Proc. 12th Int. Conf. on Entity-Relationship Approach (ER'93)
R. Elmasri, V. Kouramajian, B. Thalheim (Eds.), Springer, Berlin, LNCS 823, pp. 376–389 (1994)

Integrating the ER Approach in an OO Environment*

M. Gogolla, R. Herzig, S. Conrad, G. Denker, and N. Vlachantonis
Technische Universität Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D-38023 Braunschweig, Germany
e-mail: gogolla@idb.cs.tu-bs.de

Abstract

We translate Entity-Relationship (ER) schemas into the object-oriented specification language TROLL *light*. This language describes the Universe of Discourse (UoD) as a system of concurrently existing and interacting objects, i.e., an object community. Thereby two essential aspects, structure and behavior, are integrated in one formalism. By doing the translation from ER to TROLL *light* we preserve the visual advantages of the former and receive a formalism through the latter which can be mapped to an adequate object-oriented database system. Proceeding this way we hope our proposal for transforming ER schemas into TROLL *light* specifications provides a valuable link between structural and dynamic modeling.

1 Introduction

Nowadays the Entity-Relationship approach [Che76] has been accepted as a quasi standard [Teo90, BCN92] for the conceptual design of databases and information systems. This approach is undoubtedly a very popular tool for bridging the gap between database designers and users of database systems. One issue in favor of Entity-Relationship design is based on the fact that the graphical representation serves as an intuitively understandable platform for both database designers and users of database systems so that communication between them becomes easier. Another reason for the wide acceptance and application of this approach is due to the fact that an Entity-Relationship schema can be mapped in a systematic way to a relational database schema (see [EWH85, MMR86, TYF86, Hoh89], among others). But most approaches to Entity-Relationship modeling do not deal deeply with dynamic aspects, because the Entity-Relationship approach is used for modeling static structure though it ought not to be separated from the behavioral one. There are temporal Entity-Relationship extensions [KS91b, KS91a, HH91, RS91, Tau91] which add dynamic

*Work reported here has been partially supported by the CEC under Grant No. 6112 (COMPASS) and BMFT under Grant No. 01 IS 203 D (KorSo).

aspects to the Entity-Relationship approach, but most of them are not directed to object-oriented databases.

Recently the advent of object-oriented based technology calls for and demands database design approaches and tools resulting in object-oriented database schemas. The main difficulties in tackling this problem arise from the lack of a common model and widely accepted principles. However, one advantage of object-oriented databases is that they cover both aspects, structure and behavior, and propose in this way an adequate formalism to describe the so-called Universe of Discourse (UoD), i.e., the part of the real world to be modelled. In this framework TROLL *light* [CGH92, GCH93] was developed as an object-oriented specification language describing the UoD as a system of concurrently existing and interacting objects, i.e., an object community. TROLL *light* is a dialect of TROLL [JSHS91] and OBLOG [SSE87]. Our language follows the decision that it is better to provide a modest number of basic constructs which are clear and easily applicable than to offer a lot of redundant language features.

But because of the textual representation of TROLL *light*, the language is in some respect less suitable to support the needs of a variety of users, like database designers and system users, in view of intuitive comprehensibility. Nevertheless TROLL *light* is closer to object-oriented database systems since it provides a lot of language features capable of expressing behavioral aspects of systems. Our aim is to break the traditional way of translating Entity-Relationship schemas to relational databases. We translate Entity-Relationship schemas to TROLL *light*, preserving the visual advantages of the former and receiving a formalism through the latter which can be mapped to an adequate object-oriented database system like this has been done in [NCB91, Tar92]. After the structural design with the Entity-Relationship approach we suggest an object-oriented mapping to get on the way to an object-oriented schema.

Let us explain the TROLL *light* approach in more detail. Our specification language comes along with an integrated, open development environment [VHG+93]. The task of this environment is to give support for the creation of correct information systems on the basis of formal specifications. Some important ingredients of the environment are pictured in Figure 1. Rectangles (with upper left corner cut off) represent documents which are generated during the development process; ovals stand for tools which are invoked by the user, and arrows denote the main flow of information. We now look at the single tools item by item.

The **data type interpreter** supports the development of data type specifications. It allows to test the data type specification at hand and to validate that the specification meets the application requirements. The TROLL *light* **editor** is understood to be a syntax-directed tool which also already checks context-sensitive conditions a TROLL *light* specification has to fulfill. The TROLL *light* **animator** is designed to simulate the specified object community. It provides object windows which enables the user for instance to observe attributes, to initiate state transitions by clicking events or to formulate queries against object states. By this the informal view of the real-world fragment to be modeled is validated against the current specification. The TROLL *light* **prover** is given verification tasks by the user. It checks whether the desired properties are fulfilled by the specified TROLL *light* templates. The TROLL *light* **transformer** takes the specification of an object community and interactively generates executable code in an object-oriented programming language

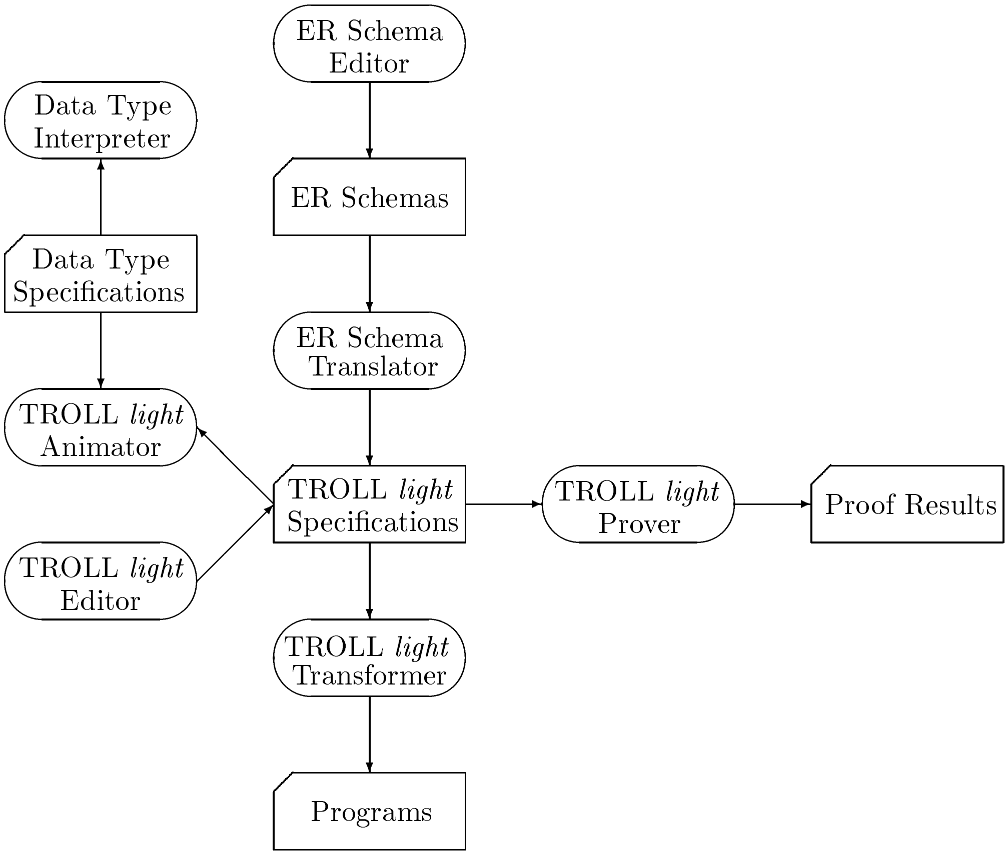

Figure 1: Some Ingredients of the TROLL *light* Development Environment.

supported by an object-oriented database system. The ER **schema editor** allows to design database schemas interactively. It fully supports all graphical features of the ER model and thus utilizes all advantages of the ER approach. Schemas obtained in this way can automatically be translated into TROLL *light* specifications by the ER **schema translator**. The advantage of this intermediate step lies in the fact that the obtained TROLL *light* descriptions can be processed with the other tools in the development environment like the TROLL *light* animator or the TROLL *light* prover.

Currently we are mainly working on the animator and the prover. But our environment is designed as an open environment. This means that new tools can be added. Nevertheless our environment is integrated in the sense that all tools have a common user interface and that they communicate with each other.

The main purpose of the present paper is to work out the basic ideas behind the ER schema translator. The remainder of the paper is organized as follows. In Section 2 we will give a short introduction to the main concepts of TROLL *light*. Afterwards we will reflect on various constructs of the Entity-Relationship approach and its extensions and show how they can be expressed in TROLL *light*. Our translation reveals that ER modelling and object-orientation can go hand in hand and support each other. Finally we give our estimation of the applicability of this proposal and its result on methodical decisions.

2 Concepts of TROLL *light*

2.1 Overview

TROLL *light* is a language for describing structural and dynamic properties of objects. With respect to the description of structural properties TROLL *light* offers concepts of semantic data models. **Attributes** describe observable properties of objects. Simple attributes are data-valued or object-valued. Over and above that complex attributes are obtainable by the application of predefined sort constructors like `SET` or `TUPLE`. TROLL *light* objects are organized into object hierarchies induced by **sub-object relationships**. Thereby a sub-object relationship must be understood as an exclusive "part-of" relationship. Normally attribute values are directly affected by event occurrences. Besides that derived attributes can be specified the state of which depends on other stored or **derived** information. In order to state derivation rules TROLL *light* incorporates an SQL-like query calculus. The query calculus of TROLL *light* also allows to specify static integrity **constraints**.

The description of dynamic properties is based on the specification of events. Events are abstractions of state-modifying operations on objects. Events on objects are described by a finite set of **event generators**. Each event generator may be accompanied by a list of parameter sorts. An event generator together with actual parameter values denotes an event. The effect of events on attributes is described by **valuation rules**. Events in different objects can be synchronized by **interaction rules**. Possible **event sequences** can be restricted to admissible ones by the means of CSP-like process descriptions.

All in all a template has the following structure:

```
TEMPLATE
  DATA TYPES     declaration of data types used in current template
  TEMPLATES      declaration of templates used in current template
  SUBOBJECTS     sub-object relationships
  ATTRIBUTES     observable properties
  EVENTS         event generators
  CONSTRAINTS    static integrity conditions
  VALUATION      effect of event occurrences on attributes
  DERIVATION     derivation rules for derived attributes
  INTERACTION    synchronization of events in different objects
  BEHAVIOR       description of admissible life cycles
END TEMPLATE
```

In the next section we show by a small example how objects in an information system may be specified by writing templates.

2.2 Examples

Let us assume that an information system is to be described which shall contain data about authors. For every author the name, the date of birth, and the number of books sold by year have to be stored. An author may change the name only once in the life.

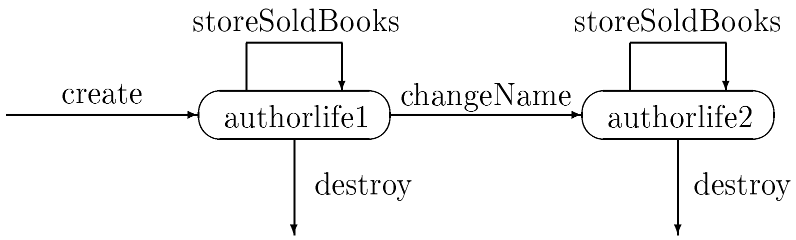

Figure 2: Behavior of Authors

An appropriate TROLL *light* specification would be:

```
TEMPLATE Author
  DATA TYPES   String, Date, Nat;
  ATTRIBUTES   Name:string; DateOfBirth:date;
               SoldBooks(Year:nat):nat;
  EVENTS       BIRTH create(Name:string, DateOfBirth:date);
                     changeName(NewName:string);
                     storeSoldBooks(Year:nat, Number:nat);
               DEATH destroy;
  VALUATION    [create(N,D)] Name=N, DateOfBirth=D;
               [changeName(N)] Name=N;
               [storeSoldBooks(Y,NR)] SoldBooks(Y)=NR;
  BEHAVIOR     PROCESS authorlife1 =
                 ( storeSoldBooks -> authorlife1 |
                   changeName -> authorlife2 |
                   destroy -> POSTMORTEM );
               PROCESS authorlife2 =
                 ( storeSoldBooks -> authorlife2 |
                   destroy -> POSTMORTEM );
               ( create -> authorlife1 );
END TEMPLATE;
```

Process descriptions as used in the `BEHAVIOR` section can be transformed to event-driven sequential machines (see Figure 2). These machines will be called *o-machines* (*o* for object) in the sequel. The states of the o-machine denote behavior states of objects.

We may assume that the information system will be populated by lots of authors so that these authors must be managed by a higher-level object. To this end usually classes are introduced in many object-oriented approaches. There is some confusion concerning the notion of class in object-oriented programming and object-oriented databases. A class is often used to cover two aspects, namely an *object factory* and an *object container* [Ban88]. In TROLL *light* object factories appear as templates while object containers are merely understood as composite objects, and composite objects have to be described by templates again. The following template `Authorclass` for such composite objects provides sub-object names for local identification of author objects.

```
TEMPLATE Authorclass
```

```
    DATA TYPES   String, Date, Nat;
    TEMPLATES    Author;
    SUBOBJECTS   Authors(No:nat):author;
    ATTRIBUTES   DERIVED NumberOfAuthors:nat;
    EVENTS       BIRTH create;
                       addAuthor(No:nat, Name:string,
                                 DateOfBirth:date);
                       removeAuthor(No:nat);
                 DEATH destroy;
    DERIVATION   NumberOfAuthors=CNT(Authors);
    INTERACTION  addAuthor(N,S,D) >> Authors(N).create(S,D);
                 removeAuthor(N)  >> Authors(N).destroy;
  END TEMPLATE;
```

Every template implies a corresponding object sort. We use names starting with an upper case letter to denote templates whereas the same name starting with the corresponding lower case letter is used to denote the corresponding object sort.

An object of sort `authorclass` will hold many author objects as private sub-objects (or components). In this example a natural number serves as a key for the identification of sub-objects. In contrast to many other data models, in TROLL *light* such keys need not be related to attributes of objects they identify. Adding an object to an author class means to create a new sub-object, and removing an object from an author class means to destroy the sub-object.

An object of sort `authorclass` may be sub-object of a higher object again, and so on. For example, a template `Libraryworld` may be specified having a sub-object of sort `authorclass` as sub-object beside other sub-objects like a book index and several libraries. Each of these libraries may have a collection of documents as sub-objects which refer to the books in the common book index (for more details see [CGH92, GCH93]).

As already mentioned it is possible to specify explicit integrity constraints within TROLL *light* templates. For this TROLL *light* contains a query calculus which is an offspring of a calculus for extended Entity-relationship schemas [HG88, GH91]. Besides static integrity constraints preconditions for valuations and for event occurrences can also be given by means of this calculus.

Let us consider an example for an integrity constraint. If we want to restrict the authors belonging to the same `authorclass` object to have together not more than 50,000 books sold in 1993, this can be expressed as follows:

```
  50,000 < SUM ( SELECT SoldBooks(PRJ2(A),1993)
                 FROM   A IN Authors
                 WHERE  DEF(SoldBooks(PRJ2(A),1993)) )
```

This constraint must be inserted into the `CONSTRAINTS` section of the `Authorclass` template. For understanding this formula some remarks are necessary. For parameterized sub-object and attribute names we assume the sub-object resp. attribute symbol without any parameters to stand for a set of tuples consisting of parameter values and a sub-object resp. an attribute value. Only those tuples are members in that set for which the sub-object exists resp. for which the attribute value is currently defined. For instance, `Authors` stands for a term of sort `SET(TUPLE(nat,author))`. In consequence, the variable `A` which becomes bound to `Authors` in the `FROM` clause is of sort `TUPLE(nat,author)`. Therefore, we use `PRJ2(A)` to project onto the second component, i.e., the `author` component, out of such a tuple. If we want to refer to attributes of other objects we have to add a parameter denoting the object the attribute of which we are interested in. In the given constraint we want to know the value of the attribute `SoldBook(1993)` of some sub-object. Because `PRJ2(A)` is evaluated to some `author` object we may use this term as additional parameter in `SoldBooks(PRJ2(A),1993)`.

Finally we note that our query calculus is safe because we require variables to be bound to finite domains, i.e., to stored information. Thereby, universal and existential quantifiers may be used as well. In [CGH92, GCH93] TROLL *light* is presented in more detail.

3 Transforming ER Schemas into TROLL *light* Templates

3.1 Translation of Basic ER schemas

Basic ER schemas according to Chen [Che76] are given by a set of entity types $\mathcal{E} = \{E_1, \ldots, E_p\}$ and a set of relationship types $\mathcal{R} = \{R_1(E_{1,1}, \ldots, E_{1,n_1}), \ldots, R_q(E_{q,1}, \ldots, E_{q,n_q})\}$ such that $E_{i,j} \in \mathcal{E}$. Both entity types and relationship types can be associated with data-valued attributes.

We propose the following general *transformation scheme*.

1. An entity type E (or a relationship type R) with attributes $A_1{:}d_1, \ldots, A_n{:}d_n$ is mapped to the following template.

```
TEMPLATE E/R
  DATA TYPES  ...;
  ATTRIBUTES  A1:d1, ..., An:dn;
  EVENTS      BIRTH create;
                    updateAi(di);
              DEATH destroy;
  VALUATION   [updateAi(D)] Ai=D;
END TEMPLATE;
```

   Here we assume that immediately after the creation of an object all attributes take undefined values. Of course this can be avoided by providing the birth event `create` with appropriate parameters.

2. ER schemas are viewed as objects in TROLL *light* for which the following object description can be given.

```
TEMPLATE ERSchema
  DATA TYPES  Nat, ...;
  TEMPLATES   E_1,...,E_p, R_1,...,R_q;
  SUBOBJECTS  E_1s(nat):e_1, ..., E_ps(nat):e_p,
              R_1s(e_{1,1},...,e_{1,n_1}):r_1, ..., R_qs(e_{q,1},...,e_{q,n_q}):r_q;
  EVENTS      BIRTH create;
                    addE_j(nat);                 addR_k(e_{k,1},...,e_{k,n_k});
                    updateE_jA_{j,i}(nat,d_{j,i}); updateR_kA_{k,i}(e_{k,1},...,e_{k,n_k},d_{k,i});
                    removeE_j(nat);              removeR_k(e_{k,1},...,e_{k,n_k});
              DEATH destroy;
  INTERACTION addE_j(N)            >> E_js(N).create;
              updateE_jA_{j,i}(N,D) >> E_js(N).updateA_{j,i}(D);
              removeE_j(N)         >> E_js(N).destroy;
              addR_k(E_1,...,E_{n_k})              >> R_ks(E_1,...,E_{n_k}).create;
              updateR_kA_{k,i}(E_1,...,E_{n_k},D) >> R_ks(E_1,...,E_{n_k}).updateA_{k,i}(D);
              removeR_k(E_1,...,E_{n_k})           >> R_ks(E_1,...,E_{n_k}).destroy;
END TEMPLATE;
```

Note that there is a special birth event for an ER schema which can be interpreted as the installation of an ER object base, and a death event denoting the shutdown of this base.

The transformation scheme has been given under the following premises. (1) On schema level entities are identified by a natural number whereas relationships are identified by the participating entities. (2) Since the behavior of objects is not subject to ER modeling in the narrow sense, object dynamics can only be reflected by canonical updates concerning the insertion and deletion of entities or relationships and attributes updates. (3) Since we assume all users to be confronted with the schema object all operations concerning the insertion and deletion of entities or relationships and attribute updates can be triggered from the schema level.

Of course, other choices could be taken as well. (1) Other keys are possible for the identification of entities, for example by referring to attributes of the objects in mind (see key concept below). (2) Proposals for the modeling of object behavior can be reflected by non-generic events as shown in Section 2. (3) Event triggering can be restricted to object level.

Another point of choice concerns the particular representation of relationships. Instead of representing relationships by objects, sometimes it seems to be more convenient to map relationship types to attributes. This is discussed in the next subsection.

3.2 Alternative Representations of Relationship Types

The representation of a relationship as an *object* is especially suited when the relationship shows distinct properties of a higher-level object, for example by having attributes which can be updated. Especially in cases where a relationship type has no attributes, the representation as an object may produce unnecessary overhead. In this case the representation of relationships as *values* may be indicated. Here value representation means that object relationships are expressed by *object-valued attributes*.

These attributes can either be stated at schema level or entity level. First we give an example for modeling relationships at *schema level*.

Assume that the attribute `Salary` of `CE` is dropped. Then the following TROLL *light* specification of the example ER schema has the same "information capacity" as the TROLL *light* specification given before.

```
TEMPLATE ERSchema
  ...
  SUBOBJECTS  Companies(nat):company;
              Employees(nat):employee;
  ATTRIBUTES  CEs:SET(TUPLE(company, employee));
  ...
  VALUATION   [create]        CEs=EMPTY;
              [addCE(C,E)]    CEs=ADD(CEs, (C,E));
              [removeCE(C,E)] CEs=REMOVE(CEs, (C,E));
  ...
END TEMPLATE;
```

Since `CEs` is now modeled as an attribute the corresponding interaction rules have been replaced by valuation rules. Note that the sort expression `SET(TUPLE(company, employee))` corresponds to a *relation* in the relational data model. Indeed, the representation of relationships given by this example resembles the way relationships are represented in a relational database.

Instead of interrelating objects at schema level object-valued attributes can only be used at *entity level*. This representation is particularly recommended for functional relationships.

Assume that an employee may only work for one company at a time. Then this relationship can adequately be described by an object-valued attribute of `Employee`.

```
TEMPLATE Employee
  ...
  TEMPLATES   Company;
  ATTRIBUTES  ...; worksFor:company;
  EVENTS      ...; updateCompany(company);
  VALUATION   ...; [updateCompany(D)] worksFor=D;
END TEMPLATE;
```

Object-valued attributes at entity level may also be used for representing N:M relationships. Then the single-valued attribute `worksFor` has to be replaced by a multi-valued attribute. The inverse relationship employs : Company $\rightarrow$ set(Employee) can be expressed by an inverse attribute. Interestingly object-oriented database management systems like ObjectStore [LLOW91] offer support for attributes and their inverses to implement N:M relationships.

3.3 Extended ER Schemas

There have been many proposals in extending ER schemas into the direction of general semantic data models [dSNF80, ABLV83, EWH85, EKTW86, Tha90, WT91, EGH$^+$92, PS92]. In contrast to the original ER approach these models mostly support object-valued attributes, complex attributes, derived attributes, constraints, keys, component relations, and ISA relations.

Object-valued attributes: In the original ER model attributes are restricted to data domains. It has been argued that this restriction is important from an ER design methodology point of view since it justifies the conceptual difference between entity and relationship types. On the other hand from a conceptual modeling point of view functional relationships between objects can be modeled more naturally by object-valued attributes. Hence many semantic data models do already support the modeling of object-valued attributes. Object-valued attributes are supported in TROLL *light* by referring to object sorts in the specification of attributes.

Complex attributes: Many non-standard applications, for example the modeling of multi-media objects, require non-standard data domains. In principle, there are two possible ways for the integration of such domains into an object model: first by referring to arbitrary user-defined *abstract data types*, second by providing *sort constructors* which directly allow the specification of multi-valued or composite attributes. Both ways are supported by TROLL *light*.

Derived attributes: Derived attributes are attributes the values of which depend on other stored or derived information. The specification of derived attributes is directly supported in TROLL *light* by the `DERIVATION` section of a template.

Constraints: The query calculus of TROLL *light* can also be used to describe a wide range of static integrity constraints on object states (e.g. cardinality constraints) in the `CONSTRAINTS` section of a template.

Key concept: In principle, the particular identity of an entity is abstract, i.e., it should be invisible to the user [KC86]. But suppose that an attribute of a certain object is to be updated. Then without a specific notion of object identity at hand a user can only describe the object in mind associatively through some related objects or values. However there will never be a guarantee that this description can be unique.

In order to facilitate object access, many models offer a key concept providing abstract objects with an additional logical identity. In its simplest form some attributes of an object are marked as keys. Here it is important to note that in the framework of TROLL *light* the specification of keys does not belong to object descriptions in the narrow sense since keys are irrelevant from the viewpoint of a single object. Keys get only important when many objects of a given sort are to be distinguished, for example by a higher-level object such as a schema. So in many cases needs for the simple identification of similar objects led to new attributes. Take for example a person's name which turns out to be more artificial than a person's color of eyes.

In the generic TROLL *light* description of an arbitrary ER schema we simply used numbers for the identification of entities, but this is not obligatory. Specific attributes of the objects which are to be identified can be used as keys as well.

Component relations: It has been recognized that component (or part) relations constitute an extremely useful modeling primitive [HGP92]. Component relations can be classified either to be *exclusive* or *shared*. Second there can be a *dependency* from the part to the whole or vice-versa.

Following the ER paradigm component relations are represented by normal relationship types, but to reduce semantic overloading separate modeling primitives may be advantageous. In TROLL *light* exclusive part relations, where the part is furthermore dependent on the whole, can be easily described by *vertical structuring* (hierarchical composition of object communities).

```
TEMPLATE Car
  TEMPLATES   Engine;
  SUBOBJECTS  Motor:engine;
  EVENTS      ...; getMotor; dropMotor; ...;
  INTERACTION getMotor  >> Motor.create;
              dropMotor >> Motor.destroy; ...;
END TEMPLATE;
```

Some semantic data models like GSM [HK87] or IFO [AH87] support the modeling of *higher-order object types* like *aggregation types* (a motor-boat is composed of a motor and a hull) or *grouping types* (a convoy is given by a set of vehicles). In many ways higher-order object types resemble component relations. Indeed, they can be viewed as special component relations where the whole depends on the parts. In this way relationship types can be viewed as special higher-order object types too. Consequently the representation of higher-order object types in TROLL *light* follows the representation of relationship types.

```
TEMPLATE ERschema
  TEMPLATES   MotorBoat, Motor, Boat ...;
  SUBOBJECTS  MotorBoats(motor, boat):motorBoat; ...;
  EVENTS      ...;
              addMotorBoat(motor,boat);
              removeMotorBoat(motor,boat);
              ...;
  INTERACTION addMotorBoat(M,B) >> MotorBoats(M,B).create;
              removeMotorBoat(M,B) >> MotorBoats(M,B).destroy; ...;
END TEMPLATE;
```

ISA relations: The specialization or ISA relation has been proved to be a fundamental concept of semantic data models [HK87]. This relation allows to model aspects of a given object, for example the role student of person. In the original ER model ISA relations would be represented by relationship types, but again to reduce semantic overloading additional modeling primitives show to be helpful.

With regard to a TROLL *light* representation it is important to observe that aspects show a similar dependency on their origin as the dependency found in part relations. Hence vertical structuring of TROLL *light* might be also used to represent aspects.

```
TEMPLATE Person
  TEMPLATES   Student;
  SUBOBJECTS  RoleStudent:student;
  EVENTS      ...; becomeStudent; ceaseToBeStudent; ...;
  INTERACTION becomeStudent    >> RoleStudent.create;
              ceaseToBeStudent >> RoleStudent.destroy; ...;
```

```
END TEMPLATE;
```

Remark: Both component and ISA relations may be accompanied by the propagation of attributes and events. This is called *inheritance*. However, it is observable that the particular direction in which attributes or events should be propagated often depends on the concrete application. In TROLL *light* the propagation of attributes must be made explicit by the specification of derived attributes, whereas the propagation of events is specified by interaction rules. The advantage of this additional effort can be seen in the increased flexibility of this approach.

4 Conclusions

The specification of large information systems with TROLL *light* requires in the first development phase a quite complete and detailed overview of the UoD. Experience shows that information system developers do not have such an overview at the beginning of the development. For overcoming this problem we have selected the ER model as a means for creating a first *structural description* of the UoD.

Our proposal for transforming ER schemas into TROLL *light* specifications provides an important link between structural and dynamic modeling. It is quite amazing to see how nice the concepts of the ER model can be formulated in an object-oriented specification language. The translation shows that ER modeling and object-orientation can assist each other.

Nevertheless it will be necessary to integrate the transformation of ER schemas into TROLL *light* also methodically into the whole software development process. In our opinion this can be achieved because the proposed transformation ideas give enough freedom for also including functional and even non-functional requirements into the transformation process which results in the possibility of generating alternative solutions.

References

[ABLV83] P. Atzeni, C. Batini, M. Lenzerini, and F. Villanelli. INCOD: A System for Conceptual Design of Data and Transactions in the Entity-Relationship Model. In: [Che83], pp. 375–410, 1983.

[AH87] S. Abiteboul and R. Hull. IFO — A Formal Semantic Database Model. *ACM Trans. on Database Systems*, 12(4):525–565, 1987.

[Ban88] F. Bancilhon. Object-Oriented Database Systems. In *Proc. 7th ACM Symp. Principles of Database Systems*, pages 152–1.562, 1988.

[BCN92] C. Batini, S. Ceri, and S.B. Navathe. Conceptual Database Design – An Entity-Relationship Approach. Benjamin-Cummings, Redwood City (CA), 1992.

[CGH92] S. Conrad, M. Gogolla, and R. Herzig. TROLL *light*: A Core Language for Specifying Objects. Informatik-Bericht 92–02, Technische Universität Braunschweig, 1992.

[Che76] P.P. Chen. The Entity-Relationship Model – Towards a Unified View of Data. *ACM Trans. on Database Systems*, 1(1):9–36, 1976.

[Che80] P.P. Chen, editor. *Proc. 1st Int. Conf. on Entity-Relationship Approach to Systems Analysis and Design (1979)*. North-Holland, Amsterdam, 1980.

[Che83] P.P. Chen, editor. *Proc. 2nd Int. Conf. on Entity-Relationship Approach to Information Modelling and Analysis (1981)*. North-Holland, Amsterdam, 1983.

[dSNF80] C.S. dos Santos, E.J. Neuhold, and A.L. Furtado. A Data Type Approach to the Entity-Relationship Approach. In: [Che80], pp. 103–1.519, 1980.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering, North-Holland*, 9(2):157–204, 1992.

[EKTW86] J. Eder, G. Kappel, A M. Tjoa, and R.R. Wagner. BIER: The Behaviour Integrated Entity-Relationship Approach. In: [Spa87], pp. 147–1.566, 1986.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The Category Concept: An Extension to the Entity-Relationship Model. *Data & Knowledge Engineering*, 1:75–1.516, 1985.

[GCH93] M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL *light*. In A. Miola, editor, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO'93)*, pages 17–32. Springer, Berlin, LNCS 722, 1993.

[GH91] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Trans. on Database Systems*, 16(3):369–416, 1991.

[HG88] U. Hohenstein and M. Gogolla. A Calculus for an Extended Entity-Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions. In C. Batini, editor, *Proc. 7th Int. Conf. on the Entity-Relationship Approach*, pages 129–1.548. North-Holland, Amsterdam, 1988.

[HGP92] M. Halper, J. Geller, and Y. Perl. "Part" Relations for Object-Oriented Databases. In G. Pernul and A M. Tjoa, editors, *Proc. 11th Int. Conf. on Entity-Relationship Approach, Karlsruhe (Germany)*, pages 406–422. Springer, LNCS 645, 1992.

[HH91] U. Hohenstein and K. Hülsmann. A Language for Specifying Static and Dynamic Integrity Constraints. In T.J. Teorey, editor, *Proc. 10th Int. Conf. on the ER-approach*, pages 389–416, San Mateo, 1991.

[HK87] R. Hull and R. King. Semantic Database Modelling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[Hoh89] U. Hohenstein. Automatic Transformation of an Entity-Relationship Query Language into SQL. In F. Lochovski, editor, *Proc. 8th Int. Conf. on the Entity-Relationship Approach*, pages 309–327, Toronto, 1989.

[JSHS91] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-Oriented Specification of Information Systems: The TROLL Language. Informatik-Bericht 91-04, TU Braunschweig, 1991.

[KC86] S.N. Khoshafian and G.P. Copeland. Object Identity. *ACM SIGPLAN Notices*, 21(11):406–416, 1986. Proc. OOPSLA.

[KS91a] G. Kappel and M. Schrefl. Object/Behavior Diagrams. In *Proc. 7th Int. Conf. on Data Engineering, Kobe (Japan)*, pages 530–539, 1991.

[KS91b] G. Kappel and M. Schrefl. Using an Object-Oriented Diagram Technique for the Design of Information Systems. In H.G. Sol and K.M. Van Hee, editors, *Dynamic Modelling of Information Systems*, pages 121–1.564. North-Holland, Amsterdam, 1991.

[LLOW91] C. Lamb, G. Landis, J. Orenstein, and D. Weinreib. The ObjectStore Database System. *Communications of the ACM*, 34(10):50–63, 1991.

[MMR86] J.A. Makowski, V.M. Makowski, and N. Rotics. Entity-Relationship Consistency for Relational Schemes. In G. Ausiello and P. Atzeni, editors, *Proc. Int. Conf. Database Theory (ICDT'86)*, pages 306–322. Springer, Berlin, LNCS 243, 1986.

[NCB91] J. Nachouki, M.P. Chastang, and H. Briand. From entity-relationship diagram to an object-oriented database. In T.J. Teorey, editor, *Proc. 10th Int. Conf. on Entity-Relationship Approach, San Mateo (California)*, pages 459–482, 1991.

[PS92] C. Parent and S. Spaccapietra. ECR+: An Object-Based Entity-Relationship Approach. In P. Loucopoulos and R. Zicari, editors, *Concepual Modeling, Databases, and CASE: An Integrated View of Information Systems Development*, pages 69–86. John Wiley & Sons, Inc., 1992.

[RS91] E. Rose and A. Segev. TOODM – A Temporal Object-Oriented Data Model with Temporal Constraints. In T.J. Teorey, editor, *Proc. Entity-Relationship Approach (ER'91)*, pages 205–230, 1991.

[Spa87] S. Spaccapietra, editor. *Proc. 5th Int. Conf. on Entity-Relationship Approach: Ten Years of Experience in Information Modeling (1986)*. North-Holland, Amsterdam, 1987.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P.M. Stoecker and W. Kent, editors, *Proc. 13th Int. Conf. on Very Large Databases VLDB'87*, pages 107–1.516. VLDB Endowment Press, Saratoga (CA), 1987.

[Tar92] Z. Tari. On the Design of Object-Oriented Databases. In G. Pernul and A M. Tjoa, editors, *Proc. 11th Int. Conf. on Entity-Relationship Approach, Karlsruhe (Germany)*, pages 389–405. Springer, LNCS 645, 1992.

[Tau91] B. Tauzovich. Towards Temporal Extensions to the Entity-Relationship Model. In T.J. Teorey, editor, *Proc. Entity-Relationship Approach (ER'91)*, pages 163–1.580, 1991.

[Teo90] T.J. Teorey. Database Modeling and Design – The Entity-Relationship Approach. Morgan Kaufmann, San Mateo (CA), 1990.

[Tha90] B. Thalheim. Extending the Entity-Relationship Model for a High-Level, Theory-Based Database Design. In: Proc. 1st Int. East-West Database Workshop, Next Generation Information System Technology, J.W. Schmidt, A.A. Stagny (Eds.), Springer, Berlin, LNCS 504, pp. 161–1.584, 1990.

[TYF86] T.J. Teorey, D. Yang, and J.P. Fry. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys*, 18(2):197–222, 1986.

[VHG+93] N. Vlachantonis, R. Herzig, M. Gogolla, G. Denker, S. Conrad, and H.-D. Ehrich. Towards Reliable Information Systems: The KORSO Approach. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proc. 5th Int. Conf. on Advanced Information Systems Engineering (CAiSE'93)*, pages 463–482. Springer, Berlin, LNCS 685, 1993.

[WT91] G. Wei and T.J. Teorey. The ORAC Model: A Unified View of Data Abstractions. In T.J. Teorey, editor, *Proc. 10th Int. Conf. on Entity-Relationship Approach, San Mateo (California)*, pages 31–58, 1991.