Management of Data (COMAD'97)
D. J. Ram (Ed.), Narosa, New Delhi, 1997, pp. 238-249

Perspectives for a Formal Framework for Schema Modification in Object Databases

Heino Gärtner

Bremen University, Department for Mathematics and Computer Science,
Database Systems Group, PO Box 330440,
D-28334 Bremen, Germany, phone +49 421 218-4951, fax: +49 421 218-3054
gaertner@informatik.uni-bremen.de

Abstract

This paper proposes a formal basis for the description of schema modification in object databases. We identify three necessary building blocks for the structural aspect of such a description: (i) basic schema and database objects (ii) properties of (classes of) schema and database objects (iii) schema modifications, i.e. transitions between schemata and transitions between database states. The semantics for the basic schema and database objects is given in a denotational style. Properties of schemata can be described as formulas. For schema modification we introduce a set of *schema modification primitives* and describe their semantics by means of pre- and postcondition in a Hoare-style notation. We focus on the structural aspects of schema modification and describe some simple properties of our framework.

1 Introduction

Schema modification is an important means for adapting an information system in an evolving environment to changing requirements. Its study has impact on many database research areas as for example views in object databases [AB91, MP96, SLT91, Sou95, Wie91], schema evolution [Zic92] and (on object level) database versions [CJ90]. An information system has several components, which can be subject to change. Figure 1 shows a layered approach as suggested e.g. in [EGH+92] to model these components.

The *data type component* describes the stateless structural part of an information system. Thereby one can think of simple abstract data types as **integer** or **real** as well as of more complex ones as **point** or **address**. The *object layer* describes the state dependent structural part of an information system. Specification in this layer is usually performed by means of semantic data models

Evolution Component	**Action Component**
Object Component	
Data Type Component	

Figure 1: Layered Approach to Conceptual Modeling

[HK87, PM88], (enhanced) Entity-Relationship models [Che76, EN94, Gog94] or object oriented data models [ABD$^+$89, BM93, KS96]. The *evolution component* describes the behavioral aspects of objects in time. They can be described with temporal logic or finite state machines. The *action component* describes the implementation independent evaluation of database update operations. This can be done e.g. by means of pre- and postconditions. Finally one could add a *process component* as a higher level layer — as for example [Saa93, HS97] do — to describe long lived application processes and work-flows.

We see basically two lines of research: On the one hand there is a magnitude of formal models for the description of the single components (e.g. [Wir90] data type component, [Gog94] object component, [Har87] dynamic components) and on the other hand there are many approaches which deal with schema modification and database evolution [BKKK87, FMZ$^+$95, Odb95, PO97, SZ87, Zic92] .

However, only few approaches (as for example [PO97]) apply formal description techniques to schema modification. In this paper we try to integrate parts of both of these research directions by combining well-known formal description techniques in a schema modification framework.

The rest of the paper is structured as follows. Section 2 deals with our notion of an object data model. First we introduce the data-level with complex values. Then we procede by adding means for capturing structural schema information, i.e. object types, attributes, relationships, specialization relationships. After that we show, how to build terms and formulas with the introduced concepts. Section 3 deals with the modification of a schema which is done by so called *modification primitives.* In Section 4 we show some simple properties of our framework and conclude in Section 5 with some remarcs on the further developement of our approach.

2 The Object Data Model

Before presenting our object model we have to introduce a few conventions: By $\mathcal{M}$ we denote the class of functions, by $\mathcal{R}$ the class of relations and by $\mathcal{S}$ the class of sets. Assume sets $S, S_1, \ldots, S_n \in \mathcal{S}$ to be given. Then $\mathcal{F}(S)$ stands for the restriction of the powersets $\mathcal{P}(S)$ of S to finite sets, $\mathcal{B}(S)$ for the set of finite multisets (or bags) over S, S^* for the set of finite lists over S, S^+ for the set of finite non-empty lists over S, and $S_1 \times \ldots \times S_n$ for the cartesian product of the sets $S_1, \ldots, S_n$.

2.1 Data-level

The basic data type component can be introduced as known from algebraic specification [Wir90]. The data signature is the central notion of the data type component. The syntax of a signature is given by the names of the sorts and the associated operations and predicates.

Definition 2.1 (Signature) The Syntax of a Signature Σ_D is given by

1. a triple (S_D, Ω_D, Π_D),
2. a function source : $\Omega_D \to S_D^*$,
3. a function destination : $\Omega_D \to S_D$,

4. a function arguments : $\Pi_D \rightarrow S_D^+$

where S_D is a set of data sorts, $\Omega_D = \{\Omega_{D,\omega}\}_{\omega \in S_D^* \times S_D}$ a set of sort induced operations and $\Pi_D = \{\Pi_{D,\pi}\}_{\pi \in S_D^*}$. $\square$

Definition 2.2 (Interpretation of a Signature) The interpretation of a signature Σ_D is given by a triple $I(\Sigma_D) = (I(S_D), I(\Omega_D), I(\Pi_D))$ with:

- $I(S_D) : S_D \mapsto \mathcal{S}$ with $\perp \in I(S_D)(s)$ for all $s \in S_D$.
- $I(\Omega_D) : \Omega_D \mapsto \mathcal{M}$ with $\omega : s_1 \times \cdots \times s_n \mapsto s$ implies
  $I(\Omega_D)(\omega) : I(S_D)(s_1) \times \cdots \times I(S_D)(s_n) \mapsto I(S_D)(s)$ for all $\omega \in \Omega_D$.
- $I(\Pi_D) : \Pi_D \mapsto \mathcal{R}$ with $\pi : s_1 \times \cdots \times s_n$ implies
  $I(\Pi_D)(\pi) \subseteq I(S_D)(s_1) \times \cdots \times I(S_D)(s_n)$ for all $\pi \in \Pi_D$.

$\square$

On the basis of sorts like **real**, **integer** and **string** one can build more complex sorts as **point = tuple(real, real)** using sort expressions.

Definition 2.3 (Sort Expressions) Let a set of interpreted data sorts S be given. The syntax of sort expressions over S is given by a set $Expr(S)$ and determined by the following rules:

- If $s \in S$, then $s \in Expr(S)$
- If $s \in S$, then $list(s) \in Expr(S)$
- If $s \in S$, then $set(s) \in Expr(S)$
- If $s \in S$, then $bag(s) \in Expr(S)$
- If $s_1, \ldots, s_n \in S$ then $tupel(s_1, \ldots, s_n) \in Expr(S)$

$\square$

Definition 2.4 (Interpretation of a Sort Expression) The interpretation of a sort expression $s \in Expr(S)$ is given by a function $I(Expr(S)) : Expr(S) \mapsto \mathcal{S}$ and determined by the following rules:

- $I(s)$ as in definition 2.2.
- $I(list(s)) := (I(s))^* \cup \{\perp\}$.
- $I(set(s)) := \mathcal{F}(I(s) \cup \{\perp\}$.
- $I(bag(s)) := \mathcal{B}(I(s) \cup \{\perp\}$.
- $I(tupel(s_1, \ldots, s_n)) := (I(s_1) \times \ldots \times I(s_n)) \cup \{\perp\}$.

$\square$

2.2 Object-level

So far we have defined means for modeling the stateless structural part of an information system. In the next step we are going to introduce the basic schema components objects, relations, subtype relationships and attributes.

Definition 2.5 (Schema)

The syntax of a schema $\mathcal{Z} = (\mathcal{O}, \mathcal{R}, \mathcal{A}, \preceq)$ is given by a set $\mathcal{O}$ of object types, a set $\mathcal{R} \subseteq \mathcal{O}_1 \times \ldots \times \mathcal{O}_n$ of relations, a set $\mathcal{A} : \mathcal{O} \cup \mathcal{R} \to Expr(S)$ of attributes and a partial order $\preceq : \mathcal{O} \times \mathcal{O}$ on the object types yielding a subtype lattice i.e.:

1. $O \preceq O$ for all $O \in \mathcal{O}$
2. $O \preceq O'$, $O' \preceq O \Longrightarrow O = O$ for all $O, O' \in \mathcal{O}$
3. $O \preceq O'$, $O' \preceq O'' \Longrightarrow O \preceq O''$ for all $O, O', O'' \in \mathcal{O}$

$\square$

Definition 2.6 (Interpretation of a schema) The interpretation of a schema $I(\mathcal{Z}) = (I(\mathcal{O}), I(\mathcal{R}), I(\mathcal{A}), I(\preceq))$ is defined in the following way.

- $I(\mathcal{O}) : \mathcal{O} \to \mathcal{S}$ with $\perp \in I(\mathcal{O})(o)$ for all $o \in \mathcal{O}$
- $I(\mathcal{R}) : \mathcal{R} \to \mathcal{S}$ with $r(o_1, \ldots, o_n) \in \mathcal{R}$ implies $I(r) \subseteq (I(\mathcal{O})(o_1) - \{\perp\} \times \cdots \times I(\mathcal{O})(o_n) - \{\perp\}) \cup \{\perp\}$
- $I(\mathcal{A}) : A \to \mathcal{M}$, $A \in \mathcal{A}$ with
  - $A : O \to d$ with $o \in \mathcal{O}$, $d \in Expr(S)$ implies $I(\mathcal{A})(A) : I(O) \to I(d)$
  - $A : R \to d$ with $R \in \mathcal{R}$, $d \in Expr(S)$ implies $I(\mathcal{A})(R) : I(O) \to I(d)$
- $I(\preceq) : I(\mathcal{O}) \to I(\mathcal{O})$ with $I(\preceq)$ injective, total and $O_1 \preceq O_2$ implies $I(O_1) \to I(O_2) \in I(\preceq)$

$\square$

A simple example schema might have the object types **person** with attribute **name : string**, **employee** with attribute **salary : integer** and **company** with attributes **cname : string** and **locations: set(string)**. Also we might have a relation **works_for: employee** $\times$ **company**:

- $\mathcal{O} =$ {employee, person, company}, employee $\preceq$ person
- $\mathcal{R} =$ {works_for}
- $\mathcal{A} =$ {name, salary, cname, locations}

In an actual database state we might have the following instances:

- $I(\mathcal{O})(\text{employee}) = \{e_1, e_2, \ldots, e_n\} \cup \{\perp\}$, $I(\mathcal{O})(\text{person}) = \{p_1, p_2, \ldots, p_m\} \cup \{\perp\}$, $I(\mathcal{O})(\text{company})$ $= \{c_1, c_2, \ldots, c_k\} \cup \{\perp\}$
- $I(\mathcal{R})(\text{works\_for}) \subseteq (\{e_1, e_2, \ldots, e_n\} \times \{c_1, c_2, \ldots, c_k\}) \cup \{\perp\}$
- $I(\mathcal{A})(\text{name}) = \{p_1, p_2, \ldots, p_m\} \to I(S)(\text{string})$
  $I(\mathcal{A})(\text{name}) = p_1 \mapsto \text{`}Smith\text{'}, \ldots$
- ...

2.3 Terms & Formulas

Upon this basis we are now able to define terms and formulas over schemata. Before doing so, we extend our notion of signature in the following way: $\Sigma = (S, \Omega, \Pi)$ with

- $S := S \cup \mathcal{O} \cup \mathcal{R}$
- $\Omega := \Omega \cup \mathcal{A}$
- $\Pi := \Pi \cup \mathcal{R} \cup \preceq$

To build terms and formulas we first need to introduce variables and assignments.

Definition 2.7 (Variables and Assignment) Let a set S of interpreted sorts and a set of variables $\mathcal{V} = \{\mathcal{V}_s\}_{s \in S}$ be given. The set of assignments B is given by

$$B := \{\beta \mid \beta : \mathcal{V}_s \to I(s)\} \text{ for all } s \in S.$$

With $\epsilon : \mathcal{V} \to \{\perp\}$ stand for the empty assignment. □

Definition 2.8 (Terms) Let an interpreted signature $\Sigma = (S, \Omega, \Pi)$ be given. The syntax of terms is given by a sort induced set $\mathcal{T}_{s \in S}$, a function $free : \mathcal{T} \to \mathcal{F}(\mathcal{V})$, a function $type : \mathcal{T} \to S$ and defined in the following way:

- If $v \in \mathcal{V}_s$, then $v \in \mathcal{T}$, $type(v) := s$ and $free(v) = \{v\}$.
- If $\omega : s_1 \times \cdots \times s_n \mapsto s \quad \in \Omega$, $t_i \in \mathcal{T}$ and $type(t_i) = s_i$, then $\omega(t_1, \ldots, t_n) \in \mathcal{T}_s$ and $type(\omega(t_1, \ldots, t_n)) := s$ with
  $free(\omega(t_1, \ldots, t_n)) := free(t_1) \cup \ldots \cup free(t_n)$

□

Definition 2.9 (Interpretation of Terms) With a given interpretation function I and an assignment β the interpretation of terms $\mathcal{T}$ is defined as follows:

- $(I, \beta)(v) := \beta(v)$.

- $(I,\beta)(\omega(t_1,\ldots,t_n)) := I(\omega)((I,\beta)(t_1),\ldots,(I,\beta)(t_n))$.

$\square$

Declarations enable us to introduce quantifiers when defining formulas.

Definition 2.10 (Declarations) Let a variable v of sort S and a term t of sort $set(S)$ be given. The syntax of declarations is given by a set $\mathcal{D}$ with functions $free, bound : \mathcal{T} \mapsto \boldsymbol{\mathcal{F}}(\mathcal{V})$ and defined as follows:

- If $v \notin free(t)$, then $\boldsymbol{v : t} \in \mathcal{D}$, $free(\text{ v : t}) := free(t), bound(\ \boldsymbol{v : t}) := \{v\}$.
- If $d \in \mathcal{D}$ and $v \notin free(t) \ \cup \ free(d) \ \cup \ bound(d)$, then $\boldsymbol{v : t; d} \in \mathcal{D}$.

$\square$

Definition 2.11 (Interpretation of Declarations) The interpretation of a declaration is a relation $I(\mathcal{D}) = \mathcal{D} \times B$. It is defined in the following way:

- $(\boldsymbol{v : t}, \beta) \in I(\mathcal{D})$ iff $\beta(v) \in (I,\beta)(t)$.
- $(\boldsymbol{v : t; d}, \beta) \in I(\mathcal{D})$ iff $\beta(v) \in (I,\beta)(t)$ and $(\boldsymbol{d}, \beta) \in I(\mathcal{D})$.

$\square$

Definition 2.12 (Formulas) The syntax of formulas is given by a set $\mathcal{P}$ and a function $free : \mathcal{P} \to \mathcal{F}(\mathcal{V})$. It is defined as follows:

- $true, false \in \mathcal{P}, free(true) = free(false) := \emptyset$.
- $\neg p \in \mathcal{P}$
  if $p \in \mathcal{P}$; $free(\neg p) := free(p)$.
- $p_1 \wedge p_2,\ p_1 \vee p_2, p_1 \Rightarrow p_2 \in \mathcal{P}$
  if $p_1, p_2 \in \mathcal{P}$;
  $free(p_1 \wedge p_2) = free(p_1 \vee p_2) = free(p_1 \Rightarrow p_2) :=$
  $free(p_1) \cup free(p_2)$.
- $\exists\ d\ p \in \mathcal{P}$
  if $p \in \mathcal{P}, d \in \mathcal{D}$ and $bound(d) \subseteq free(p)$;
  $free(\exists\ d\ p) := free(d) \cup (free(p) - bound(d))$.
- $\forall\ d\ p \in \mathcal{P}$
  if $p \in \mathcal{P}, d \in \mathcal{D}$ and $bound(d) \subseteq free(p)$;
  $free(\forall\ d\ p) := free(d) \cup (free(p) - bound(d))$.

- $\pi(t_1, \ldots, t_n) \in \mathcal{P}$
  if $\pi : s_1 \times \ldots \times s_n \in \Pi, t_i \in \mathcal{T}, type(t_i) = s_i$;
  $free(\pi(t_1, \ldots, t_n)) := free(t_1) \cup \ldots \cup free(t_n)$.

- $t_1 = t_2 \in \mathcal{P}$
  if $type(t_1) = type(t_2)$;
  $free(t_1 = t_2) := free(t_1) \cup free(t_2)$.

A formula $p \in \mathcal{P}$ is called *closed*, if $free(p) = \emptyset$.

□

Definition 2.13 (Interpretation of Formulas) The interpretation of formulas $\mathcal{P}$ is given by a relation $I(\mathcal{P}) \subseteq \mathcal{P} \times B$ and defined as follows:

- $(true, \beta) \in I(\mathcal{P}), (false, \beta) \notin I(\mathcal{P})$.
- $(\neg p, \beta) \in I(\mathcal{P})$ iff $(p, \beta) \in I(\mathcal{P})$.
- $(p_1 \wedge p_2, \beta) \in I(\mathcal{P})$ iff $(p_1, \beta) \in I(\mathcal{P})$ and $(p_2, \beta) \in I(\mathcal{P})$.
- $(p_1 \vee p_2, \beta) \in I(\mathcal{P})$ iff $(p_1, \beta) \in I(\mathcal{P})$ or $(p_2, \beta) \in I(\mathcal{P})$.
- $(p_1 \rightarrow p_2, \beta) \in I(\mathcal{P})$ iff $(\neg p_1 \vee p_2, \beta) \in I(\mathcal{P})$.
- $(\exists\ d\ p, \beta) \in I(\mathcal{P})$ iff there exists a $\beta' \in B$ with $\beta'(v) = \beta(v)$ for all $v \in \mathcal{V} - bound(d)$, $(p, \beta') \in I(\mathcal{P})$ and $(d, \beta') \in \mathcal{D}$.
- $(\forall\ d\ p, \beta) \in I(\mathcal{P})$ iff $(\neg\exists\ d\ \neg p, \beta) \in I(\mathcal{P})$.
- $(\pi(t_1, \ldots, t_n), \beta) \in I(\mathcal{P})$ iff $((I, \beta)(t_1), \ldots, (I, \beta)(t_n)) \in I(\Pi)(\pi)$.
- $t_1 = t_2 \in I(\mathcal{P})$ iff $(I, \beta)(t_1) = (I, \beta)(t_2)$.

□

With the definitions given above, other set operations can be easily introduced by syntactic means, e.g. $x \in Y \equiv \exists v : Y\ v = x$ or $X \subseteq Y \equiv \forall\ v_1 : X\ \exists\ v_2 : Y\ v_1 = v_2$.

For our purpose we also need a simple form of substitution, which can be defined by the following two rules.

Definition 2.14 (Substitution) Let a set X of sort S be given. A substitution is given by

$[X_s]^a_b := \{b\} \cup \{x \in X | x \neq a\}$ if $\exists x \in X :\ x = a$ else $\{X_s\}$

$[X_s = Y_s]^a_b := [X_s]^a_b = [Y_s]^a_b$ □

3 Modification Primitives

Now we are prepared to describe transitions between schemata by means of modification primitives. We state the semantics of these primitives using Hoare-Triples [Bac86, GS94, LS87] i.e. by specifying pre- and postconditions. A modification primitive ω is of the general form

$$\begin{aligned} &\{precondition\} \\ \omega[\mathcal{I}]&(p_1, \ldots, p_n) \\ &\{postcondition\} \end{aligned}$$

Where $\mathcal{I}$ is an index and the p_i specify the actual parameters of the modification primitive. The definition of the modification primitives imposes a certain order of the operations. It is for example impossible to define attributes without having defined the respective object type. These restrictions could easily be relaxed but would lead to unnecessary broad interfaces. In our previous work [BGL97] we described modification primitives for adding, renaming, deleting and retyping schema elements and for selecting database elements in a semi-formal way. In this paper we focus on the coupling of the basic object model and the modification primitives and exemplify this coupling with the central primitives for adding and deleting schema elements.

3.1 Adding Elements

First we describe the introduction of new structural elements or relationships in a schema, which is done by the add primitive (α).

Object Types

Adding an object type is a rather simple operation. The only precondition we need is that the object type to be added is not already in the set of object types.

$$\begin{aligned} &\{\mathcal{O} = X' \wedge X \notin \mathcal{O}\} \\ \alpha[\mathcal{O}]&(X) \\ &\{\ [\mathcal{O} = X']^{X'}_{X' \cup X}\ \} \end{aligned}$$

Since $\mathcal{O}$ is a normal set and not a multiset, we could even drop the precondition $X \notin \mathcal{O}$.

Relationships

Adding a new relationship requires the participating object types to be already defined in the schema, which is expressed by the precondition.

$$\begin{aligned} &\{\mathcal{R} = X' \wedge X \notin \mathcal{R} \wedge \{O_1, \ldots, O_n\} \subseteq \mathcal{O}\} \\ \alpha[\mathcal{R}]&(X, \{O_1, \ldots, O_n\}) \\ &\{\ [\mathcal{R} = X']^{X'}_{X' \cup X}\ \} \end{aligned}$$

Attributes

Adding an attribute to a an object or relationship requires analogously the respective element to be defined in the schema.

$$
\begin{array}{l}
\quad \{\mathcal{A} = X' \wedge X \notin \mathcal{A} \wedge D \in Expr(S_D) \wedge E \in \mathcal{O} \cup \mathcal{R}\} \\
\alpha[\mathcal{A}](X, E, D) \\
\quad \{ \left[\mathcal{A} = X'\right]^{X'}_{X' \cup X} \}
\end{array}
$$

Subtype Relationships

To introduce a new subtype relationship we do not only require the participating object types to be defined, but we also have to ensure, that we do not introduce cyclic subtype relationships. This is stated in the last part of the precondition where we make use of the transitive nature of the partial order in our type lattice.

$$
\begin{array}{l}
\quad \{O_1 \in \mathcal{O} \wedge O_2 \in \mathcal{O} \wedge \neg \exists O : \mathcal{O} \; O \preceq O_1 \wedge O_2 \preceq O\} \\
\alpha[\preceq](O_1, O_2) \\
\quad \{O_1 \preceq O_2\}
\end{array}
$$

3.2 Deleting Elements

If structural elements are removed from an environment these operations do not only affect the schema but also the database part of the environment. These changes will be reflected by the respective interpretation functions in the pre- and postconditions.

Attributes

$$
\begin{array}{l}
\quad \{X \in \mathcal{A} \wedge \left[\mathcal{A} = X'\right]^{X'}_{X' \cup X} \wedge \left[I(\mathcal{A}) = X'_i\right]^{X'_i}_{X'_i \cup I(X)} \} \\
\bar{\pi}[\mathcal{A}](X) \\
\quad \{\mathcal{A} = X' \wedge I(\mathcal{A}) = X'_i\}
\end{array}
$$

Relationship Types

We require, the respective relationship type to have no attributes before being deleted.

$$
\begin{array}{l}
\quad \{X \in \mathcal{R} \wedge \left[\mathcal{R} = X'\right]^{X'}_{X' \cup X} \wedge \left[I(\mathcal{R}) = X'_i\right]^{X'_i}_{X'_i \cup I(X)} \wedge \neg \exists A : \mathcal{A} \; X \in source(A)\} \\
\bar{\pi}[\mathcal{R}](X) \\
\quad \{\mathcal{R} = X' \wedge I(\mathcal{R}) = X'_i\}
\end{array}
$$

Subtype Relationship

$$
\begin{array}{l}
\quad\{O_1 \in \mathcal{O} \wedge O_2 \in \mathcal{O} \wedge O_1 \preceq O_2\} \\
\bar{\pi}[\preceq](O_1, O_2) \\
\quad\{O_1 \not\preceq O_2\}
\end{array}
$$

Object Types

We require, that the object type has no attributes and that it does not stand in any (subtype) relationship to another object type.

$$
\begin{array}{l}
\quad\{X \in \mathcal{O} \wedge \left[\mathcal{O} = X'\right]^{X'}_{X' \cup X} \wedge \left[I(\mathcal{O}) = X'_i\right]^{X'_i}_{X'_i \cup I(X)} \\
\qquad \wedge \neg \exists A : \mathcal{A}\ X \in source(A) \wedge \neg \exists R : \mathcal{R}\ X \in arguments(R) \\
\qquad \wedge \neg \exists O : \mathcal{O}\ X \preceq O \vee O \preceq X\} \\
\bar{\pi}[\mathcal{O}](X) \\
\quad\{\mathcal{O} = X' \wedge I(\mathcal{O}) = X'_i\}
\end{array}
$$

4 Properties

In this section we will discuss some properties of our framework, as far as it is introduced in this paper.

The main advantage of our framework is that we get a solid means for the description of the properties of the single schema modification primitives. We can for example show, that the introduction of new object types is commutative, i.e.:

$$\alpha[\mathcal{O}](X);\ \alpha[\mathcal{O}](Y) \equiv \alpha[\mathcal{O}](Y);\ \alpha[\mathcal{O}](X)$$

$$
\begin{array}{l}
\quad\{\mathcal{O} = X' \wedge X \notin \mathcal{O}\} \\
\alpha[\mathcal{O}](X) \\
\quad\{\left[\mathcal{O} = X'\right]^{X'}_{X' \cup X}\} \\
\\
\quad\{\left[\mathcal{O} = X'\right]^{X'}_{X' \cup X} \wedge Y \notin \mathcal{O}\} \\
\alpha[\mathcal{O}](Y) \\
\quad\{\underbrace{\left[\left[\mathcal{O} = X'\right]^{X'}_{X' \cup X}\right]^{X'}_{X' \cup Y}}_{\mathcal{O} = X' \cup X \cup Y}\}
\end{array}
\qquad
\begin{array}{l}
\quad\{\mathcal{O} = X' \wedge Y \notin \mathcal{O}\} \\
\alpha[\mathcal{O}](Y) \\
\quad\{\left[\mathcal{O} = X'\right]^{X'}_{X' \cup Y}\} \\
\\
\quad\{\left[\mathcal{O} = X'\right]^{X'}_{X' \cup Y} \wedge X \notin \mathcal{O}\} \\
\alpha[\mathcal{O}](X) \\
\quad\{\underbrace{\left[\left[\mathcal{O} = X'\right]^{X'}_{X' \cup Y}\right]^{X'}_{X' \cup X}}_{\mathcal{O} = X' \cup Y \cup X \equiv \mathcal{O} = X' \cup X \cup Y}\}
\end{array}
$$

Thus we get a formal instrument for investigating properties of schema modifications.

A further example for the application of our framework is to use the modification primitives as a semantic base for higher level schema modification specification languages. Consider for example the following introduction of a new object type employee.

```
create object type employee
                   name    : string;
                   salary  : integer;
                   addr    : address;
end;
```

This specification can be expressed by:

$\alpha[\mathcal{O}](employee)$; $\alpha[\mathcal{A}](name, employee, string)$;

$\alpha[\mathcal{A}](salary, employee, integer)$; $\alpha[\mathcal{A}](addr, employee, address)$

5 Conclusion

We have proposed a new approach for describing schema modifications by integrating well-known formal description techniques in a simple logical framework. Our set of modification primitives can be used as a starting point for the development of a general semantically well founded schema modification language which can serve as a target language for the interpretation and comparison of other (less formal) approaches. In this paper we focussed on the structural aspects of these changes. This is — as pointed out in the introduction — only one component of an information system. Especially integrity constraints and the dynamic aspects of an information system will need more consideration to achieve further prospect. However we provided a framework which is — due to formal treatment of schema components and instances — well suited to serve as a basis for the integration of these aspects as well.

References

[AB91] Serge Abiteboul and Anthony Bonner. Objects and Views. In *Proc. Intl. Conf. on Management of Data*, pages 238–247. ACM SIGMOD, May 1991.

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object oriented database system manifesto. Technical Report 30-89, GIP ALTAIR, Le Chesnay France, September 1989.

[Bac86] R. C. Backhouse. *Program Construction and Verification.* Prentice Hall, London, 1986.

[BGL97] Frank Buddrus, Heino Gärtner, and Sven-Eric Lautemann. First steps to a formal framework for multilevel database modifications. In *Proc. of the 8th Int'l Conf. on Database and Expert Systems Applications (DEXA).* Springer-Verlag. LNCS, 1997.

[BKKK87] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In Umeshwar Dayal and Irv Traiger, editors, *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 311–322, San Francisco, CA, May 1987. ACM. SIGMOD Record, Vol. 16, Issue 3, December, 1987.

[BM93] E. Bertino and L. Martino. *Object-Oriented Database Systems - Concepts and Architectures.* Addison-Wesley, Wokingham, 1993.

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin Cummings, Redwood City, 2 edition, 1994.

[BR97] Grady Booch and James Rumbaugh. *Unified Method for Object-Oriented Development Version 1.0.* Rational Software Corporation, 1997.

[Che76] P. P. Chen. The entity relationship model - toward an unified view of data. *ACM Trans. on Database Sys.*, 1(1):9, March 1976. Reprinted in M. Stonebraker, Readings in Database Sys., Morgan Kaufmann, San Mateo, CA, 1988.

[CJ90] W. Cellary and G. Jomier. Consistency of Versions in Object-Oriented Databases. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th VLDB Conference*, pages 432–441, Brisbane, Australia, 1990.

[EGH+92] Gregor Engels, Martin Gogolla, Uwe Hohenstein, Klaus Hülsmann, Perdita Löhr-Richter, Gunter Saake, and Hans-Dieter Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering*, 9:157–204, 1992.

[EN94] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings, Redwood City, 2 edition, 1994.

[FMZ+95] Fabrizio Ferrandina, Thorsten Meyer, Roberto Zicari, Guy Ferran, and Joëlle Madec. Schema and Database Evolution in the O_2 Object Database System. pages 170–181, Zurich, Switzerland, September 1995.

[Gog94] Martin Gogolla. An extended entity-relationship model fundamentals and pragmatics. *Lecture Notes in Computer Science*, 767, 1994.

[GS94] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math.* Texts and Monographs in Computer Science. Springer-Verlag, New York, N.Y., 1994.

[Har87] David. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

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

[HS97] Andreas Heuer and Gunther Saake. *Datenbanken. Konzepte und Sprachen.* Addison-Wesley, 1997.

[KS96] G. Kappel and M. Schrefl. *Objektorientierte Informationssysteme.* Springer-Verlag, Wien, 1 edition, 1996.

[LS87] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification (Second edition).* John Wiley and Sons, New York, N.Y., 1987.

[MP96] Renate Motschnig-Pitrik. Requirements and comparison of view mechanisms for object-oriented databases. *Information Systems*, 21(3):229–252, 1996.

[Odb95] Erik Odberg. *MultiPerspectives: Object Evolution and Schema Modification Management for Object-Oriented Databases.* PhD thesis, Department of Computer Systems and Telematics, Norwegian Institute of Technology, February 1995. 408 pages.

[PM88] Joan Peckham and Fred Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, September 1988.

[PO97] R. J. Peters and M. T. Ozsu. An axiomatic model of dynamic schema evolution in objectbase systems. In *TODS 22(1)*, 1997.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Objekt-oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.

[Saa93] Gunter Saake. *Objektorientierte Spezifikation von Informationssystemen.* Teubner Verlag, Leipzig, 1993.

[SLT91] Marc H. Scholl, Christian Laasch, and Markus Tresch. Updatable Views in Object-Oriented Databases. In C. Delobel, M. Kifer, and Y. Yasunga, editors, *Proc. 2nd Intl. Conf. on Deductive and Object-Oriented Databases (DOOD)*, number 566 in Lecture Notes in Computer Science, München, FRG, December 1991. Springer.

[Sou95] C. Souza dos Santos. Design and implementation of object-oriented views. *Lecture Notes in Computer Science*, 978, 1995.

[SZ87] Andrea H. Skarra and Stanley B. Zdonik. Type Evolution in an Object-Oriented Database. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 393–415. 1987.

[Wie91] G. Wiederhold. Views, Objects, and Databases. In *On Object-Oriented Database Systems*, pages 29–43. Springer, 1991.

[Wir90] M. Wirsing. Algebraic Specification. In J. van Leewen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 13, pages 677–788. The MIT Press, New York, N.Y., 1990.

[Zic92] Roberto Zicari. A Framework for Schema Updates in an Object-Oriented Database System. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *Building an Object-Oriented Database System: The story of O2.* Morgan Kaufmann, 1992.