Information System Behavior Specification by High-Level Petri Nets

ANDREAS OBERWEIS and PETER SANDER
Universität Karlsruhe

The specification of an information system should include a description of structural system aspects as well as a description of the system behavior. In this article, we show how this can be achieved by high-level Petri nets—namely, the so-called NR/T-nets (Nested-Relation/Transition Nets). In NR/T-nets, the structural part is modeled by nested relations, and the behavioral part is modeled by a novel Petri net formalism. Each place of a net represents a nested relation scheme, and the marking of each place is given as a nested relation of the respective type. Insert and delete operations in a nested relational database (NF2-database) are expressed by transitions in a net. These operations may operate not only on whole tuples of a given relation, but also on "subtuples" of existing tuples. The arcs of a net are inscribed with so-called Filter Tables, which allow (together with an optional logical expression as transition inscription) conditions to be formulated on the specified (sub-) tuples. The occurrence rule for NR/T-net transitions is defined by the operations union, intersection, and "negative" in lattices of nested relations. The structure of an NR/T-net, together with the occurrence rule, defines classes of possible information system procedures, i.e., sequences of (possibly concurrent) operations in an information system.

Categories and Subject Descriptors: H.1.1 [**Models and Principles**]: Systems and Information Theory—*general systems theory*; H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*data manipulation languages* (*DML*)

General Terms: Design, Languages, Management

Additional Key Words and Phrases: Behavior specification, complex objects, conceptual design, nested relations, Petri nets

1. INTRODUCTION

1.1 Specification of Information Systems

The design of information systems must reflect static, object-related system aspects as well as dynamic, activity-related system aspects. Static system aspects include the structures of objects, relationships between objects, and

Authors' address: Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, 76128 Karlsruhe, Germany; email: {oberweis; sander}@aifb.uni-karlsruhe.de.

static semantic integrity constraints. Dynamic system aspects include the operations on objects (creation, deletion, modification of objects), the synchronization of operations, and dynamic semantic integrity constraints.

1.2 Nested Relational Databases

The relational data model originated by Codd [1970] is one of the most popular data models and has become the formal basis of many commercially available database systems. Several years ago, database researchers and practitioners realized that this model has a couple of shortcomings and disadvantages. The main point of the critique was that for many database applications (e.g., technical applications, office information systems, and multimedia systems) the first-normal form is not suitable [Makinouchi 1977], and considerable interest arose to investigate database models that allow for the modeling of complex structured objects. The major extension of the relational data model in this direction is to drop the first-normal-form assumption and to allow for nonatomic attribute values. In particular, the construction of sets and tuples from atomic values is a central concept of so-called *complex object models* or *nested relations* (also called *non-first-normal-form relations*). More precisely, a nested relation is a finite relation in a mathematical sense, where components of tuples can be either atomic (i.e., nondecomposable) or a (nested) relation. Thus, a nested relation is a recursively defined data structure that is suitable for representing complex objects.

In this article, the structural part of an information system is assumed to be given by a nested relational database (NF2-database). Aspects of conceptual data design, e.g., by semantic data models [Brodie et al. 1984; Peckham and Maryanski 1988], are beyond the scope of this article.

1.3 Petri Nets

Petri nets are a graphical formalism for modeling (distributed) system behavior (see Barron [1982], Brauer et al. [1987], Genrich and Lautenbach [1981], Peterson [1981], Reisig [1985], Richter and Durchholz [1982], and Zisman [1977]). They can be used to model sequentiality, mutual exclusion, and concurrency of system activities, and they support concepts for a stepwise formalization, as well as methods for the stepwise refinement of system descriptions.

There exist different types of nets. Low-level nets (e.g., *condition/event nets, place/transition nets* [Brauer et al. 1987]) allow an easy interpretation of net components, and they are easy to understand, but rather hard to handle in case of large systems with complex behavior.

In high-level nets (e.g., *predicate/transition nets, colored Petri nets* [Jensen and Rozenberg 1991]), the basic components are more expressive and allow a more compact description than in low-level nets. They allow for an integration of static, object-related system aspects when modeling system behavior aspects. It is generally agreed that, for practical applications, high-level nets are preferable to low-level nets.

For both types of Petri nets there exists a formal theoretical basis [Brauer et al. 1987; Peterson 1981; Reisig 1985]. Several Petri net tools, such as graphical editors, simulators, and analyzers, are already commercially available.

1.4 Goal and Structure of the Article

The major intention of our work is to provide a formal executable model of information systems, including database behavior. To achieve this, we integrate concepts of nested relational structures and predicate/transition nets (Pr/T-nets) into one framework, the so-called NR/T-nets. For a concrete application, an NR/T-net could be regarded as the result of conceptual database design (which is not considered in this article) on a precise logical level.

In Section 2, we briefly summarize the major concepts of Petri nets and motivate the need for modeling structured objects in Petri nets. In the sequel, these structured objects are formally described and referred to as nested relations. Section 3 deals with nested relational databases. Two different partial orders over nested relations—inclusion order and object order—are introduced. Based on these orders, a formal semantics for insert and delete operations in nested relational databases is defined. "Filter Tables" are proposed as a new concept to model such operations. Section 4 describes how to model the behavior of information systems with high-level Petri nets. A new type of net—the so-called Nested-Relation/Transition Nets (NR/T-nets)—is introduced. In these nets, each place marking represents a nested relation, and transitions represent database operations. Filter Tables are used as arc inscriptions to specify the tuples that are to be removed from or inserted into an adjacent place when a transition occurs. As a running example, the behavior of an information system for the planning and controlling of projects in an enterprise is modeled. Section 5 surveys related approaches and shows differences to our approach. Section 6 concludes the article. Implementation aspects are considered, and an outlook to future research work is given. The appendix contains a list of symbols. We do not explain all mathematical preliminaries in full detail. The reader is assumed to have basic knowledge about lattices and partial orders [Birkhoff 1973] and about nested relations as in Schek and Scholl [1986] and Thomas and Fischer [1986].

2. PETRI NETS AND THEIR EXTENSIONS

Petri nets are a graphical formalism for the description of system behavior. A net is a bipartite graph, consisting of two disjoint sets of nodes: the set of *places* and the set of *transitions*.

Definition 2.1 ((*Petri*) *Net*). A (Petri) net [Brauer et al. 1987] is a triple $N = (PL, TR, F)$ where

(1) PL is a finite set of so-called *places*,

(2) TR is a finite set of so-called *transitions,* $PL \cap TR = \emptyset$, and

(3) $F \subseteq (PL \times TR) \cup (TR \times PL)$ is called the *flow relation* of N.

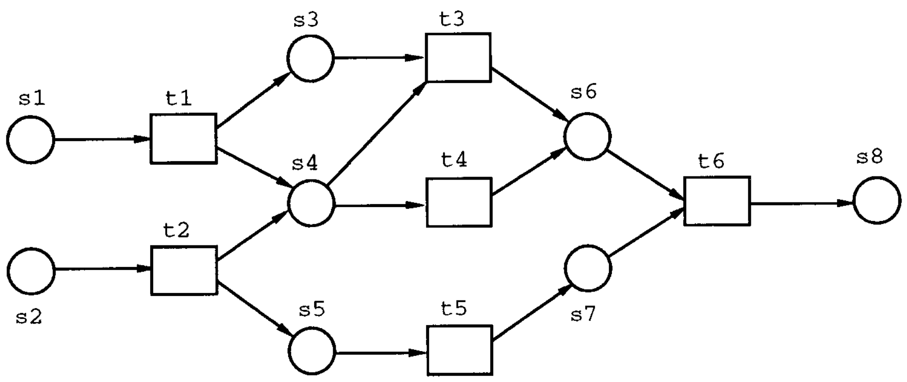


Fig. 1. Net.

In the graphical representation, places are represented as circles and transitions as boxes. Elements of F are represented as directed arcs between places and transitions. Places in a net may for example be interpreted as object storages and transitions as operations on the respective input/output objects. In general, places model state-related system aspects while transitions model activity-related system aspects.

Example 2.2 (*Net*). Figure 1 shows a net $N = (PL, TR, F)$ where $PL = \{s1, \ldots, s8\}$, $TR = \{t1, \ldots, t6\}$, and $F = \{(s1,t1), (s2,t2), \ldots, (t6,s8)\}$.

A net describes the structure of system behavior only informally. There is no formal notion of "system state" or "system behavior" in this simple type of net. However, there exist different approaches to extend the basic net concept.

In *predicate/transition nets* [Genrich and Lautenbach 1981], for instance, places represent relation schemes (predicates). A function assigns to each place a so-called *marking*, which is a relation of the respective type. The set of all place markings at a given time describes a certain global system state. A transition represents an operation on the relations in its input/output places. If a transition occurs, tuples are removed from the relations of its input places and are then inserted into the relations of its output places. A logical expression which may be assigned to a transition allows to specify certain conditions for the selection of tuples to be inserted or removed.

Example 2.3 (*Predicate/Transition Net* (*Pr/T-Net*)). The predicate/transition net in Figure 2 models the hiring process of employees in a firm. Departments are offering positions, and persons are requesting positions. Each offered position is represented as a tuple

(DEPARTMENT, POSITION, REQUESTED-PROFESSION, OFFERED-SALARY)

in the predicate OFFERED-POSITION. Each requested position is represented as a tuple

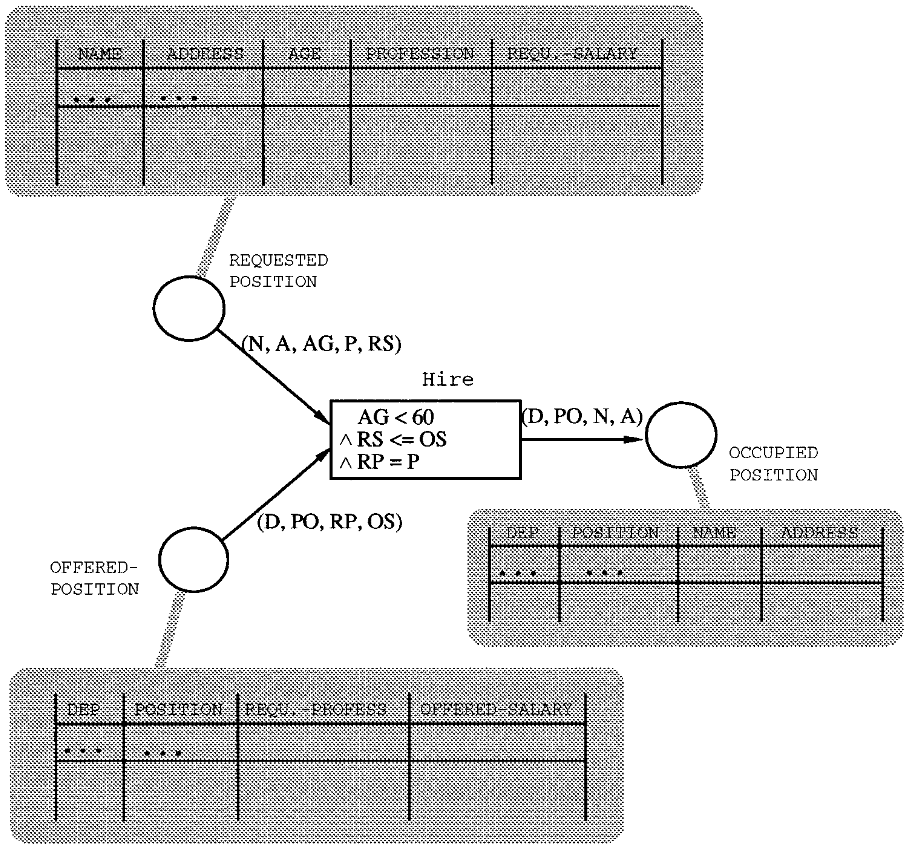


Fig. 2. Predicate/transition net.

(NAME, ADDRESS, AGE, PROFESSION, REQUESTED-SALARY)

in the predicate REQUESTED POSITION. Note that all attributes are atomic, i.e., there are no set-valued or tuple-valued attributes. A person may be hired for a certain position if the person's age is less than 60, if the requested salary is less than or equal to the offered salary, and if the requested profession is the same as the person's profession.

Hence, in Figure 2 the transition Hire is enabled, i.e., it may occur, if there exists a tuple (N, A, AG, P, RS) in the place REQUESTED POSITION and a tuple (D, PO, RP, OS) in the place OFFERED-POSITION, where $AG < 60$ and $RS \leq OS$ and $RP = P$. Additionally, the tuple (D, PO, N, A) must not yet exist in the output place OCCUPIED POSITION. If the transition Hire occurs, the respective tuples are removed from the input places, and a new tuple (D, PO, N, A) is inserted into the output place.

The problem with Pr/T nets is that, in their original form [Genrich and Lautenbach 1981], they only allow for the manipulation of flat tuples with atomic attribute values. The aim of the next sections is to integrate the concept of nested relations into the predicate/transition net formalism. The new net type to be introduced should especially support the manipulation of set-valued attributes.

3. NESTED RELATIONAL DATABASES

In this section we formally introduce nested relations, a simple, important, and mathematically well-founded class of complex structured objects. Furthermore, we define partial orders over nested relations. They serve as the formal basis for the specification of different kinds of insert and delete operations.

3.1 Nested Relations and Partial Orders

We distinguish between two aspects of a nested relational database state: the *scheme* and the *instance*. The scheme describes the structural and time-invariant part, which is usually stored in the data dictionary. The instances consist of (usually large amounts of) formatted data. The format of the instances has to match the structure of the scheme.

The definition of the structural part of nested relations is quite standard. We use a notation similar to Thomas and Fischer [1986]. Let U be a nonempty set of names, the *universe*. Then, the scheme of a database is specified by a finite, nonempty set T of *scheme equations*. Each scheme equation is of the form $R := (R_1, \ldots, R_n)$, where $n \geq 1$ and where $\{R, R_1, \ldots, R_n\}$ consists of distinct names and is a subset of U. Furthermore, each name R must not occur on the left-hand side of different scheme equations, and the scheme equations must not contain cycles. Then, for an equation $R := (R_1, \ldots, R_n)$, R is called a *scheme*, and each R_i is called an *attribute* of R. An attribute not occurring on the left-hand side of any scheme equation is called a *simple attribute*; otherwise, it is a *composite attribute*. A scheme not occurring on the right-hand side of any scheme equation is also called a *top-level scheme*. Let $(R_1, \ldots, R_k)$ be the tuple consisting of all top-level schemes of T. Then, the (meta-) scheme equation $D := (R_1, \ldots, R_k)$ specifies the *database scheme* D w.r.t. T.

Let U_{simple} denote all simple attributes of U, and let $\mathscr{d} = \{d_1, d_2, \ldots\}$ be a nonempty set of finite sets, the *domains*. Then, *dom* is a mapping from the set U_{simple} of all simple attributes into $\mathscr{d}$. An *instance* is of the form $n{:}v$ where n is called the *name* and v the *value* of the instance. Two cases are possible: if A is a simple attribute, and $a \in dom(A)$ then $A{:}a$ is an *instance of type A*. If R is a scheme, and if there exists a scheme equation $R := (R_1, \ldots, R_n)$ then $R{:}\{(\tau_{11}, \ldots, \tau_{1n}), \ldots, (\tau_{m1}, \ldots, \tau_{mn})\}$ is an *instance of type R*, where $m \geq 0$ and $i \in \{1, \ldots, m\}$, and $j \in \{1, \ldots, n\}$: τ_{ij} is an instance of type R_j. Let $Ins(R)$ denote the set of all instances of type R. Instances of type of a top-level scheme are called *top-level instances*; they are denoted by letters $r, r_1, r_2, \ldots$, and an instance $D{:}\{(r_1, \ldots, r_k)\}$ of type of a database scheme (i.e., consisting of the tuple of all top-level instances) is called a *database instance*.

Example 3.1.1. We model "direct railway connections" (DRCs) between cities by the following scheme equations:

{DRC :=(CITY, CITY2, TIMETABLE), TIMETABLE :=(DEP, ARR)}.

TIMETABLE is a scheme with attributes DEP and ARR. DRC is a top-level scheme with simple attributes CITY1 and CITY2 and the composite attribute TIMETABLE. The following domains are assigned to the simple attributes: dom(DEP) = dom(ARR) = *TIME* and dom(CITY1) = dom(CITY2) = *Char*(15), where *TIME* denotes all admissible points of time of the form *hh.mm* and *Char*(15) all strings of length $\leq$ 15. The following expressions are instances:

(a) DEP: 15.00
(b) TIMETABLE: {(DEP: 12.30, ARR: 13.30), (DEP: 13.00, ARR: 17.00)}
(c) DRC: {(CITY1: Karlsruhe, CITY2: München,
TIMETABLE: {(DEP: 08.00, ARR: 11.00),
(DEP: 09.00, ARR: 12.00),
(DEP: 12.00, ARR: 15.00)}),
(CITY1: München, CITY2: Frankfurt,
TIMETABLE: {(DEP: 06.45, ARR: 10.50),
(DEP: 12.30, ARR: 17.10)})}

In this example only (c) is a top-level instance. In the sequel we shall often use a graphical representation for top-level instances. For example, the table corresponding to (c) can be found in Example 3.1.2.

An element $(\tau_{i1}, \ldots, \tau_{in})$ of the value of an instance R:$\{(\tau_{11}, \ldots, \tau_{1n}), \ldots, (\tau_{m1}, \ldots, \tau_{mn})\}$ is also called a *tuple* of type R. For a tuple t let $t.R_k$ denote the instance of attribute R_k, i.e., for $t = (\tau_{i1}, \ldots, \tau_{in})$: $t.R_k = \tau_{ik}$. Often we explicitly distinguish between simple and composite attributes contained in a scheme. Then we write $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$, where the A_i are simple attributes, and the B_i are composite attributes.

In the sequel we shall introduce partial orders over nested relations. This can be motivated as follows. In order to describe the behavior of a database system, it must be specified how tuples can be inserted into and deleted from an instance, respectively. Note that we do not only want to insert (delete) tuples as a whole, but we also want to insert (delete) "low-level parts" of tuples into (from) existing tuples. By the definition of partial orders we shall obtain lattices of nested relations. In lattices there exist two different kinds of operations: union ($\cup$) and intersection ($\cap$). We further define a unary operation, the *negative* of instances (*Neg*) which is similar to a complement operation in Boolean lattices. These operations can be used to formally define insert and delete operations for nested relations:

—the insertion of a tuple t into an instance r can be expressed as $r \cup R$:$\{t\}$
—and the deletion of a tuple t from an instance r can be expressed as $r \cap Neg(R$:$\{t\})$.

DRC			
CITY1	CITY2	TIMETABLE	
		DEP	ARR
Karlsruhe	München	{(8.00, (9.00, (12.00,	11.00), 12.00), 15.00)}
München	Frankfurt	{(6.45, (12.30,	10.50), 17.10)}
...	...	...	

Figure 3

Example 3.1.2. We consider some "real-world" examples (Figures 3 and 4) to demonstrate the need for different kinds of insert and delete operations.

(1) Figure 3 shows direct railway connections (DRCs) between German cities (cf. Example 3.1.1 (c)). For each pair of cities there is a unique timetable (given as a set of pairs of departure and arrival times).

If a tuple is to be inserted into the instance—for example, the tuple (CITY1: Karlsruhe, CITY2: München, TIMETABLE:{(DEP: 15.00, ARR: 18.00)})—then we do not simply want to *add* this tuple to the instance. The semantics of this operation should be that the tuple is "merged" with the already existing Karlsruhe/München tuple, i.e., only (DEP: 15.00, ARR: 18.00) is inserted into the first timetable.

The semantics of deletions in this example should be similar. For instance, the deletion of the tuple (CITY1: Karlsruhe, CITY2: München, TIMETABLE:{(DEP: 8.00, ARR: 11.00)}) should lead to the deletion of (DEP: 8.00, ARR: 11.00) in the first timetable.

(2) In Figure 4 we model cars together with equipment for each car. In contrast to the DRC example, each car may be offered with different kinds of equipment (each of them having their own "identity").

Now, the semantics of an insertion, e.g., of the tuple (CAR TYPE: Golf GTD, EQUIPMENT: {(EQU. PART:airbag), (EQU. PART:radio)}) should be that the whole tuple is added to the CAR instance. It would not make sense to "merge" this tuple with an already existing tuple.

As we shall see, the definition of different partial orders over nested relations provides an appropriate formal basis for a whole class of (different) insert and delete operations. Two well-known partial orders—the *Inclusion Order* and the *Object Order*—are defined as follows.

Let r_1 and r_2 be two instances over a scheme R:

(1) *Inclusion Order*:

$r_1 \subseteq r_2 :\Leftrightarrow \forall t \in r_1 : t \in r_2$, or equivalently:

$$\forall t \in r_1: \exists t' \in r_2: \quad \text{attributes } A: t.A = t'.A.$$

CAR	
CAR TYPE	EQUIPMENT
	EQU. PART
Mercedes 230E	{airbag, leather seats}
Mercedes 230E	{airbag, leather seats, sunshine roof}
Golf GTD	{radio, sunshine roof}
...	...

Figure 4

It is well known that "$\subseteq$" is a partial order (i.e., a reflexive, transitive, and antisymmetric relation) over the set $Ins(R)$ of all possible instances.

(2) *Object Order*:

$$r_1 \leq r_2 \quad :\Leftrightarrow \quad \forall t \in r_1 : \exists t' \in r_2 : \qquad \text{simple attributes } A: t.A = t'.A \text{ and}$$

$$\text{composite attributes } B: t.B \leq t'.B.$$

The Object Order (or a very similar order, respectively) was also investigated by Abiteboul and Bidoit [1986], Bancilhon and Koshafian [1989], and Beeri and Kornatzky [1990]. The Object Order is defined recursively, and it is well known that, in general, this relationship between instances is not antisymmetric, e.g., for the instances $r_1 = R$:{(A:1,B:{(C:1)}), (A:1,B: {(C:1), (C:2)})} and $r_2 = R$:{(A:1, B:{(C:1), (C:2)})} both relationships $r_1 \leq r_2$ and $r_2 \leq r_1$ hold. This is due to the fact that the first tuple of r_1 is "contained" in the second tuple. In Bancilhon and Koshafian [1989] antisymmetry of the Object Order is enforced by restricting complex objects to so-called reduced objects, where this kind of redundancy is removed.

In our approach we need a stronger restriction and define "$\leq$" only over PNF instances (PNF = Partitioned Normal Form), a class of nested relations investigated in detail by Roth et al. [1988]. A PNF instance can be regarded as a normalized member of a class of instances in which each member represents the same information. An instance is called a PNF instance if it satisfies the following properties:

(1) it does not contain two distinct tuples that agree on the values of their simple attributes (i.e., composite attributes are functionally dependent on the set of all simple attributes) and
(2) for each tuple the values of all composite attributes are also PNF instances.

The set of all PNF instances over scheme R is denoted as $PNF(R)$. It turns out that "$\leq$" is a partial order over $PNF(R)$ (a proof can be found in Sander [1992; 1993]), and "$\leq$" induces a lattice structure $(PNF(R), \cup, \cap)$.

Example 3.1.3. Consider the following two PNF instances of type $R := (A, B, C)$, $B := (D)$, $C := (E)$. Assume that $dom(A) = dom(D) = dom(E) = \{1, 2, 3\}$.

r_1:

R		
A	B	C
	D	E
1	{1}	{2, 3}
2	{1, 2, 3}	{1, 2, 3}

r_2:

R		
A	B	C
	D	E
1	{2}	{}
2	{1, 2}	{1}
3	{}	{2, 3}

(1) Then with respect to the Object Order the least upper bound $r_1 \cup r_2$ (union) and the greatest lower bound $r_1 \cap r_2$ (intersection) yield the following results:

R		
A	B	C
	D	E
1	{1, 2}	{2, 3}
2	{1, 2, 3}	{1, 2, 3}
3	{}	{2, 3}

$r_1 \cup r_2$ (ObjectOrder)

R		
A	B	C
	D	E
1	{}	{}
2	{1, 2}	{1}

$r_1 \cap r_2$ (ObjectOrder)

(2) The situation is different for the Inclusion Order. The Inclusion Order also induces a lattice structure (which is simply a Boolean set lattice), and the least upper bound and the greatest lower bound are as follows (set-theoretic union and intersection):

R		
A	B	C
	D	E
1	{1}	{2, 3}
2	{1, 2, 3}	{1, 2, 3}
1	{2}	{}
2	{1, 2}	{1}
3	{}	{2, 3}

$r_1 \cup r_2$ (Inclusion Order)

R		
A	B	C
	D	E

$r_1 \cap r_2$ (Inclusion Order)

We can summarize, that by the Inclusion Order, values of composite attributes are tested on equality, whereas the Object Order requires the (recursively defined) containment of composite attribute values. This is also reflected by the operations shown in Example 3.1.3. The operations induced by the Inclusion Order can be characterized as "flat" operations because they do not affect the "internal" values of tuples. In contrast to this, the Object Order induces "deep" operations because tuples may change their "internal" values.

Next, we shall generalize these partial orders by combining them. Until now, the Inclusion Order and the Object Order were defined globally for each top-level scheme. Now we want to decide *locally* for each composite attribute whether the attribute values are tested on equality or on containment.

Example 3.1.4. In this example (Figure 5) we consider an instance of a hotel database. The instance contains the STYLE, EQUIPMENT, and CHARGE of rooms, together with the set of all VACANT ROOMS at a given point of time.

Now, the tuple (STYLE: single bed, EQUIPMENT: {(EQU. PART: shower)}, CHARGE: 75,-, VACANT ROOMS: {(ROOM NO.: 008)}) is to be inserted. The semantics of this operation is that ROOM NO. 008 has become a vacant room (e.g., a guest has departed), and the value 008 is to be inserted into the set of vacant rooms of the first tuple, because both tuples have identical attribute values for STYLE, EQUIPMENT, and CHARGE.

On the other hand, if we want to insert the tuple (STYLE: single bed, EQUIPMENT {(EQU. PART: shower), (EQU. PART: TV), (EQU. PART: telephone)}, CHARGE: 90,-, VACANT ROOMS: {(ROOM NO.: 217)}) the semantics is that the hotel management offers a new kind of room to its guests. In the instance shown above there is no room with the same EQUIPMENT, and it would be wrong to merge this tuple with one of the existing tuples. Instead, the whole tuple must be added to the instance.

Obviously the attribute EQUIPMENT is treated differently from the attribute VACANT ROOMS. EQUIPMENT has the "Inclusion Order semantics," i.e., values are tested on equality and are inserted and deleted as a whole. The values of VACANT ROOMS are tested on containment, and these values are merged with existing values or removed from existing values.

In order to capture both kinds of composite attribute semantics we introduce the following formalism [Sander 1992]:

Definition 3.1.5 (*Order Definition and the Corresponding Ordering*). Let T be a set of scheme equations and $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$ a top-level scheme where the A_i are simple attributes and the B_i composite attributes.

(1) Then, an *order equation* w.r.t. R is an equation of the form $\wr_R := (=A_1, \ldots, =A_k, \Phi B_1, \ldots, \Phi B_j)$ corresponding to a scheme equation $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$. Each symbol Φ_{B_i} is either the equality predicate $=_{B_i}$ (in this case B_i is called an equality attribute),

ROOM ADMINISTRATION			
STYLE	EQUIPMENT	CHARGE	VACANT ROOMS
	EQU. PART		ROOM NO.
single bed	{shower}	75,-	{007, 101, 105}
single bed	{bath, TV, fridge}	90,-	{211, 213}
twin bed	...	...	...

Figure 5

or it is $\prec_{B_i}$. Then B_i is called a *containment attribute*. A set O of order equations is called an *order definition* w.r.t. R if it satisfies the following constraint:

For the top-level scheme R there is a corresponding order equation $\prec_R := (\ldots)$ in O, and for each containment attribute B_i of R there also exists an equation $\prec_{B_i} := (\ldots)$ in O.

(2) Let R be a top-level scheme; let O be an *order definition* w.r.t. R; and let $\prec_R := (=_{A_1}, \ldots, =_{A_k}, \Phi_{B_1}, \ldots, \Phi_{B_j})$ in O. Then the *ordering defined by O* is recursively defined as follows:

$$r \prec_R S \;:\Leftrightarrow\; \forall t \in r\colon\; \exists t' \in s\colon\quad \forall \text{ simple attributes } A_i : t.A_i =_{A_i} t'.\mathrm{A}_i,$$

$$\forall \text{ composite attributes } B_i : t.B_i \, \Phi_{B_i} \, t'.B_i,$$

where each symbol $=_{A_i}$ is interpreted as an equality predicate over $dom(A_i)$. If Φ_{B_i} is of the form $=_{B_i}$ then this symbol is also interpreted as an equality predicate; otherwise—if Φ_{B_i} is of the form $\prec_{B_i}$—there exists an order equation in O that specifies the semantics of $\prec_{B_i}$.

For each composite attribute the ordering is either specified by an order equation, or it is identical with *equality*. In the first case we specify a subset relationship, whereas in the latter case sets are tested on equality and are treated as atomic values.

Example 3.1.6. Let $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$ be a top-level scheme. Then, the inclusion order over R can be specified by $\prec_R := (=_{A_1}, \ldots, =_{A_k}, =_{B_1}, \ldots, =_{B_j})$. The object order on R can be specified by the equation $\prec_R := (=_{A_1}, \ldots, =_{A_k}, \prec_{B_1}, \ldots, \prec_{B_j})$, where each order $\prec_{B_i}$ is also specified by an order equation in the same way.

Hence, Definition 3.1.5 is a generalization of the definition of the Object and the Inclusion Order. As for the Object Order this, in general, does not define a partial order in a mathematical sense because the antisymmetry condition may be violated. To obtain a partial order we restrict the set of admissible instances as follows:

Definition 3.1.7 (*Generalized PNF Instance*). Let $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$ be a top-level scheme and $\preceq_R := (=_{A_1}, \ldots, =_{A_k}, \Phi_{B_1}, \ldots, \Phi_{B_j})$ the corresponding order equation in the order definition O.

(1) An instance r of type R is in *Generalized Partitioned Normal Form* (or a *GPNF instance*) w.r.t. O, if the following holds:
   (a) r does not contain any two tuples which agree on the values of their simple attributes and their equality attributes
   (b) for each tuple the value of each containment attribute is a GPNF instance.
(2) $GPNF(R, O)$ denotes the set of all generalized PNF instances over R (w.r.t. O).

This is really a generalization, because if the instances of a top-level scheme $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$ are ordered by Inclusion Order, then $GPNF(R, O) = Ins(R)$, and if they are ordered by the Object Order then $GPNF(R, O) = PNF(R)$. Furthermore, if no composite attributes occur in a scheme (i.e., the first-normal form holds) then all these orderings are identical, because for a relation scheme $R := (A_1, \ldots, A_k)$ containing only simple attributes the corresponding order equation has to be $\preceq_R := (=_{A_1}, \ldots, =_{A_k})$, and the equation $Ins(R) = GPNF(R, O) = PNF(R)$ holds.

For formal investigations it is often necessary to distinguish explicitly between both kinds of composite attributes—equality attributes and containment attributes. Thus, we shall often use a scheme equation of the form $R := (A_1, \ldots, A_k, B_1, \ldots, B_j, C_1, \ldots, C_p)$ where the A_i denote simple attributes; the B_i denote equality attributes; and the C_i denote containment attributes.

Now, the least upper bound and the greatest lower bound of instances can be characterized in the following explicit way (a proof can be found in Sander [1992]):

PROPOSITION 3.1.8. *Let* $R := (A_1, \ldots, A_k, B_1, \ldots, B_j, C_1, \ldots, C_p)$ *be a scheme.*

(1) *Each relation* $\preceq_R$ *over* R *defined by an order definition* O *is a partial order over the set* $GPNF(R, O)$, *i.e., it is reflexive, antisymmetric, and transitive. Furthermore, all pairs* r_1, r_2 *of instances of* $GPNF(R, O)$ *have a least upper bound* $r_1 \cup_{R,O} r_2$ *and a greatest lower bound* $r_1 \cap_{R,O} r_2$. *Thus, the complete lattice* $L_{R,O} := (GPNF(R,O), \cup_{R,O}, \cap_{R,O})$ *exists and has a least element* $0_{R,O}$ *and a greatest element* $1_{R,O}$.

(2) *The operations $\cup_{R,O}$ and $\cap_{R,O}$ can be explicitly expressed in the following recursive way for each pair r_1, $r_2 \in GPNF(R,O)$:*

$$t \in r_1 \cup_{R,O} r_2 \Leftrightarrow$$

$$\begin{aligned}
& t \in r_1 \text{ and } \neg(\exists t_2 \in r_2\text{:} && \forall i \in \{1, \dots, k\}\text{: } t_2.A_i = t.A_i \text{ and} \\
& && \forall i \in \{1, \dots, j\}\text{: } t_2.B_i = t.B_i)
\end{aligned}$$

or

$$\begin{aligned}
& t \in r_2 \text{ and } \neg(\exists t_1 \in r_1\text{:} && \forall i \in \{1, \dots, k\}\text{: } t_1.A_i = t.A_i \text{ and} \\
& && \forall i \in \{1, \dots, j\}\text{: } t_1.B_i = t.B_i)
\end{aligned}$$

or

$$\begin{aligned}
& \exists t_1 \in r_1\text{: } \exists t_2 \in r_2\text{:} && \forall i \in \{1, \dots, k\}\text{: } t_1.A_i = t_2.A_i = t.A_i \text{ and} \\
& && \forall i \in \{1, \dots, j\}\text{: } t_1.B_i = t_2.B_i = t.B_i \text{ and} \\
& && \forall i \in \{1, \dots, p\}\text{: } t_1.C_i \cup_{C_i,O} t_2.C_i = t.C_i.
\end{aligned}$$

$$t \in r_1 \cap_{R,O} r_2 \Leftrightarrow$$

$$\begin{aligned}
& \exists t_1 \in r_1\text{: } \exists t_2 \in r_2\text{:} && \forall i \in \{1, \dots, k\}\text{: } t_1.A_i = t_2.A_i = t.A_i \text{ and} \\
& && \forall i \in \{1, \dots, j\}\text{: } t_1.B_i = t_2.B_i = t.B_i \text{ and} \\
& && \forall i \in \{1, \dots, p\}\text{: } t_1.C_i \cap_{C_i,O} t_2.C_i = t.C_i.
\end{aligned}$$

Example 3.1.9. We specify the following order equations for the room administration scheme of Example 3.1.4:

$$\leq_{ROOM\ ADMIN.} := (=_{STYLE}, =_{EQUIPMENT}, =_{CHARGE}, \leq_{VACANT\ ROOMS}),$$

$$\leq_{VACANT\ ROOMS} := (=_{ROOM\ NO}),$$

i.e., the composite attribute EQUIPMENT is an equality attribute whereas VACANT ROOMS is a containment attribute. Then the room administration instance r of Example 3.1.4 is a GPNF instance with respect to these order equations, because there do not exist any two tuples that agree on the values of their attributes STYLE, EQUIPMENT, and CHARGE. Furthermore, the insertion of a tuple t into r, as described in Example 3.1.4, can be specified by $r \cup_{ROOM\ ADMIN.,O}$ ROOM ADMIN.:$\{t\}$ where $\cup_{ROOM\ ADMIN.,O}$ is the union operation of the lattice induced by the partial order $\leq_{ROOM\ ADMIN.}$.

3.2 The Negative of a Nested Relation

Until now, only the insertion of tuples is formally defined. Next, we define a unary operation, the negative $Neg(r)$ of a GPNF instance r, in order to be able to express the deletion of a tuple t as $r \cap Neg(R{:}\{t\})$.

Intuitively, the negative of r contains all tuples (w.r.t. the underlying domains and orderings) not present in r. For the following definitions, let R be a scheme with a scheme equation $R := (A_1, \ldots, A_k, B_1, \ldots, B_j, C_1, \ldots, C_p)$, and let $\{_R$ be a partial order (specified by an order definition O) over $GPNF(R,O)$.

Definition 3.2.1 (*Negative of an Instance*). Let $1_{R,O}$ denote the greatest GPNF instance of the scheme R. Then, the *negative of a GPNF instance* r of type R, abbreviated as $Neg_{R,O}(r)$, is defined as

(1) If R contains no containment attribute C_i (i.e., $p = 0$): $Neg_{R,O}(r) = R{:}\{t | t \in 1_{R,O},\ t \notin r\}$

(2) Otherwise (i.e., $p \geq 1$): $Neg_{R,O}(r) = R{:}\{t | t$ satisfies the following condition (*)$\}$

(*) for all attributes A_i and B_i: $t.A_i \in dom(A_i)$ and $t.B_i \in Ins(B_i)$, and if $\quad t' \in r$ such that for all attributes A_i and B_i: $t.A_i = t'.A_i$, and $t.B_i = t'.B_i$ then for each containment attribute C_i: $t.C_i = Neg_{C_i,O}(t'.C_i)$, else for each containment attribute C_i: $t.C_i = 1_{C_i,O}$.

The next example shows that, in general, the negative of an instance is not identical to a Boolean complement.

Example 3.2.2. Let R be the scheme $R := (A, B)$, $B := (C)$, and $dom(A) = dom(C) = \{1, 2\}$. A is a simple attribute, and B is a containment attribute (specified by an order definition O).

We regard the top-level instance $r_1 := R{:}\ \{(A{:}1, B{:}\{(C{:}1)\})\}$. Then, the following instance r_2 is the *negative* of r_1:

$$r_2 := R{:}\ \{(A{:}1, B{:}\{(C{:}2)\}), (A{:}2, B{:}\{(C{:}1), (C{:}2)\})\}.$$

We obtain the following (because of the definition of $\cup_{R,O}$ and $\cap_{R,O}$):

$$r_1 \cup_{R,O} r_2 = R{:}\ \{(A{:}1, B{:}\{(C{:}1), (C{:}2)\}), (A{:}2, B{:}\{(C{:}1), (C{:}2)\})\} = 1_{R,O} \text{ and}$$

$$r_1 \cap_{R,O} r_2 = R{:}\{(A{:}1, B{:}\{\ \})\}\ (\textit{not} \text{ the least instance } O_{R,O})$$

A complement $C(r)$ of r must satisfy the equations $r \cup C(r) = 1$ and $r \cap C(r) = 0$ (where 0 is the least element, and 1 is the greatest element of the lattice, respectively). This does not hold in this example because the intersection of r_1 and r_2 does not yield the least instance $0_{R,O}$.

Now, we have provided appropriate means to express the deletion of tuples in a formal and sound way.

Example 3.2.3. (1) In the DRC scheme of Example 3.1.2 (Figure 3) the attribute TIMETABLE is interpreted as a containment attribute. Then the instance r given in the example is a GPNF instance (since no equality attributes occur it is also a PNF instance), and the deletion of the tuple

t = (*CITY*1: Karlsruhe, CITY2: München, TIMETABLE:{(DEP: 8.00, ARR: 11.00)})

can be specified by $r \cap_{R,O} Neg_{R,O}(R{:}\{t\})$. This leads to the deletion of (DEP: 8.00, ARR: 11.00) in the timetable of the Karlsruhe/München tuple.

(2) In the CAR scheme of Example 3.1.2 (Figure 4) the composite attribute EQUIPMENT must be specified as an equality attribute. Otherwise, the instance would not be a GPNF instance. If we want to delete the tuple

t = (CAR TYPE: Mercedes 230E, EQUIPMENT: {(EQU. PART: airbag)})

from the instance of Example 3.1.2 (part 2) by the expression $r \cap_{R,O} Neg_{R,O}(R{:}\{t\})$, then the instance remains unchanged because this tuple t is not present in the instance. But if we do the same with the tuple

t'=(CAR TYPE: Mercedes 230E, EQUIPMENT: {(EQU. PART: airbag), (EQU. PART: leather seats)})

then the whole first tuple is removed from the instance.

(3) Finally, we consider the ROOM ADMINISTRATION example (Example 3.1.4). As already demonstrated in Example 3.1.9 the attribute EQUIPMENT is an equality attribute, and VACANT ROOMS is a containment attribute. Then the semantics of the expression $r \cap_{R,O} Neg_{R,O}(R{:}\{t\})$, where r is the instance of Example 3.1.4 and t = (STYLE: single bed, EQUIPMENT: {(EQU. PART: shower)}, CHARGE: 75,-, VACANT ROOMS: {(ROOM NO.: 007)}) is that only the room number 007 is removed from the set of vacant rooms. If we do the same with the tuple t' = (STYLE: single bed, EQUIPMENT: {(EQU. PART: shower)}, CHARGE: 75,-, VACANT ROOMS: {(ROOM NO.: 007), (ROOM NO.: 101), (ROOM NO.: 105)}) then all rooms are deleted from the first tuple, i.e., the tuple remains in the instance, but the set of vacant rooms becomes empty.

3.3 The Insert and Delete Operations Specified by Terms and Filter Tables

In this section we describe how the selection of information and the insertion and deletion of tuples can be specified in a more general, parametrized way by so-called top-level terms. This is to express *classes* of operations in a uniform way.

Definition 3.3.1 (*Term*). We assume that a database scheme is given by a set T of scheme equations. Let U be the names of attributes and schemes occurring in T, and let Var be a nonempty set of variables (denoted by strings of uppercase letters, e.g., X, Y, Z, XY, UVW, . . .).

Then, the set of all *terms* is defined recursively as follows:

(a) If $R \in U$ and $X \in Var$ then $R{:}X$ is a *term of type* R. We say X *represents values of type* R.

(b) If $A \in U$ is a simple attribute, and $a \in dom(A)$, then $A{:}a$ is a *term of type* A.

(c) If $R \in U$ is a composite attribute specified by the equation $R := (R_1, \ldots, R_n)$ then $R{:}\{(\tau_{11}, \ldots, \tau_{1n}), \ldots, (\tau_{m1}, \ldots, \tau_{mn})\}$ is a *term of type* R, where $m \geq 0$ and $i \in \{1, \ldots, m\}$, and $j \in \{1, \ldots, n\}$: τ_{ij} is a term of type R_j. Each expression $(\tau_{i1}, \ldots, \tau_{in})$ is called a *tuple* of the term.

(d) For a term $n{:}v$, n is called the *name* and v the *value* of the term.

Only expressions which can be constructed by (a)–(c) are *terms*. A term of type "top-level scheme" is called a *top-level term*.

Thus terms are recursively composed according to their type. The set of all terms containing no variables is identical to the set of all instances. For example, valid terms are (cf. Example 3.1.2)

(a) DEP: X,

(b) TIMETABLE: {(DEP: 8.00, ARR: Y), (DEP: U, ARR: V)},

(c) DRC: Y,

(d) DRC: {(CITY1: X, CITY2: Frankfurt, TIMETABLE: Z)}.

Only (c) and (d) are top-level terms.

Next, we introduce so-called labeled top-level terms in order to select information from instances and to specify insert and delete operations.

Definition 3.3.2 (*Labeled Top-Level Term*). Let R be a top-level scheme of a set of scheme equations T. Then, a *labeled top-level term* w.r.t. R (or a labeled R-term) is a pair (τ, O) consisting of a top-level term τ of type R and an order definition O w.r.t. R. $LT(R)$ denotes the set of all labeled top-level terms w.r.t. R.

Each labeled top-level term has an equivalent graphical representation called *Filter Table* which is easier to read. This formalism is adopted from the graphical query language *Query by Example* [Zloof 1975] known from the field of relational databases.

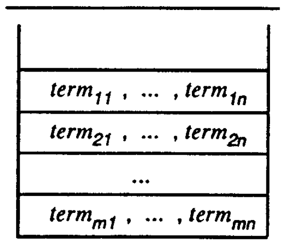
A labeled top-level term is transformed into a Filter Table as follows:

—Each term $\tau = R{:}\{(\tau_{11}, \ldots, \tau_{1n}), \ldots, (\tau_{m1}, \ldots, \tau_{mn})\}$ is transformed into a table of the form:

$term_{11}$, ... , $term_{1n}$
$term_{21}$, ... , $term_{2n}$
...
$term_{m1}$, ... , $term_{mn}$

—The strings $term_{ij}$ in the table represent transformed terms: each of the terms τ_{ij} is also transformed into a table if its value consists of a set of tuples. Otherwise—i.e., if the term is of the form $R{:}X$ or $A{:}a$—it is simply transformed into "X" or "a" respectively.

—If in the previous transformation steps the attribute of a term is a (composite) equality attribute, then the transformed term is marked by an overline, e.g., for the term above:



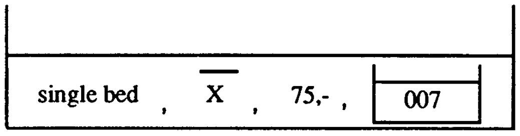
Such a term of type "equality attribute" is called a *closed term*; otherwise (i.e., for a containment attribute) it is called an *open term*.

Example 3.3.3. The following labeled terms (τ, O) are specified with respect to the hotel database of Example 3.1.4:

(1) τ = ROOM ADMINISTRATION: {(STYLE: single bed, EQUIPMENT: X, CHARGE: 75,-, VACANT ROOMS: {(ROOM NO.: 007)})},

$O = \{ \{_{ROOM\ ADMIN.} := (=_{STYLE}, =_{EQUIPMENT}, =_{CHARGE}, \{_{VACANT\ ROOMS}),$
$O = \ \{_{VACANT\ ROOMS} := (=_{ROOM\ NO})\}$

This term is represented graphically by the following Flter Table:

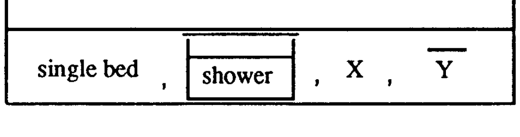


As EQUIPMENT is an equality attribute, all terms of type EQUIPMENT are closed terms. Thus, the variable X is marked by an overline.

(2) τ = ROOM ADMINISTRATION: {(STYLE: single bed, EQUIPMENT: {(EQU. PART: shower)}, CHARGE: X, VACANT ROOMS: Y)},

$O = \{ \{_{ROOM\ ADMIN} := (=_{STYLE}, =_{EQUIPMENT}, =_{CHARGE}, =_{VACANT\ ROOMS}) \}$

This term is represented graphically by the following Filter Table:



Obviously the mapping from labeled top-level terms to Filter Tables is one-to-one, and in the sequel we do not distinguish between both kinds of representations. As already mentioned, labeled top-level terms will be used to specify insert and delete operations. For example, the terms of Example 3.3.3 can be used to specify the following delete operations:

(1) Select a tuple where STYLE = single bed, CHARGE = 75,-, and where 007 is an element of the set of vacant rooms. Delete room number 007 from the set of vacant rooms of this type, and assign the respective EQUIPMENT value to the variable X.
(2) Select a tuple where STYLE = single bed and where shower is the only equipment part. Delete the whole tuple, and assign the CHARGE value to the variable X and the VACANT ROOMS value to the variable Y.

The insert operation is inverse to the delete operation. In order to define these operations formally, the variables contained in a term must be instantiated by values of the corresponding domains. Therefore, we define the notion of instantiation.

Definition 3.3.4 (*Instantiation of a Term*). Let (τ, O) be a labeled top-level term.

(1) An *instantiation* Θ of τ is a mapping from the set of all variables occurring in τ into the set of all instance values represented by the variables. Let $\tau\Theta$ denote a term τ where each variable is replaced by its Θ-value. Note, that if a term has no variables, the only possible instantiation is the *empty instantiation*.
(2) Assume that the term τ of a labeled term (τ, O) is of type R, and let $GPNF(R,O)$ be the set of GPNF instances with respect to the order definition O. Then, an instantiation Θ of τ is *valid* if $\tau\Theta \in GPNF(R,O)$.

We are only interested in valid instantiations, because we do not want to leave the class of GPNF instances in our formalism. Now we can specify deletions and insertions with respect to a valid instantiation in a formal way. Let (τ, O) be a labeled top-level term (a Filter Table), $\prec_R$ the partial order specified by O, and let $r \in GPNF(R,O)$:

Deletion. Let Θ be a valid instantiation of τ. Then the deletion of $\tau\Theta$ from r is formally defined as

$$r \cap_{R,O} Neg_{R,O}(\tau\Theta).$$

Insertion. Let Θ be a valid instantiation of τ. Then the insertion of $\tau\Theta$ into r is formally defined as

$$r \cup_{R,O} \tau\Theta.$$

As we shall see in Section 4, this formalism of Filter Tables is well suited for behavior modeling by high-level Petri nets.

Example 3.3.5. Consider again the Filter Tables of Example 3.3.3.

Deletion. If a deletion is to be expressed, there is only one possible instantiation in both cases (1) and (2) such that $\tau\Theta \{_{R,O}\, r$ (where r is the hotel database instance of Example 3.1.4). If in (1) X is instantiated with {(EQU. PART: shower)}, then the desired operation (described before Definition 3.3.4) is expressed. In (2) X is mapped to 75,- and Y to {(ROOM NO.: 007), (ROOM NO.: 101), (ROOM NO.: 105)} in order to express the desired operation (also mentioned above).

Insertion. We explain what kind of insertions can be specified by the Filter Table (1). Assume that the room number 007 is not present in the instance r (because this room is occupied at the moment). Then, this room is inserted into the set of vacant rooms of the first tuple if X is instantiated with {(EQU. PART: shower)}. On the other hand, if X were instantiated by any other equipment combination, e.g., {(EQU. PART: shower), (EQU. PART: fridge)}, then the *whole* instantiated tuple would be inserted into the instance (because no other room with this combination of STYLE, EQUIPMENT, and CHARGE exists).

4. MODELING INFORMATION SYSTEM BEHAVIOR BY NESTED-RELATION/TRANSITION-NETS

Our starting point is the net formalism of Section 2. The aim of this section is to integrate the concept of nested relations into the formalism. The new net type to be introduced (so-called Nested-Relation/Transition-nets, abbreviated *NR/T-nets*) should meet the following objectives:

—it should properly extend the Pr/T-net formalism of Section 2,

—it should have a formal semantics, and

—it should be very expressive with respect to the manipulation of nested relations, i.e.,
  (a) it must allow one to express complex selection conditions, such as set membership, set equality, subset, etc.
  (b) it must allow one to delete and insert not only tuples on the top level, but also subinstances of nested relations (as shown in Section 3).

The first extension to ordinary nets is to interpret places as nested relation schemes with attribute values that may in turn themselves be (nested) relations. This extension yields the so-called NR-nets, which visualize the *structural* part of NR/T-nets. NR/T-nets are defined later on.

Definition 4.1 (*Nested-Relation-Net* (*NR-Net*)). A nested-relation-net is a tuple $NN = (\varphi, N)$, such that

(1) $\varphi = (U, T, \mathcal{d}, dom)$ is a signature of the net, where U a universe, i.e., a set of (attribute and scheme) names, T a set of scheme equations, $\mathcal{d} = \{d_1, d_2, \ldots\}$ a set of domains, and $dom{:}U_{\text{simple}} \rightarrow \mathcal{d}$ a function.

(2) $N = (PL, TR, F)$ is a net (cf. Definition 2.1) where $PL = \{R_1, \ldots, R_n\}$ equals the set of top-level schemes of T.

Example 4.2 (*Nested-Relation-Net*). An information system for planning and controlling projects in an enterprise is to be designed. The (simplified) lifecycle of a project is as follows:

—If a project proposal is accepted, then a department with the required equipment is selected. A project proposal consists of a project number, the required equipment (given as a set of hardware and software component names), and the required personnel (given as a set of needed project members, each of them with a set of required skills).

—In the next step, employees having the required skills are selected from the respective department. An employee cannot participate in more than one project. To simplify the example, we assume that in every selected department there are enough employees available or that additional employees can be hired.

—The proposed project is initiated.

—Finally, a running project is either canceled or successfully finished. In both cases an internal message is sent to the department that was responsible for the project, and the employees involved in the project become available for other projects.

The structure of the possible behaviors related to a project can be modeled by an NR-net *Project-Life-Cycle* = (φ, N) as follows:

φ = (U, T, $\mathcal{d}$, *dom*), where
U = {DEPARTMENT, EMPLOYEE, PROJECT-PROPOSAL, RUNNING-PROJECT, DEP#, PERSONNEL, EQUIPMENT, . . .}
T = {DEPARTMENT := (DEP#, PERSONNEL, EQUIPMENT, PROJECTS),
EMPLOYEE := (EMP#, SKILLS, STATUS),
PROJECT-PROPOSAL :=(PRO#, EQUIPMENT, PP-MEMBERS),
PROJ-EMP := (PRO#, EMPLOYEES),
PROJ-DEP := (PRO#, DEP#, DEP-PERS, PP-MEMBERS),
RUNNING-PROJECT := (PRO#, DEP#, RP-MEMBERS),
CANCELED-PROJECT := (PRO#, DEP#),
PERSONNEL := (EMP#),
EMPLOYEES := (EMP#),
DEP-PERS := (EMP#),
EQUIPMENT := (EQ-NAME),
PP-MEMBERS := (PPM#, SKILLS),
RP-MEMBERS := (EMP#),
PROJECTS := (PRO#),
SKILLS := (SKILL)
etc. . . . }

DEP#, EMP#, PRO#, STATUS, SKILL, and EQ-NAME are atomic attributes. If an employee already participates in a project then the value of the attribute STATUS equals 1; otherwise the value equals 0.

$\mathcal{d}$ = {{d1, d2, . . .}, {p1, p2, . . .}, . . .}
dom(DEP#) = {d1, d2, . . .},
dom(PRO#) = {p1, p2, . . .}, etc.

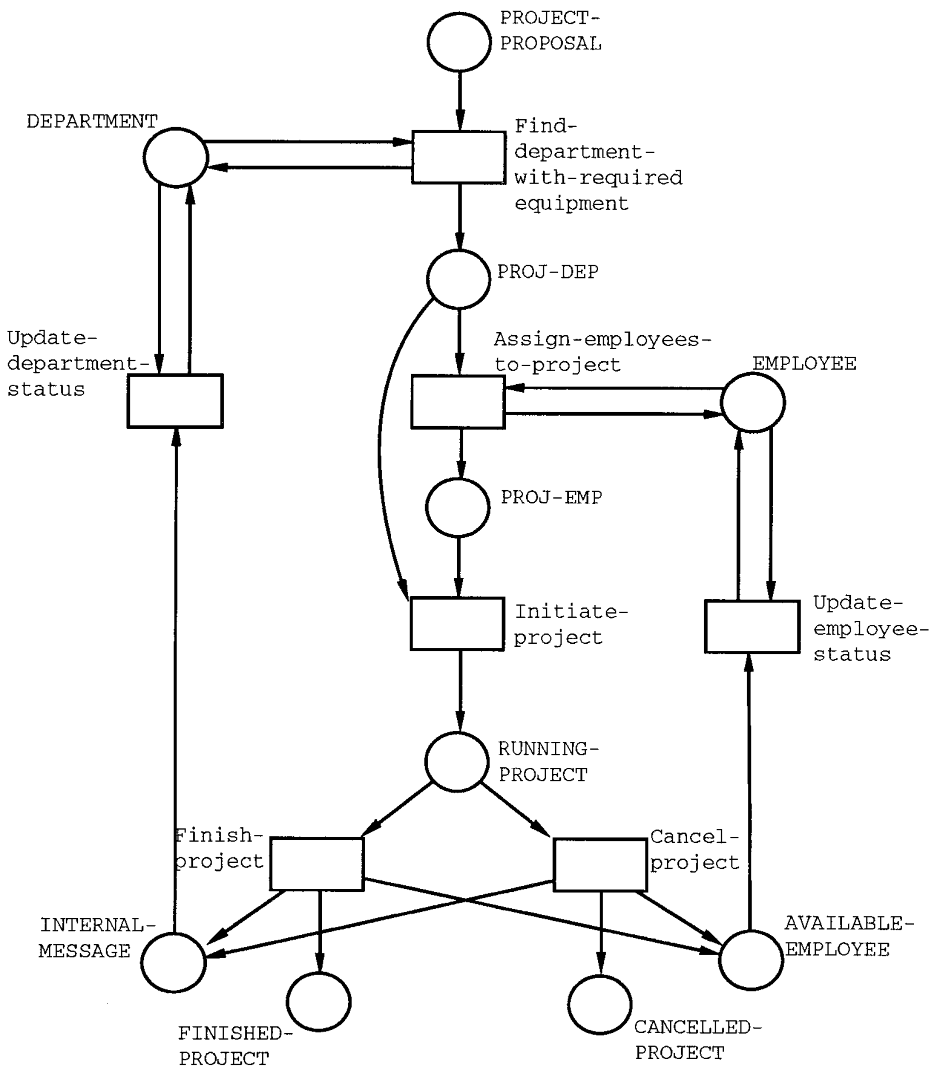


Fig. 6. Example NR-net.

The net N is shown in Figure 6. Each place R represents the respective top-level scheme "R := . . ." in T. In the graphical representation we omit the right-hand sides of the scheme equations.

In Figure 6 it is, for example, modeled that a transition Finish-project requires a running project as input and produces an internal message, a finished project, and a set of available employees as output. The transition is not yet specified in more detail.

An NR-net describes the *structure* of possible system behaviors (procedures), but it does not describe concrete states or state transitions. Each transition is interpreted as a class of changes applied to the possible instances in the adjacent places. In the sequel we define NR/T-nets.

Informally speaking, an NR/T-net consists of an NR-net plus the following concepts:

—an *initial marking*, i.e., a mapping that assigns to each place (i.e., to each top-level scheme) a nested relation instance.
—two concepts to select and delete tuples from places and to insert tuples into places:
  (a) *arc inscriptions*: arcs are inscribed with Filter Tables.
  (b) *transition inscriptions*: transitions are inscribed with logical expres sions.
—*occurrence rule*: defines in which state a transition is enabled and specifies which changes are related to a transition occurrence.

Definition 4.3 (*Nested-Relation/Transition-Net* (*NR/T-Net*)). A nested-relation/transition-net is a five-tuple $NRT = (NN, P, AI, TI, M^0)$, such that

(1) $NN = (\varphi, N)$ is an NR-net; $\varphi = (U, T, d, dom)$; $N = (PL, TR, F)$ (cf. Definition 4.1).
(2) P is a set of typed predicate symbols to specify membership, equality, etc. between instances. Each element $p \in P$ has a fixed interpretation $I(p)$, which is a relation of the respective arity over instance values and tuples.
(3) Let $LT(R)$ denote the set of all labeled R-terms (Filter Tables). Then, AI is a function that assigns to each element f in F a labeled term of $LT(R)$, where R is the adjacent place (scheme).
(4) $TI{:}TR \rightarrow LF$ assigns to each transition a logical formula that must be contained in the following set LF:
  —$true \in LF$
  —If $p \in P$ is an n-ary predefined predicate symbol, and $v_1, \ldots, v_n$ are variables or values of terms or tuples contained in values of terms, then $p(v_1, \ldots, v_n) \in LF$
  —If f_1 and $f_2 \in LF$ then $f_1 \wedge f_2, f_1 \vee f_2, \neg f_1 \in LF$.
(5) A marking M is a mapping that assigns to each place $R_i \in PL$ an instance of type R_i. M^0 is a marking, the so-called *initial marking*.

In the graphical representation of a nested-relation/transition-net the transition inscription "true" is omitted. For predicates in logical formulas we use infix instead of prefix notation (e.g., we write "X = Y" instead of "=(X, Y)"). To give the reader an intuition of how Definition 4.3 can be applied in a "real-life example," we discuss the running example of this section in an informal way. Afterward, we regard the contents of Definition 4.3 in more detail.

Example 4.4. Figure 7 shows an initial marking for the places of the running example in Figure 6.

The marking is given as a set of instances for the schemes DEPARTMENT, EMPLOYEE, RUNNING-PROJECT, and PROJECT-PROPOSAL, PROJ-EMP,

DEPARTMENT			
DEP#	PERSONNEL	EQUIPMENT	PROJECTS
	EMP#	EQ-NAME	PRO#
d1	{1, 3, 5, 11}	{e2, e3, e5}	{p11, p33}
d2	{2, 8}	{e2}	{ }
d3	{6, 9}	{e1, e2, e3}	{p21}
d4	{4, 7}	{e2, e3}	{p23}
d5	{10, 12}	{e2, e3, e9}	{p29}

RUNNING-PROJECT		
PRO#	DEP#	RP-MEMBERS
		EMP#
p11	d1	{5}
p21	d3	{6, 9}
p23	d4	{4, 7}
p29	d5	{10, 12}
p33	d1	{11}

EMPLOYEE		
EMP#	SKILLS	STATUS
	SKILL	
1	{s1, s2, s3, s4, s5}	0
2	{s1, s2, s3}	0
3	{s1, s2, s3}	0
4	{s1, s2, s3, s4, s5}	1
5	{s1, s2}	1
6	{s1, s2, s3, s4, s5, s7}	1
7	{s1, s2, s5}	1
8	{s6}	0
9	{s1, s3, s4, s5}	1
10	{s2, s5, s6}	1
11	{s1, s2, s3, s4}	1
12	{s2, s3, s4, s5}	1

PROJECT-PROPOSAL			
PRO#	EQUIPMENT	PP-MEMBERS	
	EQ-NAME	PPM#	SKILLS
			SKILL
p12	{e2, e5}	{(1, (2,	{s1, s3, s4}), {s1, s2, s3})}

Fig. 7. Initial marking for DEPARTMENT, RUNNING-PROJECT, EMPLOYEE, and PROJECT-PROPOSAL.

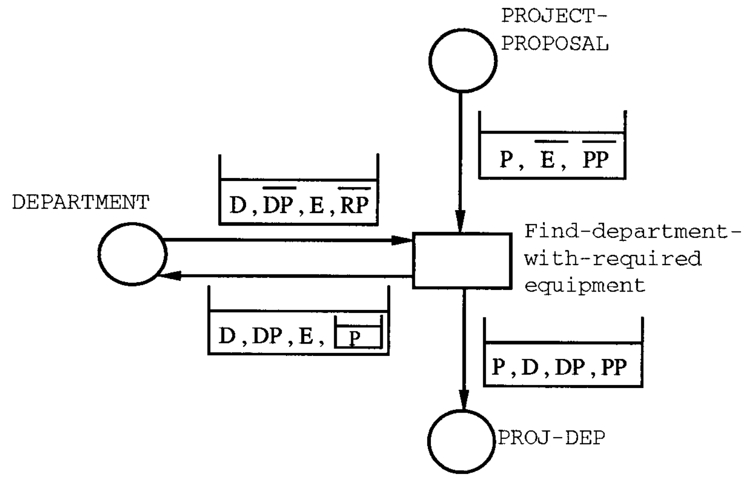

PROJECT-PROPOSAL
P, $\overline{E}$, $\overline{PP}$
DEPARTMENT
D, $\overline{DP}$, E, $\overline{RP}$
Find-department-with-required equipment
D, DP, E, $\boxed{P}$
P, D, DP, PP
PROJ-DEP

Fig. 8. A transition of an NR/T-net.

PROJ-DEP. Note that the markings of PROJ-EMP and PROJ-DEP are empty in the initial state.

In the initial state (marking), there exists one proposed project, and five projects are currently running. Four employees are not involved in any project (i.e., STATUS = 0).

In Figure 8 a small part of an NR/T-net of the project lifecycle described in Example 4.2 is given. It shows a single transition and the adjacent places. The transition has two incoming and two outgoing arcs. Each arc is inscribed with a labeled top-level term, which is equivalently written as a Filter Table. The transition Find-department-with-required-equipment is inscribed with the logical formula "*true*," which is omitted in the graphical representation.

An *occurrence* of a transition, which means a data flow from the input places to the output places, can be explained as follows. First of all, each variable must be instantiated with a value of the corresponding attribute domain. Equal values must be assigned to equal variables in the environment of a transition. As described in the previous section, for each Filter Table an ordering is specified by the open and closed terms occurring in the Filter Table. This determines two things: (a) how tuples are deleted from (or inserted into, respectively) the instances of the adjacent places (cf. Section 3.3) and (b) the conditions for which a transition is *enabled* (i.e., can occur). A transition is enabled with respect to a marking and a valid instantiation if

—The instantiated terms at the incoming arcs are contained (w.r.t. the underlying ordering) in the marking (instances) of their respective adjacent places. This implies instantiated closed terms must be contained as a whole in the marking of a place, and open terms must be a “substructure” of these markings.

—The instantiated terms at the outgoing arcs are not contained (w.r.t. the underlying ordering) in the markings of their respective adjacent places.

—The logical formula inscribed to the transition evaluates to “true.”

Then, the tuples specified by the Filter Tables of the incoming arcs are deleted from the corresponding markings, and the tuples contained in the Filter Tables of the outgoing arcs are inserted into the markings of the output places.

Example 4.4 (*Continued*). For the transition Find-department-with-required-equipment of Figure 8 we obtain the following:

—The Filter Tables which are assigned to the incoming arcs select a PROJECT-PROPOSAL tuple and a DEPARTMENT tuple, respectively. The PROJECT-PROPOSAL tuple consists of a PRO#, the EQUIPMENT, and the PP-MEMBERS. If the transition occurs the whole tuple is deleted from the PROJECT-PROPOSAL instance (because in the Filter Table *all* “relation-valued” variables are closed terms, i.e., marked with an overline). The department is a tuple consisting of a DEP#, the PERSONNEL, the EQUIPMENT, and the running PROJECTS of the department. If the transition occurs the specified tuple is not deleted as a whole, because the variable E is an open term. The DEP#, the PERSONNEL, and the PROJECTS remain unchanged in the instance. Only the specified EQUIPMENT set is removed from the instance.

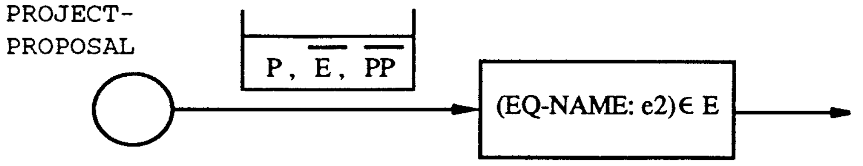
Note that the variable E occurs in both Filter Tables of the incoming arcs. One occurrence is marked with an overline; the other one is not. This is equivalent to a subset relationship. The equipment of the selected department must be a subset of the equipment required for the project proposal.

Tuples are inserted into the instances of the output places (in an inverse way) if the transition occurs. A new project P and the equipment E (which was removed by the deletion) are inserted into the DEPARTMENT instance. By the other Filter Table PRO#, DEP#, PERSONNEL, and the required PP-MEMBERS are inserted into the PROJ-DEP instance (which was assigned the empty instance by the initial marking).

Note that a set of “independently enabled” transitions may occur concurrently in one step. Informally, a set of transitions is independently enabled if each element is enabled, and additionally no two transitions in this set access the same (sub-) tuple.

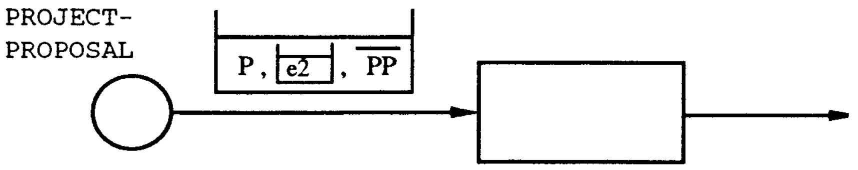
The occurrence of transitions in a given initial marking leads to so-called *follower markings*. The exact meaning of an enabled transition and a transition occurrence will be formally defined later on. In the following example we consider the arc inscriptions and transition inscriptions (cf. Definition 4.3 parts (3) and (4)) in more detail.

Example 4.5. We regard the PROJECT-PROPOSAL instance of the running example in order to show how information can be selected from this instance by different kinds of Filter Tables and transition inscriptions.



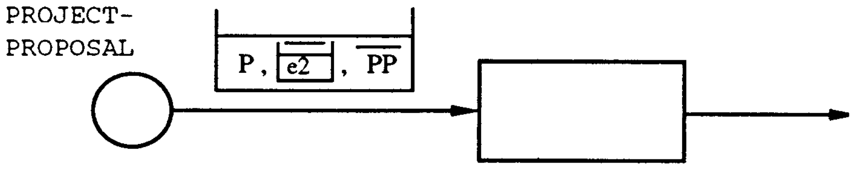
(1)

By this transition inscription and the given Filter Table a *whole tuple* is deleted from the instance PROJECT-PROPOSAL (when the transition occurs) if the EQUIPMENT value contains the part e2.



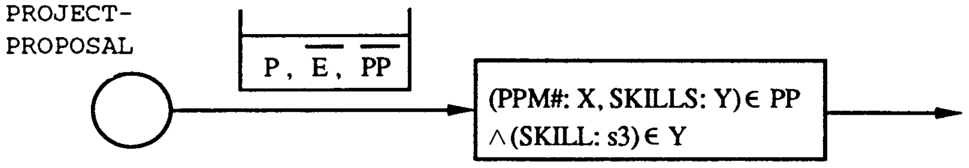
(2)

This transition is very similar to the first one. The selected tuple must satisfy the same constraint as in (1), i.e., the EQUIPMENT value must contain the value e2. In contrast to (1) the tuple is not deleted as a whole, only the selected part e2 is removed from the EQUIPMENT value.



(3)

This transition is again similar to (1) and (2). In this case a whole tuple is deleted, and the EQUIPMENT value must contain the part e2 and no further part.



(4)

By this transition inscription and the given Filter Table a whole tuple is selected, and the PP-MEMBERS value must contain a tuple which has the skill s3 in its set of skills.

By the following steps we slightly restrict the set of possible NR/T-nets to so-called *valid NR/T-nets*. This can be motivated and summarized as follows:

—First, the insert operation for tuples into the instance of a place is determined by the ordering of a Filter Table that is assigned to an incoming arc. If there is more than one incoming arc, then all respective Filter Tables must specify the *same ordering*. Otherwise, ambiguities concerning the insertion of tuples may occur. Thus, it makes sense to assign this unique ordering to the adjacent place and to talk about *the ordering of the place*.

—Second, the initial marking must satisfy the restriction imposed by these orderings. In particular, if a place R has an ordering O, then the initial marking must satisfy $M^0(R) \in GPNF(R,O)$. A marking that satisfies this condition is called a *valid marking*.

—Third, all Filter Tables at the outgoing arcs of a place must specify orderings that are *compatible* with the ordering of the place. This has technical reasons that, if violated, would also cause ambiguities concerning the deletion of tuples. More precisely, the ordering of an "outgoing Filter Table" must be less or equally restrictive than the ordering of the place, in the sense that each instance of a place is also a GPNF instance w.r.t. the ordering of the Filter Table.

The last point is formalized as follows:

Definition 4.6 (*Comparison of Orderings*). Let R be a top-level scheme defined by the scheme equation $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$. The A_i denote simple attributes, and the B_i denote composite attributes. Let O and O' be order definitions w.r.t. R (cf. Definition 3.1.5), which specify orderings $\prec_{R,O}$ and $\prec_{R,O'}$, respectively.

The ordering $\prec_{R,O}$ is *more or equally restrictive* than $\prec_{R,O'}$ (denoted as $\prec_{R,O} \sqsubseteq \prec_{R,O'}$) if the following holds for each composite attribute B_i:

B_i is containment attribute w.r.t. $\prec_{R,O'}$
$\Rightarrow$ B_i is containment attribute w.r.t. $\prec_{R,O}$ and $\prec_{B_i,O} \sqsubseteq \prec_{B_i,O'}$.

In this case we also say $\prec_{R,O'}$ is *less or equally restrictive than* $\prec_{R,O}$.

Equivalently, an ordering $\prec_{R,O}$ is more or equally restrictive than another ordering $\prec_{R,O'}$ if each equality attribute w.r.t. $\prec_{R,O}$ is also an equality attribute w.r.t. $\prec_{R,O'}$. It can be shown that $\sqsubseteq$ is a partial order on orderings. We easily obtain the following result:

PROPOSITION 4.7. *Let R be a top-level scheme with the scheme equation $R := (A_1, \ldots, A_k, B_1, \ldots, B_j)$. Let O and O' be order definitions w.r.t. R*

which specify orderings $\{_{R,O}$ *and* $\{_{R,O'}$*, respectively. Then, it follows that*

$$\{_{R,O} \sqsubseteq \{_{R,O'} \Rightarrow GPNF(R,O) \subseteq GPNF(R,O').$$

The proof is omitted. It is a trivial consequence of the definitions.
Now, all notions are available to define valid NR/T-nets:

Definition 4.8 (*Valid NR/T-Net*). Let $NRT = (NN, P, AI, TI, M^0)$ be a nested-relation/transition-net such that $NN = (\varphi, N)$ is an NR-net, $\varphi = (U, T, d, dom)$, and $N = (PL, TR, F)$.
NRT is called a *valid NR/T-net* if the following three conditions hold:

(1) For each place R and for all pairs of incoming arcs $(tr_1, R), (tr_2, R) \in F$, $AI(tr_1, R) = (\tau_1, O_1)$ and $AI(tr_2, R) = (\tau_2, O_2) \Rightarrow O_1 = O_2$. In this case we have the following definition: if R has at least one incoming arc then let O be the unique ordering of the term $AI(tr, R) = (\tau, O)$ assigned to any incoming arc (tr, R). In this case we say *O is the ordering of R*, abbreviated as $o(R)$. Otherwise, if R has no incoming arc, we define $o(R)$ as the most restrictive ordering such that $M^0(R) \in GPNF(R,o(R))$ (this ordering is unique!).
(2) Assume that (1) holds, i.e., each place R is assigned a unique ordering $o(R)$. Then, we require that M^0 is a *valid marking*, i.e., for each place R: $M^0(R) \in GPNF(R,o(R))$.
(3) Assume that (1) holds. Then, we require for each place R and for each labeled term (τ,O) assigned to an outgoing arc $(R, tr) \in F$ (i.e., $AI(R, tr) = (\tau, O)$):

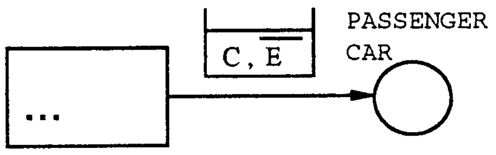
$$\{_{R,o(R)} \sqsubseteq \{_{R,O}.$$

The first two restrictions are very natural. One may ask whether the third restriction, which obviously restricts the flexibility of the design of NR/T-nets, is a severe one. This is discussed in the next example.

Example 4.9. (1) We regard the transition depicted in Figure 8. Obviously, condition (1) is satisfied because each place has at most one incoming arc. Now consider the places PROJECT-PROPOSAL, DEPARTMENT, and PROJ-DEP.
The place PROJECT-PROPOSAL has no incoming arc, and we assume the most restrictive ordering for this place, such that the PROJECT-PROPOSAL instance of Figure 7 is in GPNF. This holds for the object order. Thus, each composite attribute must be a containment attribute. DEPARTMENT has one incoming arc. The ordering of DEPARTMENT is also the object order, because in the respective Filter Table of the incoming arc all attributes are specified as containment attributes. The same argument holds for PROJ-DEP. It is easy to see that the instances of Figure 7 are PNF instances. Thus they are GPNF instances with respect to these orderings. As the object order is the most restrictive ordering, all places satisfy condition (3).

CAR	
CAR TYPE	EQUIPMENT
	EQU. PART
Mercedes 230E	{airbag, leather seats}
Mercedes 230E	{airbag, leather seats, sunshine roof}
Golf GTD	{radio, sunshine roof}
...	...

(a)

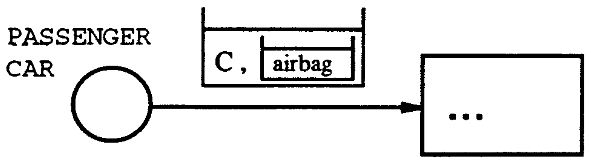


(b)

Fig. 9. (a) A CAR instance; (b) incoming arc.

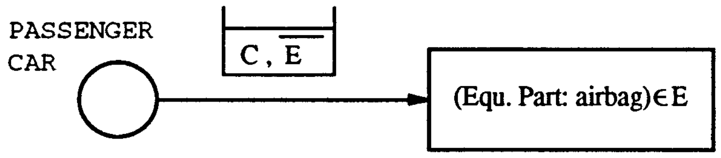
(2) We give another example to motivate condition (3) in Definition 4.8. We regard the CAR example shown in Example 3.1.2 (Figure 9(a)), and we assume that tuples are inserted into this instance by the Filter Table (Figure 9(b)).

In this case EQUIPMENT is an equality attribute because of Definition 4.8 (1). Now it is not allowed to specify the selection of tuples by an outgoing arc in the following way:



This Filter Table violates condition (3) in Definition 4.8 because the ordering of the incoming arc in Figure 9(b) is less restrictive than the ordering of the outgoing arc. This may cause ambiguities. If, for example, C is instantiated by the value "Mercedes 230E" there are *two* possible choices to delete the specified tuple.

Hence, we must postulate that a *whole* tuple must be specified and deleted, i.e., the corresponding EQUIPMENT term must be a closed term. A unique selection can be achieved in the following way, where a whole tuple is deleted:



This kind of ordering is enforced by Definition 4.8 (3), which thus can be summarized as follows for places with at least one incoming arc:

—*If a term in the Filter Table of the incoming arc is a closed term, then the corresponding term in the Filter Table of each outgoing arc must also be a closed term!*

In the subsequent definitions, the semantics of NR/T-nets is formally specified. At first we define the truth value of logical formulas that are inscribed to transitions. This is almost standard.

Definition 4.10 (*Evaluation of Logical Formulae*). The truth value $TV_\Theta(f)$ of a logical formula f with respect to an instantiation Θ is recursively defined as follows:

$TV_\Theta \mathrm{LF} \rightarrow \{\text{true, false}\}$ where
$TV_\Theta(\mathit{true}) = \text{true}$,
$TV_\Theta(p(v_1, \ldots, v_n)) = \text{true iff } (\Theta(v_1), \ldots, \Theta(v_n)) \in I(p)$,
$TV_\Theta(f_1 \wedge f_2) = \text{true iff } TV_\Theta(f_1) = \text{true and } TV_\Theta(f_2) = \text{true}$,
$TV_\Theta(f_1 \vee f_2) = \text{true iff } TV_\Theta(f_1) = \text{true or } TV_\Theta(f_2) = \text{true}$,
$TV_\Theta(\neg f_1) = \text{true iff } TV_\Theta(f_1) = \text{false}$.

We now give the formal definition for a transition to be enabled. We use the following notation: the set $\{s \in PL | (s, tr) \in F\}$ of *input places* of a transition tr is denoted as $\cdot tr$; the set $\{s \in PL | (tr, s) \in F\}$ of *output places* of a transition tr is denoted as $tr\cdot$.

Definition 4.11 (*Enabled Transition*). Let $NRT = (NN, P, AI, TI, M^0)$ be an NR/T-net such that $NN = (\varphi, N)$ is an NR-net, $\varphi = (U, T, d, dom)$, and $N = (PL, TR, F)$. Let tr be a transition and Θ an instantiation of all variables in the environment of tr and M a valid marking. Θ is assumed to be valid for all labeled terms in the environment of tr with respect to their orderings.

The transition *tr is enabled w.r.t.* Θ *and the marking M* if the following conditions hold:

(1) $R \in \cdot tr$: If $AI(R, tr) = (\tau, O)$ then $\tau\Theta \lbrace_{R,O} M(R)$.
(2) $R \in tr\cdot$: If $AI(tr, R) = (\tau, O)$ then $\tau\Theta \cap_{R,O} M(R) \neq \tau\Theta$.
(3) $TV_\Theta(TI(tr)\Theta) = \text{true}$.

The first condition says that the instantiated term $\tau\Theta$ of an incoming arc must be completely contained in the instance $M(R)$ of the adjacent place R. Due to the second condition, it is required that the term $\tau\Theta$ of an outgoing arc not be contained in the instance $M(R)$ of the adjacent place, and the last condition says that the transition inscription evaluates to true.

If a transition is enabled it can occur. The effects of a transition occurrence are described in the next definition.

Definition 4.12 (*Transition Occurrence*). Let $NRT = (NN, P, AI, TI, M^0)$ be an NR/T-net such that $NN = (\varphi, N)$ is an NR-net, $\varphi = (U, T, d, dom)$, and $N = (PL, TR, F)$. Let tr be a transition and Θ a valid instantiation of all variables in the environment of tr and M a valid marking. Let tr be enabled w.r.t. Θ and M.

Then, the *occurrence of tr* is defined by (1)–(3). It implies that the marking M is changed in one indivisible step to a new marking M' as follows:

(1) $R \in PL \setminus (tr{\cdot} \cup {\cdot}tr)$: $M'(R) = M(R)$,

(2) $R \in {\cdot}tr$: If $AI(R,tr) = (\tau, O)$ then $M'(R) = M(R) \cap_{R,O} Neg_{R,O}(\tau\Theta)$,

(3) $R \in tr{\cdot}$: If $AI(tr,R) = (\tau, O)$ then $M'(R) = M(R) \cup_{R,O} \tau\Theta$.

Note that the lattice operations described in Section 3 are essential in this definition. In (1), all instances being not adjacent to the transition remain unchanged. In (2) and (3), all Filter Tables of the incoming arcs cause a delete operation in the respective instances, and, vice versa, the Filter Tables of the outgoing arcs cause an insertion into the instances being adjacent to these arcs.

Furthermore, if the marking M is valid, i.e., each instance assigned to a place R is a GPNF instance with respect to $o(R)$, then the same holds for the new marking M'. This is due to Proposition 4.7, Definition 4.8 (c), and to the definition of the lattice operations $\cap_{R,O}$ and $\cup_{R,O}$.

Now we are going to complete the running example. The next example shows a valid NR/T-net corresponding to the main part of the NR-net in Figure 5.

Example 4.13 (*The Running Example*). Starting with a certain marking, the occurrences of transitions may generate marking sequences. We now explain one possible marking sequence for the NR/T-net given in Figure 10. For the initial marking M^0 given in Figure 7 (the markings of PROJ-DEP and PROJ-EMP are empty instances), the transition Find-department-with-required-equipment is enabled for the project-proposal p12 and department d1. Variable P is instantiated with p12, and D with d1. Both occurrences of variable E are instantiated with the set {e2, e5} to express that the required equipment is a subset of the available equipment in the department d1.

The Filter Table assigned to the arc leading from the place PROJECT-PROPOSAL to the transition Find-department-with-required-equipment only contains closed terms. Hence, the whole tuple is removed from the instance in place PROJECT-PROPOSAL when the transition occurs. The variables in the Filter Table of the other incoming arc are instantiated with the values d1, {1, 3, 5, 11}, {e2, e5}, {p11, p33}, respectively. Only the EQUIPMENT set

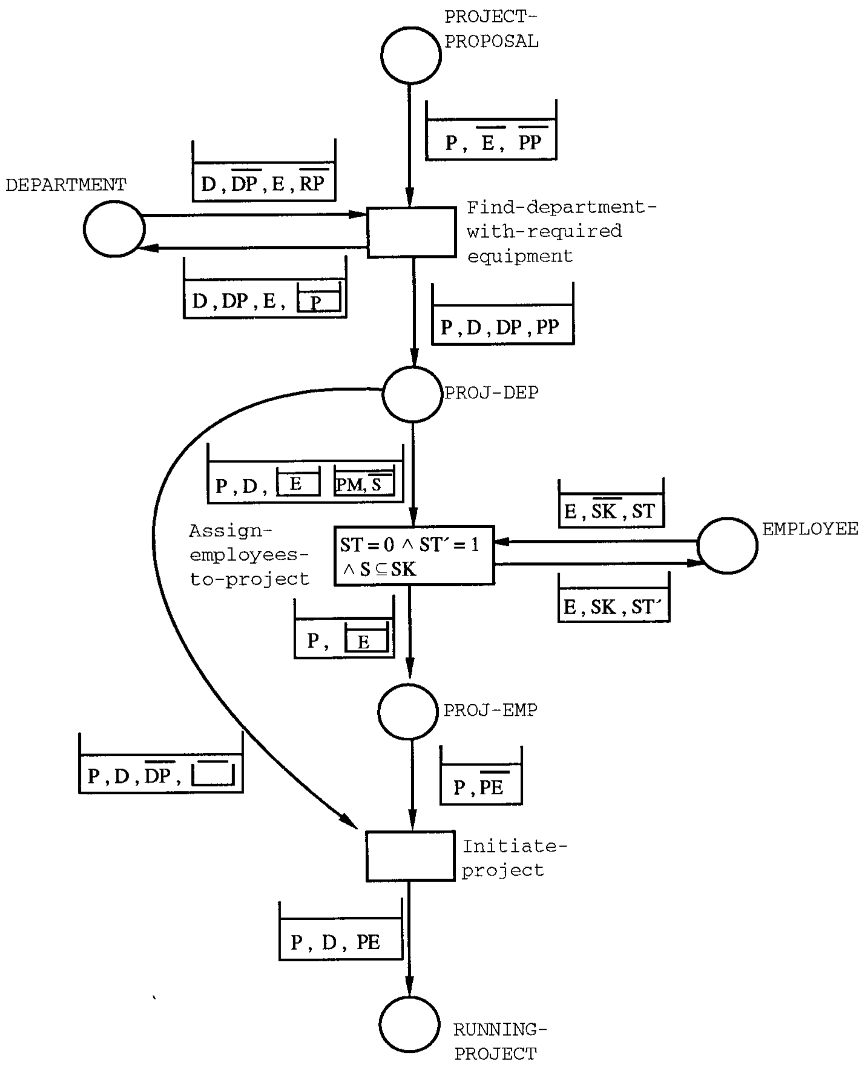


Fig. 10. Example NR/T-net (without initial marking).

{e2, e5} is deleted (because of the ordering specified by the Filter Table). Due to the outgoing arc two things happen simultaneously when the transition occurs. The EQUIPMENT set {e2, e5} is reinserted into the DEPARTMENT instance, and the new PRO# is also inserted into the DEPARTMENT instance. Furthermore, the tuple shown in Figure 11 is inserted into the instance of the place PROJ-DEP. Note that all changes related to the transition Find-department-with-required-equipment are done in one indivisible step.

The occurrence of transition Find-department-with-required-equipment in marking M^0 leads to a new marking M^1. The instance of the place PROJ-DEP is shown in Figure 11.

PROJ-DEP				
PRO#	DEP#	DEP-PERS	PP-MEMBERS	
		EMP#	PPM#	SKILLS
				SKILL
p12	d1	{1,3,5,11}	{(1, (2,	{s1, s3, s4}), {s1, s2, s3})}

Fig. 11. Marking M^1 of place PROJ-DEP.

Now regard the transition Assign-employees-to-project. For the new marking M^1 this transition is enabled for the given tuple in PROJ-DEP and an employee (tuple of the EMPLOYEE instance) who has (at least) the skills of one of the required project members. This implies that the employee's set of skills must contain either the set {s1, s3, s4} or the set {s1, s2, s3}. Furthermore, these employees must have the status 0 (i.e., they are not yet involved in any project). Only employees 1 and 3 satisfy these conditions. Hence, possible instantiations for variable E are 1 or 3.

Let us assume that Assign-employee-to-project first occurs for employee 1 with respect to the first required project member. This means the variables are instantiated as follows: $\Theta(P) = p12$, $\Theta(D) = d1$, $\Theta(E) = 1$, $\Theta(PM) = 1$, $\Theta(S) = \{s1, s3, s4\}$, $\Theta(SK) = \{s1, s2, s3, s4, s5\}$, $\Theta(ST) = 0$, $\Theta(ST') = 1$. Then the transition occurs, and (among others) the value 1 and the tuple (1, {s1, s3, s4}) are deleted from the DEP-PERS value and the PP-MEMBERS value of the PROJ-DEP instance, respectively. The specified EMPLOYEE tuple is deleted as a whole from the EMPLOYEE instance. The tuple (p12, {1}) is inserted into the PROJ-EMP instance, and the EMPLOYEE tuple is reinserted into the EMPLOYEE instance with the new status $ST' = 1$.

Assign-employee-to-project then occurs in a very similar way for employee 3 with respect to the second required project member. The resulting marking M^2 of the instances PROJ-DEP and PROJ-EMP is shown in Figure 12.

For the marking M^2, the transition Initiate-project is enabled if the variables are instantiated as follows: $\Theta(P) = p12$, $\Theta(D) = d1$, $\Theta(DP) = \{5, 11\}$, and $\Theta(PE) = \{1, 3\}$. According to the Filter Table assigned to the arc between PROJ-DEP and Initiate-Project, the value of PP-MEMBERS of the selected tuple in PROJ-DEP must be the empty set, which is actually true for M^2. This means that each required project member in the former project proposal has been "matched" with an employee in the selected department d1.

When Initiate-Project occurs, the selected tuples are removed from PROJ-DEP and PROJ-EMP, and the new tuple (p12, d1, {1, 3}) is inserted into RUNNING-PROJECT. This yields the marking M^3 of place RUNNING-PROJECT shown in Figure 13.

To conclude this chapter we discuss the major properties of our approach.

—*NR/T-nets are a proper extension of Pr/T-nets*, because for first-normal form relations each NR/T-net can be represented by an equivalent

PROJ-DEP				
PRO#	DEP#	DEP-PERS	PP-MEMBERS	
		EMP#	PPM#	SKILLS
				SKILL
p12	d1	{5, 11}	{ }	

PROJ-EMP	
PRO#	EMPLOYEES
	EMP#
p12	{1, 3}

Fig. 12. Marking M^2 of places PROJ-DEP and PROJ-EMP.

RUNNING-PROJECT		
PRO#	DEP#	RP-MEMBERS
		EMP#
p11	d1	{5}
p12	d1	{1, 3}
p21	d3	{6, 9}
p23	d4	{4, 7}
p29	d5	{10, 12}
p33	d1	{11}

Fig. 13. Marking M^3 of place RUNNING-PROJECT.

Pr/T-net. For 1NF relations, only whole tuples are deleted and inserted from/into an instance, and each transition is only inscribed with formulae not containing set operators and set terms. *NR/T-nets have a formal semantics*. The most subtle point is the formal definition of insert and delete operations by means of different lattices and partial orders. This is also an extension of Pr/T-nets, because for Pr/T-nets the corresponding operations can be expressed by union, intersection, and complement operations of a Boolean set lattice.

— NR/T-nets are easy to understand. This is our personal opinion and must be viewed relative to the complexity of nested relational structures. For example, algebraic query languages for nested relations are more complicated (and less expressive) than our Filter Table notation.

—*NR/T-nets are very expressive* because of the following three reasons. (This article is not intended to characterize the expressive power in detail, but the reader may have got an impression by the last example.)

(1) In the net formalism it is possible to specify sequential, alternative, and concurrent state transitions.

(2) NR/T-nets allow one to specify arbitrary (sub-) structures of nested instances. It is possible to delete and to insert these structures from/into instances, respectively.

(3) They allow one to specify complex selection conditions as set membership, set equality, subset, etc. These conditions can be expressed by the Filter Table formalism and, on the other hand, by transition inscriptions.

5. RELATED APPROACHES

We have presented a novel approach to describe structural as well as behavioral aspects of information systems within one approach, the so-called NR/T-nets. The major contribution is the formal definition of insert and delete operations for nested relations and of valid NR/T-nets and their transition occurrences.

In the original definition of predicate/transition nets [Genrich and Lautenbach 1981], places represent normalized relation schemes (i.e., schemes with simple attributes only). Some approaches also allow set-, list-, or tuple-valued attributes in the marking of places [Heuser and Richter 1992; Horndasch et al. 1985; Oberweis 1988; Richter and Durchholz 1982]. In these approaches, it is only possible to access *whole tuples* in the environment of a transition, i.e., to delete whole tuples and to insert whole tuples. Direct access to substructures is not supported. Our direct access to substructures could be simulated, e.g., by first removing a tuple from a place, then manipulating the respective substructure, and finally inserting the whole tuple back into its original place. Note that in this case concurrency is unnecessarily restricted, since no two transitions can access the same tuple at the "same time," whereas in our novel approach it is possible for different transitions to access different (disjoint) subtuples of the same tuple in one step.

In the literature, several inscription languages were introduced for places, arcs, and transitions in high-level Petri nets. Some of these languages are

—database language oriented [Horndasch et al. 1985; Richter and Durchholz 1992; Sibertin-Blanc 1986],

—logic oriented [Genrich and Lautenbach 1981],

—algebraic [Battiston et al. 1988; Billington 1989; Krämer 1987; Reisig 1991],

—programming language oriented [Albert et al. 1989; Baldassari and Bruno 1988].

For many of these languages a formal semantics has not been provided, and—in contrast to our Filter Table formalism—none of them matches all of the properties discussed at the end of Section 4.

Our concept of Filter Tables has been influenced by Zloof [1982] and Luo and Yao [1981] where similar concepts are used to model office procedures. *Office-By-Example* [Zloof 1982] is an extension of Query-By-Example [Zloof 1975] to describe simple structured office procedures consisting of operations on normalized relations only. *Form-Operation-By-Example* (*FOBE*) [Luo and Yao 1981] also allows one to compose programs from single database queries, which are specified in a Query-By-Example-like manner. FOBE additionally supports nested relations, which are typical for office documents. In contrast to our approach, FOBE is not based on different partial orders. The last point makes it less expressive than our Filter Tables. In the graphical formalism, FOBE only provides existentially quantified access to nested sets, and it is impossible to specify set equality or a subset relationship.

In both approaches the provided control structures are too restrictive to model complex procedures adequately.

6. CONCLUSIONS AND FUTURE RESEARCH

Our major intention of this research project was to provide a formal executable database model that includes database behavior. This is the basis for, e.g., verification of system specification, code generation, reachability analysis, and parallel execution of database operations.

Another important feature is the simulation of processes in a database environment. Simulation with an NR/T-net can be done by repeating the following two steps. For a given marking the simulator computes all enabled transitions. Then a subset of the enabled transitions is selected to occur in one step. The selection of transitions to occur may be done automatically (according to a certain occurrence policy, e.g., always fire a maximal subset of enabled transitions) or manually. Currently an existing simulator for predicate/transition nets [Mochel et al. 1993; Oberweis 1988] is adapted to NR/T-Nets.

Furthermore, the concept of NR/T-nets is being integrated into an existing methodology for information system design based on Petri nets and semantic data modeling. This methodology is supported by an existing development tool, INCOME [Lausen et al. 1989; Oberweis et al. 1994] which provides graphical editors, dictionary support, generators, and prototyping facilities. Based on INCOME, the NR/T-net concept is to be validated in practical case studies.

There are also some open questions:

—Questions related to available analysis techniques for NR/T-nets have not been addressed in this article. For several kinds of high-level Petri nets, methods for computing invariants [Jensen and Rozenberg 1991] have been proposed as the basis for formal analysis, e.g., deadlock analysis and reachability analysis. These results can be transferred to NR/T-nets by defining a homomorphism between both types of net.

—The expressive power of Filter Tables should be investigated in detail. In Sander [1992; 1993] it was shown that most algebraic operations for nested relations (known from the literature) can be expressed easily in a similar (rule-based) formalism. These results also apply to NR/T-nets.

—NR/T-nets can be extended by assigning a nonempty *set of Filter Tables* to each arc and not only a single one. This makes it possible to access the instance of a place by different Filter Tables (possibly with different orderings) in one step. The problem is that these Filter Tables may be conflicting in the sense that they can access overlapping (sub-) tuples of an instance. It is an interesting task to look for an appropriate notion of *serializability* in order to exclude these conflicts.

—For practical applications additional high-level constructs for, e.g., starting, stopping, and interrupting activities and processes can be included into the language. Furthermore, an application-oriented role concept can be integrated into NR/T-nets. These extensions already exist in other Petri net-based languages and should be part of the development tool's user interface.

APPENDIX

List of Symbols

AI	function that assigns a Filter Table to each arcs of an NR/T-net
dom	mapping from U_{simple} into $\mathcal{d}$
$d_1, d_2, \ldots$	domains
$\mathcal{d}$	set of domains
D	database scheme
φ	signature of a nested-relation-net
F	flow relation of a net, $F \subseteq (PL \times TR) \cup (TR \times PL)$
$GPNF(R,O)$	set of all GPNF instances over scheme R w.r.t. O
Θ	instantiation
$Ins(R)$	set of all instances over scheme R
LF	set of possible logical formulas over a set of predicate symbols P
$LT(R)$	set of labeled terms (Filter Tables) of type R
$L_{R,O}$	lattice of GPNF instances over scheme R
M	marking (of an NR/T-net)
M^0	initial marking
N	net
NN	nested-relation-net
NRT	nested-relation/transition-net (NR/T-net)
$Neg_{R,O}(r)$	negative of an instance r
O	order definition
$o(R)$	ordering of a place R in an NR/T-net
PL	finite set of places of a net
$PNF(R)$	set of PNF instances over scheme R
$r, r_1, r_2, \ldots$	top-level instances
$R, R_1, R_2, \ldots$	scheme and composite attribute names, places in an NR/T-net
$\cdot tr$	input places of a transition tr
$tr\cdot$	output places of a transition tr
T	set of scheme equations

TI	function that assigns a logical formula to each transition of a NR/T-net
TR	finite set of transitions
τ	term
TV_{Θ}	function that assigns a truth value to each logical formula
U	universe
U_{simple}	set of simple attributes of U
Var	nonempty set of variables
$\subseteq$	inclusion order
$\leq$	object order
$\{_{R}$	ordering for instances of type R in an order definition O
$\cup_{R,O}$	union (least upper bound) in the set $GPNF(R,O)$
$\cap_{R,O}$	intersection (greatest lower bound) in the set $GPNF(R,O)$
$\sqsubseteq$	relation between orderings to compare their restrictiveness
$0_{R,O}$	least GPNF instance w.r.t. the scheme R and order definition O
$1_{R,O}$	greatest GPNF instance w.r.t. the scheme R and order definition O

ACKNOWLEDGMENT

We wish to thank Wolffried Stucky, Wolfgang Weitz, and the anonymous referees for many useful suggestions helping us to improve the quality of the article.

REFERENCES

Abiteboul, S. and Bidoit, N. 1986. Non first normal form relations: An algebra allowing data restructuring. *J. Comput. Syst. Sci. 33,* 361–393.

Albert, K., Jensen, K., and Shapiro, R. 1989. DESIGN/CPN. A tool package supporting the use of colored Petri nets. *GI Petri Net Newslett. 32,* 22–26.

Baldassari, M. and Bruno, G. 1988. An environment for object-oriented conceptual programming based on PROT nets. In *Advances in Petri Nets 1988,* G. Rozenberg, Ed. Lecture Notes in Computer Science, vol. 340, Springer-Verlag, Berlin, 1–19.

Bancilhon, F. and Koshafian, S. 1989. A calculus for complex objects. *J. Comput. Syst. Sci. 38,* 326–340.

Barron, J. 1982. Dialogue and process design for interactive information systems using Taxis. In *Proceedings of the SIGOA Conference on Office Information Systems.* ACM, New York, 12–20.

Battiston, E., De Cindio, F., and Mauri, G. 1988. OJBSA Nets: A class of high level nets having objects as domains. In *Advances in Petri Nets 1988,* G. Rozenberg, Ed. Lecture Notes in Computer Science, vol. 340. Springer-Verlag, Berlin, 20–43.

Beeri, C. and Kornatzky, Y. 1990. The many faces of query monotonicity. In the *International Conference on Extending Database Technology.* Lecture Notes in Computer Science, vol. 416. Springer-Verlag, Berlin, 120–135.

Billington, J. 1989. Mony-sorted high level nets. In *Proceedings of the 3rd Workshop on Petri Nets and Performance Models.* IEEE, New York, 166–179.

BIRKHOFF, G. 1973. *Lattice Theory.* 3rd ed. AMS Colloquium Publication, vol. 25. AMS, Providence, R.I.

BRODIE, M. L., MYLOPOULOS, J., AND SCHMIDT, J. W., Eds. 1984. *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages.* Springer-Verlag, Berlin.

BRAUER, W., REISIG, W., AND ROZENBERG, G., Eds. 1987. *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986. Part I.* Lecture Notes in Computer Science, vol. 254. Springer-Verlag, Berlin.

CODD, E. F. 1970. A relational model for large shared data banks. *Commun. ACM 13,* 6, 377–387.

GENRICH, H. J. AND LAUTENBACH, K. 1981. System modelling with high level Petri Nets. *Theor. Comput. Sci. 13,* 109–136.

HEUSER, C. A. AND RICHTER, G. 1992. Constructs for modeling information systems with Petri Nets. In *Application and Theory of Petri Nets 1992,* K. Jensen, Ed. Lecture Notes in Computer Science, vol. 616. Springer-Verlag, Berlin, 224–243.

HORNDASCH, A., STUDER, R., AND YASDI, R. 1985. An approach to (office) information systems design based on general net theory. In *Proceedings of the IFIP TC8.1 TFAIS85 Conference.* North-Holland, Amsterdam.

JENSEN, K. 1991. Coloured Petri Nets: A high level language for system design and analysis. In *Advances in Petri Nets 1990,* G. Rozenberg, Ed. Lecture Notes in Computer Science, vol. 483. Springer-Verlag, Berlin, 342–416.

JENSEN, K. AND ROZENBERG, G., Eds. 1991. *High Level Petri Nets.* Springer-Verlag, Berlin.

KRAMER, B. 1989. Concepts, syntax, and semantics of SEGRAS. A specification language for distributed systems. GMD-Bericht. Nr. 179, R. Oldenbourg Verlag, München, Wien.

LAUSEN, G., NEMETH, T., OBERWEIS, A., SCHÖNTHALER, F., AND STUCKY, W. 1989. The INCOME approach for conceptual modelling and prototyping of information systems. In *Proceedings of CASE '89. The 1st Nordic Conference on Advanced Systems Engineering.*

LUO, D. AND YAO, S. B. 1981. Form operation by example—A language for office information processing. In *Proceedings of the International Conference on Management of Data* (*ACM-SIGMOD*), Y. E. Lien, Ed. ACM, New York, 212–223.

MAKINOUCHI, A. 1977. A consideration on normal form of not-necessarily-normalized-relation in the relational data model. In *Proceedings of the International Conference on Very Large Databases.* VLDB Endowment Press, Saratoga, Calif., 447–453.

MOCHEL, T., OBERWEIS, A., AND SANGER, B. 1993. INCOME/STAR: The Petri Net simulation concepts: Systems analysis—modelling—simulation. *J. Model. Simul. Syst. Anal. 13,* 21–36.

OBERWEIS, A. 1988. Checking database integrity constraints while simulating information system behaviour. In *Proceedings of the 9th European Workshop on Application and Theory of Petri Nets.* 299–308.

OBERWEIS, A., SCHERRER, G., AND STUCKY, W. 1994. INCOME/STAR: Methodology and tools for the development of distributed information systems. *Inf. Syst. 19,* 8, 643–660.

PECKHAM, J. AND MARYANSKI, F. 1988. Semantic data models. *ACM Comput. Surv. 20,* 3 (Sept.), 153–189.

PETERSON, J. L. 1981. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, N.J.

REISIG, W. 1985. *Petri Nets.* EATCS Monographs on Theoretical Computer Science, vol. 4. Springer-Verlag, Berlin.

REISIG, W. 1991. Petri nets and algebraic specifications. *Theor. Comput. Sci. 80,* 1–34.

RICHTER, G. AND DURCHHOLZ, R. 1982. IML inscribed high level Petri Nets. In *Information Systems Design Methodologies: A Comparative Review,* T. W. Olle, H. G. Sol, and A. A. Verrijin-Stuart, Eds. North-Holland, Amsterdam, 335–368.

ROTH, M., KORTH, H., AND SILBERSCHATZ, A. 1988. Extended algebra and calculus for nested relational databases. *ACM Trans. Database Syst. 13,* 4, 389–417.

SANDER, P. 1992. Boolean lattices of nested relations as a foundation for rule-based database languages. *Data Knowl. Eng. 8,* 2, 93–130.

SANDER, P. 1993. An order-based rule language for nested relations. Ph.D. dissertation, Univ. Karlsruhe, Karlsruhe, Germany. In German.

SCHEK, H.-J. AND SCHOLL, M. H. 1986. The relational model with relation-valued attributes. *Inf. Syst. 11,* 2, 137–147.

SIBERTIN-BLANC, C. 1986. High level Petri Nets with data structure. In *Proceedings of the 6th Workshop on Petri Nets.*

THOMAS, S. AND FISCHER, P. C. 1986. Nested relational structures. In *Advances in Computing Research III, The Theory of Databases,* P. Kanellakis, Ed. JAI Press, Greenwich, Conn., 269–307.

ZISMAN, M. D. 1977. Representation, specification and automation of office procedures. Ph.D. dissertation, Univ. of Pennsylvania, Wharton School of Business, Philadelphia, Pa.

ZLOOF, M. M. 1975. Query-By-Example: The invocation and definition of table and forms. In *Proceedings of the International Conference on Very Large Data Bases,* D. S. Kerr, Ed. VLDB Endowment Press, Saratoga, Calif., 1–24.

ZLOOF, M. M. 1982. Office-By-Example: A business language that unifies data and word processing and electronic mail. *IBM Syst. J. 21,* 3, 272–304.

Received December 1992; revised August 1994; accepted July 1995