguish workstation and monitor entities.

In the first step of the transformation process, reference attributes are created. They are determined, e.g., through inclusion dependencies, which have been gathered from reverse engineering of the relational data-

```
interface Person {
    attribute string name;
    attribute Address address;};

interface Student : Person {
    attribute string stud_id;
    attribute Address parent_address;};

interface Employee : Person {
    attribute string emp_id;
    relationship Office office
        inverse Office::employees;
    relationship set<Institute> institutes
        inverse Institute::employees;};

interface Institute {
    attribute string name;
    attribute Address address;
    attribute Text description;
    relationship set<Employee> employees
        inverse Employee::institutes;};

 interface Text {
     attribute list<string> content;};
```

```
interface Address {
   attribute string street;
   attribute string zip;
   attribute string city;};

interface Hardware {
    attribute string serial_nr;
    relationship Office office
        inverse Office::hardware;};

interface Workstation : Hardware {
    attribute string ip;};

interface Monitor : Hardware {
    attribute string screen_size;};

interface Office {
    attribute string floor;
    attribute string number;
    relationship set<Employee> employees
        inverse Employee::office;
    relationship set<Hardware> hardware
        inverse Hardware::office;};
```

Figure 12: Object-oriented schema

base. The cardinalities of a reference attribute can be determined through the presence of primary keys, candidate keys and null values in the relational schema.

Most existing transformation rules change the structure of the SOT schema. SOTs can be split vertically or horizontally. Vertical split was applied for all SOTs except InstEmp, such that the address attributes street, zip and city, and the office attributes office_floor and office_nr have been moved to new SOTs. As an example of horizontal splitting, hardware has been specialized into workstations and monitors, depending on the value of the attribute type. Afterwards, the attributes ip and screen_size have been shifted to Workstation and Monitor, respectively. Merging two SOTs into a single one was applied for those SOTs obtained from splitting addresses and offices, resulting in single SOTs Address and Office.

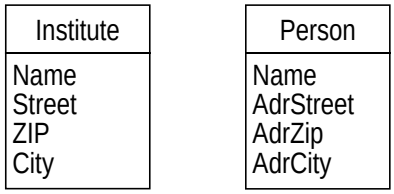
It can be distinguished whether duplicates lead to multiple objects or a single object. In case of the SOT Address, every address is mapped into one object, such that identical addresses lead to distinct objects. In case of the SOT Office, duplicates have been removed, such that every combination of office_floor and office_nr leads to one object. Applying the first variation results in a one-to-one relationship, whereas the second variation results in a one-to-many relationship.

The initial SOT InstEmp has been replaced by a many-to-many relationship between Institute and Employee in both SOTs. This relationship is denoted by a bidirected arrow linking the attributes institutes and employees, and represents an inverse relationship as known from the ODMG standard [13].

Another group of transformation rules supports the creation of collection attributes. This way the five description attributes of Institute have been transformed into a single list attribute. Finally, SOTs and attributes which became obsolete can be removed, for example, the SOT InstEmp.

5.2 Data Migration

For the following subset of the university example, we demonstrate the use and the effects of two transformation rules. The example in Fig. 13 presents an initial SOT schema comprising two SOTs Institute and Person. The initial extensions of both SOTs are shown on the right side. The transformation rules as introduced in section 4.2 have to be applied for obtaining the target SOTs presented in Fig. 15.



Institute

id	Name	Street	ZIP	City
id_1	IfI	Winterthurerstr. 190	8057	Zurich
id_2	GIUZ	Winterthurerstr. 190	8057	Zurich

Person

id	Name	AdrStreet	AdrZIP	AdrCity
id_3	Oeler	Gujerstr. 10	9200	Gossau
id_4	Berger	Rigistr. 23	3855	Brienz

Figure 13: Subset of SOT schema with extensions

The effects on schema and instance level of applying this transformation rule for both Institute and Person are illustrated in Fig. 14. The pattern is initialized with s_1 = Institute and A_1 = {Street, ZIP, City} as well as with s_1 = Person and A_1 = {AdrStreet, AdrZIP, AdrCity}. The resulting current extensions of the SOTs Institute, Person, Address and PersAddr are defined by the following algebraic expressions, based on the initial extensions $e_{\text{Institute}}$ and e_{Person}:

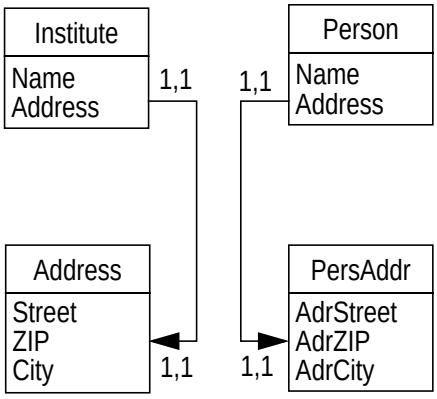
$E_{\text{Institute}} = \chi_{\text{Address} : \lambda x\,.\, newid(\pi_I x,\, \text{Address})} \pi_{O_{\{\text{Name}\}}} e_{\text{Institute}}$

$E_{\text{Address}} = \iota[\lambda x\,.\, newid(\pi_I x,\, \text{Address}) \oplus \pi_{T_{\{\text{Street, ZIP, City}\}}} x] e_{\text{Institute}}$

$E_{\text{Person}} = \chi_{\text{Address} : \lambda x\,.\, newid(\pi_I x,\, \text{PersAddr})} \pi_{O_{\{\text{Name}\}}} e_{\text{Person}}$

$E_{\text{PersAddr}} = \iota[\lambda x\,.\, newid(\pi_I x,\, \text{PersAddr}) \oplus \pi_{T_{\{\text{AdrSreet, AdrZIP, AdrCity}\}}} x] e_{\text{Person}}$

The effects on schema and instance level of applying this transformation rule is illustrated in Fig. 15. The pattern is initialised with s_1 = Address, s_2 = PersAddr, $al_1 = \langle \text{Street, ZIP, City} \rangle$ and $al_2 = \langle \text{AdrStreet, AdrZIP, AdrCity} \rangle$. The current extension of the SOT Address is defined by the following algebraic expression, in the second case again based on the initial extensions $e_{\text{Institute}}$ and e_{Person}:



Institute

id	Name	Address
id_1 id_2	IfI GIUZ	id_5 id_6

Person

id	Name	Address
id_3 id_4	Smith Glenn	id_7 id_8

Address

id	Street	ZIP	City
id_5 id_6	Winterthurerstr. 190 Winterthurerstr. 190	8057 8057	Zurich Zurich

PersAddr

id	AdrStreet	AdrZIP	AdrCity
id_7 id_8	Gujerstr. 10 Rigistr. 23	9200 3855	Gossau Brienz

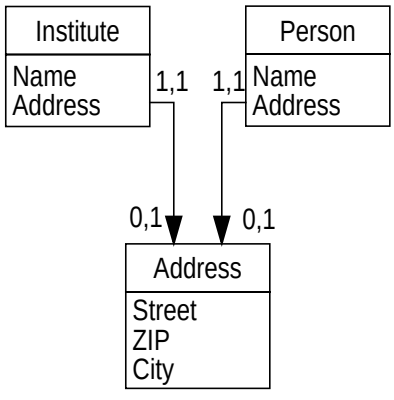
Figure 14: Effects of applying vertical SOT split

$$E_{\text{Address}} = E_{\text{Address}} \cup \chi_{\langle \text{AdrStreet, AdrZIP, AdrCity} \rangle : \langle \text{Street, ZIP, City} \rangle} E_{\text{PersAddr}}$$

$$E_{\text{Address}} = \iota[\lambda x \,.\, newid(\pi_I x, \text{Address}) \oplus \pi_{T_{\{\text{Street, ZIP, City}\}}} x] e_{\text{Institute}} \cup \chi_{\langle \text{AdrStreet, AdrZIP, AdrCity} \rangle : \langle \text{Street, ZIP, City} \rangle}(\iota[\lambda x \,.\, newid(\pi_I x, \text{PersAddr}) \oplus \pi_{T_{\{\text{AdrSreet, AdrZIP, AdrCity}\}}} x] e_{\text{Person}})$$

This expression can be rewritten such that the map operator is embedded in the image operator, and only one iteration over the objects in the extension e_{Person} is necessary.

$$E_{\text{Address}} = \iota[\lambda x \,.\, newid(\pi_I x, \text{Address}) \oplus \pi_{T_{\{\text{Street, ZIP, City}\}}} x] e_{\text{Institute}} \cup \iota[\lambda x \,.\, \chi_{\langle \text{AdrStreet, AdrZIP, AdrCity} \rangle : \langle \text{Street, ZIP, City} \rangle}(newid(\pi_I x, \text{PersAddr}) \oplus \pi_{T_{\{\text{AdrSreet, AdrZIP, AdrCity}\}}} x)] e_{\text{Person}}$$



Institute

id	Name	Address
id_1 id_2	IfI GIUZ	id_5 id_6

Person

id	Name	Address
id_3 id_4	Smith Glenn	id_7 id_8

Address

id	Street	ZIP	City
id_5 id_6 id_7 id_8	Winterthurerstr. 190 Winterthurerstr. 190 Gujerstr. 10 Rigistr. 23	8057 8057 9200 3855	Zurich Zurich Gossau Brienz

Figure 15: Effects of merging SOTs

6 Conclusions and Future Work

In this paper we studied the problem of migrating relational databases into object-oriented databases. Both paradigms follow different design strategies. Moreover, legacy databases often contain specific drawbacks which should be overcome when reengineering them. The differences in design strategies motivate schema transformation techniques being more powerful and flexible than existing ones, which mostly generate structurally identical object-oriented schemas.

In the second part, we presented a framework which provides a formal foundation for the migration of legacy relational schemas and data into object-oriented databases. This is the first approach which supports automatic data migration for complex schema transformations. The concept presented here allows straightforward implementation of a tool. In particular we described:

- The complete migration process from relational schema and data to object-oriented schema and data.
- A data model in which the migration process is embedded.
- The concept of transformation rules supporting complex schema transformations.

We currently evaluate and validate our approach by migrating a large database managing bank accounts. The relational schema consists of 100 relations with altogether 3.500 attributes and 180 views. First experiences indicate the feasibility of this approach. Furthermore, we are currently working on the following extensions:

- Tool support: how can a tool support or simplify the transformation of large schemas and how can it control the transformation process? To what extent can heuristics support choosing the best-suited transformation rules?
- Updates: for simplicity we assumed that there are no updates on the database during the migration process. How can updates on the source database be propagated through the migration process such that incremental migration is possible?
- Optimization: How can the cost of the migration process be reduced by using algebraic rewriting and parallelization?

Acknowledgements

We thank Ruxandra Domenig, Dirk Jonscher and Martin Schönhoff for their helpful comments on an earlier version of this paper.

References

[1] S. Amer-Yahia, S. Cluet, and C. Delobel. Bulk loading techniques for object databases and an application to relational data. *Proc. Int'l Conf. on Very Large Databases (VLDB)*, pages 534–545, New York, August 1998.

[2] M. Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. *Proc. 13th Int'l Conf. on ER Approach*, pages 403–419, Manchester, UK, 1994.

[3] C. Batini, S. Ceri, and S. Navathe. *Conceptual database design: an entity-relationship approach*. Benjamin/Cummings Publishing Company, Inc., 1992.

[4] C. Beeri. Query languages for models with object-oriented features. In *Advances in Object-Oriented Database Systems*, NATO ASI Series, chapter 3, pages 47–72. Springer-Verlag, 1994.

[5] C. Beeri and Y. Kornatzky. Algebraic optimization of object-oriented query languages. *Proc. 3rd Int'l Conf. on Database Theory*, pages 72–88, Paris, France, December 1990.

[6] A. Behm. Semi object types - a data model and an algebra for database migration. Unpublished, 1999

[7] A. Behm. Transformation rules for database migration in the SOT model. Unpublished, 1999

[8] A. Behm, A. Geppert, and K. R. Dittrich. On the migration of relational schemas and data to object-oriented database systems. *Proc. 5th Int'l Conf. on Re-Technologies for Information Systems*, pages 13–33, Klagenfurt, Austria, December 1997.

[9] M. Blaha and W. Premerlani. A catalog of object model transformations. *Proc. 3rd Working Conf. on Reverse Engineering*, pages 87–96, Monterey, California, November 1996.

[10] M. Blaha and W. Premerlani. Detailed design. In *Object-Oriented Modeling and Design for Database Applications*, chapter 10. Prentice-Hall, 1998.

[11] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, 1994.

[12] M. Castellanos and F. Saltor. Semantic enrichment of database schemas: An object oriented approach. *Proc. 1st Int'l Workshop on Interoperability in Multidatabase Systems*, pages 71–78, 1991.

[13] R. G. G. Cattell. *The object database standard: ODMG 2.0*. Morgan Kaufmann, 1997.

[14] R. Chiang, T. Barron, and V. Storey. Reverse engineering of relational databases: Extraction of an EER model from a relational database. *Data & Knowledge Engineering*, 12:107–142, 1994.

[15] S. Cluet and C. Delobel. A general framework for the optimization of object-oriented queries. *Proc. SIGMOD Int'l Conf. on Management of Data*, pages 383–392, New York, NY, June 1992.

[16] S. Cluet and G. Moerkotte. Classification and optimization of nested queries in object bases. In *Bases de Donnees Avancees*, pages 331–349. 1994.

[17] K Davis and A Arora. Converting a relational database model into an entity-relationship model. *Proc. 6th Int'l Conf. of the Entity Relationship Approach*, pages 271–285, New York, 1987.

[18] C. Fahrner and G. Vossen. Transforming relational database schemas into object-oriented schemas according to ODMG-93. *Proc. Int'l Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 429–446, Singapore, December 1995.

[19] J. Fong. Converting relational to object-oriented databases. *SIGMOD Record*, 26(1):53–58, March 1997.

[20] A. Geppert, K. R. Dittrich, V. Goebel, and S. Scherrer. The NO2 data model. Technical Report 93.09, Institut fuer Informatik der Universität Zürich, 1993.

[21] J. Hainaut, M. Chandelon, C. Tonneau, and M. Joris. Contribution to a theory of database reverse engineering. In *IEEE Working Conf. on Reverse Engineering*, pages 161–170, Baltimore, May 1993.

[22] J. Hainaut, V. Englebert, J. Henrard, J. Hick, and D. Roland. Requirements for information system reverse engineering support. *Proc. IEEE Working Conf. on Reverse Engineering*, Toronto, July 1995.

[23] J-L. Hainaut, C. Tonneau, M. Joris, and M. Chandelon. Transformation-based Database Reverse Engineering. *Proc. 12th Int'l Conf. on the Entity-Relationship Approach*, pages 364–375, Dallas, Texas, December 1993.

[24] J. Jahnke, W. Schäfer, and A. Zündorf. Generic fuzzy reasoning nets as a basis for reverse engineering relational database applications. Proc. *6th European Software Engineering Conf.*, 1997.

[25] P. Johannesson. A method for transforming relational schemas into conceptual schemas. *Proc. 10th Int'l Conf. on Data Engineering*, pages 190–201, Houston, February 1994.

[26] D. Jordan. Some comparisons between object and relational database technology. In *C++ Object Databases, Programming with the ODMG Standard*, chapter 17. Addison-Wesley, 1997.

[27] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. Proc. *19th VLDB Conf.*, pages 120–133, Dublin, Ireland, 1993.

[28] G. Mitchell, S.B. Zdonik, and U. Dayal. Optimizations of object-oriented query languages: Problems and approaches. In *Advances in Object-Oriented Database Systems*, NATO ASI Series, chapter 6, pages 119–146. Springer-Verlag, 1994.

[29] J-M. Petit, F. Toumani, J-F. Boulicaut, and J. Kouloumdjian. Towards the reverse engineering of de-

normalized databases. *Proc. Int'l Conf. on Data Engineering*, pages 218–227. IEEE, 1996.

[30] 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.

[31] G. Saake, S. Conrad, I. Schmitt, and C. Türker. Object-oriented database design: What is the difference with relational database design. In *ObjectWorld Frankfurt'95 – Conf. Notes, Wednesday, October 11*, pages 1–10, 1995.

[32] G. Saake, R. Jungclaus, and C. Sernadas. Abstract data type semantics for many-sorted object query algebras. *Proc. 3rd Symp. on Mathematical Fundamentals of Database and Knowledge Base Systems MFDBS-91*, pages 291–307, Rostock, Germany, 1991.

[33] G. M. Shaw and S. B. Zdonik. A query algebra for object-oriented databases. *Proc. Int'l Conf. on Data Engineering*, pages 154–162, Los Angeles, CA, February 1990.

[34] A. Umar. *Application (Re)Engineering - Building Web-Based Applications and Dealing with Legacies*. Prentice Hall Int'l, London, UK, 1997.

[35] S. L. Vandenberg and D. J. DeWitt. Algebraic support for complex objects with arrays, identity, and inheritance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data)*, May 1991

[36] M. Vermeer and P. Apers. Object-oriented views of relational databases incorporating behavior. *Proc. 4th Int'l Conf. for Advanced Application (DASFAA)*, Singapore, April 1995.