Stepwise Re-Engineering and Development of Object-Oriented Database Schemata

Martin Gogolla$^{(A)}$, Anne Kathrin Huge$^{(A)}$, Bodo Randt$^{(B)}$

Universität Bremen$^{(A)}$, FB3, AG Datenbanksysteme,
Postfach 330440, D-28334 Bremen

STN Atlas Elektronik$^{(B)}$, Abteilung Verkehrssimulation,
Seebaldsbrücker Heerstr. 235, D-28309 Bremen

Abstract

We present a general approach for re-engineering of object-oriented database schemata. The approach consists of four dependent steps: (1) description of the data within the underlying basic data format, (2) application of a powerful semantic data model in order to construct a semantic database schema, (3) translation of the achieved schema into a general object model, and (4) implementation of the object schema in a concrete object-oriented database system. As an instantiation of this general procedure we report on a case study carried out in an industrial context where an Extended Entity-Relationship model was used as the semantic data model and ObjectStore as the implementation platform.

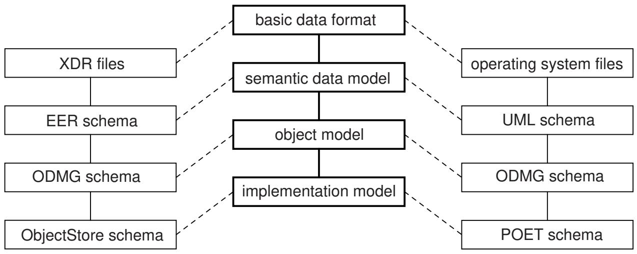

Figure 1. Steps for Re-Engineering Object-Oriented Database Schemata

1. Introduction

From-scratch database schema development and re-engineering of database schemata are special cases of general software development activities. Thus well-developed techniques known from the software engineering world should be applied here as well. This means, that one should develop software in identifiable, small, and precisely defined steps. Each step yields a resulting document, e.g. a specification or a piece of code, which is input to further steps.

The choice of suitable languages for these steps is crucial. It is advisible to use languages requiring to specify implementation details as late as possible. The more general and implementation-independent the chosen language, the greater is the possiblity to re-use decisions and results in contexts which are slightly different from the originally one.

The application of this stepwise process to database schema development is captured in Fig. 1 which shows in the middle part the general steps for re-engineering of object-oriented database schemata: (1) description of the data within the underlying basic data format, (2) application of a powerful semantic data model in order to construct a semantic database schema, (3) translation of the achieved schema into a general object model, and (4) implementation of the object schema in a concrete object-oriented database system. As an instantiation of this general procedure, the left part of the diagram pictures our concrete development process whereas the right part of the figure captures an alternative choice of languages realizable within the general approach.

In the project we report on, we started with XDR file descriptions, developed from those definitions an Extended Entity-Relationship (EER) schema, transformed this schema into a schema conform to the ODMG standard, and as the last step we implemented a prototype of the schema in the ObjectStore database system. Later in the project it turned out that not ObjectStore but the Poet database system was to be used for the running system. Therefore our design decision to proceed in small steps and especially to have available the ODMG schema (without the ObjectStore implementation details) made it possible to smoothly switch from the implementation platform ObjectStore to the implementation platform Poet. Other typical choices of languages could for instance be simple, flat operating systems files as the starting point of the re-engineering activity. An attractive alternative to the EER model is the UML notation which has been established during the run time of our project.

The structure of the rest of the paper is as follows. Section 2 shortly sketches the starting point for the project. Afterwards, Sect. 3 introduces the semantic data model we employed, namely our Extended Entity-Relationship model. Here, we also show the central ER schema of our application. In Sect. 4 we point out how to translate the achieved semantic data model schema into a general object schema conform to the ODMG standard. Due to space limit we cannot go into the detail of the ObjectStore implementation and left out this section. The paper ends with a discussion of related approaches in Sect. 5 and some conclusions in Sect. 6.

2. Motivation for Re-Engineering

Before presenting the different schemata, we briefly sketch the situation we found when starting with our work. Our application area is traffic simulation where a databasis toolset serves as a base for generating data describing streets and street crossings and building house fronts linked with these streets and railways. Streets and railways are represented as polygons. The application domain of the data (and the software built thereon) is the simulation of traffic situations during training of tram and truck drivers. The generation of a databasis consists of generating elements like streets, street crossings, places and rails. These elements are assigned to point features, line features, and areal features. Point features describe things like crossings, cars, persons, and traffic lights. Line features serve for entities like streets, roads, and rails. Areal features are constructs for generating entities like places and traffic isles. In this way, a databasis is composed by a number of features which can be partioned in point, line, and areal ones. Each feature consists of at least one point.

Originally, the data relevant for modeling our case study was represented in hierarchical flat files, namely in the SUN XDR-format (E**x**ternal **D**ata **R**epresentation). One of the main disadvantages of this form of data organisation is its representation in a file system so that during database generation in the modeling process sharing of data by more than one designer is not possible. The representation in hierarchical files is similar to a tuple representation where a number of elements, each of some fixed type, describe a user-defined type. Each user-defined type is described by a `struct` definition containing attributes. Relationships among user-defined types are constructed in a hierarchical manner so that a user-defined type references another user-defined type as an attribute type.

3. Employing an Extended Entity Relationship Model

In the specification phase of our case study we describe the basic elements of the database toolset for traffic simulation in terms of an EER Schema. The basic elements mainly preserve the original structure of the data initially represented in XDR format. Since the area of mapping a conventional (hierarchic, network, relational) logical database schema to a corresponding semantic data model schema has been extensively studied [NA91], [JK89], [CBS94], [DA87], we do not concentrate on this step. Our primary goal was the development of an object-oriented database schema with an EER schema as the basis. The advantages of using an EER model are (1) ease of understanding of the underlying concepts due to the graphical representation, (2) independence of the used database management system, and (3) a precise formal foundation [Gog94]. In particular, the last point guarantees an unambigious transformation into an object-oriented database schema. Using the EER model for re-engineering purposes means we employ it both in backward direction when mapping an existing conventional schema into a conceptual EER schema and in forward direction when transforming the resulting conceptual schema into an object-oriented schema. For this purpose, the EER schema can be considered as an intermediate implementation-independent schema.

3.1. EER Modeling Primitives

We now describe the central notion of this section, the EER schema, because we want to demonstrate a part of our case study with the EER model as shown in Fig. 2. Data types can be predefined or user-defined. In order to allow multi-valued types, the type constructors set, list, bag, and tuple can generate pretty arbitrary new types.

Besides the basic constructs, object-valued attributes and type constructions can be used. Object-valued attributes allow for composing objects out of other objects. Type constructions are modeling primitives to provide for specialization and generalization. In the EER diagram, object-valued attributes are represented by ovals which are added by an extra arrow pointing to the entity type giving the value of the attribute. Type constructions are represented by triangles connecting input types at the base line with output types at the opposite point.

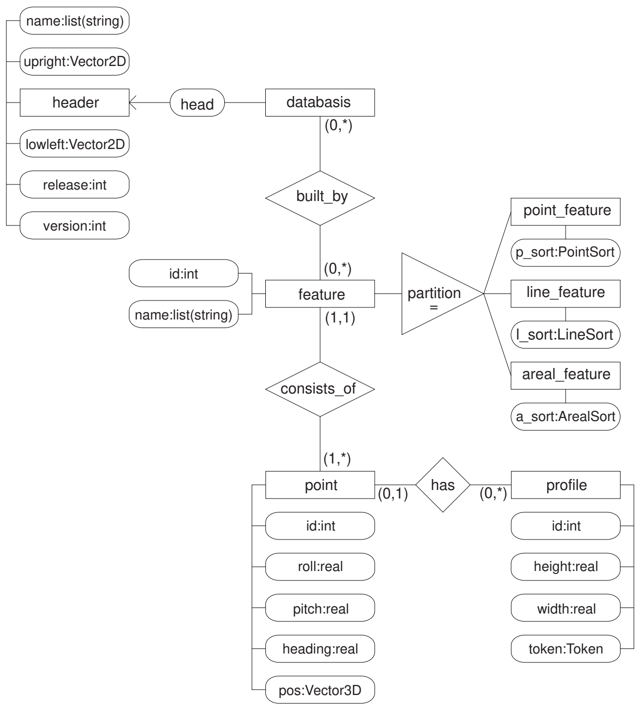

Figure 2. Extended Entity-Relationship Diagram (Excerpt)

3.2. Data Level

One main aspect of the EER model is the strict separation of the data level and the object level. Instances of the data level (values) do not change their properties over time (or when going from one system state to the next one) whereas the instances of the object level (objects) do. The modeling of objects generates an object level. Attribute domains, which are specified by data types, constitute the data level.

In this section, we briefly describe the user-defined data types important for the understanding of our case study. Besides the predefined data types `int`, `real` and `string`, the user-defined data types `Vector2D`, `Vector3D`, `PointSort`, `LineSort`, `ArealSort`, and `Token` are needed. Instead of defining these data types formally, we give an intuitive description.

The data types `Vector2D` and `Vector3D` describe tuples and triples of reals for the specification of two- and three-dimensional coordinates, resp. `PointSort`, `LineSort`, and `ArealSort` are enumeration types designed to describe the kind of point, line, and areal feature, resp. The data type `Token` is an enumeration type as well. It describes the kind of profile for the specification of certain road points in a databasis.

3.3. EER Diagram of the Case Study

We now present (a part of) the EER diagram for our case study as given in Fig. 2. The central entity types were already present in the original XDR format and the relationships were implemented by XDR `struct` components. First, we identify the basic constructs of the EER model, the entity types `databasis`, `header`, `feature`, `point_feature`, `line_feature`, `areal_feature`, `point`, and `profile`. Between the entity types `databasis` and `feature`, there is the relationship type `built_by`. Relationships between `feature` objects and `point` objects are described by the relationship type `consists_of` and between `point` objects and `profile` objects by the `has` relationship type.

The cardinalities in the EER schema are denoted by lower and uperbounds indicating the participation of the entity type in the relationship.

Next, we concentrate on the additional concepts like the partioning of `feature`. In our example, we have the constructed entity types `point_feature`, `line_feature`, and `areal_feature` as a specialization of `feature`. Since, `feature` is here a partition into `point_feature`, `line_feature`, and `areal_feature`, these three specialized feature types are disjoint, which means that a `feature` is either a `point_feature` or a `line_feature` or an `areal_feature`, but not any two of them at the same time.

Object-valued attributes allow us to specify the attribute `head` of `databasis` with `header` as attribute domain.

Apart from these structural schema features we have further restricted the allowed database states by integrity constraints. We do not mention them here because our central target language, the ODMG schema language, does unluckily not provide an explicit concept for such constraints.

4. From the EER Schema to an ODMG Schema

4.1. Object Schema

After modeling an Extended Entity-Relationship schema, it is desirable to present a conversion into an object schema, in particular an ODMG object schema. Thereby we capture structural and behavioural characteristics of an application in a uniform manner and facilitate the following implementation. The advantage of specifying an ODMG schema is its independence of the underlying OODBS and the presence of a global framework defining the main aspects of an object-oriented standard model. The transformation into an ODMG schema facilitates the implementation in a concrete OODBS.

Object types of an ODMG object schema are specified with the Object Definition Language ODL [Cat96]. In the following, we briefly describe the constructs of the object type interface specification. Besides type characteristics such as inheritance, the interface specification defines instance properties and operations.

Type characteristics are properties which are common to all instances of an object type. Such characteristics include supertype information and extent naming. The indication of supertypes enables the inheritance of structure and behaviour of the supertypes. The extent of an object type gives access to the set of all current instances of the type.

Instance properties of an object type are the attributes and relationships of its instances. The attributes describe the abstract state of a type. For each instance of an object type there exist specific values for each of its defined attributes. The ODMG object model supports data-valued and object-valued attributes. Object-valued attribute enable objects to reference other ones.

Relationships define links between instances of object types. In contrast to the EER model allowing arbitrary n-ary relationships, the ODMG object model supports only binary relationships which are defined between at most two types. N-ary relationships cannot be modeled directly, and therefore must be worked out as binary relationships. In our example however, we could do all modeling tasks with simple binary relationships.

A relationship is defined implicitly by the declaration of traversal paths which enable applications to use the logical connections between the objects participating in the relationship. The definition of the traversal path includes designation of the target type, ordering information, and information about the inverse traversal path, i.e. the name of the relationship found in the target type. Cardinality information is included in the specification of the target type. Cardinality greater than one is realized by use of a collection type (Set, Bag, List) in order to specify unordered or ordered members on the target side [Cat96].

Besides the structural part of object types which is defined by attributes and relationships, the behaviour of types is specified by operations. Operations are defined by signatures which define the name of an operation, the name and type of each parameter, and the types of values to be returned.

4.2. Transformation

After this short introduction to the essential constructs of the ODMG object model, let us consider our application example as shown in Fig. 3. Each entity type of the EER schema is transformed into an object type of the ODMG schema. Data-valued attributes can be transformed immediately without further change providing the existence of implementations for the non-standard data types by C++ classes. Only for object-valued attributes, we have to decide whether to represent them as object-valued attributes or as relationships between this object type and the type of the attribute. Both representations are possible in the ODMG schema. Since it is undesirable to have more than one possible construct in the target model for explicit constructs in the starting model, we constrain the transformation of object-valued attributes to equivalent object-valued attributes in the ODMG schema in order to support understandability of the transformation.

```
interface Feature
 ( extent features )
 { attribute Short id;
   attribute List<String> name;
   relationship Set<Point> consists_of
                inverse Point :: composes;
   void search_next ();
   void search_prev ();};
```

Figure 3. ODMG Object Type Description for Part of the EER Schema

Relationship types of the EER schema are integrated as traversal paths in the relationship specification of those object types participating in the relationship. In general, special attention must be paid to the transformation of n-ary relationships in the EER schema which must be represented by binary relationships modeling each respective link of the n-ary relationship. Such relationships however do not occur in our case.

As relationship types in EER schemata are usually concentrated to a single name describing the relationship and since we have neglected role names for the participating entity types in our EER schema, it is necessary to find adequate names for the traversal paths in the relationship specification (including a name for the inverse relationship). In the ODMG model, relationships itself are not named, but the traversal paths are. In this sense, the names for the traversal paths could be considered as rolenames of the participating object types. It seems to be adequate to adopt the name of a relationship type as one traversal pathname in the relationship specification of the respective object type. But we are are aware of the fact this proceeding this way, the name of a relationship type converts to a traversal pathname. The respective inverse traversal pathname results in a corresponding naming convention of the inverse relationship.

Let us have a view at the transformation of the relationship `consists_of` into corresponding traversal paths between the participating types `Feature` and `Point` in

Fig. 2. The relationship type `consists_of` converts to two traversal paths between the object types `Feature` and `Point`. For example, a feature `consists_of` points and a point `composes` a feature. This means, the direction from `Feature` to `Point` preserves the name of the relationship type, `consists_of`, and the inverse direction from `Point` to `Feature` requires a new naming, namely `composes`. The fact that both traversal paths apply to the same relationship type in the EER schema is indicated by the `inverse`-clause in both of the traversal path declarations.

The generalization and specialization construct in the EER model, also called type constructions in [Gog94], is realized in the inheritance specification of the ODMG model. Therefore, all supertypes of an object type are listed in its interface header in order to allow inheritance. Each supertype must be specified in its own type definition. Considering our case study, this means that the specification of the object types `Point_Feature`, `Line_Feature`, and `Areal_Feature` corresponding to the entity types `point_feature`, `line_feature` and `areal_feature` will be completed by the indication of its supertype `Feature`.

In addition to the steps of the transformation process, there is a need to specify the extension of an object type in the extent specification. Usually, we adopt for the extent naming the same name as for the object type but in plural form and starting with a lower case letter.

Since an ODMG schema enables the modeling of behaviour in terms of operations, an object schema can be completed by adding operation signatures to each object type. Concerning our case study, we add the operations `search_next()` and `search_prev()` to the object type `Feature` for demonstration purposes. These operations will be inherited by the specialized object types `Point_Feature`, `Line_Feature` and `Areal_Feature`.

In Fig. 3 we have shown the transformation by only describing the object type `Feature`. The rest of the transformation can be done anologously to this.

5. Related Work

Since the advent of object-oriented database systems, there have been several efforts to present transformations from this approach to the (Extended) Entity-Relationship model. Nearly all of them describe transformation rules in order to be capable to combine the (Extended) Entity-Relationship model with a concrete object-oriented model. More specifically, there exists a series of works relating the Entity-Relationship model with an object-oriented model, see [Kil91], [NCB91], [NNJ93], [MGG95], and [Fon95]. In order to bridge the gap between Entity-Relationship modeling and object-oriented programming, [Kil91] proposes the category type as a new mechanism and a possibility to describe relationships between types. In [NCB91], the definition of an object-oriented database schema is proposed starting from an Entity-Relationship model and integrating the data by presenting a set of rules. A similar approach is presented in [NNJ93] where a set of mapping rules for the transformation of Entity-Relationship schemas into object-oriented schemas is introduced. More recently, [MGG95] describe a transformation from an EER schema into an OO schema by using a clustered form of the EER schema in order to improve understandability and to reduce complexity of a conceptual schema. [Fon95] proposes a series of transformation steps from an EER model to an OMT (Object Modeling Technique) model for the purpose of object-oriented database design and re-engineering purposes.

Mappings from conventional schemas into ER schemas are treated in [JK89], [NA91], and [TYF86]. The former two articles propose methodologies for translating relational schemas into conceptual schemas and extended ER schemas, resp. In the latter article, a methodology in the other direction from ER modeling to relational database design is described. [CSGS94] introduce a general approach for semantically enriching relational databases and mapping them into the BLOOM (BarceLona Object Oriented Model).

Another approach is described in [BMP94] where an extension of the ER model is transformed into F-Logic [KLW93] which enables the subsequent generic transformation into the object-oriented database system ONTOS.

A further area concerning transformations is re-engineering with the goal of migrating from traditional to object-oriented databases. In this field, [FV95] have extensively studied formal transformations in both directions between an extended ER model and conventional (relational, hierarchical) models and object-oriented models, especially the ODMG model. In combining reverse and forward database engineering, [PH95] proposes the development of a legacy system in terms of three different views: a structural, functional, and behavioural view. A more global, database management system independant methodology for database reverse engineering is presented in [HTJC93] where a two-phase process of (1) data structure extraction and conceptualization and (2) standard schema transformations is proposed.

Newer approaches rely on the integration of several databases in order to produce federated views in a homogeneous and consistent manner as in [HP97], [BFN94]. To achieve consistent federations, a series of transformations relying on the ODMG object model are needed there.

6. Conclusion

Starting from a given data description in a low-level format, we have developed a schema for the same data in an object-oriented database. We have done this development in several well-defined steps with well-defined intermediate documents providing the opportunity for alternative design decisions. These intermediate schemata allowed for re-use of development steps when an alternative implementation platform was introduced.

Thus, we have succesfully applied classic techniques known from the software engineering field for re-engineering object-oriented databases. We have benefitted mainly also from the ODMG schema language ODL which provided an intermediate, implementation-independent layer in the development of the different schemata.

References

[BFN94] R. Busse, P. Fankhauser, and E. Neuhold. Federated Schemata in ODMG. In *Proc. 2nd East/West Database Workshop 1994*, 1994.

[BMP94] J. Biskup, R. Menzel, and T. Polle. Transforming an Entity-Relationship Schema into Object-Oriented Database Schema. Technical Report 17-94, Informatik-Bericht, Uni Hildesheim, 1994.

[Cat96] R.G.G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Francisco, 1996.

[CBS94] 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, 1994.

[CSGS94] M. Castellanos, F. Saltor, and M. García-Solaco. Semantically Enriching Relational Databases into an Object-Oriented Semantic Model. In *DEXA '94*, LNCS 856, pages 125–134. Springer-Verlag, 1994.

[DA87] K.H. Davis and A.K. Arora. Converting a Relational Database Model into an Entity-Relationship Model. In S. March, editor, *6th Int. Conf. on Entity-Relationship Approach*, 1987.

[Fon95] J. Fong. Mapping Extended Entity Relationship Model to Object Modeling Technique. *Sigmod Record*, 24(3):18–22, 1995.

[FV95] C. Fahrner and G. Vossen. Transformation relationaler Datenbank-Schemas in objektorientierte Schemas gemäß ODMG-93. In *BTW*. Springer, 1995.

[Gog94] M. Gogolla. *An Extended Entity-Relationship Model.Fundamentals and Pragmatics*. Springer-Verlag, Berlin, Heidelberg, 1994.

[HP97] U. Hohenstein and V. Plesser. A Generative Approach to Database Federation. In D.W. Embley and R.C. Goldstein, editors, *Int. Conf. on Conceptual Modeling*, LNCS 1331, pages 422–435. Springer, 1997.

[HTJC93] J.-L. Hainault, C. Tonneau, M. Joris, and M. Chandelon. Schema Transformation Techniques for Database Reverse Engineering. In R.A. Elmasri, V. Kouramajian, and Bernhard Thalheim, editors, *Proc. 12th Int. Conf. on ER-Approach*, LNCS 823, pages 364–375, Berlin, 1993. Springer.

[JK89] P. Johannesson and K. Kalman. A Mehtod for Translating Relational Schemas into Conceptual Schemas. In F.H. Lochovsky, editor, *Proc. 8th Int. Conf. on ER-Approach*, pages 271–285, 1989.

[Kil91] M.F. Kilian. Bridging the Gap between O-O and E-R. In T.J. Teorey, editor, *Proc. 10th Int. Conf. on ER-Approach*, pages 445–458, 1991.

[KLW93] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. Technical Report 93/06, Department of Computer Science, SUNY at Stony Brook, 1993.

[MGG95] R. Missaoui, J.-M. Gagnon, and R. Godin. Mapping an Extended Entity-Relationship Schema into a Schema of Complex Objects. In M.P. Papazoglou, editor, *14th Int. Conf. OOER*, LNCS 1021, pages 204–215, Berlin, 1995. Springer.

[NA91] S.B. Navathe and A.M. Awong. Abstracting Relational and Hierarchical Data with a Semantic Data Model. In *Proc. 6th Int. Conf. on ER-Approach*, pages 459–474, 1991.

[NCB91] J. Nachouki, M.P. Chastang, and H. Briand. From Entity-Relationship Diagram to Object-Oriented Database. In T.J. Teorey, editor, *Proc. 10th Int. Conf. ER-Approach*, pages 459–474, 1991.

[NNJ93] B. Narasimham, S.B. Navathe, and S. Jayaramam. On Mapping ER and Relational Models into OO Schemas. In R.A. Elmasri, V. Kouramajiam, and B. Thalheim, editors, *Proc. 12th Int. Conf. on ER-Approach*, pages 402–413, Arlington, Texas, 1993.

[PH95] G. Pernul and H. Hasenauer. Combining reverse with forward database engineering–a step forward to solve the legacy system dilemma. In *DEXA '95*, LNCS 978. Springer-Verlag, 1995.

[TYF86] T. Teorey, D. Yang, and J. Frey. A Logical Design Methodology for Relational Databases Using the EER Model. *ACM Transaction on Database Systems*, 18(2):197–220, 1986.