How to Transfer Concepts of Abstract Data Types to Petri Nets?

H. Ehrig, A. Merten, J. Padberg

email: ehrig@cs.tu-berlin.de, aneme@cs.tu-berlin.de, padberg@cs.tu-berlin.de

Technische Universität Berlin
Franklinstr. 28/29
D - 10587 Berlin
Germany

22nd September 1997

Abstract

The concept of abstract data types is one of the fundamental concepts for software system specification. Although Petri nets have been used with success for the specification of various kinds of software and communication based systems, there is no abstraction concept for Petri nets similar to abstract data types up to now. In this paper we propose a new concept, called *abstract process types*, for this purpose. We discuss the idea of abstract process types based on the new concept of *open processes*. This is an abstraction of Petri net processes which includes designated interface places with autonomous actions caused by the environment. The new concepts are illustrated by some well–known examples of place/transition and algebraic high–level nets. The formalization of these concepts is subject of a forthcoming paper.

1 Introduction

The aim of the project "Petrinetz–Technologie (PNT)", (see [WRE95]) is to analyse and improve the use of Petri net techniques in practical software system development. On the background of the experience with several other semi–formal and formal specification techniques in software development projects there is the important question how to transfer well–established abstraction and structuring concepts, like abstract data types and modules to Petri nets.
In each of the practical case studies of the PNT–project a large number of Petri nets has been developed, in order to specify a more or less complex communication based system. Although the different application processes, short tasks, of the systems are clear on an informal level, it is difficult to distinguish how each task of the system is realized in terms of Petri nets or Petri net processes.
In fact, the connection of all the Petri nets in each of the case studies is modeling the union of all the tasks of each system. But if we consider a specific task there is no net, subnet or process in the classical sense of Petri nets which models exactly this task. The classical notion of a Petri net process is too narrow to model such tasks, because it cannot interact with the environment.
The concept of abstract data types is suitable to model the different tasks of a system, at least if the tasks can be expressed by functions from input to output data. Such functional tasks correspond exactly to the operations of an abstract data type. In communication–based systems, however, the tasks are more likely application processes which are nondeterministic, nonterminating and reactive with respect to the environment.
It is the purpose of this paper to propose a new concept, called *abstract process types*, which combines the idea of abstract data types in the area of software system specification with processes in the sense of Petri nets and tasks in the sense of communication based and reactive system. Moreover, we hope that this notion of abstract process types is also a basis in order to develop a new semantics for Petri nets, which includes all the data type and process aspects and tasks in a suitable way.

2 From Abstract Data to Abstract Process Types

The concept of abstract data types was developed in the 70'ties and has become a fundamental concept of programming and specification of software systems. A *data type* is a collection of data domains and operations on these domains. Typical basic examples are stacks and queues, but also editors, compilers and in principle software systems with distinguished functionality can be considered as data types. An *abstract data type* is a class of data types which is closed under renaming of data domains and operations and hence independent of any specific representation. There are several well–known specification techniques for abstract data types, especially algebraic specifications (see [EM90]), which offer various structuring and refinement techniques in order to support the software development process. The initial algebra semantics of an algebraic specification $SPEC$ is represented by the corresponding quotient term algebra T_{SPEC}. In fact, the corresponding abstract data type is the class of all algebras $\mathcal{A}$, which are isomorphic to T_{SPEC}. This motivates that such an abstract data type can be represented uniquely up to isomorphism by the data domains (resp. base sets) and operations of a suitable algebra A isomorphic to T_{SPEC} (see figure 1).

The basic idea of *abstract process types* is to replace *operations* of *abstract data types* by suitable processes, called *open processes*, which can be considered as an abstraction of processes in the sense of Petri nets (see [Roz87] [Rei85]). In the same way as all the operations of an abstract data type are defined w.r.t. a given family of data domains, the base sets of the algebra A, the open processes of an abstract process type are defined w.r.t. a given base net. This means that the data domains / base sets of an abstract data type, i.e. the carrier of the algebra A representing the abstract data type, correspond to a given Petri net, called base net in this context of abstract process types. (see figure 1). An important property of algebras and abstract data types is the possibility to construct terms of operations, which can be considered as derived operations while a specific operation of an abstract data type corresponds to a specific task of the corresponding system each derived operation corresponds to a combination of tasks. Similar to the construction of terms of operations we want to have a mechanism for the concatenation or synchronization of open processes. This allows to construct derived open or closed processes, which correspond to combinations of tasks of the system. These derived processes can be constructed from a given abstract process type, but they do not belong explicitly to the abstract process type.

Finally, the representation independence of abstract data types should be transfered to abstract process types. A suitable notion of isomorphism for process types is required, such that an abstract process type can be represented uniquely up to isomorphism by a (concrete) process type. In fact there are already suitable notions for general morphisms and isomorphisms for different kinds of Petri nets (see [MM90], [PER95], [Pad96]), which can be extended to process types.

Abstract Data Type	**Abstract Process Type**
Data Domains / Base Sets Operations	Base Net Open Processes
Terms of Operations	Concatenation/Synchronization of Processes
Representation Independence by Abstraction up to Isomorphism	

Figure 1: Analogy of Abstract Data and Abstract Process Types

Finally, let us point out that our concept of abstract process types is not restricted to a specific kind of Petri nets, but can be defined for any kind of net including elementary, place/transition and algebraic high–level nets. Moreover, the same idea should be applicable also to other kinds of process specification techniques, like CCS, CSP, π–calculus, process algebras, statecharts and graph transformations. In some of these cases, however, it might be more suitable to consider abstraction in the sense of bisimulation equivalence instead of isomorphisms.

3 Open Processes and Abstract Process Types

In this section we want to propose a notion of *open processes* which is suitable to define *abstract process types*. The classical notion of a process p for a Petri net N is a pair $p = (P, f : P \to N)$ consisting of a occurrence net P together with a net morphism $f : P \to N$ (see [Pet77], [GR83], [BD87], [DMM91] and [MM90] for various notions of process). Nevertheless, for the purpose of considering process notions analogously to abstract data types we especially consider the following aspects for processes given by the morphism $f : P \to N$:

1. We do not want to restrict P to be an occurrence net, but would like to allow that P belongs to a given class of nets, called *process nets*. This class may be the class of all occurrence nets, the class of all nets, or some other suitable subclass of nets.

2. We do not want to consider a fixed initial marking for P, but allow to consider P with different kinds of markings corresponding to different input data for operations of abstract data types.

3. We want to distinguish between *normal places* of P and *interface places*, where the interface places of P are those places which are able to interact with the environment (see [NPS95]). In this context the environment of P are all items of the given net N which do not belong to the image $f(P)$ of P in N.

This leads to the following conceptual definition based on a given class of nets without initial markings, called class of process nets, and a given class of net morphisms:

Definition 1 (Open Process)
Given a Petri net N (with initial marking M_0) we have:

1. An **open process** p of N is a pair $p = (P, f : P \to N)$ consisting of a process net P and a net morphism $f : P \to N$.

2. The **context** $C(p)$ is the net N except of the image $f(P)$, i.e. $C(p) = N - f(P)$.

3. An **interface place** x of an open process $p = (P, f : P \to N)$ is a place x of P, such that the image $f(x)$ in N is adjacent to some transition in the context $C(p)$ of P in N. In this case x is also called interface place of the process net P.

4. An **autonomous action** of a marked process net (P, M) w.r.t. an interface place x of P is a step from (P, M) to (P, M') where M' and M may be different at most on the interface place x.

5. An **instantiation** of an open process $p = (P, f : P \to N)$ is a marking M of P, such that $f(M) \leq M_0$. In this case the marked process net (P, M) can be executed by firing of transitions and/or autonomous actions.

Given two open processes p_1 and p_2 of the same net N concatenation and synchronization of p_1 and p_2 have to be defined in such a way that the result p_3 becomes again an open process of N. More details will be given by the examples in the next section. Note, that an open process whose image is not initially marked may be executed by autonomous actions occuring on interface places. An open process that is neither initially marked nor has interface places, corresponds to an isolated and not initially marked subnet. This cannot be executed in the base net either.
Now we are able to give the following conceptual definition of concrete and abstract process types.

Definition 2 (Abstract Process Type)
1. a **concrete process type**, short process type,

$$PT = (N, (p_i)_{i \in I})$$

consists of a Petri net N, called **base net**, and a family $(p_i)_{i \in I}$ of open processes p_i of N.

2. An **abstract process type**, short APT, is an isomorphism class of concrete process types.

Finally, let us point out that for a given Petri net N there are several ways to define a process type with base net N. One extreme is to consider the family of all open processes of N, the other extreme to consider the empty family. In general, however, a suitable finite nonempty family of open processes should be more adequate to model all the tasks of a system. Vice versa, given the tasks of the system on an informal level it seems to be useful to model first each task $t_i (i \in I)$ by a process net P_i having in mind that specific places should become interface places. In a second step these process nets P_i should be combined to a Petri net N, such that each P_i can be extended to an open process p_i of N leading to a process type $PT = (N, (p_i)_{i \in I})$.

4 Examples of Abstract Process Types for Low and High-Level Nets

In this section we demonstrate our concepts with two simple examples. However, the motivation for the new concepts in this paper is due to the study of large reactive and communication based systems modeled by high–level nets. Our first well–known example is the place/transition net of dining philosophers introduced in 1971 by Dijkstra. Our second example is an extension of this net to a *Restaurant of Dining Philosophers* represented as an algebraic high–level net.

4.1 Abstract Process Types in Place/Transition Nets

Figure 2 shows the base net for five dining philosophers sitting around a table with five sticks lying at the corresponding sides of the philosophers. Each philosopher shares his left and right sticks with his left and right neighbour, respectively. A philosopher can either eat or think. A thinking philosopher can start eating if his both sticks lying on the table. The initial marking consists of tokens on the places $think_1, ..., think_5$ and on the places $side_1, ..., side_5$, meaning that all philosophers are thinking. The abstract process type has the following representation

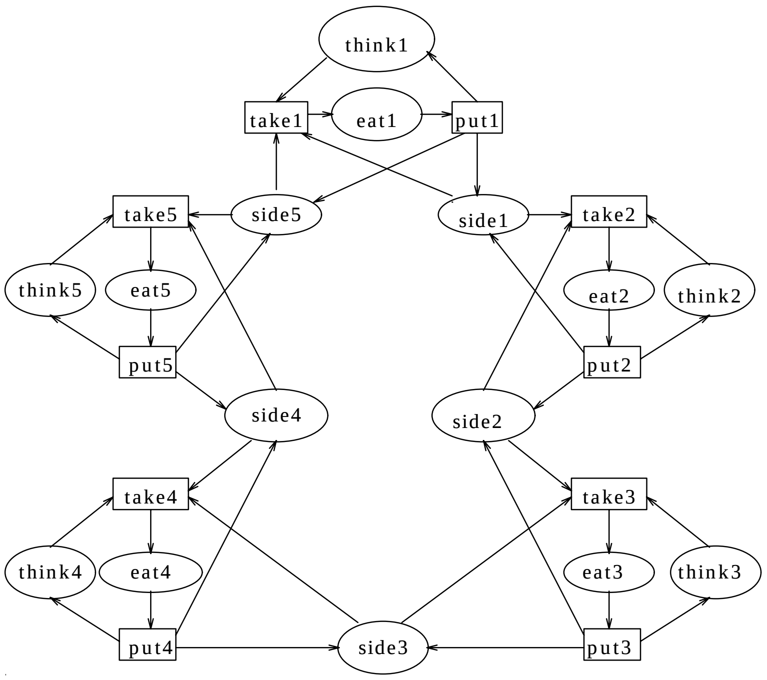


Figure 2: Base net of process type for dining philosophers (N)

$$PT = (N, (P_1, f_1), (P_2, f_2), (P_3, f_3), (P_4, f_4), (P_5, f_5))$$

consisting of the base net N in figure 2 and five open processes, one for each philosopher. The process nets P_3 and P_4 are illustrated in the left part of figure 3. Note, that in these examples the morphism – an inclusion – is denotated by the names of the places. The intended morphism maps places of the open process net to the places with the same name in the base net. The places with dashed lines represent interface places, denoting the communication with other processes by autonomous actions.

With $M = think_4$, the marking of the process net P_4 and $f(M) \leq M_0$ we get an instantiation (see 1(5))of the open process (P_4, f_4) as a marked net (P_4, M). The event that $philosopher_3$ stops eating and puts his sticks back to the table, leads to a new marking $M' = think_4 + side_3$ of the process net P_4. It is also possible to take a token from an interface place (1(3)). We denote this change of the marking of an interface place without firing of a transition as an autonomous action. A possible firing sequence for the marked process net (P_4, M) is given by

$$think_4 \overset{[side_3]}{\longrightarrow} think_4 + side_3 \overset{[side_4]}{\longrightarrow} think_4 + side_3 + side_4 \overset{[take_4>}{\longrightarrow} eat_4 \overset{[put_4>}{\longrightarrow} \ldots$$

with $\overset{[pl]}{\longrightarrow}$ as a symbol for autonomous actions on place pl and $\overset{[t>}{\longrightarrow}$ for the usual firing of a transition t. More complex processes constructed from those defined in our process type PT can

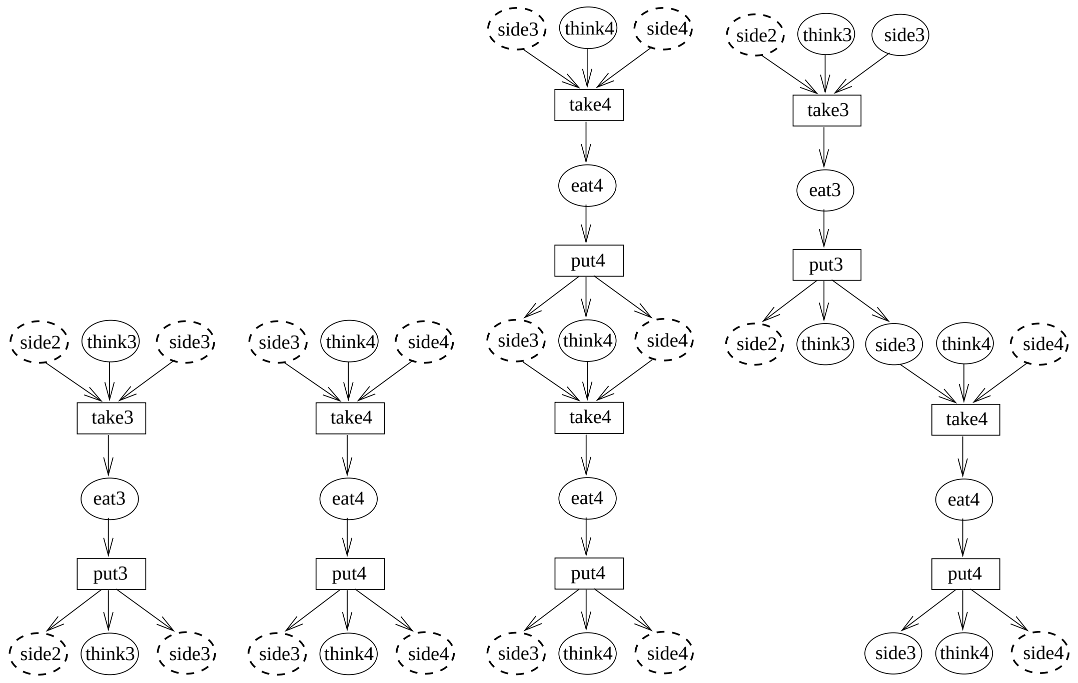
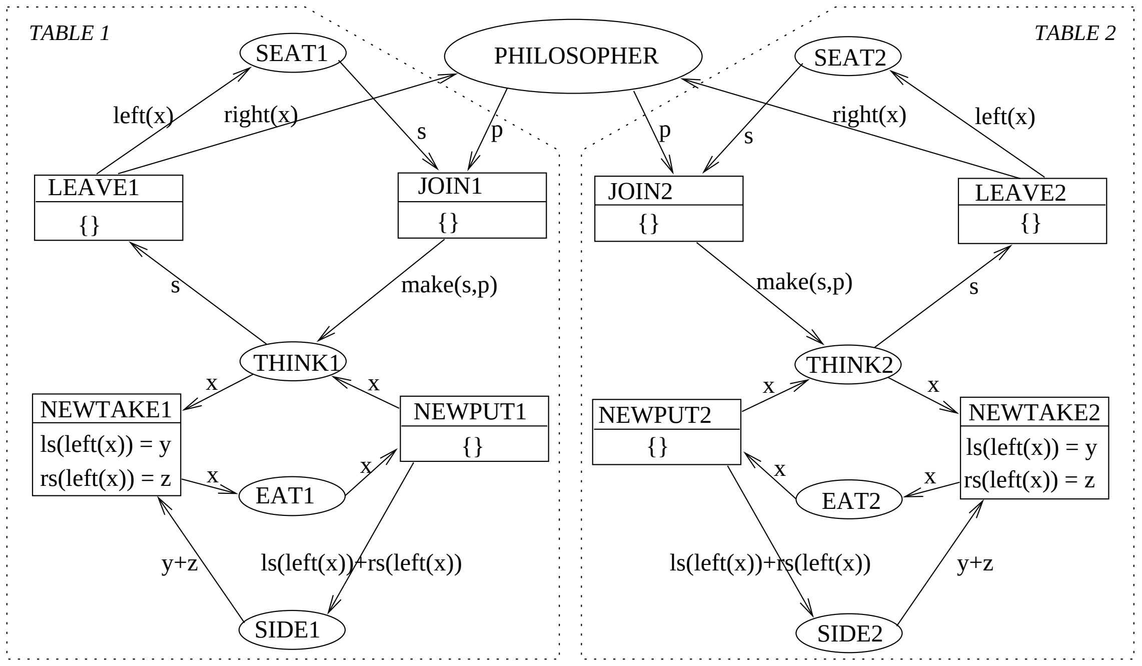


Figure 3: Two open processes $(P_3, f_3), (P_4, f_4)$, concatenation of (P_4, f_4) with itself and synchronization of (P_3, f_3) and (P_4, f_4)

be described by combination principles like concatenation and synchronization(see figure 3). The concatenation of two processes is defined in terms of gluing the output places of the first process to the input places of the second process provided that the input and output places have the same image in the base net. A special case of concatenation of processes is the synchronization. In this case interface places become closed by gluing, such that the places after gluing are no longer interface places. This is due to the fact that all transitions in the pre- or post domain of thus places covered by the synchronized process. Synchronization means that the second open process has to wait for the first open process. Concatenation and synchronization in general allow the construction of an infinite number of open processes for this example.

4.2 Abstract Process Types in Algebraic high–level Nets

In this second part we illustrate the applicability of our approach for high–level nets, especially for algebraic high–level nets (AHL–nets).



diphi $=$

sorts :	$philo, stick, seat$
opns :	$p_1, ..., p_5 :\to philo$
	$st_1, ..., st_5 :\to stick$
	$s_1, ..., s_5 :\to seat$
	$make : seatphilo \to seat \times philo$
	$ls : seat \to stick$
	$rs : seat \to stick$
	$left : seat \times philo \to seat$
	$right : seat \times philo \to philo$
eqns :	$ls(s_1) = st_1, ls(s_2) = st_2, ls(s_3) = st_3, ls(s_4) = st_4, ls(s_5) = st_5$
	$rs(s_1) = st_5, rs(s_2) = st_1, rs(s_3) = st_2, rs(s_4) = st_3, rs(s_5) = st_4$

$\boldsymbol{A_{diphi}}$ **:**

A_{philo}	$=$	$\{uta, anne, otto, elli, ingo, ulla, anke, ilka, udo\}$
A_{stick}	$=$	$\{a, e, i, o, u\}$
A_{seat}	$=$	$\{ua, ae, ei, io, ou\}$
$make_A(x, y)$	$=$	tuple of x, y, for $x \in A_{seat}, y \in A_{philo}$
$ls_A(x)$	$=$	first letter of x, for $x \in A_{seat}$
$rs_A(x)$	$=$	last letter of x, for $x \in A_{seat}$
$left_A(x)$	$=$	left part of x, for $x \in A_{seat} \times A_{philo}$
$right_A(x)$	$=$	right part of x, for $x \in A_{seat} \times A_{philo}$

Figure 4: The restaurant of dining philosophers: base net (N), specification ***diphi*** and algebra $\boldsymbol{A_{diphi}}$

The AHL–net in figure 4 is an extension of dining philosophers (see [PER95]). The restaurant consists of two tables, each with five seats and five sticks. The philosopher are able to join a table, assuming there is a vacant seat at this table, or to leave a table. In contrast to the first example in section 4.1 there are individual token for philosophers and sticks which are explicitly presented in the algebra that belongs to the AHL-net. According to the general concept of AHL-

nets the inscriptions of arcs consist of a linear sum of terms denoting which data elements are consumed and created in case of firing a transition. On the one hand the firing of a transition depends on the presence of tokens on places that belong to the preconditions of a transition. On the other hand the firing depends on the satisfaction of the firing condition (denotated inside the rectangles of transitions).
Let us consider the following abstract process type represented by

$$PT = (N, (P_1, f_1), (P_2, f_2), (P_3, f_3), (P_4, f_4), (P_5, f_5), (P_6, f_6))$$

where (P_1, f_1), (P_2, f_2) and (P_3, f_3) are shown in figure 5 (numbered 1, 2 and 3) and (P_4, f_4), (P_5, f_5) and (P_6, f_6) are similar open processes for the second table on the right hand side of the base net in figure 4. Due to the small site of our example the specification and algebra are the same as in the base net N. This is not a required characteristic of an open process for AHL–nets.

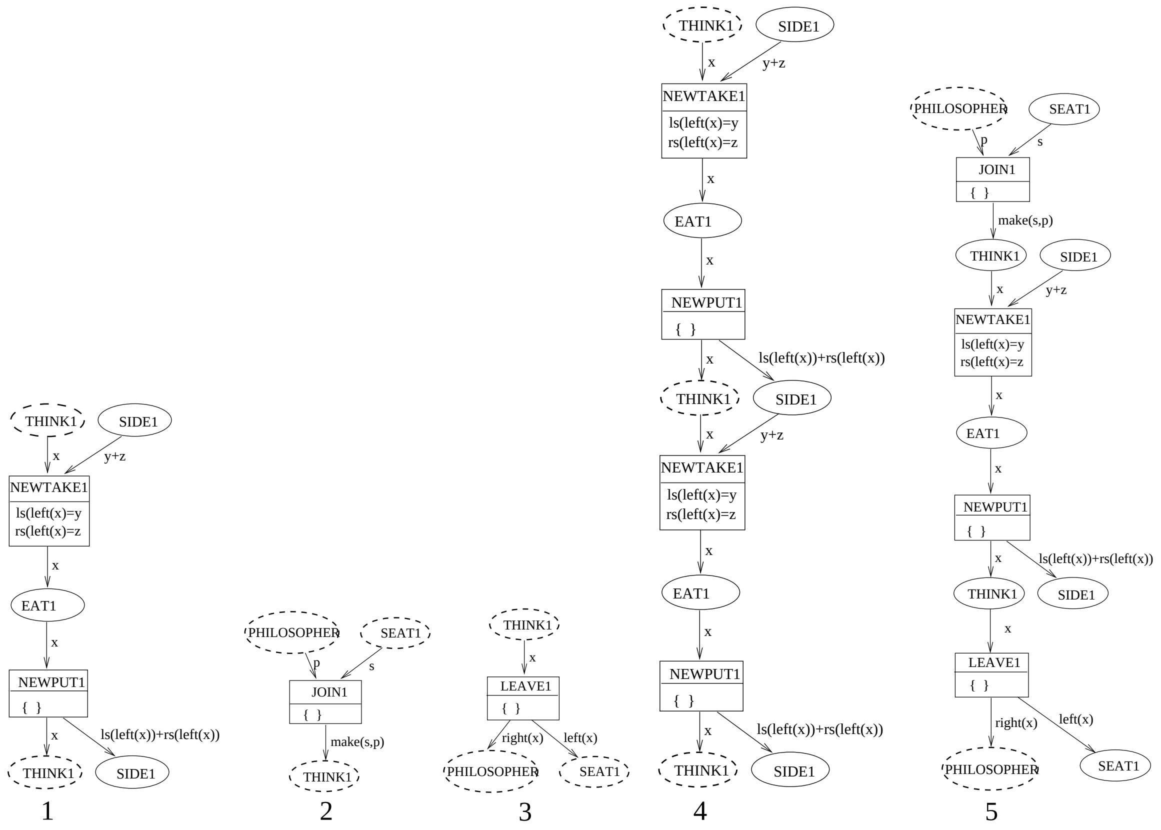


Figure 5: Open processes of the base net N

The open processes presented in figure 5 can be interpreted in the following way:

1. If there are two sticks available on the appropriate sides of the thinking philosophers they can take the sticks and start eating. Afterwards they put the sticks back onto the table and think again.

2. Philosophers take a seat, join $table_1$ and think.

3. Thinking philosophers leave $table_1$.

4. This open process is the concatenation of the first open process with itself.

5. This open process is constructed by synchronisation of the second, first and third open process in figure 5. Note, that the places $think_1$ and $seat_1$ are interface places of the given three open processes, but become closed in the synchronized process net. This is due to the fact that the image of $think_1$ and $seat_1$ in N is not adjacent to some transition in the context of this synchronized open process.

Note, that the idea of AHL-nets implies the use of the set of all philosophers instead of a single philosopher within one open process as considered in section 4.1. Thus the distinguished tasks are concerned with all philosophers at one specific table instead of one specific philosopher. This is also the reason for the place $side_1$ to be closed, that is no interface place
Instantiation of open processes and execution of marked processes as well as concatenation and synchronization in the high–level case are defined similar to the corresponding notion in the low–level case presented in section 4.1 with the distinction that token are elements of the domain of an algebra instead of black token.

5 Conclusion

In this paper we have motivated, that it is useful to transfer the concept of abstract data types, well–known from software system specification, to Petri nets. The new concept proposed in this paper, called abstract process type is motivated by several case studies including a lecture of E. Schnieder in Berlin on *European Train Control Systems*. It is based on the notion of an open process, which is able to interact with the environment. The discussion in this paper is only a first step towards a theory of abstract process types for communication based and reactive systems. Having in mind the benefits of abstract data types for software system design and development we hope that abstract process types based on Petri nets may play a similar role for design and development of communication based and reactive systems. Future work especially includes the application of the introduced concepts to the motivating case studies and systems. We are convinced that this concept of abstract process types is also a suitable basis for a module concept for Petri nets comparable with module specifications in [EM90] based on algebraic specifications.

References

[BD87] E. Best and R. Devillers, *Sequential and Concurrent Behaviour in Petri Net Theory*, Theoretical Computer Science (Elsevier, 1987), no. 55, 87–136.

[DMM91] P. Degano, J. Meseguer, and U. Montanari, *Axiomatizing the Algebra of Net Computations and Processes*, Tech. Report TR 1/91, Dipartmento Informatika, Universitá di Pisa, 1991.

[EM90] H. Ehrig and B. Mahr, *Fundamentals of algebraic specifications 2 , module specifications and constraints*, EACTS-Monographs in Theoretical Computer Science, vol. 21, Berlin, 1990.

[GR83] U. Goltz and W. Reisig, *The Non-Sequential Behaviour of Petri Nets*, Information and Computation, Academic Press, 1983, pp. 125–147.

[MM90] J. Meseguer and U. Montanari, *Petri nets are monoids*, Information and Computation **88** (1990), no. 2, 105–155.

[NPS95] M. Nielsen, L. Priese, and V. Sassone, *Characterizing Behavioural Congruences for Petri Nets*, Proceedings CONCUR'95, LNCS, vol. 962, Springer Verlag, 1995, pp. 175–189.

[Pad96] J. Padberg, *Abstract Petri Nets: A Uniform Approach and Rule-Based Refinement*, Ph.D. thesis, Technical University Berlin, 1996, Shaker Verlag.

[PER95] J. Padberg, H. Ehrig, and L. Ribeiro, *Algebraic high-level net transformation systems*, Mathematical Structures in Computer Science **5** (1995), 217–256.

[Pet77] C. A. Petri, *Non-Sequential Processes*, Interner Bericht ISF-77-5, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, 1977.

[Rei85] W. Reisig, *Petri nets*, EATCS Monographs on Theoretical Computer Science, vol. 4, Springer-Verlag, 1985.

[Roz87] G. Rozenberg, *Behaviour of elementary net systems*, Advances in Petri nets 1986 (W. Brauer, W. Reisig, and G. Rozenberg, eds.), Springer Verlag Berlin, 1987, pp. 60–94.

[WRE95] H. Weber, W. Reisig, and H. Ehrig, *Concept, Theoretical Foundation, and Validation of an Application Oriented Petri Net Technology*, Proposal for a "Forschergruppe" to the German Research Council (DFG), 1995, (Accepted as a DFG-project from April 1996 to March 1999).