Visual Specification of Complex Database Actions

Gregor Engels
Leiden University, Dept. of Computer Science
P.O. Box 9512, NL-2300 RA Leiden
The Netherlands
engels@wi.leidenuniv.nl

Perdita Löhr
Quantum GmbH
Emil-Figge-Straße 83, D-44227 Dortmund
Germany
loehr@quantum.de

Abstract

The paper presents an approach to specify in an integrated way by visual, diagrammatic languages the structural and behavioural aspects of database applications. Hereby, well-known extended Entity-Relationship diagrams are employed to specify the structural aspects. The behavioural aspects of a database application are specified by using ViAL (Visual Action Language). ViAL specifications are a special kind of data (or better object) flow diagrams, where so-called elementary actions are used as basic building blocks. These elementary actions are automatically derived from a given EER diagram. They guarantee that after finishing their execution all inherent integrity constraints are fulfilled. The paper explains the features of the language ViAL and gives some illustrating examples.

1 Introduction

Nowadays, semantic data models are regarded as convenient means to specify database applications [11]. Most of these approaches offer corresponding visual specification languages which eases both to develop and to comprehend the specification. The most well-known representatives of such languages are languages based on the ER model and its extensions. They are widely used in scientific as well as commercial software development. Meanwhile, the ER approach can even be found in commercial products like ORACLE CASE, as well as in fully extended software development methods like Modern SA [17]. Additionally, the ER concepts impacted the development of object oriented approaches which partially adopted them for their structural specification parts, e.g., OMT [14] or OOA [3].

Since database applications besides structural parts tend to more and more encompass dynamic parts, suitable languages are needed to specify the application's behaviour and functionality. Those languages can be termed suitable if they fulfill the following two requirements:

1. Developped behaviour specifications are highly integrated with the structure specification, i.e., they obey the inherent integrity constraints imposed by the structure specification.

2. They are on the same language level as the language to define the structural part and are also visual, diagrammatic languages.

Usually, the classical approach to function specification, i.e., data flow diagrams, is added to the structure specification language. Unfortunately, this approach do not fulfill the first requirement. Thus, the well-known integration gap between conventional data and function specification arises.

An alternative approach to integrate data and function specification can be found in ACM/PCM [1]. There, the graphically specified data structures are also used to express data flow. For this purpose, the database schema is extended by basic operations referring to the object types. Since it is not possible to specify more elaborated control flow in these extended schemata, most of the remaining functionality has to be specified in a traditional, i.e., separate and textual, manner.

To our knowledge, such work on combining structural and behavioural modelling has not been continued in the semantic data modelling community. But, it has returned back as an interesting and important topic in the object-oriented modelling world. While industrial object-oriented methods like OMT [14] still lack a real integrated specification of structure and behaviour, some research results offer appropriate solutions. As an example, Petri net based approaches as for instance Object/Behaviour Diagrams [12] are mentioned here.

In this paper, we also propose an approach which offers a **visual specification language** to model actions **tightly bound** to the specification of a database schema, and thus fulfills the above mentioned two requirements. The language is called ViAL – Visual Action Language. It was developed during a DFG[1] project called CADDY (Computer-Aided Design of non-traditional Databases). In this context, ViAL is implemented as part of a prototype design environment for information systems [6]. In contrast to the above mentioned, non-integrated approaches ViAL focus on

- a specification of actions *highly integrated* with the database schemata. For this purpose, elementary actions are derived from the database schema and are automatically offered as graphic specification primitives in ViAL.

- *inclusion of arbitrary structured data* queried from the database. To support this facility, arbitrary queries can be separately defined and connected to a graphic symbol. In turn, this query symbol is offered in ViAL as a specification primitive. Furthermore, special constructs are available to handle the object sets delivered by the queries.

- an *intuitively comprehensible representation.* This is provided by using graphic symbols in ViAL which on the one hand, resemble the symbols of the used semantic data model, and on the other hand, adopt the well-known concepts of data flow diagrams.

[1] DFG is the abbreviation for the German Research Council - Deutsche Forschungsgemeinschaft

2 Specification of Conceptual Database Schemata

To show how to specify actions in terms of ViAL, we roughly introduce the specification of a database schema first, since ViAL's specification primitives base on it. For short: a specification of a database schema consists of three parts [5]: an extended Entity-Relationship (EER) diagram, definitions of application oriented data types and and definitions of static integrity constraints.

2.1 EER diagram

An EER diagram describes the object types of the database application and their relationships. It is an extension of the ER model by Chen [2] and enhances it by the well-known concepts of generalization/specialization, association, and aggregation. Figure 1 gives an EER diagram which models a simple aspect of flight reservation. You can see in this example schema that the basic constructs equal those of the classical ER model. Few concepts are denoted slightly different, for example, key attributes of an object type are marked by a black dot on the connecting edge, while optional attributes are marked by an hollow dot. A new construct, for example, is the type constructor which is denoted as a triangle in the schema. It describes the specialization of persons into the special object type PASSENGER. Attributes may be data-valued like Name or object-valued like AdrSet. Object-valued attributes (or components) are dependent on the existence of their owners. Futhermore, attributes may be multivalued like set, list, or bag, as, for instance, AdrSet. Further details on this data model can be found in [5].

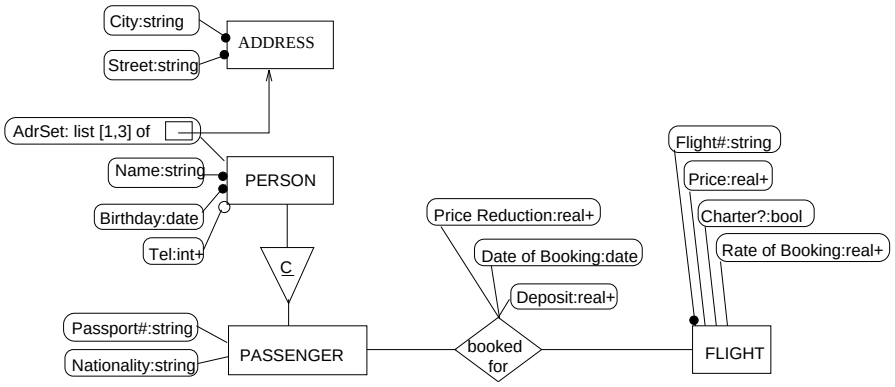


Figure 1: Example EER digram refering to booking of flights

2.2 Application-oriented Data Types

Data types are specified by using an algebraic approach. This allows to introduce application-oriented data types like `point, date, colour` besides the

standard types like `int, real` into a specification, together with type specific operations.

2.3 Static Integrity Constraints and Queries

Static integrity constraints are expressed as formulas of an EER calculus. This calculus was especially developed for and adapted to the needs of an EER language. It allows to define variables and predicates over EER specific constructs [8].

On top of the EER calculus, an EER oriented query language is defined: SQL/EER [10]. This language facilitates to query the database on the same level as the defined EER diagram. An example of an SQL/EER query is:

```
SELECT pa.Name, (   SELECT f.Flight#
                    FROM f IN FLIGHT
                    WHERE pa booked_for f )
FROM pa in PASSENGER
WHERE pa.Nationality = "Dutch"
```

This query selects for all Dutch passengers their names and the flight numbers they are booked for.

3 ViAL - Concepts and Syntax

Having the specification of the database schema on hand, we now introduce the visual language ViAL [4, 7, 9, 16]. ViAL enables to graphically specify complex (trans-)actions corresponding to the existing database schema. ViAL specifications are interactively executable by an interpreter, and, thus, offer the possiblity to test the specification of structure and behaviour of a database application in a prototype manner. For this purpose, ViAL offers the following concepts:

ViAL comprehends a set of **graphic symbols to express functionality**: the processing symbols. This set may be distinguished into two main groups of symbols: on the one hand, general symbols which include arbitrary actions or queries (cf. figure 3 and 4), and on the other hand, specific symbols which are derived from the existing database schema. The latter are shown in figure 2: rectangular symbols denoting operations on objects, diamond-shaped symbols for operations on relationships, and triangular symbols which represent the dynamic specialization/generalization of objects in type hierarchies.

Specifying an action in ViAL means to construct it from the **basic stock** of graphic symbols available. This basic stock encompasses the above standard operations for each object type, each relationship type and each specialization/generalization which are all automatically derived from the existing database schema. They are called **elementary actions** and are offered in ViAL like a construction kit with which to start the specification (cf. figure 2).

Each elementary action is (internally) composed of basic actions. Basic actions modify exactly one database object. There are basic actions

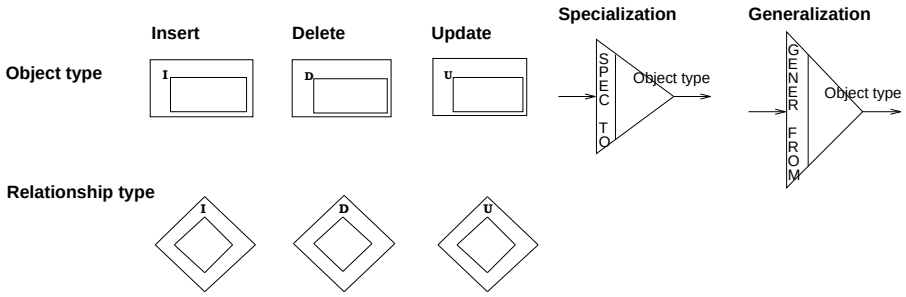


Figure 2: Graphical representation of elementary actions

- to insert or delete an instance of an object or relationship type
- to add or remove a component of an object
- to add or delete the membership of an object in a certain type construction
- to update attribute values of an existing object.

All these basic actions can automatically be derived from a given EER diagram. An important feature of elementary actions is that after the execution of an elementary action the database results in a consistent database state. Here consistency means that all EER diagram inherent integrity constraints are fulfilled.

Let us illustrate an elementary action by the example of the insertion of a new passenger. The automatically generated realization, here given in pseudo-code, is as follows:

```
elem_insert_PASSENGER (Name : string, Birthday : date) : PASSENGER
BEGIN
        /* check whether PASSENGER already exists */
        SELECT p
        FROM p IN PASSENGER
        WHERE p.Name = Name AND p.Birthday = Birthday

        IF p EXISTS
        THEN error
        ELSE /* check whether PERSON already exists */
                SELECT p
                FROM p IN PERSON
                WHERE p.Name = Name AND p.Birthday = Birthday

                IF p NOT EXISTS
                THEN /* insert new PERSON */
                        p := basic_insert_PERSON( Name, Birthday )
```

```
                basic_add_comp_AdrSet( p )
            END IF;

            /* specialize PERSON to PASSENGER */
            basic_specialize_PASSENGER( p )
        END IF;
END elem_insert_PASSENGER;
```

The example shows that several basic actions, known as **update propagations** [15], are needed to yield a new consistent database state. Elementary actions describe minimal sequences of basic actions starting and resulting in a consistent database state [4]. All concrete data values have to be provided by the user during the interative execution of an elementary action. These are, for instance, the key values mentioned in the parameter list of this elementary action, as well as, for instance, concrete values for Passport# and Nationality, which have to be given by the user during the execution of basic_specialize_PASSENGER.

Besides the elementary actions, the basic stock includes so-called **existential queries**. Likewise, these queries are automatically generated for each object type (cf. figure 3). The issues of such a query is to check whether a particular object, determined by its key values, exists or not. If it exists, the query delivers, depending on the kind of edge (see below), either the object itself or a positive signal (along the edge marked by a black dot), otherwise a negative signal is sent (along the edge marked by a hollow dot).

These elements of the basic stock are provided for a ViAL specification from the beginning. This proceeding guarantees that all composed actions are highly integrated with the according database schema and obey all inherent integrity constraints.

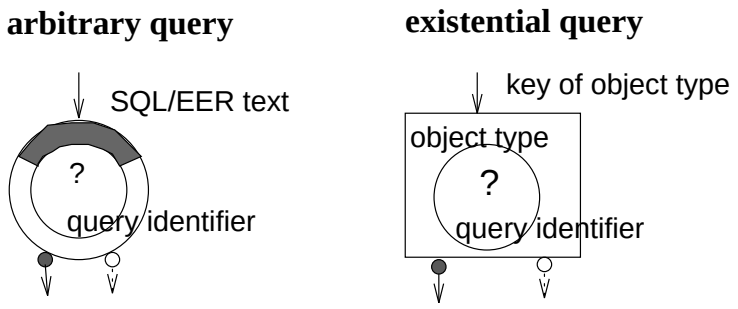


Figure 3: Graphical representation of queries

Furthermore, ViAL allows to include **arbitrary queries separately defined in SQL/EER** into an action specification. For this purpose, a particular symbol is offered which hides the textual representation of the query and may be used in a ViAL specification (cf. figure 3). The symbol for each such query is included in the basic stock of graphic ViAL symbols and, thus, available

for each specification of an action. Such queries enable to process arbitrary combinations of data from the database and to incorporate them into a ViAL action. This proceeding supports one of the typical tasks of a database centered information system: processing of arbitrary data collections.

ViAL offers an **easy to handle abstraction concept**. Actions under development are identified by a name introduced by a declaration symbol (cf. figure 4). Such already specified (or even incompletely specified) actions can be (re)used in a currently developed action like calling a procedure in a conventional programming language (cf. figure 4). Like arbitrary queries, these procedure declaration and invocation symbols are included in the basic stock of graphic elements.

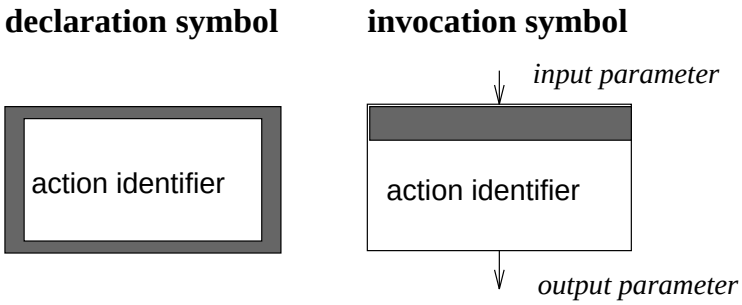


Figure 4: Graphical representation of procedures

To specify data flow and to connect the processing symbols, ViAL offers **data flow edges**. Three different types of such edges are available (cf. figure 5).

Via *object flow edges* (single or double arrow) objects or sets of objects flow through the specified processes. *Signal edges* handle boolean values, for example, the information whether an existential query has found an object or not. *Edges for error handling* connect an action with a standard error handling procedure in order to save a developer from tedious error handling details at the specification level.

ViAL supports **multivalue oriented data processing**. For this purpose, the double arrowed object edges are offered (cf. figure 5). Multivalued data are usually delivered by queries. From these, the data sets flow via the double arrowed object edges to the processing symbols. To handle these data sets conveniently, the processing symbols may be marked with an "asterisk" which means that they iteratively execute the data sets. The iteration need not be explicitly defined since the processing symbols are provided with an implicit cursor concept similar to those known from the combination of programming and database languages. Every element of figures 2 and 3 can be used in this context.

ViAL offers basic constructs to **control** the data flow (cf. figure 5).

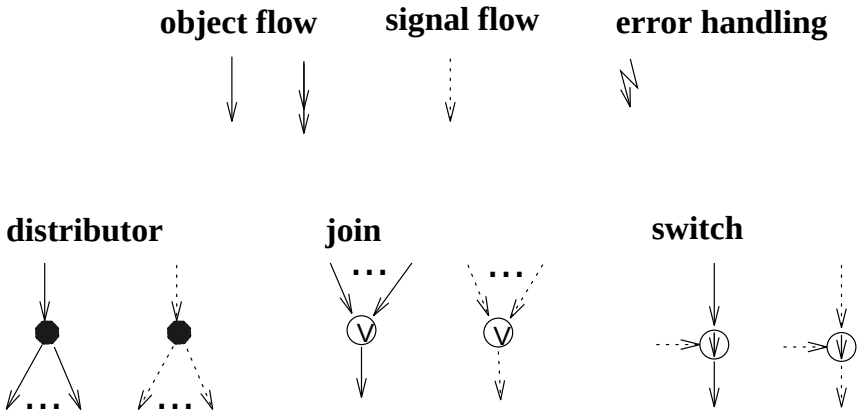


Figure 5: Data flow and control

The *distributor* takes one ingoing object or signal flow and distributes it to each of the outgoing edges. This allows to distribute data to parallel branches within one action.

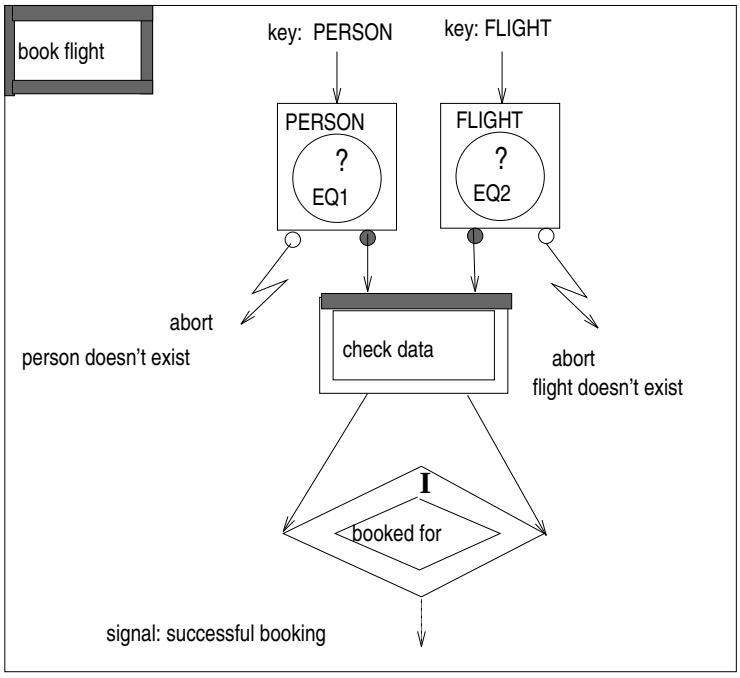
The *join operator* joins different branches of information flow. Only one of the ingoing edges is allowed to carry information (exclusive or) which is then delivered to the outgoing edge.

Switch represents an operator to constrain data flow. The ingoing information (from the top of the operator) is delivered to the outgoing edge only if a constraint is satisfied represented by the left hand signal edge.

4 Example

Let us illustrate how a (more or less) complex action like `booking of a flight` can be specified using the ViAL construction kit (cf. figure 6). The action given in the figure refers to the introduced EER diagram in figure 1.

The complex action `booking of a flight` is constructed from two existential queries EQ_1 and EQ_2, another complex action `check data`, and the elementary action `elem_insert_booked_for`. Its input parameters are the `key:PERSON` and `key:FLIGHT`. At the beginning of the action, the two existential queries EQ_1 and EQ_2 ask interactively for the values of the key attributes of a PERSON and a FLIGHT. If the specified objects exist, the action continues, otherwise it stops, switching to the standard error handling. In case of continuation, one object for PERSON and one object for FLIGHT is delivered by the corresponding existential queries to the complex action `check data`. This action itself has to be specified as a ViAL action which is shown in the lower part of the ViAL diagram.



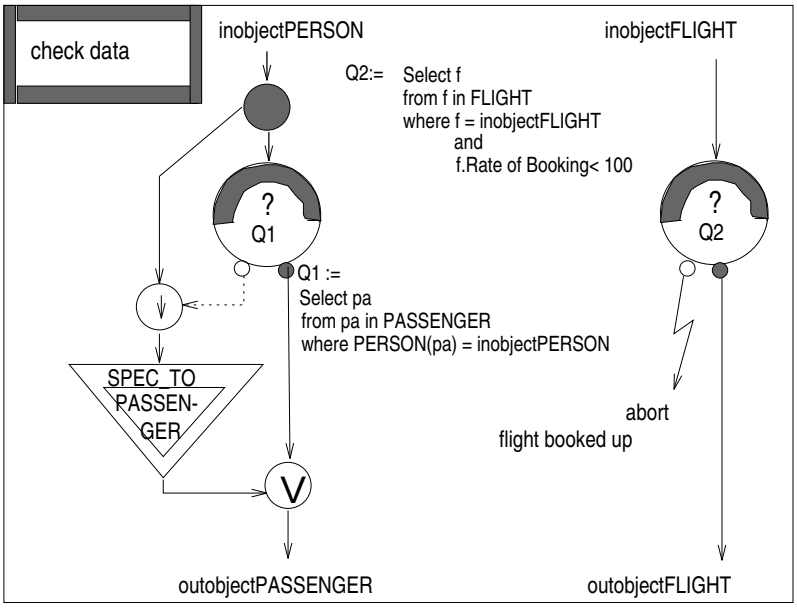


Figure 6: Example action `book flight`

`Check data` only starts to execute if every ingoing edge carries values. In such case, it checks whether the incoming object values are valid for the following action parts. This can be done in parallel. On the left hand side, the query Q_1 checks whether the delivered `PERSON` object already exists as a `PASSENGER` object. If not (outgoing negative signal edge of Q_1), the `PERSON` object is specialized to the required `PASSENGER` object, otherwise the `PASSENGER` object found by the former query Q_1 will be delivered. In parallel, the query Q_2 checks whether the flight corresponding to the specified `FLIGHT` object still has free seats. If not, is the flight booked up and the `booking of a flight` is aborted. Otherwise the `FLIGHT` object is delivered. The textual representation of the SQL/EER queries are only entered to support the comprehensibility of the diagram. In general, the text of a query is hidden and only shown to the user on request.

Then, `check data` delivers its outgoing objects to the elementary action `elem_insert_booked_for` which belongs to the relationship type `booked_for` of the EER diagram. To start its execution, the elementary action need both a `PASSENGER` object and a `FLIGHT` object in order to enter the corresponding relationship properly. If the booking action finished successfully, `booking of a flight` delivers a signal.

Even from this small example, it can be seen how the construction principle of ViAL works: elements of the basic stock like existential queries, user defined queries, and elementary actions may be chosen and combined to build a complex action. Other not yet defined actions like `check data` can be used as procedures. Since the primitive specification elements of the basic stock are automatically derived from the database schema, the integration and consistency between data and function specification is guaranteed. The graphic style of the language supports the understanding of the entire specification.

To gain experience, we began to use ViAL in small student projects at the university. In [13], ViAL was successfully employed to specify a complexer and larger application. By this practice, we learned that it is difficult to specify highly interactive applications with ViAL because no temporary variables exist which allow to hold information beside the data stored in the database. Positive experience was made with the concepts of procedural abstraction, the easy inclusion of database queries into the actions, and the convenient processing of multivalued data which in the whole lead to an easier specification of actions within a database application.

References

[1] M.L. Brodie and E. Silva. Active and Passive Component Modelling: ACM/PCM. In T. Olle, H.G. Sol, and A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: A Comparative Review, Proc. IFIP WG8.1 Working Conference, Nordwijkerhout*, pages 41–92. North-Holland, 1982.

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

[3] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[4] G. Engels. Elementary Actions on an Extended Entity-Relationship Database. In *Proc. 4th Int. Workshop on Graph Grammars and Their Application to Computer Science, LNCS 532*, pages 344–362. Springer, Berlin, 1991.

[5] 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*, 9(2):157–204, 1992.

[6] G. Engels and P. Löhr-Richter. CADDY: A Highly Integrated Environment to Support Conceptual Database Design. In G. Forte, N. Madhavji, and H. Müller, editors, *Proc. 5th Int. Workshop on Computer-Aided Software Engineering, Montreal, Kanada*, pages 19–22. IEEE Computer Society Press, 1992.

[7] K. Gerlach. *Ein Interpreter für visuell spezifizierte komplexe Aktionen auf EER-Datenbanken.* Technical University of Braunschweig, Master Thesis, 1992.

[8] M. Gogolla and U. Hohenstein. Towards a semantic view of an extended Entity-Relationship model. *ACM Transactions on Database Systems*, 16:369–416, 1991.

[9] C. Hennemann and J. Schacht. *Entwurf und Implementierung einer Sprache zur visuellen Spezifikation von Aktionen auf erweiterten Entity-Relationship Databanken.* Technical University of Braunschweig, Master Theses, 1991.

[10] U. Hohenstein and G. Engels. Formal semantics of Entity-Relationship-based query language. *Information Systems*, 17(3):209–242, 1992.

[11] R. Hull and R. King. Semantic Database Modeling: Survey, Applications, Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1989.

[12] G. Kappel and M. Schrefl. Object/Behaviour Diagrams. *Proc. 7th IEEE International Conference on Data Engineering*, Kobe, Japan, April 1991.

[13] P. Löhr-Richter. *Generische Methoden für die frühen Entwurfsphasen von Informationssystemen.* PhD thesis, Technical University of Braunschweig, 1993.

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[15] P. Scheuermann, G. Schiffner, H. Weber Abstraction Capabilities and Invariant Properties Modelling within the Entity-Relationship Approach. In P.P. Chen (ed.): *Proc. of the 1st Int. Conference on Entity-Relationship Approach*, Los Angeles (California), 121–140, 1980.

[16] M. Wolff. *Eine Sprache zur Beschreibung Schema-abhängiger Aktionen in einem erweiterten Entity-Relationship-Modell.* Technical University of Braunschweig, Master Thesis, 1989.

[17] E. Yourdon. *Modern Structured Analysis.* Prentice Hall, 1989.