Formal Development of Concurrent Systems using Algebraic High-Level Nets and Transformations

Leila Ribeiro Hartmut Ehrig Julia Padberg

Technical University Berlin

Abstract

In this paper we present algebraic high-level nets: a combination of algebraic specifications and Petri nets. Algebraic specifications are used to specify the data-structure and Petri nets to specify the data-flow. This combination is a very powerful description technique. Moreover we introduce net transformations, net fusions and net unions as structuring techniques. Fusions and unions can be considered as horizontal structuring mechanisms in the sense that we combine nets to obtain a new net consisting of the given components. Fusions capture the notion of sharing of subnets, while unions are kinds of net composition. Net transformations are based on concepts from graph grammars because net refinements are defined via productions. Transformations can be seen as high-level vertical refinements. Moreover we show compatibility between these techniques, i.e. in our framework vertical refinement and horizontal structuring of nets are compatible.

Resumo

Neste artigo são apresentadas redes algébricas de alto nível: uma combinação de especificações algébricas e redes de Petri.Especificações algébricas são usadas para a descrição da estrutura dos dados e redes de Petri para a descrição do fluxo dos dados. Esta combinação é uma técnica de especificação muito poderosa. Transformações de, fusões e uniões de redes também são introduzidas neste artigo como técnicas para a estruturação da especificação. Fusões e uniões são usadas como mecanismos de estruturação horizontal no sentido em que redes sào combinadas para se obter redes maiores. Transformações de redes são baseadas em conceitos de gramática de grafos, refinamentos de redes são definidos em termos de produções, e podem ser vistos como refinamentos de redes de alto nível. Em nosso formalismo existe compatibilidade entre refinamentos verticais e estruturaç ao horizontal.

1 Introduction

Formal specification techniques for concurrent systems can be divided in two main groups: the group following the interleaving approach and the group following the true concurrency approach. The interleaving approach is based on the idea that although the processes may run concurrently, the observation of their behavior is a sequence of events. Following this line there are many specification methods using different kinds of processes in process algebras [4], CCS [27] and CSP [18]. In the true concurrency approach, as the name says, events that occur concurrently are not ordered, leading then to a partial order of events. The most famous example of method belonging to this group are Petri nets [30, 31]. Graph transformations [9] and Actor systems [2] are also models of true concurrency. All these methods have a solid theoretical background and have been used in different kinds of practical applications (for Petri nets see [23]).

All these specification techniques deal mostly with the behavioral aspect of the systems, very few informations concerning data types (if at all) are included in these methods. That means that they are suitable only for the description of the data-flow of the system, but not for specifying the data itself. Research on formal specifications of data types has been done already for a long time and there are many well-established methods that allow us to prove nice properties of the specifications, like consistency and correctness. Among the many data type specification techniques, we have methods based on denotational semantics, e.g. VDM [5] and Z [1], algebraic specifications [13], and methods following other paradigms like object-orientation and logic programming.

Until a certain stage in the process of software development one can specify data-type and data-flow separately, but as these two belong together in the system that is being specified they should be put together also in the specification. One of the main aims of using formal methods to specify software is to be able to prove properties, especially correctness, of systems. If we use different formalisms to describe data-types and data-flow, and prove that these two specifications are correct separately, we cannot conclude by default that putting them together will yield a correct system (this would be the case even using the same formalism without a suitable notion of composition). For this reason many attempts have been made to present methods in which it is possible to specify data-type and data-flow within the same framework. Examples of these are LOTOS [24], process algebras [4], projection specifications [15], the SMOLCS approach [3].

Now recall the usual textbook definition (see [31]) of a place/transition net as a 3-tuple (P,T,F) where P is a nonempty, finite set of places, T a finite set of transitions and F is a mapping $F : (P \times T) \uplus (T \times P) \to N$ that yields the casual dependency relation of the transitions and the places. F represents the pre- and post-conditions for the firing of the transitions. The algebraic way to describe a place/transition-net is to view the pre- and post-conditions not as a mapping from pairs to natural numbers (that represent the numbers of tokens involved in the switchings of transitions) but as functions from transitions to the free commutative monoid $P^{\oplus}$ over the places of the net [26]$(pre, post : T \to P^{\oplus})$. For example, $pre(t1) = 3 \times p1 \oplus p2$ means that for the switching of the transition $t1$ it is necessary that place $p1$ has at least 3 tokens and place $p2$ at least one token.

A place/transition net based on groups, short P/T-G-Net, (see [7, 10, 14]) $N = (P,T,pre,post)$ consists of sets P and T (places and transitions, respectively) and functions pre and $post$:$T \to P^{\odot}$ (from T into the free abelian group $P^{\odot}$ over P with addition $\oplus$ and subtraction $\ominus$) . The G in the name P/T-G-Net stands for *group*. An element of $P^{\odot}$ is called *marking* of a P/T-G-Net. The marking $m_1 = 2 \times p_1 \oplus 3 \times p_2$ means that we have 2 tokens in place p_1 and 3 tokens in place p_2. In fact, in this model we can also have negative markings like $m_2 = \ominus p_3$ corresponding to a negative number of tokens in p_3. This allows to formulate the successor marking m' of m after switch of transition t in a purely algebraic way by $m' = m \ominus pre(t) \oplus post(t)$ P/T-nets based on groups are the appropriate algebraic framework to study invariants of nets [7].

Colored Petri nets [22] and predicate/transition nets [16] were the first models based on Petri nets

to describe data-type and data-flow in a unified framework. The main idea of these kinds of nets is to have colored tokens, i.e. different kinds of tokens belonging to some token sorts. This idea was further developed by combining Petri nets with algebraic specifications (see [32]) leading to our notion of algebraic high-level nets [21, 7, 10, 14]. The key idea is not to use any kind of token sets more or less ad hoc chosen, but to use an algebraic specification to describe these token sorts. This way we can take advantage from the already existing concepts from the algebraic specification theory, e.g. modularization concepts and semantical constructions.

The combination of algebraic specifications and Petri nets is very fruitful because it brings a new insight to Petri nets, solving some of the problems of this approach from the software engineering point of view (see section 2). In section 3 we give the definition of algebraic high-level nets and introduce a file server's example. We give also structuring techniques for algebraic high-level nets (section 4). Vertical structuring can be achieved through net transformations, and the concepts of fusion and union can be considered as horizontal structuring. In fact, these constructions can be used also for other kinds of Petri nets whose corresponding categories satisfy suitable properties (see [29]).

We assume the reader to be familiar with some basic notions of algebraic specifications in the sense of [13].

Acknowledgments: This paper has been developed within a forthcoming German-Brazilian Cooperation on software engineering techniques, partially supported by a CNPq-grant for Leila Ribeiro, a DFG-grant for Julia Padberg and by the ESPRIT Basic Research working groups COMPASS and COMPUGRAPH.

2 What are the problems with Petri nets from software engineering point of view?

In this section we start with some common objections against classical Petri nets, like P/T nets, from a software engineering point of view. In a second step we analyze each of these objections in more detail, show how far the corresponding problems have been studied in the literature already and sketch also some new solutions which will be studied in the subsequent sections in more detail.

Some main objections against P/T-nets as a model for concurrency and as a specification technique for distributed systems from a software engineering point of view are given in the following four slogans:

- *Petri nets lack abstraction*
  This means that the concept of Petri nets is a low level concept, comparable with machine level programming languages, which does not allow higher level abstraction, structuring principles and suitable compositionality which is essential for specification of distributed systems.

- *Petri nets lack data-type handling*
  Usually the objects that are created/deleted/modified by a Petri net are treated as "black dots" (tokens). One knows that there is an object in a place, but it is not possible to say which object it is, what are its properties, etc. Moreover, it is impossible to specify which kind of modifications an object suffers by the firing of a transition. For example, when a transition fires, it consumes tokens from some places and put new tokens in some other places, but we do not know whether this new tokens represent the same objects as the deleted ones, completely different objects or some modification on the deleted objects. From the software engineering point of view this is a weak point because the data-types play a very important role in the specification of a system. It must be possible to represent exactly which are the tokens in each place, and how the tokens that are created by transitions relate to the consumed tokens to specify completely a software system.

- *Petri nets lack refinement*
  This common objection means that usually the net structure of a Petri net is considered to be fixed and therefore Petri nets that these are no suitable techniques for stepwise refinement of net items and net structure as usual in top-down or bottom-up development techniques.

- *Petri nets lack structuring*
  Very often when specifying practical systems, the size of the Petri net that is obtained as specification is so big that it is quite difficult to understand what is going on there. In other specification methods, the use of suitable horizontal structuring techniques tackles with this problem, whereas for Petri nets there is no well-established notion of structuring.

In the following we will analyze how far these objections are really justified and what has been done concerning these problems in the last few years. The answer will be presented again in slogans (see the headlines of 2.1,2.2 and 2.3), but in this case together with a more detailed reasoning.

2.1 Algebraic high-level nets are abstractions of Petri nets

Now let us reconsider our first and second common objections which claims that Petri nets lack abstraction and data-type handling. In fact, several different notions of high-level nets have been introduced in the last years already (see [23]). We will focus especially on algebraic high-level nets, a combination of Petri nets with algebraic specifications.

The first important step to achieve a higher level of abstraction for Petri nets is the concept of colored Petri nets introduced by Jensen [22]. This concept allows colored tokens, i.e., different kinds of tokens in the places, and also colored transitions, i.e., different modes under which transitions can be switched. If different kinds of tokens are represented by data elements of an algebraic data type, arcs of the nets are labeled by terms with variables of the corresponding algebraic specification and a mode of a transition is given by an assignment for the variables of the terms on adjacent arcs of the transition we obtain the notion of algebraic high-level nets (see [21, 8, 10]). According to [23], most practical applications of Petri nets in recent years are based on some kind of high-level nets. Complex systems like computer chips, communication protocols, flexible manufacturing systems, radar surveillance and electronic funds transfer had been specified using these nets.

The essential idea of combining Petri nets with algebraic specifications was given already by Vautherin in [32]. In section 3 of this paper we present a revised version of algebraic high-level nets and a flattening construction $FLAT$ from algebraic high-level nets to P/T-G-Nets which is compatible with markings and switching behavior on both levels. That means that for each algebraic high-level net there exists an equivalent place/transition net w.r.t. markings and switching of transitions. In general, the flattened version of an algebraic high-level net is an infinite place/transition net. The flattening construction allows to extend all notions, like marking graphs and invariants from place/transition nets to algebraic high-level nets. For a given AHL-Net AN we can compute the marking graph and the invariants for the corresponding P/T-G-Net $FLAT(AN)$. On the other hand it is also useful to study these notions directly for high-level nets. In these papers it was also shown that invariants for algebraic high-level nets can be computed in a compositional way.

2.2 Net transformation systems are high-level refinements for Petri nets

Now let us reconsider the objections against Petri nets claiming that the net structure of Petri nets is not dynamic and does not allow suitable refinements. In fact, there are several attempts to define refinements of places and transitions in nets, which might be considered as low level refinements. We want to consider also high-level refinements leading to the notion of net transformation systems.

Net transformation systems have been introduced for place/transition nets and algebraic high-level nets in our paper [28]. They are based on productions $p : L \rightarrow R$, where the left hand side L and the right hand side R are nets. Similar to Chomsky grammars and graph grammars a production $p : L \rightarrow R$ is applied to a net N by finding a suitable occurrence of L in N and replacing L by R leading to a new net N'. This kind of replacement can be considered as a high-level refinement of nets. In section 4 of this paper we study transformation systems for algebraic high-level nets. In fact, we are able to extend some main results concerning parallelism and concurrency from the theory of graph grammars to AHL-Net-transformation systems using the general theory of high-level replacement systems [11]. This allows to analyze concurrency in nets not only on the token level but also on the level of transformations of the net structure, what is very important for stepwise refinement of algebraic high-level nets as specifications for distributed systems.

2.3 Net fusions and unions are horizontal structuring for Petri nets

Together with net transformations, we also present the notions of fusion and union as net structuring techniques. Fusion [20] can be interpreted as identification of shared subnets. To increase readability and understanding of a Petri net, we allow the same subnet may be represented twice (or more times) within a net by putting a "virtual link" among the occurrences of this subnets such that in the semantical level this subnet is represented only once and has all links from its occurrences. Union and disjoint union are different ways of composing nets, with or without some intersection between the component nets. Of course, disjoint union is a special case of union. We show that net fusions and unions and net transformations are compatible. This compatibility between vertical and horizontal structuring is a very desired property in a software engineering framework.

3 Algebraic high-level nets

As discussed already in section 2.1, algebraic high-level nets, short AHL-Nets, are a combination of place/transition nets and algebraic specifications. In an AHL-Net tokens are specified by the algebraic specification part. The pre- and post-condition functions define not only how many tokens are involved in each switching of a transition, but also which tokens are involved. AHL-Nets allow a higher level of abstraction as we will see in the file server's example presented as an AHL-Net in this section. In a first step towards AHL-Nets we define a corresponding notion of net signature.

3.1 Definition (AHL-SIG)

An algebraic high-level net signature, short AHL-Sig, $NS = (SIG, P, T, pre, post)$ consists of a signature $SIG = (S, OP)$, sets P and T (places and transitions, respectively) and functions $pre, post : T \rightarrow (T_{OP}(X) \times P)^{\oplus}$ assigning to each $t \in T$ an element of the abelian group over the cartesian product of terms $T_{OP}(X)$ with variables in a suitable set X and the set P of places. The pre (post) function assigns to each transition a sum of terms together with their places that are consumed(created) by switching this transition. □

Remark: For simplicity we assume that the set of variables X used in the term algebra $T_{OP}(X)$ is essentially the same for all AHL-Sigs. In fact, X is obtained by a fixed set X_{fix} of variables which is indexed by the set of sorts S of the signature $SIG = (S, OP)$, i.e. $X = X_{fix} \times S$.

Using this net signature we add equations and a corresponding algebra to get the notion of an algebraic high-level net.

3.2 Definition (AHL-NET)

An algebraic high-level net, short AHL-Net, $AN = (NS, E, A, cond)$ consists of an AHL-Sig $NS = (SIG, P, T, pre, post)$, a set of equations E over the signature, a (SIG, E)-algebra A and a function $cond : T \to \mathcal{P}_{fin}(EQNS(SIG))$ assigning to each $t \in T$ a finite set $cond(t)$ of equations. This set represents the conditions that must be satisfied for each transition to switch (see 3.4). □

Remark: This notion of AHL-Nets based on AHL-Sigs is a revised version of corresponding notions in [32], [21], [7], [8] and [10]. Previously the notions of algebraic high-level net schemes corresponding to our signatures included equations already and schemes as well as nets were sorted. The latter means that for each place $p \in P$ we have a unique sort $s \in S$ such that only terms and data elements of sort s are allowed for place p. Our revised version is mixed-sorted because it allows terms and data elements of different sorts for each place. This is more flexible for applications and easier in the mathematical notation. An extended notion of AHL-Nets including capacities and initial markings is given in [10].

3.3 Example: File server

A very simple file server can be represented as a net shown in figure 3.1. There are two places: the file server (*FS*) and the storage of files (*STO*), two transitions corresponding to the mode in which a request for a file is made: read only (*READ*), read/write (*WRITE*) and one transition corresponding to the end of a write access, releasing files (*REL*). The basic idea is to prevent that two processes write simultaneously on the same file. Using the classical P/T-nets one would have to have a control scheme for each file, since it is not possible to distinguish among the tokens representing the files. Using AHL-Net-nets we can model this file server with a very simple net. We assume to have already the specifications of bool, filename, file and proc, together with their corresponding signatures, and assume that the sorts corresponding to these names are defined there. The AHL-Sig for this file server $NS = (SIG, P, T, pre, post)$ is given as follows:

SIG: sig-fs = sig-bool + sig-filename + sig-file + sig-proc

sorts: Req,Status,Fs-request,Fs-answer,Storage,Sto-elem

opns:
read, *write*, *rel* :→ Req
av, not-av :→ Status
fs-req : Proc × Filename × Req → Fs-request
fs-ans : Proc × Filename × File → Fs-answer
sto : Filename × File × Status → Sto-elem
empty-sto :→ Storage
ins-sto : Storage × Sto-elem → Storage
is-in-sto : Storage × Filename → Bool
file-in-sto : Storage × File → Bool
exists : Storage × Sto-elem → Bool
get-file : Storage × Filename → File
chg-sto : Storage × Filename × Status → Storage
is-av : Storage × Filename → Bool

$P = \{FS, STO\}$
$T = \{READ, WRITE, REL\}$
$pre = \{READ \mapsto (fs\text{-}req(p, fn1, read), FS) \oplus (s, STO),$
$\quad WRITE \mapsto (fs\text{-}req(p, fn1, write), FS) \oplus (s, STO),$
$\quad REL \mapsto (fs\text{-}req(p, fn1, rel), FS) \oplus (s, STO)\}$
$post = \{READ \mapsto (fs\text{-}ans(p, fn1, get\text{-}file(s, fn1)), FS) \oplus (s, STO),$
$\quad WRITE \mapsto (fs\text{-}ans(p, fn1, get\text{-}file(s, fn1)), FS) \oplus (chg\text{-}sto(s, fn1, not\text{-}av), STO),$
$\quad REL \mapsto (chg\text{-}sto(s, fn1, av), STO)\}$

Each request of a file can be done by the operation *fs-req* having as arguments the process that is requesting a file, the name of this file and which kind of request is being made (write, read or release the file). The answer of the file server is the operation *fs-ans*, that sends to the process its requested file. The storage consists of a table with filenames, files and the current status of the files, that can be available or not available for writing (in case another process is already writing on this file). The remaining operations are meant to answer if a filename occurs in the storage, if a file is already in the storage, if a storage element (filename, file and status) is in the storage, to get a file from the storage, to change the status of a file in the storage and to answer if a file is available for writing. In this step we give only the signature, the equations for this operations are added when we define the corresponding AHL-Net. As the pre- and post-conditions assign for each transitions a sum of terms with variables over the signature, we have to define the set of variables to be used. We use as variables the subset X' of the set X defined in 3.1: $X' = \{(p, \text{Proc}), (fn1, \text{Filename}), (fn2, \text{Filename}), (file1, \text{File}), (file2, \text{File}), (req, \text{Req}), (st1, \text{Status}), (st2, \text{Status}), (s, \text{Storage})\}$. As we do not use the same variable name for variables of different sorts, in the rest of the paper we just write the variable name without causing any confusion.

The *pre*-function assigns for each transition which tokens must be in which places for this transition to switch. For example, the switching of the transition *READ* requires that a token corresponding to the term $fs\text{-}req(p, fn1, \underline{read})$ is in the FS-place (request from some process p to read a file $fn1$) and ("and" is denoted by the plus operator of abelian groups $\oplus$) that a token corresponding to s is in the STO-place (the storage).

Note that we do not assign sorts to places, that means that elements belonging to different sorts may be at the same place. In this example, the place FS may have tokens from sort Fs-request and Fs-answer(in fact, it may have even other kinds of tokens, but these are explicitly used by the transitions that are connected to this place).

In order to get an AHL-Net $AN = (NS, E, A, cond)$ for this AHL-Sig NS we add a set of equations E to the signature SIG, written $\underline{\underline{\text{fs}}} = (\underline{\underline{\text{sig-fs}}}, E)$, give a (SIG, E)-algebra $A_{\underline{\underline{fs}}}$, and a function *cond*:

$E =$ equations of bool,filename, file, proc and equations specifying the storage as a list of files, such that one file does not have different names, one filename does not correspond to different files and one file can not be available and not available at the same time, and the operations on the storage as usual list operations.

$A_{\underline{\underline{\text{fs}}}}$:

$A_{Bool} = \{\mathbf{T}, \mathbf{F}\}$

$A_{File} = \{\mathbf{file1}, \mathbf{file2}, \mathbf{file3}\}$

$A_{Filename} = \{\mathbf{fn1}, \mathbf{fn2}, \mathbf{fn3}\}$
(each of the elements of A_{File} represents a file. Here we abstracted from more concrete representations of files)

$A_{Proc} = N$

$A_{Req} = \{\mathbf{read}, \mathbf{write}, \mathbf{rel}\}$

$A_{Status} = \{\mathbf{av}, \mathbf{not\text{-}av}\}$

$A_{Fs\text{-}request} = A_{Proc} \times A_{Filename} \times A_{Req}$

$A_{Fs\text{-}answer} = A_{Proc} \times A_{Filename} \times A_{File}$

$A_{Sto\text{-}elem} = A_{Filename} \times A_{File} \times A_{Status}$

$A_{Storage} =$ each of the elements of $A_{Storage}$ is a list of $A_{Sto\text{-}elem}$ satisfying the equations of the specification.

$cond = \{READ \mapsto is\text{-}in\text{-}sto(s, fn1), WRITE \mapsto is\text{-}av(s, fn1), REL \mapsto\sim is\text{-}av(s, fn1)\}$

This net AN can be graphically represented as it is shown in figure 3.1. In each transition we add to its name the set of equations that must hold for this transition to switch. The labels of the arcs represent the elements of the places which are involved in the switching of the transitions. Each label should be written as an element of $(T_{OP}(X) \times P)^{\oplus}$ but we avoid this in the

graphical representation by writing, for example, $y \oplus z$ instead of $(y, PLACE) \oplus (z, PLACE)$. The tokens that may be in the places are specified by the algebraic specification fs. They belong to the carriers of the algebra A_{fs}. For example, if there are tokens $(34, \mathbf{fn3}, \mathbf{write})$ in FS and $\langle(\mathbf{fn1}, \mathbf{file1}, \mathbf{av}), (\mathbf{fn2}, \mathbf{file2}, \mathbf{not\text{-}av}), (\mathbf{fn3}, \mathbf{file3}, \mathbf{av})\rangle$ in STO (from carriers $A_{Fs\text{-}request}$ and $A_{Storage}$ resp.), the transition $WRITE$ may switch (its condition is true for these tokens). The consequence then is that these tokens are removed from their places and new tokens are generated (according to the *post* function). In this example, the token $(34, \mathbf{fn3}, \mathbf{file3})$ would be put in the place FS and a new storage in which the entry for the file **fn3** is not-av is put into the STO place.

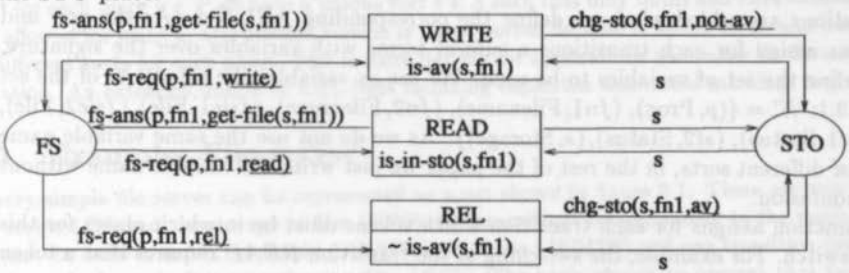


Figura 3.1

3.4 Definition (Operational behavior of AHL-Nets)

Given an AHL-Net $AN = (SIG, P, T, pre, post, E, A, cond)$ *as presented in 3.1 and 3.2 we define:*

1. *The set of consistent transition assignments, is* $CT = \{(t, ass_A) | t \in T, ass_A : Var(t) \to A$ *s.t.* A *satisfies the equations* $cond(t)$ *with variables* $Var(t)$ *under the assignment* $ass_A\}$. $Var(t)$ *is the set of variables that occur in the condition equations* $cond(t)$ *and in the pre and post conditions* $pre(t)$ *and* $post(t)$ *for each transition* $t \in T$.

2. *The set of place vectors* PV, *also called marking group of* AN, *is the free abelian group* $PV = (A \times P)^{\circledcirc}$ *Within this context* A *is considered to be* $A = \biguplus_{s \in S} A_s$, *where* A_s *is the domain of* A *of sort* s.

   An element of PV *is called a marking of the AHL-Net* AN.

3. *The A-induced functions* $pre_A, post_A : CT \to PV$ *of the AHL-Net* AN *are defined for all* $(t, ass_A) \in CT$ *by*

$$pre_A(t, ass_A) = ASS_A(pre(t)) \text{ and } post_A(t, ass_A) = ASS_A(post(t))$$

   where $ASS_A = (T_{OP}(Var(t)) \times P)^{\circledcirc} \to (A \times P)^{\circledcirc} = PV$ *is defined on generators by* $ASS_A(term, p) = (ass_A(term), p)$ *for each* $p \in P$, *and* $term \in T_{OP}(Var(t))$.

4. *Given a marking* $m \in PV$ *and a consistent transition assignment* $(t, ass_A) \in CT$ *the successor marking* $m' \in PV$ *of* m *is defined by* $m' = m \ominus pre_A(t, ass_A) \oplus post_A(t, ass_A)$

□

Remark: The operational behavior of AHL-Nets can easily be extended to define marking graphs with nodes colored by markings and edges by consistent transition assignments.

3.5 Fact (FLATTENING)

For each AHL-Net AN there is a P/T-G-Net FLAT(AN), called flattening of AN, which is behavioral equivalent to AN in the sense that both have the same marking groups and successor markings.

Construction: For each AHL-Net $AN = (NS, E, A, cond)$ *with* $NS = (SIG, P, T, pre, post)$ *we have* $FLAT(AN) = (P_F, T_F, pre_F, post_F)$ *with* $P_F = (A \times P)$, $T_F = CT$, $pre_F = pre_A$ *and* $post_F = post_A$. □

Remark: Although the construction of $FLAT(AN)$ may suggest that AN and $FLAT(AN)$ are almost equal, they are essentially different concerning the sets of places and transitions and the corresponding net structure.

4 AHL-net structuring

In this section we present notions for vertical refinement as well as for horizontal structuring of AHL-Nets. For vertical refinement we use AHL-Net transformations, what yields a high-level refinement concept for Petri nets. For horizontal structuring we present the notions of net fusion and union. Fusions allow us to identify different occurrences of the same subnet within a net and the two different kinds of union can be used to compose nets. In our framework vertical refinement and horizontal structuring are compatible with each other, i.e. AHL-Net transformations are compatible with fusions and unions.

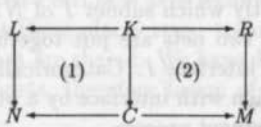
4.1 AHL-Net transformation systems

Net transformation systems are closely related with to graph grammars (see [9]). Each *production p* of a transformation system consists of a left-hand side net L, a right-hand side net R, an interface net K that consists of items that are preserved by the rule (are in L and R), and net morphisms $l : K \to L$ and $r : K \to R$ relating the items that are preserved from L to R.

Net morphisms are homomorphic mappings between the corresponding net components.

In order to apply a production p to a net N we must find a match of L in N given by a net morphism $m : L \to N$, and a so-called gluing-condition must be satisfied. This condition assures that after the deletion of the items that are not preserved by the rule (not in K), the remaining net is still a well-defined net.

A *direct transformation* of a net N to M via production $p = (L \leftarrow K \rightarrow R)$ at match $m : L \to N$, denoted by $N \overset{p}{\Longrightarrow} M$ is shown in the following diagram, where C is the context net (N after the deletion of items by the rule and before the addition of the new items from R).



In a categorical framework, the nets and the net morphisms in this diagram belong to the category **AHLNET**. The net morphisms in this production belong to a distinguished class $\mathbf{M}_{AHLNET}$ of injective morphisms and the squares (1) and (2) are pushouts in this category (see [14]). ¿From a categorical point of view a net transformation $p : N \Longrightarrow M$ is a pair of

pushouts ((1),(2)). This allows to show compatibility of transformations with different kinds of categorical constructions which preserve colimits (see [14]). Some of these results are stated in section 4.5.

A *transformation sequence* $N \stackrel{*}{\Longrightarrow} M$ (short: transformation or derivation) from N to M means $N \cong M$ or a sequence of $n \geq 1$ direct transformations $N = N_0 \stackrel{p_1}{\Longrightarrow} N_1 \stackrel{p_2}{\Longrightarrow} ... \stackrel{p_n}{\Longrightarrow} N_n = M$

Two direct transformations $N \stackrel{p1}{\Longrightarrow} M1$ and $N \stackrel{p2}{\Longrightarrow} M2$ are called *parallel independent* if the overlap of the left-hand sides of the productions $p1$ and $p2$ in N consists at most of common gluing items of $p1$ and $p2$. Dually, a transformation sequence $N \stackrel{p1'}{\Longrightarrow} X1 \stackrel{p2'}{\Longrightarrow} M$ is called *sequential independent* if the overlap of the right-hand side of the first production with the left-hand side of the second production consists at most of common gluing points. In this case the direct transformations $N \stackrel{p1'}{\Longrightarrow} X1$ and $X1 \stackrel{p2'}{\Longrightarrow} M$ are also called sequential independent.

An *AHL-Net transformation system* $ATS = (AN_0, P, \Rightarrow, T)$ consists of a start net AN_0, a set P of productions, the direct derivation (transformation) relation $\Rightarrow$ and a class T of terminal labeled AHL-Nets.

If we consider the start net AN_0 as the most abstract net, and the productions as refinements, we can see applications of the productions as stepwise refinement.

4.2 Example (File server's transformations)

Let us reconsider the file server's example from section 3.3. Now assume that we decided to specify in more detail how the write procedure should be done. To do this we write a rule that says how the $WRITE$ transition really works. This rule works like a procedure in programming languages: the code corresponding to the procedure (right-hand side of the rule) replaces the procedure call (left-hand side). The formal parameters are represented by the gluing net (K). In the example, the refinement of the $WRITE$ transition is a net with a new write transition and a place that keeps a table with names of the files that are not available. This way it is easier to verify whether a file is available or not. The production p corresponding to this refinement is shown in figure 4.1 (for reasons of space, we show only the left and hand-side nets, the gluing net consists only of the nodes FS and STO - therefore they are marked with a double line in the figure).. The production in figure 4.1 deletes the old write procedure and adds a new one. The application of the production p to the net of figure 3.1 is shown also in figure 4.1.

4.3 Union

Unions are used to compose nets. We distinguish two kinds of union: union of two nets $N1$ and $N2$ with respect to an interface I and disjoint union of $N1$ and $N2$. In the first case, we say explicitly which subnet I of $N1$ and $N2$ may be shared in the composed net. In the disjoint union, two nets are put together without any sharing. This is a special case of union with empty interface I. Categorically, the disjoint union is given by a coproduct construction and the union with interface by a pushout construction. Both cases are quite useful in the software development process.

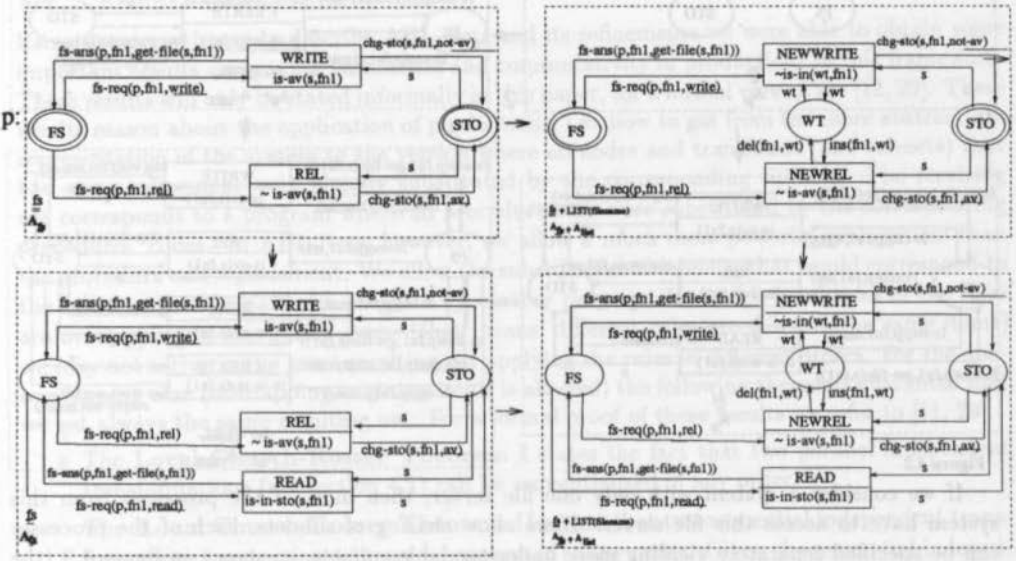


Figura 4.1

Software systems are usually developed by teams, where each person is responsible for a part of the system. For example, imagine that two different persons are in charge of the construction of a file server. The first one was responsible for the part shown in section 3.3 and the other should provide a subnet for the creation of new files in the storage. This subnet if shown in the net 2 of figure 4.2. Clearly there are places, namely FS and STO that should be shared by both parts when putting them together. The union of these two nets with the places FS and STO as interface is shown in figure 4.2 (the result of the union is net 3 in this figure).

On the other hand, sometimes it is the case that we do not really want to glue all the shared parts, but just put the nets together and leave this identification of common parts to a step further. If we have, for example, a file server place that is used by many different processes within this net, if we would connect each of these processes to the file server this could cause a difficulty in the graphical representation and consequently in the understanding of the system. The disjoint union of the nets corresponding to these processes and the file system is shown in figure 4.3. The nodes FS are drawn with thicker lines because they are fusion nodes (see following section).

4.4 Fusions

The notion of fusion was informally introduced in [20] for colored Petri nets as a technique to set virtual links between places resp. transitions, meaning that these places resp. transitions represent the same. These are in a sense places or transitions that are shared. We have defined fusion formally for AHL-Nets in [29]. We define fusion of subnets, therefore fusion of single places or transitions are special cases.

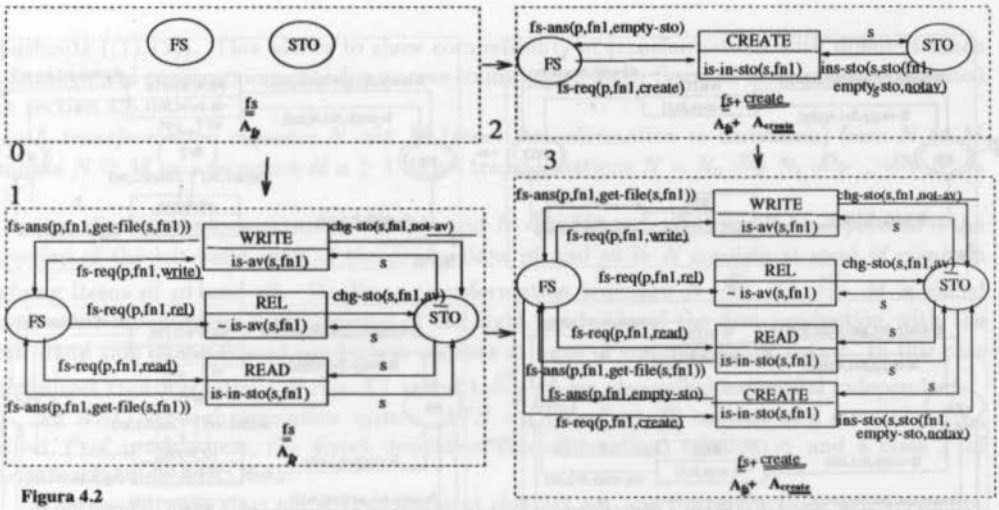


Figura 4.2

If we consider a system with only one file server, then most of the processes from this system have to access this file server. If we allow sharing of subnets, each of the processes can be specified separately yielding more understandable subnets, as shown in figure 4.3 (the fusion nodes are represented by circles with thicker lines). The different representation is also important if the subnets are drawn in different sheets of paper to warn the specifier that some parts are shared by other subnets, or in other words, these shared parts are the interface between his subnet and the rest of the system. Formally we obtain a net where all fusion subnets are "glued" from nets with fusion nodes by the application of the fusion construction defined in [29] (categorically, a coequalizer construction).

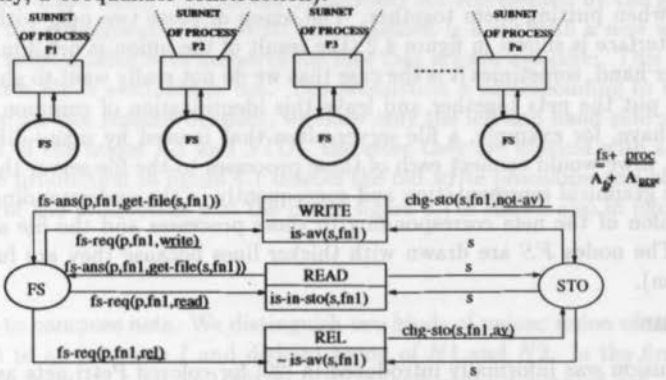


Figura 4.3

There is a general relationship between fusion and union. In [29] we proved that we get equivalent nets when we do the union of nets $N1$ and $N2$ with interface I and when we first do the disjoint union of $N1$ and $N2$ and then the fusion of the occurrences of I in the resulting net.

4.5 Theoretical results

Using category theory to describe AHL-Nets and its refinements we were able to obtain some important results concerning parallelism and commutativity of productions in this framework. These results will only be stated informally in this paper, for a formal version see [12, 29]. These results reason about the application of productions, i.e., how to get from the more abstract net representation of the system to the version where all nodes and transitions (and subnets) that had some refinement were already substituted by the corresponding subnets. The resulting net corresponds to a program where all procedure calls were substituted by the corresponding procedures' codes. In AHL-Nets, however, we allow a much more powerful replacement than the procedure call replacement. We allow the substitution of subnets, that would correspond to the substitution of a part of a program by another program. If the left-hand-sides of the rules are overlapping in non-gluing items (that means, different rules are deleting the same items) we may not arrive in the same resulting net applying the rules in different orders. For the non-overlapping case (overlapping in gluing items is allowed) the following theorems guarantee that we get always the same resulting net. For a formal proof of these results we refer to [11, 29].

- The **Local Church-Rosser Theorem I** states the fact that two parallel independent transformations (see section 4.1) can be sequentialized in any order.

  The **Local Church-Rosser Theorem II** means that two sequential independent transformations (see section 4.1) can be computed in any order. Given the sequential independent transformation sequence $N \stackrel{p}{\Longrightarrow} M \stackrel{p'}{\Longrightarrow} X$, then we have a sequential independent transformation sequence $N \stackrel{p'}{\Longrightarrow} M' \stackrel{p}{\Longrightarrow} X$ as well.
- **Parallel productions** are constructed by the componentwise disjoint union of two productions p and p' and is denoted by $p + p'$. The **Parallelism Theorem** asserts that there are two operations ANALYSIS and SYNTHESIS, that are inverse to each other. With ANALYSIS we can divide a parallel transformation into two sequential independent transformation sequences; and with SYNTHESIS we can combine two sequential independent transformations into one parallel transformation. The use of both operations yields the same net.
- The **Compatibility between transformations and unions theorem** states that a transformation and a union that are parallel independent can be sequentialized in any order.
- The **Compatibility between transformations and fusions theorem** states that a transformation and a fusion that are parallel independent (i.e. the overlap of the left-hand side of the rule and the fusion subnet consist at most of gluing items) can be sequentialized in any order.

5 Conclusions and future work

In this paper we have discussed some problems of classical place/transition nets from the software engineering point of view, addressed how these problems have been handled until now, and proposed a framework based on algebraic specifications, Petri nets and high-level replacement systems in which these problems are solved in an elegant and general way. Integration of these three concepts within a common framework in a well-defined way was possible due to the use of category theory to describe and analyze AHL-Nets and AHL-Net transformation

systems. Algebraic high-level nets provide a more abstract and easier way do describe systems than the usual place/transition nets. Another important feature of our framework is the use of net transformations as stepwise refinement for Petri nets, in particular for AHL-Nets. Net transformations are a very powerful and well-defined vertical refinement technique. Moreover, for the horizontal structuring of the nets we provide two kinds of union constructions and a fusion construction, and show that these constructions are compatible with net transformations, that means, we show the compatibility of vertical refinement and horizontal structuring in our framework.

There is already a computer tool to aid in the construction of AHL-Nets. The DJR-system [17] was developed in the Technical University of Berlin and using it it is possible to describe and simulate AHL-Nets. Together with the development of the theory, we intend to bring the new features to the DJR-system. We intend also to integrate the existing system AGG (an algebraic graph grammar system) [25] with the DJR-system to have in the same environment also a computer aided stepwise refinement for AHL-Nets.

An interesting application for the techniques developed within this paper is the HDMS project (*Heterogeneous distributed management system*) performed by the German Heartcenter Berlin (DHZB) and the Technical University of Berlin (TUB) and an abstract version of it, called HDMS-A (see [6]), studied within the BMFT-project KORSO on correct software development. The aim of HDMS is to develop an information system to support all kinds of services for the staff and patients within a hospital. The specifications of this system within HDMS-A are meant to be written in an algebraic specification language. As the system is very large, the first model of the system was made using Petri nets, in fact, a semi-formal combination of condition/event nets, short C/E nets, algebraic specification pieces and informal requirements in natural language. As the resulting C/E net is quite large, some modularization concepts were used in an informal way to break it into smaller parts. But the use of C/E nets, where tokens are "black boxes", lead to the fact that many actions that could have been done in parallel had to be sequentialized. Inspired by these concepts, we developed the formal structuring concepts described in this paper. Moreover, a description of the HDMS-A-system using AHL-Nets is in preparation, allowing a much more precise specification and also a higher degree of parallelism in the system. We intend to continue working on structuring concepts because they play a central role in description of real systems. One of the ideas in this direction is to use the concept of net transformations not only as refinements but also as a module concept for Petri nets.

Another important area are invariants for AHL-Nets, which were already defined in [8]. Reachability graph analysis is a powerful analysis tool to find out properties of Petri nets. The main drawback of this method is the very big size of the resulting graph. In [19] it was demonstrated that reachability graphs of high-level nets are much smaller than of the corresponding P/T nets. The definition of reachability graphs for AHL-Nets is of great importance for the analysis of this kind of nets.

Topics like guarded arcs (allowing a variable number of tokens in each switching of a transition) and dynamic nets (called invocation transitions in [20]: nets that change their structure according to firing of some transitions) are also very interesting and will be analyzed within the framework of AHL-Nets.

References

[1] J. R. Abrial. Programming as a mathematical exercise. In C. A. R. HOARE, editor, *Mathematical logic and programming languages.* Prentice-Hall International, 1985.

[2] G.A. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* The MIT Press Cambridge, Massachusetts, London, England, 1988.

[3] E. Astesiano and G. Reggio. An outline of the SMolCS approach. In M. Venturini Zilli, editor, *Mathematical models for the semantics of parallelism*, volume 280 of *Lecture Notes in Computer Science.* Springer Verlag Berlin, 1987.

[4] J. A. Bergstra and J. W. Klop. Algebra of communicating processes. In *CWI Monographs I*, 1986. Proc. CWIU Symp. Math. and Comp. Sci.

[5] D. Bjørner and C. B. Jones, editors. *The Vienna Development Method: the meta-language*, volume 61 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1978.

[6] F. Cornelius, Marcus Klar, and Michael Löwe. Ein Fallbeispiel für KORSO: Ist-Analyse HDMS-A. Technical report, Technical University of Berlin, 1993. To appear.

[7] C. Dimitrovici and A. Heise. Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften. Technical Report 342/6/91, Technical University of Munich, jul. 1991.

[8] C. Dimitrovici, U. Hummert, and L. Petrucci. Composition and net properties of algebraic high-level nets. In *Advances of Petri-Nets, Lecture Notes in Computer Science 483.* Springer, 1991.

[9] H. Ehrig. Introduction to the algebraic theory of graph grammars. In *1st Int. Workshop on Graph Grammars and their Application to Computer Science and Biology, Lecture Notes in Computer Science 73*, pages 1–69. Springer, 1979.

[10] H. Ehrig, M. Große-Rhode, and A. Heise. Specification techniques for concurrent and distributed systems. Technical Report 92/5, Technical University of Berlin, jan. 1992. Invited paper for 2nd Maghr. Conference on Software Engineering and Artificial Intelligence, Tunis,1992.

[11] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in High Level Replacement Systems. *Mathematical Structures in Comp. Sci.*, 1:361–404, 1992.

[12] H. Ehrig, H.-J. Kreowski, and G. Taentzer. Canonical Derivations in High-Level Replacement Systems. Technical Report 6/92, University of Bremen, 1992.

[13] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications*, volume 6 of *EACTS-Monographs in Theoretical Computer Science.* Springer, Berlin, 1985.

[14] H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. Technical Report 93-06, Technical University of Berlin, 1993. To appear in the Proc. of the ADT-COMPASS Workshop'92, Caldes de Malavella, Spain.

[15] H. Ehrig, F. Parisi-Presicce, P. Boehm, C. Rieckhoff, C. Dimitrovici, and M. Große-Rhode. Algebraic data type and process specifications based on projection spaces. *Theoretical Computer Science*, 332:23–43, 1987.

[16] H.J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[17] D. Giesel, J. Krüger, and R. Jeschke. Grundkonzepte und Implementierung eines Netzw--erkzeugs für Algebraische High-Level Netze. Technical Report 90/34, Technical University of Berlin, 1990.

[18] C. A. R. Hoare. *Communicating sequential processes.* Prentice-Hall International, London, 1985.

[19] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Reachability trees for high-level Petri nets. *High-level Petri nets: theory and application*, pages 319–350, 1992.

[20] P. Huber, K. Jensen, and R. M. Shapiro. Hierarquies in coloured Petri nets. *High-level Petri nets: theory and application*, pages 215–243, 1992.

[21] U. Hummert. *Algebraische High-Level Netze.* PhD thesis, Technical University of Berlin, Department of Computer Science, 1989.

[22] K. Jensen. Coloured petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[23] K. Jensen and G. Rozenberg, editors. *High-level Petri nets: theory and application.* Springer-Verlag, Berlin, 1992.

[24] LOTOS - A formal description technique based on temporal ordering of observational behaviour. Information Processing Systems - Open Systems Interconnection **ISO DIS 8807**, jul. 1987. (ISO/TC 97/SC 21 N).

[25] M. Löwe and M. Beyer. AGG — An Implementation of Algebraic Graph Rewriting. Accepted at the Fifth Int. Conf. on Rewriting Techniques and Applications, 1993.

[26] J. Meseguer and U. Montanari. Petri nets are monoids. In *Proc. Logic in computer science*, Edinburgh, 1988.

[27] R. Milner, editor. *A calculus for communicationg systems*, volume 92 of *Lecture Notes in Computer Science.* Springer-Verlag, 1980.

[28] J. Padberg. Theory of High-Level Replacement Systems with Application to Petri Nets. Diplomarbeit, Technical University of Berlin, 1992.

[29] J. Padberg, H. Ehrig, and L. Ribeiro. Algebraic High-Level Net- Transformation Systems. Technical Report 93-12, Technical University of Berlin, 1993. To appear in Mathematical Structures in Computer Science.

[30] C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[31] W. Reisig. *Petri nets.* Springer Verlag, 1985.

[32] J. Vautherin. Parallel specification with coloured Petri nets and algebraic data types. In *Proc. of the 7th European Workshop on Application and Theory of Petri nets*, pages 5–23, Oxford, England, jul. 1986.