Modelling the Behaviour of NF^2-Database Applications*

Andreas Oberweis, Peter Sander and Wolffried Stucky

Universität Karlsruhe
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
D-W-7500 Karlsruhe
Germany

Tel.: +(49)(721)6084283
Fax: +(49)(721)693717
E-Mail: {oberweis|sander|stucky@aifb.uni-karlsruhe.de}

Topics
Information system design, Petri net, NF^2-database

Abstract
In this paper a new type of high level Petri nets is introduced for modelling procedures in NF^2-database applications. Each net defines a class of possible system procedures, i.e. sequences of (possibly concurrent) operations, in an NF^2-database application. Places (predicates) in these so-called nested relation/transition nets (NR/T-nets) represent schemes of unnormalized relations ("nested relations"). The marking of each place is a (possibly empty) nested relation of the respective type. Each transition represents a specific type of operation on the relations in the adjacent places. These operations may not only operate on whole tuples of a given relation but also on "subtuples" of existing tuples. Arcs in a net are inscribed with so-called filter tables which allow (together with an optional logical expression as transition inscription) to formulate conditions on the specified (sub-) tuples.

1 Introduction

The conceptual design of information systems must include static, object related system aspects as well as dynamic, activity related system aspects. *Static system aspects* concern the structure of objects, relationships between objects and static semantic integrity constraints. *Dynamic system aspects* concern the operations on objects (creation, deletion, modification of objects), the synchronisation of operations and dynamic semantic integrity constraints.

First, we consider static system aspects. One of the most popular data models which has become the formal basis of many commercially available database systems is the *relational data model* originated by Codd [Cod70]. However, 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) the first normal form is not suitable [Mak77], and considerable interest arose to investigate database models which allow for modelling

* Extended version of a paper which will appear in J.E. Urban (Ed.): Proc. Seventeenth Annual International Computer Software and Applications Conference COMPSAC 1993, Phoenix/Arizona

complex objects. The major extension of the relational data model in this direction is to drop the first normal form assumption and to allow for non-atomic attribute values. In particular, the construction of sets and tuples from atomic values is a main 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. not decomposable) or also a nested relation. Thus, a nested relation is a recursively defined data structure which is suitable for modelling complex objects. In this paper nested relations are used to model the structural aspects of information systems.

On the other hand, dynamic system aspects can be formally described by Petri nets. They can be used to model sequential, alternative and concurrent system activities, and they support concepts for a stepwise formalization as well as formal methods for the stepwise refinement of system descriptions. There exist different types of nets: Low level nets (*condition event nets, place transition nets* [BRR87]) provide a simple interpretation of net components and are easy to understand but are rather hard to handle in case of large systems with complex behaviour. In high level nets (e.g. *predicate transition nets* [Gen87], *coloured Petri nets* [Jen92], see also [JeR91]) on the other hand the basic components are more expressive and provide a more compact description than low level nets. They allow for the integration of static, object related system aspects when modelling system behaviour. It is generally agreed that, for practical applications, high level nets are preferable to low level nets. Several Petri net tools like graphical editors, simulators and analyzers are commercially available [Fel92].

In this paper we introduce a novel type of high level Petri nets, the so-called *nested-relation/transition nets*. Each net defines a class of possible system procedures, i.e. sequences of (possibly concurrent) operations, in a complex object database application. *Places* (*predicates*) represent schemes of unnormalized relations ("nested relations"). The *marking* of each place is a (possibly empty) nested relation of the respective type. Each *transition* represents a specific type of operation on the relations in the adjacent places. These operations may not only operate on whole tuples of a given relation but also on "subtuples" of existing tuples. Arcs in a net are inscribed with so-called *filter tables* which allow (together with an optional logical expression as transition inscription) to formulate conditions on the specified (sub-)tuples.

The paper is structured as follows: The next section describes some properties of operations in complex object databases. Based on these discussions a semantics is proposed for insert and delete operations in complex object databases. Filter tables are introduced as a new concept to express these operations. Section 3 introduces Petri nets and nested-relation/transition nets (NR/T-nets), respectively. In Section 4 it is shown that *fact transitions* in NR/T-nets can be used for modelling static semantic integrity constraints in complex object databases. Section 5 concludes the paper and discusses some open problems.

Due to space limitations we cannot give all formal definitions needed for the introduction of nested-relation/transition nets. The interested reader is referred to [ObS92] which is an extended, formal version of this paper.

2 Operations in NF^2-Databases

In this section we describe nested relations, a simple but important and mathematically well-founded class of complex objects. We show how so-called filter tables will serve as the basis for the specification of different kinds of insert and delete operations for nested relations.

We distinguish two aspects of a nested relational database: 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.

For the definition of the structural part of nested relations we use a notation similar to [ThF86]. 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 $R := (R_1,...,R_n)$, where $n \geq 1$ and where the set $R, R_1,...,R_n$ 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. R is called a *scheme* and each R_i is called an *attribute* of R. An attribute not occuring on the left-hand side of any scheme equation is a *simple attribute*, otherwise it is a *composite attribute*. A scheme not occuring on the right-hand side of any scheme equation is also called a *top-level scheme*. Let $(R_1,...,R_k)$ be the tuple consisting of all top-level-schemes of T. Then, the (meta-) scheme equation $D := (R_1,...,R_k)$ specifies the database scheme D w.r.t T.

Let U_{simple} denote all simple attributes of U, and let $\dot{d} = \{d_1,d_2,...\}$ be a nonempty set of sets, the *domains*. Then *dom* is a mapping from the set U_{simple} of all simple attributes into $\dot{d}$. 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 with a scheme equation $R{:=}(R_1,...,R_n)$ then $R{:}\{(\tau_{11},...,\tau_{1n}),...,(\tau_{m1},...,\tau_{mn})\}$ is an instance of type R, where $m \geq 0$ and $\forall i \in \{1,...,m\}$ and $j \in \{1,...,n\}$: 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 $r, r_1, r_2,...$ An element $(\tau_{11},...,\tau_{1n})$ of the value of an instance $R{:}\{(\tau_{11},...,\tau_{1n}),...,(\tau_{m1},...,\tau_{mn})\}$ is also called a tuple of type R.

Example:
An information system for the management of a conference is to be designed (compare [CRIS82]). We only consider the activities related to the program committee here:
Papers are submitted to the program committee chairperson (PC chairperson). Each paper has a title, a non-empty set of authors and a topic given as a single keyword. The program committee consists of a set of program committee members (referees). For each referee a name, a set of topics and a set of already assigned papers is given. Each paper is identified by a paper number (PAP#). For every paper the PC chairperson selects three different referees such that the paper´s topic belongs to every selected referee´s set of topics. The referees propose to reject or to accept a paper. The PC chairperson decides to reject or to accept a paper depending on the vote of the majority of the

referees. Finally, accepted papers are collected to sessions with respect to their topics. For a conference a set of topics is given.

The following scheme equations describe the structural system aspects:

```
SUBMITTED-PAPER := (TITLE,TOPIC,AUTHORS)
AUTHORS := (AUTHOR)
REFEREE := (REF#,NAME,TOPICS,PAPERS)
TOPICS := (TOPIC)
PAPERS := (PAP#)
PAPER-COUNT := (PAP#)
PAPER-TO-BE-REFEREED := (PAP#,TITLE,TOPIC,AUTHORS,REFEREE-SET)
REFEREE-SET := (REF#)
REJECTED-PAPER := (TITLE,TOPIC,AUTHORS)
ACCEPTED-PAPER := (TITLE,TOPIC,AUTHORS)
REFEREE-REPORT := (REF#,PAP#,VOTE)
SESSION := (TOPIC,TALKS)
TALKS := (TITLE,AUTHORS)
```

REFEREE, e.g., is a top-level scheme with simple attributes REF# and NAME and composite attributes TOPICS and PAPERS. TOPICS is a scheme with attribute TOPIC, PAPER-COUNT is a scheme with attribute PAP#. dom(VOTE) = {0,1}, *dom*(NAME) = *dom*(TOPIC) = STRING, *dom*(REF#) = *dom*(PAP#) = INTEGER. Note, that the value "1" for the VOTE-attribute in REFEREE-REPORT is interpreted as *accept* and the value "0" as *reject*.

Instead of the linear syntax mentioned above we shall use a more readable, tabular representation for top-level instances. Figure 2.1 shows the tabular representation of the REFEREE-instance, Figure 2.2 shows an instance for the PAPER-TO-BE-REFEREED-scheme.

■

In the sequel we shall discuss operations for nested relational instances. This can be motivated as follows: In order to describe the behaviour of a database system, it must be specified how tuples can be inserted into and deleted from an instance. 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. These operations can be used to define insert and delete operations for nested relations. Later we define the occurrence rule for transitions in nested-relation/transition nets based on these operations.

REFEREE			
REF#	NAME	TOPICS	PAPERS
		TOPIC	PAP#
1	n1	{t3,t4,t5}	{1,6,8,9,10,11}
2	n2	{t1,t2,t3}	{1,2,4,5,7,12,13}
3	n3	{t1,t2}	{ }
4	n4	{t1,t3,t4,t5}	{3,6,8,9,10,11}
5	n5	{t4,t5}	{3,6,8,9,10,11}
6	n6	{t1,t2,t3,t5}	{1,2,3,4,5,7,12,13}
7	n7	{t1,t2,t5}	{2,4,5,7,12,13}

Figure 2.1: Example instance for the REFEREE-scheme

PAPER-TO-BE-REFEREED				
PAP#	TITLE	TOPIC	AUTHORS	REFEREE-SET
			AUTHOR	REF#
1	ab	t3	{a1,a2,a3,a4}	{1,2,6}
2	cd	t1	{a5,a6,a7}	{2,6,7}
3	ef	t5	{a8,a9}	{4,5,6}
4	gh	t2	{a10,a11}	{2,6,7}
5	ij	t1	{a12,a13}	{2,6,7}
6	kl	t4	{a14,a15}	{1,4,5}
7	mn	t1	{a16,a17}	{2,6,7}
8	op	t3	{a18}	{1,4,5}
9	qr	t5	{a19}	{1,4,5}
10	st	t4	{a20,a21}	{1,4,5}
11	uv	t5	{a22}	{1,4,5}
12	wx	t2	{a23,a24}	{2,6,7}
13	yz	t2	{a25,a26}	{2,6,7}

Figure 2.2: Example instance for the PAPER-To-BE-REFEREED-scheme

Let for example the REFEREE-instance in Figure 2.1 be given. To assign paper 25 to referee 1 we can specify an insert-operation as follows:

Insert the tuple (REF#:1,NAME:n1,TOPICS:{ },PAPERS:{(PAP#:25)}).

This does not mean that the tuple is to be inserted as an "autonomous" tuple into the relation, but that the tuple is "merged" with the already existing tuple

(REF#:1,NAME:n1,TOPICS:{(TOPIC:t3),(TOPIC:t4),(TOPIC:t5)},
PAPERS:{(PAP#:1),...,(PAP#:11)}).

The result should be the tuple

(REF#:1,NAME:n1,TOPICS:{(TOPIC:t3),(TOPIC:t4),(TOPIC:t5)},
PAPERS:{(PAP#:1),...,(PAP#:11),(PAP#:25)}).

An example delete operation for the REFEREE-instance in Figure 2.1 could be:

Delete the tuple (REF#:1,NAME:n1,TOPICS:{(TOPIC:t4)},PAPERS:{ })

which reflects a situation, where referee 1 is no longer interested in papers of topic t4. One interpretation of this operation could be, not to delete anything, since exactly this tuple is not contained in the instance. But it is meant that the topic t4 is to be deleted from the set of topics of referee 1.

These examples show that tuples to be inserted or to be deleted sometimes must be treated as sub-tuples of already existing tuples. On the other hand there also exist examples where it is more appropriate to insert or delete a tuple only as one whole, indivisible unit. This observation is also reflected by the different proposals for nested relational algebras existing in the literature: By the set theoretic operations (union and intersection) of the algebras proposed in [ScS86, PDG89, Col90] tuples are treated as autonomous tuples. [AbB86, Hul89, RKS88] on the other hand belong to the sub-tuple approaches.

In the sequel we informally describe a combination of both approaches. The theoretical foundations are presented in detail in [San92] and [ObS92]. In our approach it is not only possible to treat a tuple either as a sub-tuple or as an autonomous tuple, but to mix up both points of view. More precisely,

for every composite attribute we specify *locally* (i.e. independently from all other attributes) whether to access the respective values only as a whole or to access also subsets of these values. This is achieved by so-called *filter tables* which allow to delete or to insert information from /into an instance in a parametrized way.

Conceptually, filter tables can be considered as an extension of literals in 1st order predicate logic without function symbols. Such a 1st order literal is something like an "example tuple" consisting of constants and variables. A literal can be used very well to describe the insertion and deletion of information in a 1NF relational database. For example, a literal of the form $(X_1,...,X_n)$ – where each X_i is a variable or a constant – specifies the insertion (or deletion, respectively) of tuples which can be obtained by mapping the variables of the literal to constants of the repective domains. This simple mechanism is not appropriate for nested relations because it does not fit well to the hierarchical structure of nested relations: it is only possible to access values on the highest level of the attribute hierarchy, and tuples can be selected only as autonomous tuples.

Thus, we propose a generalization of this approach, the so-called *filter tables*. This extension of literals can be characterized as follows:

- Filter tables have a hierarchical structure corresponding to a nested relational scheme. This allows to access values which are located on a "deep level" of the attribute hierarchy.
- By filter tables, in general, we do not specify single tuples but sets of tuples. This also corresponds to the structure of nested relations and enables us to specify and access set-values.
- We distinguish two types of terms in filter tables: *open terms* and *closed terms*. Syntactically, closed terms are marked by an overline whereas open terms are unmarked. Semantically this does express the difference between sub-tuples and autonomous tuples: a closed term, e.g. the variable $\overline{X}$, represents an access to a whole, indivisable set-value, i.e. this variable must be instantiated by a whole set-value existent in a tuple. On the other hand an open term describes the access to subsets, i.e. the term can be instantiated with a subset of an existing set.

A filter table contains a set of tuples. Each tuple consists of terms, which may be variables (uppercase letters), atomic constants (lowercase letters) or other filter tables. Each term represents values of the respective domain (of the corresponding attribute). If a term represents set-valued instances then the term is either open or closed.

In order to visualize their hierarchical structure we have chosen a graphical notation for filter tables. This might remind the reader to the graphical query language *Query by Example* [Zlo75], a declarative language for 1NF relational databases. After the next example we briefly explain the theoretical background and formal semantics of our filter table formalism.

Example (continued):

Figure 2.3 shows some filter tables, which may be interpreted as insert- or, alternatively, as delete-operations for the REFEREE-instance. For each filter table we discuss both, the insert- and the delete-operation.

Note, that uppercase letters represent variables which must be instantiated, and all other strings denote values, i.e. elements of the respective domain.

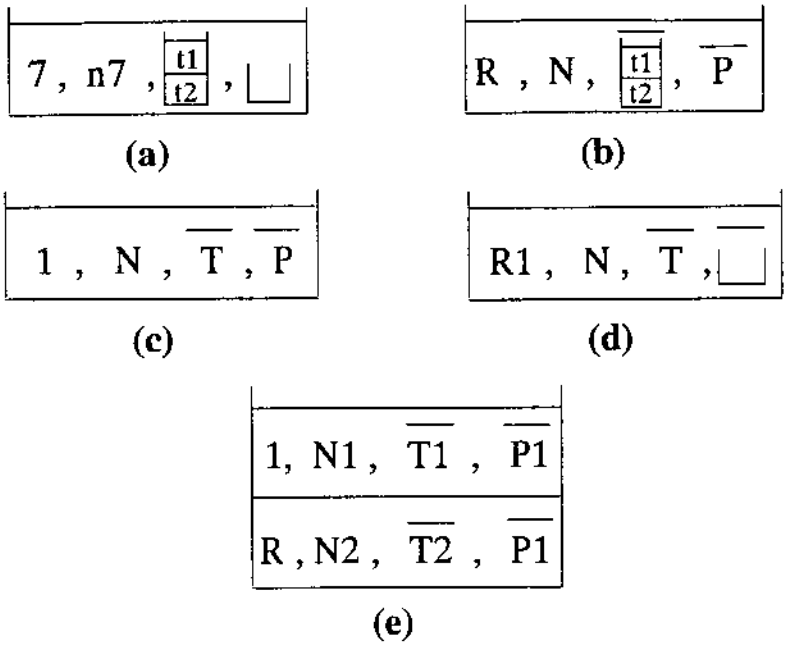


Figure 2.3: Example filter tables for the REFEREE-instance

(a) *Insertion*:

Insert a (new) referee with REF# 7 and name n7 into the instance of REFEREE. The set of topics of this referee consists of elements t1 and t2. If a referee-tuple with REF# 7 and name n7 exists in the REFEREE-instance, then the topics t1 and t2 are inserted into the existing set of topics of this tuple. Else the referee-tuple with REF# 7, name n7, topics t1 and t2 and empty set of papers is inserted as a new autonomous tuple into the REFEREE-instance.

Deletion:

Delete the topics t1 and t2 from the set of topics of referee with REF# 7. This operation is possible only if a referee with REF# 7 and name n7 is present in the instance.

(b) *Insertion:*

Insert a (new) referee tuple, where the set of topics only consists of t1 and t2. The variables R, N and P must be instantiated appropriately.

Deletion:

Delete a referee tuple where the set of topics only consists of t1 and t2. With respect to the instance in Figure 2.1 this filter table might be instantiated by the values belonging to referee 3.

If we slightly modify the filter table and omit the overline of the TOPICS term, then the meaning is as follows: Delete the topics t1 and t2 from the set of topics of one of the existing referees.

(c) *Insertion*:

Insert a (new) tuple with REF# 1 as autonomous tuple. The variables must be appropriately instantiated. Note that this insertion is performed even if a tuple with REF# 1 already exists.

Deletion:

Delete a tuple with REF# 1.

(d) *Insertion*:

Insert a (new) referee, to whom no paper is assigned. Variables R1, N and T must be instantiated appropriately.

Deletion:

Delete a referee, to whom no paper is assigned.

(e) *Insertion*:

Insert referee 1 and one other referee who has the same papers as referee 1.

Deletion:

Delete referee 1 and one other referee who has the same papers as referee 1.

■

Thus, values which are obtained by instantiating a closed term are treated as atomic values, and a tuple can be deleted as a whole only if *all* set-valued terms are closed. Otherwise, i.e. if there exists at least one open term, only the values of the open terms can be removed from the corresponding tuple in an instance. The insertion is inverse to the deletion of tuples, and it is uniquely defined how set-values are grouped together by an insert operation. If a term is closed then the value to be inserted must appear as a whole in the resulting instance, otherwise the value can be grouped together with an already existing value.

The theoretical background of filter tables can be summarized as follows:

Filter tables are a graphical short hand notation for the following two independent concepts: On the one hand, they can be considered as a complex logical term, and the instantiation of the variables of this term yields a (nested relational) instance. On the other hand, the distinction between open and closed terms implicitly defines an *ordering* on nested relational instances (i.e. a reflexive, transitive and antisymmetric relation). This ordering induces a lattice structure with a union (lub), intersection (glb) and complement operation on instances. Then, the insertion of an instance (the instantiated filter table) into another instance is simply the union of both, and the deletion of an instance from another instance is the intersection of the second instance with the complement of the first instance.

A formalization of these issues and a proof of the soundness of our definitions can be found in [San92] and [ObS92]. Due to space limitations we do not present it in this paper.

3 Nested Relation/Transition Nets

In this section we shall describe how procedures in complex object database applications can be modelled with a novel type of high-level Petri nets, the so-called NR/T-nets (Nested Relation/Transition nets). NR/T-nets combine the previously described concept of nested relations with the Petri net formalism. We start with some basic definitions related to Petri nets.

Petri nets [BRR87] are a graphical formalism for the description of system behaviour. A net is a bipartite graph, consisting of two disjoint sets of nodes, the set of *places* and the set of *transitions*. Places and transitions are connected by directed arcs.

Definition 3.1: (Petri) Net

A (Petri) net is a triple $N = (PL, TR, F)$ where

- PL is a finite set of so called *places*,
- TR is a finite set of so called *transitions*, $PL \cap TR = \emptyset$,
- $F \subseteq (PL \times TR) \cup (TR \times PL)$ is called the *flow relation* of N.

■

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

The set $\{s \in PL \mid (s,t) \in F\}$ of *input places* of a transition t is denoted as $\bullet t$, the set $\{s \in PL \mid (t,s) \in F\}$ of *output places* of a transition t is denoted as $t\bullet$. $\bullet t \cup t\bullet$ is called *environment* of t.

Example (continued):

Figure 3.1 shows a net describing a part of the behaviour of the conference information system informally. Note that exception handling, e.g. handling of missing referee reports, is not considered here.

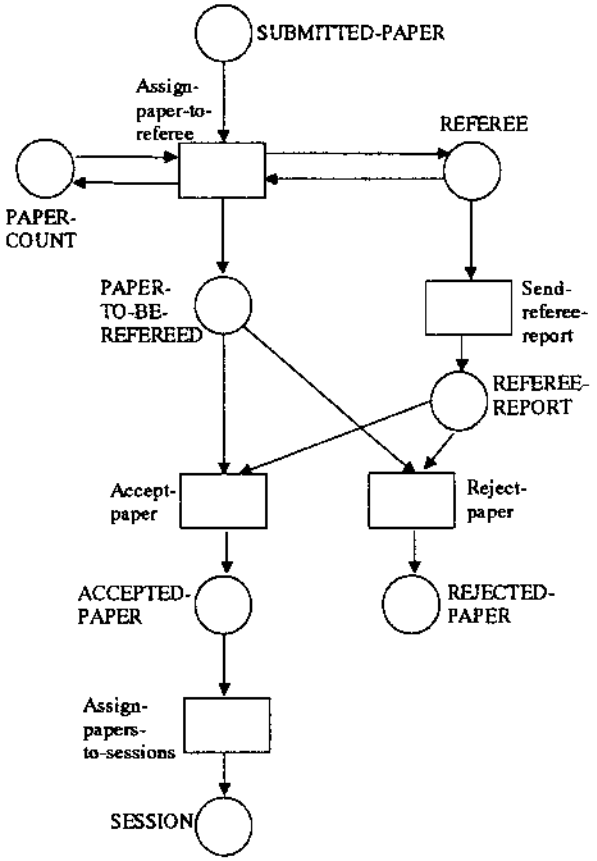


Figure 3.1: Net describing program committee activities

■

A net describes the structure of system behaviour only informally. There is no formal notion of system state or system behaviour in this simple type of net. However, there exist different approaches to extend the basic net concept, e.g. *condition/event nets* or *place/transition nets* [BRR87]. In *predicate/transition nets* [Gen87] (and similar in *coloured Petri nets* [Jen92]), each place represents a relation scheme (predicate). In *strict* predicate/transition nets the marking of a place is a particular relation, the set of all place markings at a given time describes a certain 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 in its input places and tuples are inserted into the relations in its output places. A logical expression, which may be assigned to a transition allows to specify certain conditions on the tuples to be inserted and removed.

The problem with predicate/transition nets is that they only allow the manipulation of flat tuples with atomic attributes. The aim of the next section is to integrate the concept of unnormalized relations into the predicate/transition net formalism. The new net type should be easy to understand and should have a formal semantics. It should especially support the manipulation of tuple sub-structures, i.e. set valued attributes.

The first extension to predicate/transition nets is to interpret places as nested relation schemes which have attribute values being themselves (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 3.2: Nested-Relation Net (NR-Net)
A nested-relation net is a net where the places are interpreted as top-level schemes with respect to a given set of scheme equations.

■

Example (continued):
The net in Figure 3.1 which shows the structure of the possible behaviours related to the conference information system can be interpreted as an NR-net as follows.

Let T={`SUBMITTED-PAPER:=(TITLE,TOPIC,AUTHORS)`, ...} be the set of scheme equations as in Section 2. The top-level schemes in T are assigned to the respective places in the net. In the graphical representation, the right-hand sides of the scheme equations are omitted.

■

An NR-net describes the structure of possible system behaviours (procedures) but it does not describe concrete states or state transitions. In the sequel we extend NR-nets to 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 which 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:
  - *arc inscriptions*: arcs are inscribed with filter tables.
  - *transition inscriptions*: transitions are inscribed with logical expressions.
- *occurrence rule*: defines in which state a transition is enabled and specifies which changes are related to a transition occurrence.

Definition 3.3: Nested-Relation/Transition Net (NR/T-Net)
A nested-relation/transition net is a 4-tuple $NRT = (NN, AI, TI, M^0)$ such that

(1) $NN = (PL,TR,F)$ is an NR-net.

(2) AI is a function that assigns to each element f in F a filter table. $AI(f)$ corresponds to the type of the adjacent place.

(3) TI assigns to each transition t a logical formula. $TI(t)$ may contain typed predicate symbols p to specify membership, equality, etc. between instances, where each p has a fixed interpretation $I(p)$ which is a relation of the respective arity over instance-values and tuples.

(4) M^0 assigns to each place an *initial marking*, i.e. an instance corresponding to the respective scheme.

■

Example (continued):

Figure 3.2 shows an example for an NR/T-net of the program committee activities. For predicates in logical formulas we use infix instead of prefix notation. We write e.g. "X=Y" instead of "=(X,Y)".

As start marking (which is omitted in the graphical representation) we consider a situation, where 13 papers are already in the state "to be refereed", two additonal papers are newly submitted and 4 referee reports are already available:

Place PAPER-COUNT:

PAPER-COUNT
PAP#
13

Place SUBMITTED-PAPER:

SUBMITTED-PAPER		
TITLE	TOPIC	AUTHORS
		AUTHOR
ba	t1	{a27,a28}
dc	t4	{a31}

Place SESSION:

SESSION		
TOPIC	TALKS	
	TITLE	AUTHORS
		AUTHOR
t1	{ }	
t2	{ }	
t3	{ }	
t4	{ }	
t5	{ }	

Place REFEREE-REPORT:

REFEREE-REPORT		
REF#	PAP#	VOTE
4	3	0
5	3	0
6	3	0
2	5	1

For the marking of the places REFEREE and PAPER-TO-BE-REFEREED see Figure 2.1 and 2.2. The start marking of all other places is empty.

■

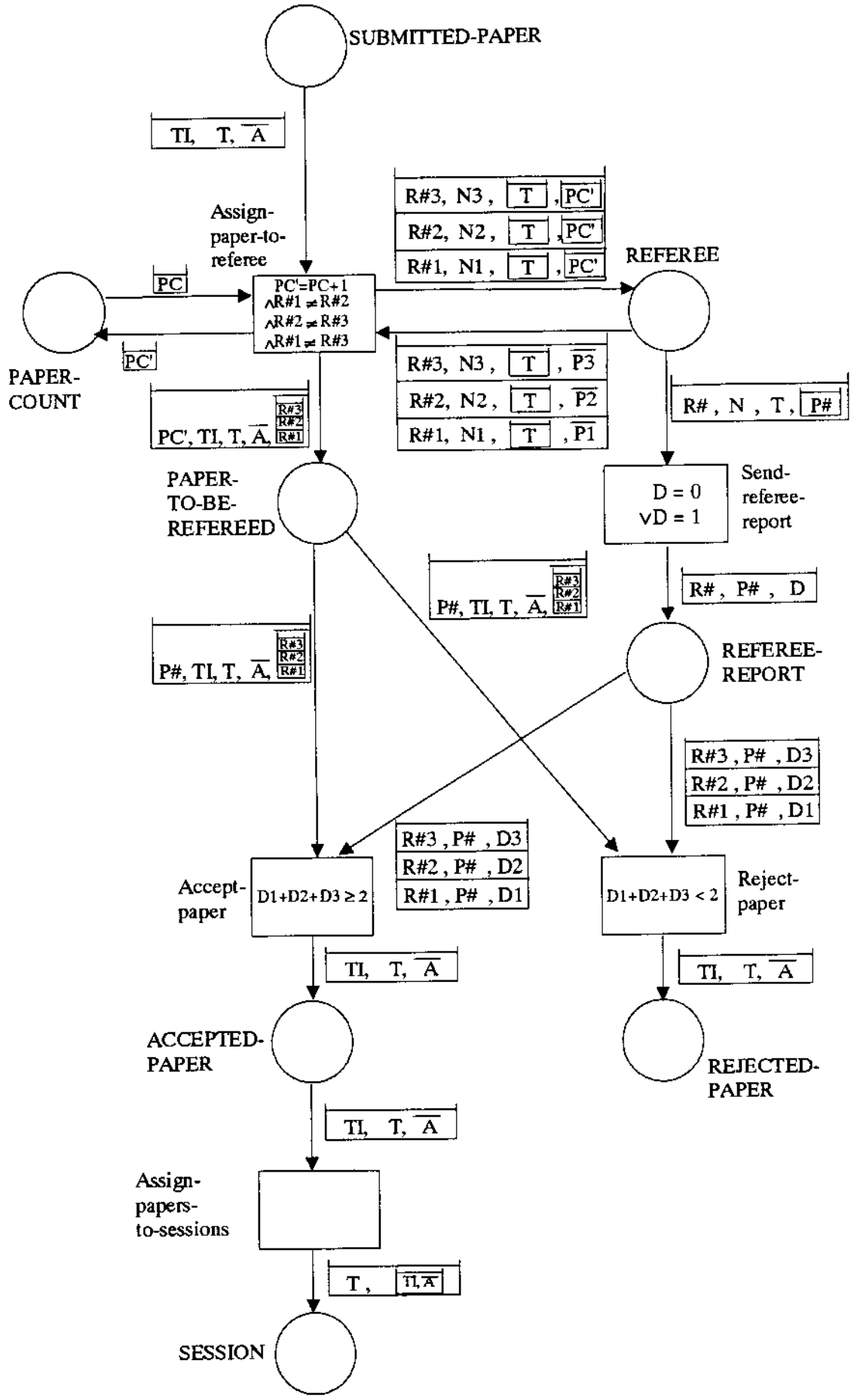


Figure 3.2: NR/T-net

Transitions represent schemes of (local) marking changes. A transition t is enabled for a given marking and a given instantiation[1] of the variables in t's environment, i.e. t may occur, if

(i) 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 "sub-structure" of these markings.

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

(iii) the logical formula inscribed to the transition t evaluates to "true".

A transition which is enabled may occur. If an enabled transition occurs, tuples are removed from its input places and tuples are inserted into its output places.The occurrence of transitions in a given initial marking leads to so-called *follower markings*.

The occurrence of transition t for a given marking M and an instantiation Θ means that M is changed to a new marking M' which is defined as follows:

- the tuples specified by the filter tables of the incoming arcs with respect to the instantiation Θ are deleted from the corresponding instances. Speaking formally, a deletion is an intersection of the instance with the complement of the instantiated filter table. These operations depend on the underlying ordering (specified by open and closed terms in filter tables).
- the tuples contained in the filter tables of the outgoing arcs with respect to the instantiation Θ are inserted into the markings of the output places. Formally, an insertion is a union of the instance with the instantiated filter table (w.r.t. the underlying ordering).
- for every place not belonging to the environment of t $M(t) = M'(t)$, i.e. the marking remains unchanged.

Both operations, deletion and insertion can be explained informally as in Section 2.

Example (continued):

Starting with a certain initial marking the occurrences of transitions may generate marking sequences. We now explain one possible marking sequence for the NR/T-net given in Figure 3.2. For the initial marking given before, the transitions `Assign-paper-to-referee`, `Send-referee-report` and `Reject-paper` are enabled.

Transition `Assign-paper-to-referee` is enabled for different instantiations, e.g. for the submitted paper with title `ba`, paper count value `13` and referees 3, 4 and 7. If the transition occurs for this instantiation, then the respective tuple

```
(TITLE:ba,TOPIC:t1,AUTHORS:{(AUTHOR:a27),(AUTHOR:a28)})
```

is completely removed from the current instance in place `SUBMITTED-PAPER`, since the `AUTHORS`-term is closed in the respective filter table. The value in `PAPER-COUNT` changes from `13` to `14`. In the filter table assigned to the arc leading from place `REFEREE` to transition `Assign-paper-to-referee` the `TOPICS`-term is open whereas the `PAPERS`-term is closed. Hence the

[1] Note, that equal variables in the environment of a transition must be assigned equal values.

TOPIC-value t1 is removed from the tuples with referee 3, 4 and 7 in place REFEREE. In the filter table assigned to the arc leading into the place REFEREE, the TOPICS- and the PAPERS-term are both open terms. The topic t1 and the paper number 14 are inserted into the respective attribute values of the three referee tuples. All terms denoting set-valued attributes (AUTHORS and REFEREE-SET) in the filter table assigned to the arc leading to place PAPER-TO-BE-REFEREED are closed. Hence a new tuple is inserted into the place PAPER-TO-BE-REFEREED with PAP#-value 14, and set of referees consisting of the referee numbers of the selected referees 3, 4 and 7. The new marking of place REFEREE and PAPER-TO-BE-REFEREED is as follows:

REFEREE			
REF#	NAME	TOPICS	PAPERS
		TOPIC	PAP#
1	n1	{t3,t4,t5}	{1,6,8,9,10,11}
2	n2	{t1,t2,t3}	{1,2,4,5,7,12,13}
3	n3	{t1,t2}	{14}
4	n4	{t1,t4,t5}	{3,6,8,9,10,11,14}
5	n5	{t4,t5}	{3,6,8,9,10,11}
6	n6	{t1,t2,t3,t5}	{1,2,3,4,5,7,12,13}
7	n7	{t1,t2,t5}	{2,4,5,7,12,13,14}

PAPER-TO-BE-REFEREED				
PAP#	TITLE	TOPIC	AUTHORS	REFEREE-SET
			AUTHOR	REF#
1	ab	t3	{a1,a2,a3,a4}	{1,2,6}
2	cd	t1	{a5,a6,a7}	{2,6,7}
3	ef	t5	{a8,a9}	{4,5,6}
4	gh	t2	{a10,a11}	{2,6,7}
5	ij	t1	{a12,a13}	{2,6,7}
6	kl	t4	{a14,a15}	{1,4,5}
7	mn	t1	{a16,a17}	{2,6,7}
8	op	t3	{a18}	{1,4,5}
9	qr	t5	{a19}	{1,4,5}
10	st	t4	{a20,a21}	{1,4,5}
11	uv	t5	{a22}	{1,4,5}
12	wx	t2	{a23,a24}	{2,6,7}
13	yz	t2	{a25,a26}	{2,6,7}
14	ba	t1	{a27,a28}	{3,4,7}

For this marking the transitions Assign-paper-to-referee, Send-referee-report and Reject-paper are again enabled.

Reject-paper is enabled for paper 3. All referees (REF# 4, 5 and 6) voted *reject*, i.e. the VOTE-value is always "0" and D1+D2+D3 < 2. If Reject-paper occurs, the complete tuple with P#-value 3 is removed from place PAPER-TO-BE-REFEREED (since all terms denoting set-valued attributes are closed in the respective filter table). The three referee reports with PAP#-value 3 are removed from place REFEREE-REPORT. A new tuple having the same TITLE-, TOPIC- and AUTHORS-values as the removed tuple from PAPER-TO-BE-REFEREED is inserted into the empty instance REJECTED-PAPER.

For the new marking the transitions Assign-paper-to-referee and Send-referee-report are enabled. Send-referee-report is, e.g., enabled for PAP# 14 with respect to

referee 3, 4 and 7. Let us assume that referee 3 and 4 vote *accept* ("1") and referee 7 votes *reject* ("0"). Three transition occurrences - possibly in one step - lead to the new marking, where three new tuples are inserted into the instance of place REFEREE-REPORT. For each referee the paper number 14 is removed from the PAPERS-instance since the respective PAPERS-term is open. The new marking of REFEREE-REPORT then looks as follows:

REFEREE-REPORT		
REF#	PAP#	VOTE
3	14	1
4	14	1
7	14	0
2	5	1

Let us now consider transition Accept-paper, which is - for the current marking - enabled with respect to paper 14. Two referees voted *accept*, i.e. their VOTE-value is "1", one referee voted *reject*, i.e. the VOTE-value is "0" and D1+D2+D3 ≥ 2. If Accept-paper occurs, the complete tuple with PAP#-value 14 is removed from place PAPER-TO-BE-REFEREED (since all terms denoting set-valued attributes are closed in the respective filter table). The three referee reports with PAP#-value 14 are removed from place REFEREE-REPORT. A new tuple having the same TITLE- and AUTHORS-values as the removed tuple from PAPER-TO-BE-REFEREED is inserted into the empty instance ACCEPTED-PAPER.

Finally, transition Assign-papers-to-sessions can occur.

For the new marking transition Assign-papers-to-sessions is enabled with respect to paper 14 and session with topic t1. If the transition occurs, the tuple (TITLE:ba, TOPIC:t1,AUTHORS:{a26,a27}) is completely removed from the place ACCEPTED-PAPER (since the AUTHORS-term is closed). On the other hand, the tuple (TOPIC:t1, TALKS: {(TITLE:ba,AUTHORS:{a26,a27})}) is merged with the existing tuple (TOPIC:t1, TALKS:{}) in place SESSION, since the TALKS-term is open. The final marking of place SESSION is:

SESSION		
TOPIC	TALKS	
	TITLE	AUTHORS
		AUTHOR
t1	{(ba,	{a26,a27})}
t2	{ }	
t3	{ }	
t4	{ }	
t5	{ }	

■

In the long version of this paper [ObS92] we consider some additional aspects of NR/T-nets. Mainly, these are constraints which ensure that a net is well-defined. In this case we talk about *admissible NR/T-nets*. The constraints are:

- First, the insert operation for tuples into the instance of a place is determined by the ordering (specified by the combination of open and closed terms) of a filter table which 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.
- Third, all filter tables at the outgoing arcs of a place must specify orderings which are *compatible* to the ordering of the place. This has technical reasons which, if they were violated, would also cause ambiguities concerning the deletion of tuples.

For a more detailed discussion of these aspects we refer the interested reader again to [ObS92].

4 Modelling Static Semantic Integrity Constraints by Fact Transitions

Static semantic integrity constraints restrict the set of possible system states to the set of *correct* system states. Static semantic integrity constraints are part of the conceptual schema of a database.

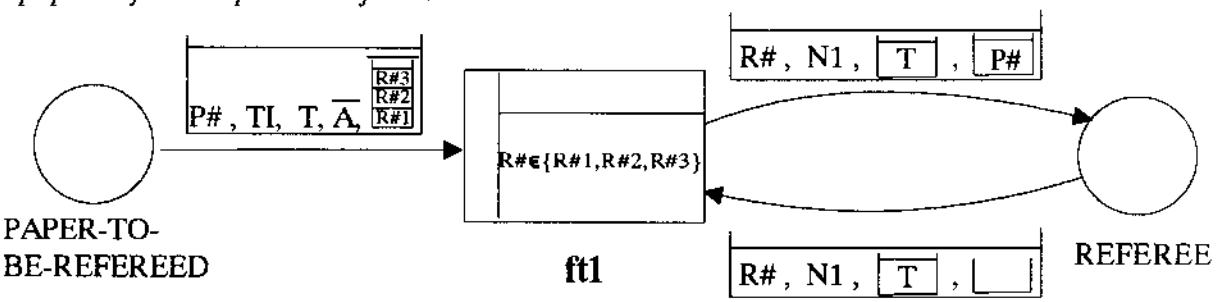
Fact transitions [GeL78] are transitions which are not enabled for any reachable net marking. Hence a fact transition represents an invariant assertion about the net behaviour. In [HeR86, Vos87] it has been proposed to use fact transitions for the specification of static semantic integrity constraints. The semantics of a fact transition now is as follows: a marking where a fact transition is enabled represents an invalid system state, i.e. a state where the respective integrity constraint is violated. Hence a net must be designed such that in no reachable marking a fact transition is enabled. Alternatively, certain exception handling mechanisms must be provided to handle situations, where a fact transition is enabled, i.e. where a constraint is violated.

In this section we shall present some examples to show different kinds of semantic integrity constraints, which can be expressed by fact transitions in NR/T-nets. Graphically, fact transitions are represented as usual with an inscribed F.

Example (continued):

For the NR/T-net in Figure 3.2 the following constraints can be specified:

(i) *Each paper, whose set of referees contains a certain referee, must be contained in the set of papers of the respective referee.*

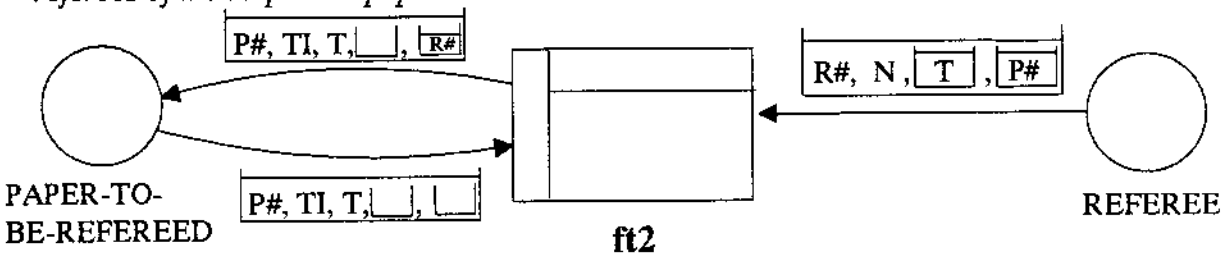


For a given marking, **ft1** is enabled, i.e. for the respective system state the constraint is violated, if the following three conditions hold:

- there is a tuple in place PAPER-TO-BE-REFEREED with paper number P#, title TI, topic T, and referees R#1, R#2, R#3.
- there is a tuple in place REFEREE with referee number R#1 or R#2 or R#3, name N1, where topic T of paper P# is contained in the referee´s set of topics.
- there is no tuple in place REFEREE with referee number R#1 or R#2 or R#3, name N1, where topic T is contained in the set of topics and P# is contained in the set of papers.

(note that R# in the inscription of the incoming and outgoing arc of place REFEREE must be instantiated to the same value.)

(ii) *Each referee, whose set of papers contains a certain paper, must be contained in the set of referees of the respective paper.*

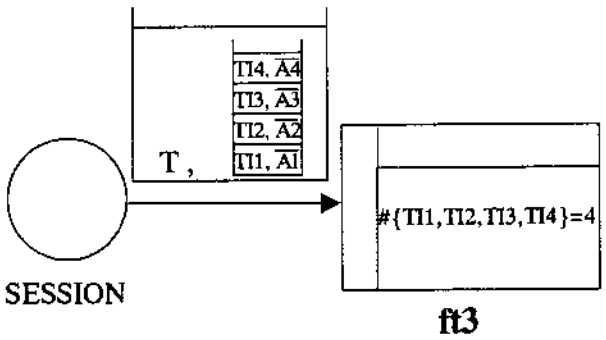


For a given marking, **ft2** is enabled if the following three conditions hold:

- there is a tuple in place REFEREE with referee number R#, name N, where topic T is contained in the set of topics and paper P# is contained in the set of papers.
- there is a tuple in place PAPER-TO-BE-REFEREED with paper number P#, title TI and topic T.
- there is no tuple in place PAPER-TO-BE-REFEREED with paper number P#, title TI, topic T and referee R# in the set of referees.

Note, that both constraints (i) and (ii) cannot be violated by the NR/T-net given in Figure 3.2. The constraints are guaranteed by the inscriptions of the incoming and outgoing arcs of transition `Assign-paper-to-referee`.

(iii) *No session contains more than 3 talks.*



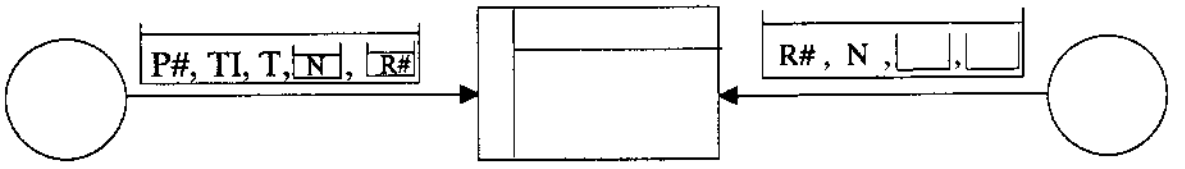
For a given marking, **ft3** is enabled if the following condition hold:

- There is a tuple in the place SESSION with topic T and at least 4 different talks.

For the given net in Figure 3.2, it is possible that a state is reached, where the constraint (iii) is violated, i.e. where the respective fact transition **ft3** is enabled. Two modifications are possible

to guarantee that the constraint is satisfied: First, transition `Assign-papers-to-sessions` might be modified such that no more than three papers can be assigned to a single session. Alternatively, it is possible to tolerate a temporary violation of the constraint. An additional transition could be introduced to split sessions with too many papers into smaller sessions.

(iv) *No paper must be assigned to a referee who belongs to the paper's set of authors.*



For a given marking, **ft4** is enabled if the following two conditions hold:

- there is a tuple in place `PAPER-TO-BE-REFEREED` with author `N` in the set of authors and referee `R#` in the set of referees.
- there is a tuple in place `REFEREE` with number `R#` and name `N`.

The fulfillment of the constraint can be simply guaranteed by assigning the following inscription to the transition `Assign-paper-to-referee` in Figure 3.2:

$$N1 \notin A \wedge N2 \notin A \wedge N3 \notin A$$

5 Conclusion and Outlook

In this paper we have proposed a novel high-level type of Petri nets which integrates concepts of nested relational structures and predicate/transition nets into one framework, the so-called NR/T-nets. The main new contributions of NR/T-nets are:

- Places are unnormalized relational schemes, the marking of a place is a nested relational instance.
- Arcs are inscribed with so-called filter tables, which allow to express complex selection conditions on hierarchically structured relational instances.
- The transition occurrence rule is modified such that it becomes possible to directly access (insert and delete) sub structures of existing tuples.

Some other extensions to high level Petri nets exist which also allow, e.g., set-, list- and tuple-valued attributes in the marking of places [BaB89, HeR92, HSY85, Jen92, Obe88, RiD82, Sib86]. 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 sub-structures is not supported. Our direct access to sub-structures must be simulated by first removing a tuple from a place, then manipulating the respective sub-structure 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.

It is planned to integrate the concepts described here into INCOME/STAR [OSS92] which is an environment for the development of distributed information and control systems. INCOME/STAR combines the modelling of structural system aspects based on semantic data models with system behaviour modelling based on high-level Petri nets. It provides graphical editors, generators, prototyping facilities and design dictionary support. One specific objective of the available simulator for predicate/transition nets is that it supports the integration of fact transitions for net simulation [Obe88]. The graphical editor for predicate/transition nets and the simulator must be appropriately extended to allow editing and simulation of NR/T-nets.

There are several future research directions:

- Questions related to available analysis techniques for NR/T-nets must be addressed.
- The expressive power of filter tables should be investigated in detail.
- NR/T-nets can be extended by assigning a nonempty set of filter tables to each arc instead of 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.

References

[AbB86] S. Abiteboul, N. Bidoit: Non first normal form relations: an algebra allowing data restructuring, Journal of Computer and System Sciences, 33, 1986, pp. 361-393

[BaB89] M. Baldassari, G. Bruno: An environment for object-oriented conceptual programming based on PROT nets, in: G. Rozenberg (Ed.): Advances in Petri Nets 1988, Springer-Verlag, 1989, pp. 1-19

[BRR87] W. Brauer, W. Reisig, G. Rozenberg (Eds.): Petri Nets: Central Models and their Properties, Advances in Petri Nets 1986, Part I, LNCS 254, Springer-Verlag, 1987

[Cod70] E.F. Codd: A relational model for large shared data banks, Comm. ACM, 13, 6, 1970, pp. 377-387

[Col90] L.S. Colby: A recursive algebra for nested relations, Information Systems, Vol. 15, No. 5, 1990, pp. 567-582

[CRIS82] T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (Eds.): Information Systems Design Methodologies. A Comparative Review, Proc. of the IFIP WG 8.1 Working Conference, North-Holland Publuishing Company, Amsterdam, New York, Oxford, 1982

[Fel92] Feldbrugge, F.: Petri net tool overview 1992, Special Volume of Petri Net Newsletter, Gesellschaft für Informatik, Special Interest Group on Petri Nets and Related System Models, April 1992.

[GeL78] H.J. Genrich, K. Lautenbach: Facts in place/transition nets, in: Mathematical Foundations of Computer Science 1978, Springer-Verlag, 1978

[Gen87] H.J. Genrich: Predicate/transition nets, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.): Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, LNCS 254, Springer-Verlag, 1987, pp. 207-247

[HeR86] C.A. Heuser, G. Richter: On the relationship between conceptual schemata and integrity constraints on databases, in: T.B. Steel, R. Meersman (Eds.): Proc. of the IFIP WG2.6 Workshop Conference on Database Semantics (DS-1), North-Holland Publ. Comp., 1986, pp. 27-39

[HeR92] C.A. Heuser, G. Richter: Constructs for modeling information systems with Petri nets, in: K. Jensen (Ed.): Application and Theory of Petri Nets 1992, LNCS 616, Springer-Verlag, 1992, pp. 224-243

[HSY85] A. Horndasch, R. Studer, R. Yasdi: An approach to (office) information system design based on general net theory, in: Proc. IFIP TC8.1 TFAIS85, North-Holland Publishing Company, 1985

[HTY89] R. Hull, K. Tanaka, M. Yoshikawa: Behavior Analysis of Object-Oriented Databases: Method Structure, Execution Trees, and Reachability, in: W. Litwin, H.-J. Schek (Eds.): Foundations of Data Organization and Algorithms, LNCS 367, Springer-Verlag, 1989

[Hul89] G. Hulin: A relational algebra for nested relations in Partitioned Normal Form, Manuscript M 318, Philips Research Laboratory, 1989

[Jen92] K. Jensen: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Vol. 1, Springer-Verlag, 1992

[JeR91] K. Jensen, G. Rozenberg (Eds.): High-Level Petri Nets, Springer-Verlag, 1991

[Mak77] A. Makinouchi: A consideration on normal form of not - necessarily - normalized relation in the relational data model, in: Proc. of the Intl. Conference on Very Large Databases, 1977, pp. 447-453

[Obe88] A.Oberweis: Checking database integrity constraints while simulating information system behaviour, in: Proc. of 9th European Workshop on Application and Theory of Petri Nets, Venice/Italy, 1988, pp. 299-308

[ObS92] A. Oberweis, P. Sander: The specification of complex object behaviour by high-level Petri nets, Universität Karlsruhe, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Forschungsbericht 254, October 1992

[OSS92] A. Oberweis, G. Scherrer, W. Stucky: INCOME/STAR: Process model support for the development of information systems, Forschungsbericht 252, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, August 1992

[PDG89] J. Paradaens, P. DeBra, M. Gyssens, D. VanGucht: The Structure of the Relational Database Model, Springer-Verlag, Berlin, Heidelberg, New York, 1989

[RiD82] G. Richter, R. Durchholz: IML inscribed high-level Petri nets, in: T.W. Olle, H.G. Sol, A.A. Verrijn-Stuart (Eds.): Information Systems Design Methodologies: A Comparative Review, North-Holland Publishing Company, 1982, pp. 335-368

[RKS88] M.A. Roth, H.F. Korth, A. Silberschatz: Extended algebra and calculus for nested relational databases, ACM Transactions on Database Systems, Vol. 13, No. 4, 1988, pp. 389-417

[San92] P. Sander: Boolean lattices of nested relations as a foundation for rule-based database languages, Data&Knowledge Engineering, 8, 2, 1992, pp. 93-130

[ScS86] H.-J. Schek, M.H. Scholl: The relational model with relation-valued attributes, Information Systems, 11, 2, 1986, pp. 137-147

[Sib86] C. Sibertin-Blanc: High level Petri nets with data structure, in: Proc. of the 6th Workshop on Petri nets, Espoo/Finland, 1986

[ThF86] S. Thomas, P.C. Fischer: Nested relational structures, in: P. Kanellakis (Ed.): Advances in Computing Research III, The Theory of Databases, JAI Press, Greenwich, 1986, pp. 269-307

[Vos87] K. Voss: Nets in data bases, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.): Petri Nets: Applications and Relationships to other Models of Concurrency, Advances of Petri Nets 1986, LNCS 255, Springer-Verlag, 1987, pp. 234-257

[Zlo75] M.M. Zloof: Query-By-Example: The invocation and definition of tables and forms, in: D.S. Kerr (Ed.): Proc. Intl. Conference on Very Large Data Bases, Framingham/Massachusetts, 1975, pp. 1-24