PETRI NET MODULES

Julia Padberg*
*Technical University Berlin
Institute for Software Engineering and Theoretical Computer Science
Germany
email: padberg@cs.tu-berlin.de

Abstract: Here we present a new module concept for Petri nets that is based on the component concepts of *Continuous Software Engineering (CSE)*. According to that concept two distinguished interfaces are required. These are import and export interfaces. The import describes the assumptions on the environment, e.g. in terms of used components. The export gives an abstraction of the functionality and presents e.g. the offered services.
The modules for Petri we introduce here consist of three nets for the import, the body of the module and the export. The import net IMP states the prerequisites the modules assumes. The body net BOD represents the internal functionality. The export net EXP gives an abstraction of the body that can be used by the environment. We provide module operations to compose larger modules from basic ones. Operations to compose Petri net modules are crucial as the main purpose is composition. In most approaches module concepts come along with just one operation. A great advantage is achieved having different possibilities to compose modules, as it increases the convenience of modeling large systems. We propose three different operations, composition, disjoint union, and union. Our main result in this contribution is that these module operations are compatible with each other.

1. Introduction

The main motivation for Petri net modules is the modeling of component-based systems. Software components are an useful and widely accepted abstraction mechanism. Components are deployed during the entire software life cycle, from analysis to maintenance. The component concept as suggested in (Müller and Weber, 1998; Weber, 1999) for *Continuous Software Engineering (CSE)* is the basic concept for our approach.
Especially in early software development phases modeling with Petri nets is a common approach. Unfortunately Petri nets do not yet support this component idea. In this paper we present Petri net modules as an abstraction mechanism that corresponds to the component concept of CSE. This concept is realized for Petri

nets using the following basic ideas:

- Components consist of three nets: the import net IMP, the export net EXP, and the body net BOD. The import net presents those parts of the net that need to be provided from the "outside". The export net is that what the net module presents to the "outside. The body is the realization of the export using the import.
- The relation between import IMP and body BOD is given by a plain morphism. Export EXP and body $BOD$are related by a substitution morphism, that allows mapping one transition to a subnet.
- Different module operations are required for the flexible composition of modules. At the moment we have union of modules and composition of modules.
- Compatibility between the operations needs to be guaranteed. Provided some conditions are satisfied the result of first union and then composition or vice versa is the same up to isomorphism.
- The original algebraic module concept is based on a loose semantics. Up to now there is no loose semantics defined for Petri nets or other Petri net classes. This is future research and will be investigated in forthcoming papers.

In the area of Petri nets various structuring concepts have been proposed during the last 40 years, some of these are even called modules or modular approach. On the one hand there are hierarchical concepts (e.g. (Jensen, 1992; Buchholz, 1994; He, 1996; Fehling, 1993)). The system is described in an abstract net, while special functionalities are encapsulated into subnets and called like procedures by merging transitions or places and exchange of tokens. On the other hand there is a large variety of connector mechanisms between modules.The communication, coordination or cooperation among components via signals is given by a relation between input and output (e.g. (Christinsen and Hansen, 1994; Sibertin-Blanc, 1994; Desel *et al.*, 2000; Deiters and Gruhn, 1994)). In this way the exchange of tokens or only read-operation of modules is described by special features like read arcs, inhibitor arcs etc. In other approaches places and transitions of modules are merged by well-defined operations (e.g. (Kindler, 1995; Battiston *et al.*, 1991*b*; Battiston *et al.*, 1991*a*; Broy and Streicher, 1992)).
In contrast to these our approach is based on concepts of *Continuous Software Engineering (CSE)*. The advantage is that we must not extend the underlying formalism of Petri nets, but compose modules by means of a connector component. Moreover, there are no approaches that provide different composition operations with explicit compatibility results and composition operations are usually only given implicitly. The approach to Petri net modules as presented in this paper is closely related to algebraic specification modules (Ehrig and Mahr, 1990). Their impact for practical concepts in software engineering has been relevant from the beginning (Weber and Ehrig, 1988). The transfer of these concepts to process description techniques is a recent development. It has been started in (Simeoni, 1999) where modules for graph transformation systems and local action systems have been investigated. A general framework for component concepts based on High-Level Replacement Systems (Ehrig *et al.*, 1991) is presented in (Ehrig and Orejas, 2001). To sum up different process modeling techniques like graph transformation systems, Petri nets and many others besides, the relation between export and body is here formalized in a more general way by transformation rules instead of morphisms and our concept for Petri nets modules fits in very well. Moreover, the Petri net modules we introduce can be considered an instantiation of the generic component concept presented in (Ehrig *et al.*, 2002*c*) (also in this volume). In 3.4. we discuss this relation.
This paper is organized as follows: First we introduce Petri net modules in Section 2. The module operations are presented in Section 3. In both sections we use a small example to illustrate the introduced notions. A summary and future work are given in the conclusion in Section 4.

2. Petri Net Modules

In this section we define Petri net modules. More precisely the notions and results we present in this paper are given for place/transition nets. Since we use a categorical approach these notions and results easily can be transferred to other Petri net classes provided they are given categorically.
Petri net modules consist of three nets, namely the import net IMP, the body net BOD, and the export net EXP. These are related with two different kinds of morphisms. So we first need to define the different notions of morphisms and to invest their specific properties. The use of those constructions and conditions can be seen in the subsequent section (Section 3.) where module operations are introduced.
Due to space limitations we have skipped the proofs here. All proofs can be found in (Padberg, 2001). Here we give only the basic constructions that are required for understanding.

2.1. Preliminaries

First we give a short intuition of the underlying basics. The precise definitions can be found in (Padberg, 2001). Here we use the algebraic notion of Petri nets as introduced in (Meseguer and Montanari, 1990). Hence a Petri net is given by the set of transitions and the set of places and the pre- and post domain function.
$N = T \underset{post}{\overset{pre}{\rightrightarrows}} P^{\oplus}$, where $P^{\oplus}$ is the free commutative monoid over P, or the set of finite multisets over P. So an element $w \in P^{\oplus}$ can be presented as a linear sum $w = \Sigma_{p \in P} \lambda_p p$ and we can extend the usual operations and relations as $\oplus$, $\ominus$, $\leq$, and so on. Nevertheless we need the pre- and post-sets as well. Hence we have as usually ${}^\bullet t$ the set of all places in the pre- domain of a transition t. Analogously $t^\bullet$, ${}^\bullet p$ and $p^\bullet$. Moreover we need to state how often is a basic element with in an element of the free commutative monoid given. We define this for an element $p \in P$ and a word $w \in P^{\oplus}$ with $w_{|p} = \lambda p \in P^{\oplus}$.
Subnets $N' \in \mathcal{P}(N)$ of a given net N with $N' \subseteq N$ can be easily defined by subsets of places and transitions, where the pre- and postdomain of transitions may be extended. $\mathcal{P}(N)$ denotes the set of all subnets. Note that this subnet relation is *not* an inclusion in terms of plain morphisms.

2.2. Morphisms of Petri Nets

Morphisms are the basic entity in category theory; they can present the internal structure of objects and relate the objects. So they are the basis for the structural properties a category may have and can be used successfully to define various structuring techniques.
Based on the algebraic notion of Petri nets (Meseguer and Montanari, 1990) we use simple homomorphisms that are generated over the set of places. These morphisms map places to places and transitions to transitions. They preserve firing and they yield nice categorical properties as cocompleteness.

Plain morphisms are presented as usual by an arrow $\longrightarrow$.

Definition 1 (Plain Morphisms)
A plain morphism $f : N_1 \longrightarrow N_2$ is given by $f = (f_P, f_T)$ with $f_P : P_1 \longrightarrow P_2$ and $f_T : T_1 \longrightarrow T_2$ so that

$$pre_2 \circ f_T = f_P^{\oplus} \circ pre_1$$

and post analogously.
These morphisms give rise to the category $\mathbf{PN}$. ◇

A more elaborate notion of morphisms are substitution morphisms. These map places to places as well. But they can map a single transition to a whole subnet. Hence they *substitute* a transition by a net. These

morphisms are more complicated and do net yield nice categorical properties. But they capture a very broad idea of refinement and hence are adequate for the relation between the export net and the body net. Subsequently substitution morphisms are presented by an undulate arrow $\rightsquigarrow$.

Definition 2 (Substitution Morphisms)
A substitution morphism $f : N_1 \rightsquigarrow N_2$ *is given by* $f = (f_P, \mathfrak{f}_T)$ *with* $f_P : P_1 \longrightarrow P_2$ *and* $\mathfrak{f}_T : T_1 \longrightarrow \mathcal{P}(N_2)$ *with* $\mathfrak{f}_T(t) := N_2^t \subseteq N_2$ *such that* $N_2^t = (P_2^t, T_2^t, pre_2^t, post_2^t)$

- $f_P({}^\bullet t) \in P_2^t$ *and*
- $f_P(t^\bullet) \in P_2^t$

Composition of substitution morphisms $f : N_1 \rightsquigarrow N_2$ *and* $g : N_2 \rightsquigarrow N_3$ *is given by:*
$g \circ f := (g_P \circ f_P, \mathfrak{g}_T \circ \mathfrak{f}_T)$ *where* $g_T \circ f_T : T_1 \longrightarrow \mathcal{P}(N_3)$ *is defined by* $g_T \circ \mathfrak{f}_T(t) := \bigcup_{x \in T_2^t} N_3^x$. *Since all* $N_3^x \subseteq N_3$ *this construction is well defined.* ◇

Note that these morphisms do not preserve properties. They can be restricted in order to preserve liveness (see for example (Urbášek and Padberg, 2002)).

Example 3 (Substitution Morphisms)

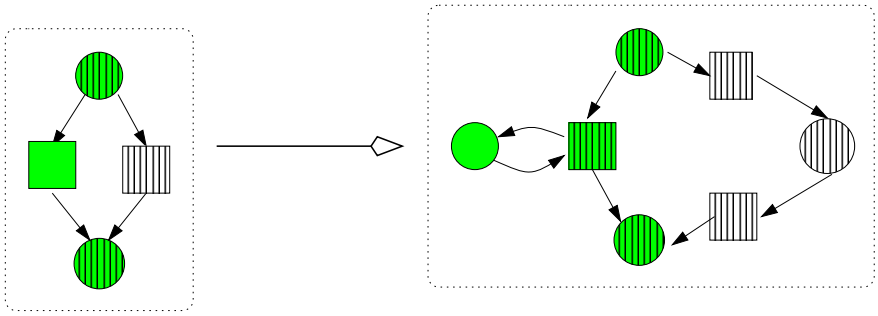


Figure 1. A substitution morphism.

The green [1] transition is mapped to the subnet consisting of green places and transitions. The striped transition is mapped to the subnet consisting of striped places and transitions.
The places in the pre- and post-domain have to be preserved in following sense: Places in the pre-domain of the transition are mapped to places in the subnet to which the transition has been mapped. ◇

Conversely we can determine the net that surrounds a transition, i.e. all the adjacent places and the transition itself present a subnet of the target net.

Definition 4 (Net of a Transition)
Given a transition $t \in T$ *for some net* N, *then* $net(t)$ *the net surrounding* t *is given by :*

$$net(t) := (P^t, T^t, pre^t, post^t)$$

with

- $P^t = {}^\bullet t \cup t^\bullet$,

[1] It may be grey if you have a black & white presentation. Then we assume the readers substitute susequently green by grey.

- $T^t = \{t\}$, *and*
- $pre^t : T^t \longrightarrow P^{t\oplus}$ *with* $pre^t(t) = pre_1(t)$, *analogously* $post^t$

◇

The net surrounding a transition is the basis to relate the two notions of morphisms. Plain morphisms are a special case of substitution morphisms, where each transition is mapped the surrounding net of its target transition.
The other way round: if a substitution morphisms maps all transitions to subnets containing only one transition, then it is plain as well.

Lemma 5 (Plain Morphisms and Substitution Morphisms)
Each plain morphism $f : N_1 \longrightarrow N_2$ *given by* $f = (f_P, f_T)$ *can be expressed as a substitution morphism* $f' : N_1 \rightsquigarrow N_2$ *given by* $f' = (f_P, \mathfrak{f}_T)$ *where* $\mathfrak{f}_T(t) = f(net(t))$.

Moreover, if a substitution morphism $f : N_1 \rightsquigarrow N_2$ *by* $f = (f_P, \mathfrak{f}_T)$ *has for all* $t \in T_1$ $\mathfrak{f}_T(t) = net(t')$ *for some* $t' \in T_2$ *so that* $f_P^{\oplus}(pre_1(t)) = pre_2(t')$ *and* $f_P^{\oplus}(post_1(t)) = post_2(t')$ *then it is plain with* $f_T(t) := t'$. ◇

Proof is trivial.

The above fact can be expressed in categorical terms as well. Then we have a functor relating the two categories of Petri nets with the different morphisms.

Lemma 6 (Category $\mathbf{SPN}$ of Petri Nets with Substitution Morphisms)
$\mathbf{SPN}$ *consisting of Petri nets and substitution morphisms is a category. Moreover,* $\mathbf{PN}$ *is a (full) subcategory of* $\mathbf{SPN}$ *and we have the inclusion functor* $I : \mathbf{PN} \longrightarrow \mathbf{SPN}$, *with* $I(f) := f'$ *as in Lemma 5.* ◇

Proof is trivial.

The inclusion functor defined above is cocontinuous, that means colimits are preserved. Given a colimit diagram in $\mathbf{PN}$ then the inclusion into $\mathbf{SPN}$ yields again a colimit diagram.

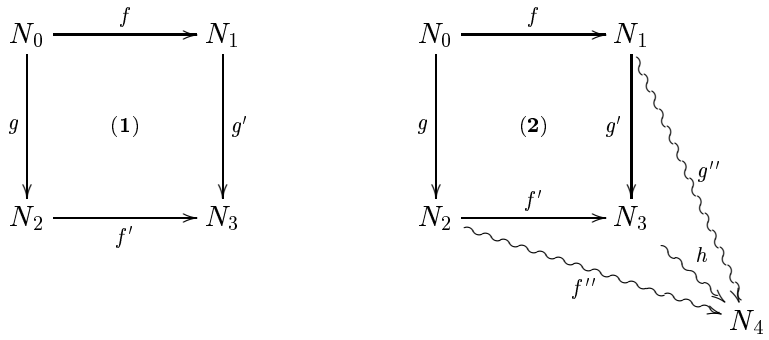
This is an essential result as it allows defining the module operations in terms of operation on Petri nets in Section 3.

Lemma 7 (Inclusion $I : \mathbf{PN} \longrightarrow \mathbf{SPN}$ is Cocontinuous)

◇

In the subsequent proof we show the preservation of pushouts explicitly as they are one of the main constructions we use.
The proof can be found in (Padberg, 2001) here we just sketch the preservation of pushouts.

Proof Sketch:
Given the following pushout **(1)** *in* $\mathbf{PN}$*, then we have the commutative square* **(2)** *in* $\mathbf{SPN}$ *as* I *is a functor.*



Let $f'' : N_1 \rightsquigarrow N_4$ *and* $g'' : N_2 \rightsquigarrow N_4$ *with* $g'' \circ f = f'' \circ g$.
We construct unique $h : N_3 \rightsquigarrow N_4$ *where* h_P *is the uniquely induced morphism by pushout* P_3 *in* $\mathbf{Set}$.
$\mathfrak{h}_T : T_3 \longrightarrow \mathcal{P}(N_4)$ *is given by*
$\mathfrak{h}_T(g'(t_1)) := \mathfrak{g}''_T(t_1)$
$\mathfrak{h}_T(f'(t_2)) := \mathfrak{g}''_T(t_2)$ *with* $t_i \in T_i$
$\mathfrak{h}_T$ *is well defined as* T_3 *is pushout is* $\mathbf{Set}$. $\surd$

So, colimits in $\mathbf{PN}$ are colimits in $\mathbf{SPN}$ as well. But these consistmerely ofplain morphisms. For diagrams where one morphism is plain and the other is a substitution morphism we need the subsequent condition that ensures the existence of pushouts.

Definition 8 (Compatibility Condition for Pushouts)
Given a plain morphism $f : N_0 \longrightarrow N_1$ *and a substitution morphism* $g : N_0 \rightsquigarrow N_2$ *the* PO-compatibility condition *is satisfied if:*
For all $t_0, t'_0 \in T_0$ *we have* $f_T(t_0) = f_T(t'_0)$ *implies* $\mathfrak{g}_T(t_0) \cong \mathfrak{g}_T(t'_0)$.
We the call f *and* g *jointly injective.* $\diamond$

Lemma 9 (Pushouts in $\mathbf{SPN}$)
In the category $\mathbf{SPN}$ *we have pushouts for a plain morphism* $f : N_0 \longrightarrow N_1$ *and a substitution morphism* $g : N_0 \rightsquigarrow N_2$ *if the* PO-compatibility condition *is satisfied.* $\diamond$

Due to space limitation we refer again to (Padberg, 2001) and give here only the construction of the pushout.

Proof Sketch:
Given a plain morphism $f : N_0 \longrightarrow N_1$ *and a substitution morphism* $g : N_0 \rightsquigarrow N_2$ *so that the* PO-compatibility condition *is satisfied, then we have* $N_3 := (P_3, t_3, pre_3, post_3)$ *where*

- $P_3 = P_1 +_{P_0} P_2$ *is pushout in* $\mathbf{Set}$,
- $T_3 := T_1 \setminus \{t_1 \in T_1^t | t \in T_0\} \uplus T_2$,
  hence we have: $t_3 \in T_3$ *implies* $t_3 \in T_1 \dot{\vee} t_3 \in T_2$ $(*)$
  and,

- $pre_3 = \begin{cases} g'_P(pre_1(t_3)) & ; t_3 \in T_1 \\ f'_P(pre_2(t_3)) & ; t_3 \in T_2 \end{cases}$

  $post_3$ *is defined analogously.*

N_3 is well defined due to $(*)$. $\surd$

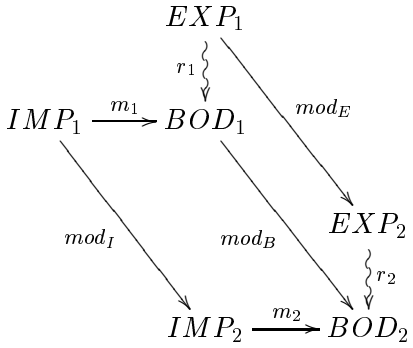
2.3. The Category of Modules of Petri Nets

We now introduce modules of Petri nets. The import is mapped to the body net by a plain morphism. The export net is mapped to the body net by a substitution morphism. This substitution morphism expresses the refinement of the export by the body net.

Definition 10 (Petri Net Modules)
A Petri net module $MOD = (IMP, EXP, BOD)$ consists of two morphisms $IMP \xrightarrow{m} BOD \xleftarrow{r} EXP$ where m is a plain morphism and r is a substitution morphism.

A (Petri) net module morphism $mod : MOD_1 \longrightarrow MOD_2$ is given by $mod = (mod_I, mod_E, mod_B)$ with $mod_I : IMP_1 \longrightarrow IMP_2$, $mod_E : EXP_1 \longrightarrow EXP_2$, and $mod_B : BOD_1 \longrightarrow BOD_2$, where mod_I, mod_E, mod_B are plain and the following conditions hold:

1. $mod_B \circ m_1 = m_2 \circ mod_I$
2. $mod_B \circ r_1 = r_2 \circ mod_E$



This gives rise to the category $\mathbf{PNMod}$ *of Petri net modules.* $\diamond$

Example 11 (Simple Module)
We have an example with three modules that describe the process of writing urging and offering letters. The module Mod_W given in Figure 2 provides the writing of a letter via the export to the environment. It imports two precise processes for the urging and offering letters. In Figure 3 we show the corresponding modules. In Example 16 we then illustrate the composition of these modules.
The module Mod_W in Figure 2 has the import net IMP_W with the two transitions Urge and Offer and their adjacent places. These are mapped to the net $BODY_W$ by a non-injective morphism. The import describes that this module assumes these two transitions to be abstractions. The export net EXP_W consists of a transition and two adjacent places.

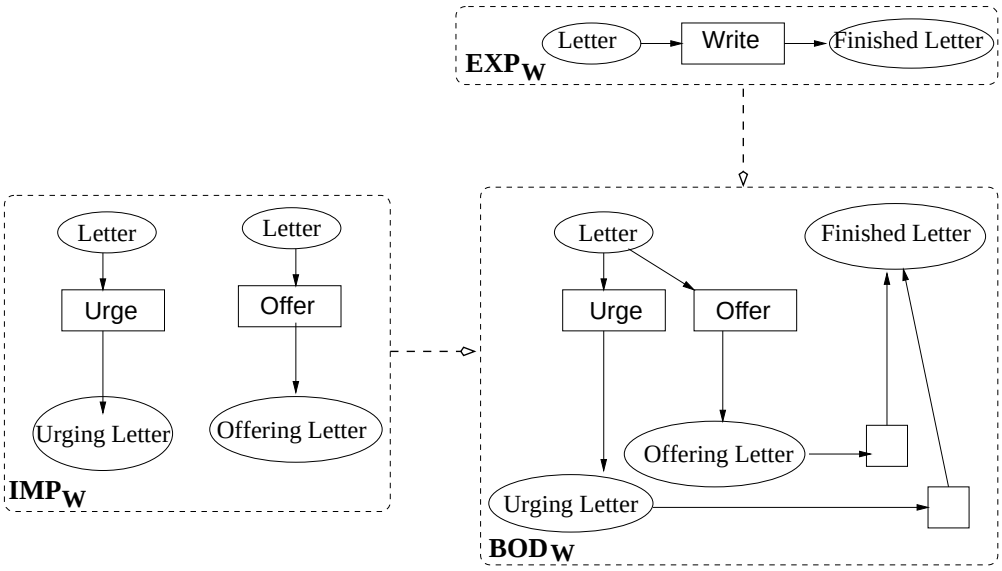


Figure 2. MOD_W.

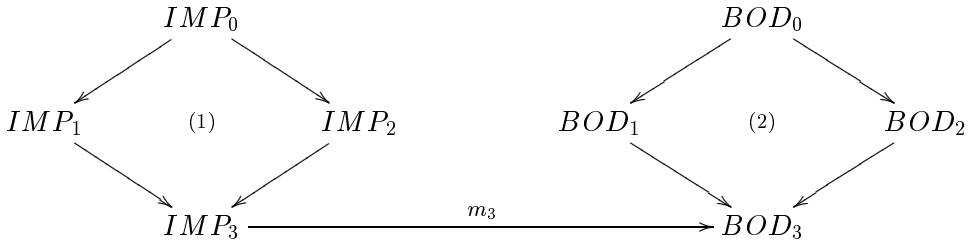
The places are mapped to the corresponding places in the body net BOD_W. The transition is mapped to the whole net BOD_W. Hence the morphism $EXP_W \rightsquigarrow BOD_W$ is a substitution morphism.
The export abstracts from the possibilities to write either an urging or offering letter. ◇

Next we state that we can construct arbitrary colimits in the category **PNMod**. In (Padberg, 2001) we construct the initial object, arbitrary coproducts and pushouts for the proof. Here we merely state the construction of pushouts as these are the formal foundation of the union operation for modules given in the next section.

Lemma 12 (PNMod is cocomplete)

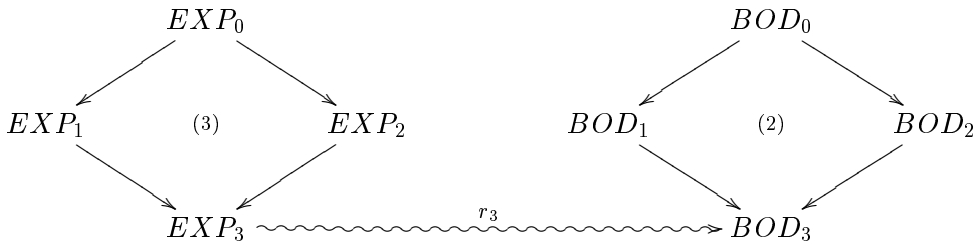
◇

Proof Sketch:
PNMod *has pushouts that are denoted by* $MOD_3 := MOD_1 +_{MOD_0} MOD_2$ *for* $mod_1 : MOD_0 \longrightarrow MOD_1$ *and* $mod_2 : MOD_0 \longrightarrow MOD_2$.
We have a component-wise construction, so we obtain the subsequent diagram



where (1) and (2) are pushouts in the category **PN** *and* $m_3 : IMP_3 \longrightarrow BOD_3$ *is the induced pushout morphism. As* $IMP_1 \longrightarrow BOD_1$ *and* $IMP_2 \longrightarrow BOD_2$ *are plain so is* m_3.

Analogously, we obtain – as I is cocontinuous – from pushout (3) in $\mathbf{SPN}$ *the induced morphism* r_3 : $EXP_3 \rightsquigarrow BOD_3$*:*



Hence we obtain $MOD_3 = (IMP_3 \xrightarrow{m_3} BOD_3 \overset{r_3}{\rightsquigarrow} EXP_3)$ *as the pushout.* $\surd$

3. Composition of Petri Net Modules

In order to construct models of large systems we propose to use a variety of operations to put modules together. We call these module operations. The use of various module operations directly rises the question whether the nodules obtained by different operations ina still consistent. In order to guarantee this we prove compatibility between the module operations.
In this section we define three module operations, namely union, disjoint union, and composition. Susequently we state their compatibility (for the proof see (Padberg, 2001)). These are important results since it ensures the consistency of modules that are composed in various ways.

3.1. Module Operations

The first operation we introduce is the *Disjoint Union*. Disjoint union is a special case of the *Union* of modules. Union allows a defined overlapping of the modules that is glued together.

Definition 13 (Disjoint Union of Petri Net Modules)
Given Petri net modules $MOD_i = (IMP_i \xrightarrow{m_i} BOD_i \overset{r_i}{\rightsquigarrow} EXP_i)$ *for* $i \in I$ *then there is* $MOD := \uplus_{i \in I} MOD_i$ *given by the coproduct in category* $\mathbf{PNMod}$. $\diamond$

In Example 16 we use the disjoint union for combining the modules Mod_O and MOD_U.

Definition 14 (Union of Petri Net Modules)
Given the Petri net modules $MOD_i = (IMP_i \xrightarrow{m_i} BOD_i \overset{r_i}{\rightsquigarrow} EXP_i)$ *for* $i = 0, 1, 2$ *with the net module morphisms* $mod_1 : MOD_0 \longrightarrow MOD_1$ *and* $mod_2 : MOD_0 \longrightarrow MOD_2$ *then there is* $MOD_3 := MOD_1 +_{MOD_0} MOD_2$ *given by the corresponding pushout in category* $\mathbf{PNMod}$. $\diamond$

The next module operation *Composition* describes the hierarchical composition of two modules, where the first module uses the export of the second one for its import. This results in a new module with the import of the second one, the export of the first one and a new composed body.

Definition 15 (Composition of Petri Net Modules)

Given two Petri net modules $MOD_i = (IMP_i \xrightarrow{m_i} BOD_i \overset{r_i}{\rightsquigarrow} EXP_i)$ *for* $i = 1, 2$ *and let* $h : IMP_1 \rightsquigarrow EXP_2$ *be a morphism so that the PO-compatibility conditions is satisfied for*

m_1 *and* $r_2 \circ h$*, then the composition* $MOD_3 = MOD_2 \odot_h MOD_1$ *is defined by* $MOD_3 = (IMP_2 \xrightarrow{m_1' \circ m_2} BOD_3 \xrightarrow{r_2' \circ r_1} EXP_1)$ *with:*

$$\begin{array}{ccccc} & & & & EXP_1 \\ & & & & \downarrow{\scriptstyle r_1} \\ & & IMP_1 & \xrightarrow{m_1} & BOD_1 \\ & & \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle r_2'} \\ & & EXP_2 & PO & \\ & & \downarrow{\scriptstyle r_2} & & \\ IMP_2 & \xrightarrow{m_2} & BOD_2 & \xrightarrow{m_1'} & BOD_3 \end{array}$$

◇

Example 16 (Module Operations)
In Example 11 we have the import IMP_W of two transitions with disjoint sets of places. To match this import we first do a disjoint union of the following modules Mod_O and MOD_U (see Figure 3).

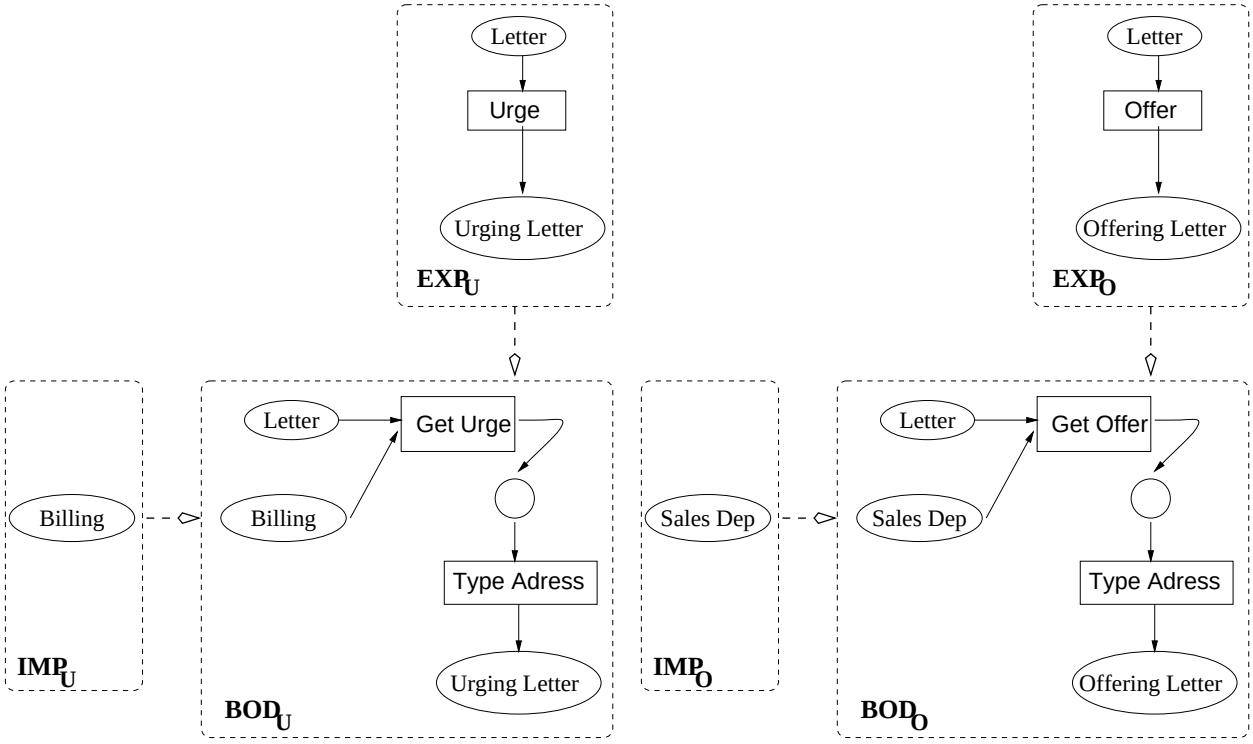


Figure 3. MOD_U and MOD_O

The resulting module MOD_{UO} is the composed with module MOD_W In Figure 4 we show the construction with the intermediate nets. BOD_{WUO} is the pushout of $BOD_{UO} +_{IMP_{UO}} BOD_W$. In Figure 5 we finally see the result of the composition. Note that $IMP_{WUO} = IMP_{UO}$ and $EXP_{WUO} = EXP_W$.

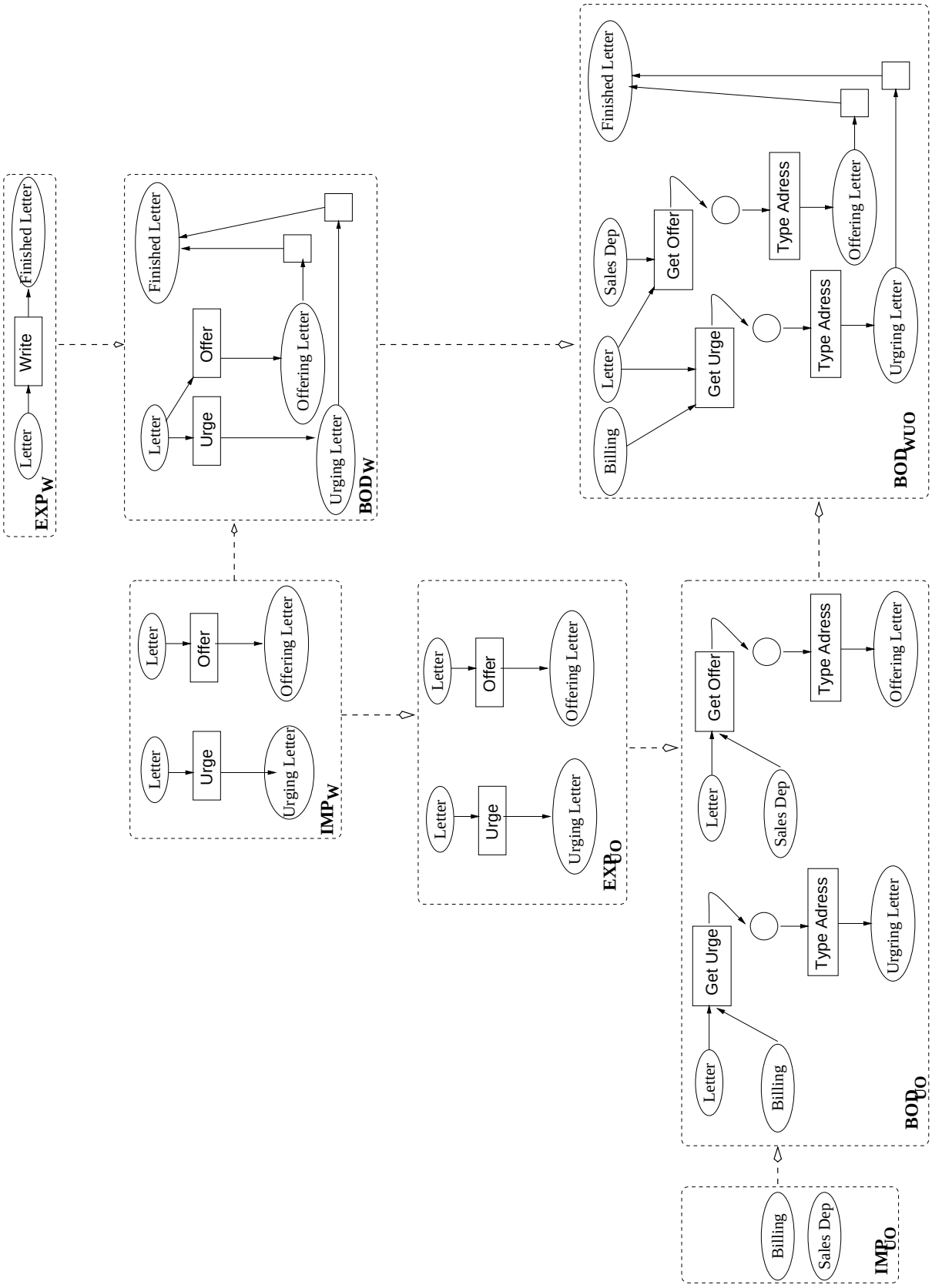


Figure 4. Composition of Mod_{UO} and MOD_W

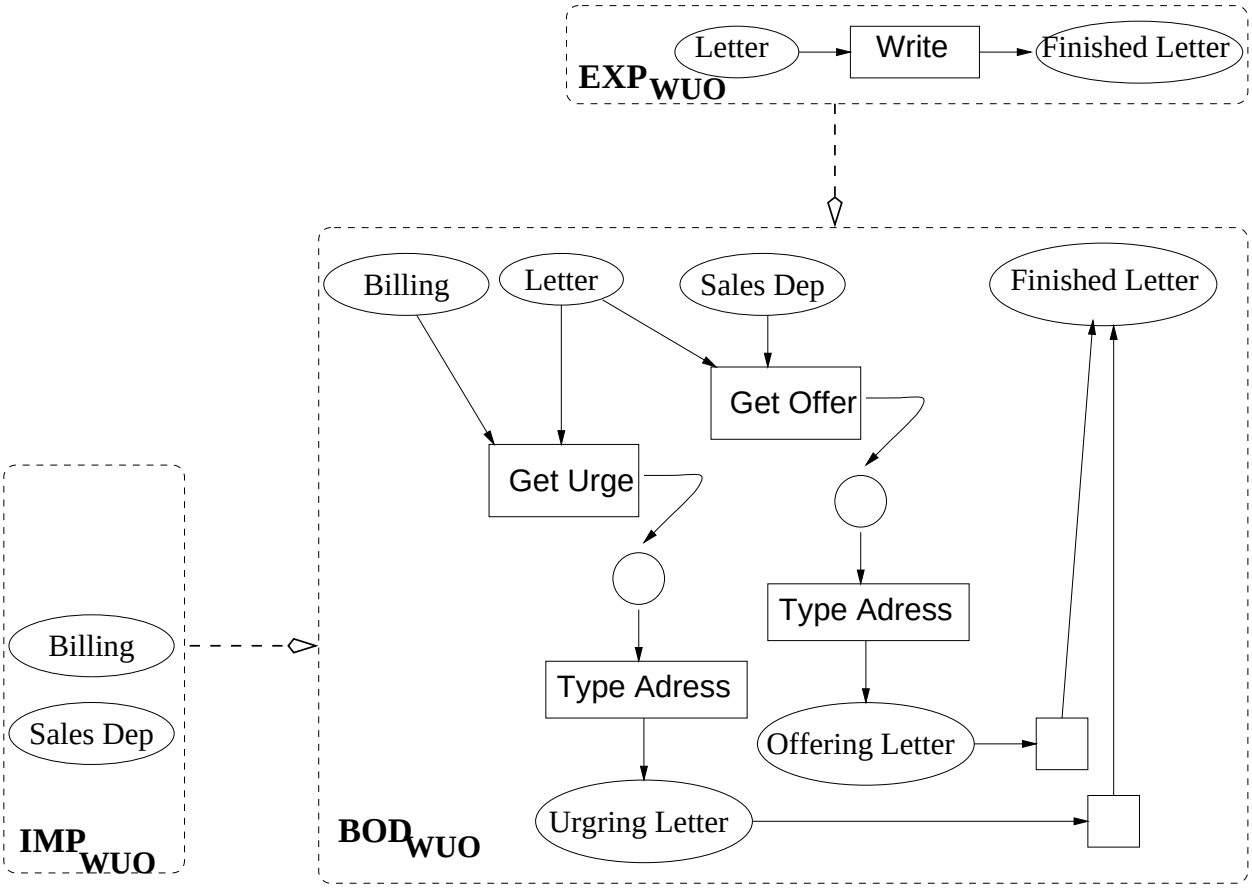


Figure 5. Mod_{WUO}

◇

3.2. Compatibility Results

We now come to the main result of this paper: The compatibility of the module operations with each other. This yields a kind of distributivity law for composition and (disjoint) union. Other kinds of such laws are stated informally in Subsection 3.3.
The compatibility of union and composition requires a technical condition ensuring that pushouts that preserve jointly injective morphisms, i.e. that preserve they the conditions given in Definition 8. here we skip this condition and again refer to (Padberg, 2001).

Theorem 17 (Compatibility of Union and Composition)
Given $f^1 : MOD_0 \longrightarrow MOD_1$ *and* $g^1 : MOD_0 \longrightarrow MOD_2$, *as well as* $f^{1\prime} : MOD'_0 \longrightarrow MOD'_1$ *and* $g^{1\prime} : MOD'_0 \longrightarrow MOD'_2$ *We have*

$$(MOD_1 +_{MOD_0} MOD_2) \odot_h (MOD'_1 +_{MOD'_0} MOD'_2)$$
$$\cong$$
$$(MOD_1 \odot_{h_1} MOD'_1) +_{(MOD_0 \odot_{h_0} MOD'_0} (MOD_2 \odot_{h_2} MOD'_2)$$

provided that the compositions via h, h_0, h_1, *and* h_2 *are well-defined, the compatibility condition (Definition 5 in (Padberg, 2001)) is satisfied, and the following compatibility condition holds:*

$$h_1 \circ f_I^1 = {f_E^1}' \circ h_0$$

and

$$h_2 \circ g_I^1 = {g_E^1}' \circ h_0 \qquad \diamond$$

For the proof see (Padberg, 2001).
As a direct consequence of the above Theorem we have that union is also compatible with disjoint union. This follows directly since disjoint union is a special case of union.

Corollary 18 (Compatibility of Disjoint Union and Composition)
Given $(MOD_i)_{i \in I}$ *and* $(MOD_i')_{i \in I}$ *then we have*

$$\biguplus_{i \in I} (MOD_i \odot_{h_i} MOD_i') \cong \biguplus_{i \in I} MOD_i \odot_h \biguplus_{i \in I} MOD_i'$$

provided that the compositions via h, *and* $(h_i)_{i \in I}$ *are well-defined.* $\diamond$

3.3. Laws for Module Operations

Here we state informally the basic laws we have for module operations These laws make use of the nice properties colimits have. Clearly, they result from the underlying theory and do not present a surprise for readers familiar with category theory. Hence we do not treat then in detail. Here we also have skipped the assumptions.
Nevertheless such laws are of eminent importance for the practical use. So we have stated them here explicitly.

Associativity

$$MOD_1 \uplus (MOD_2 \uplus MOD_3) \cong (MOD_1 \uplus MOD_2) \uplus MOD_3$$

$$MOD_1 \odot (MOD_2 \odot MOD_3) \cong (MOD_1 \odot MOD_2) \odot MOD_3$$

due to properties of colimits

Commutativity

$$MOD_1 \uplus MOD_2 \cong MOD_2 \uplus MOD_1$$

$$MOD_1 +_{MOD_0} MOD_2 \cong MOD_2 +_{MOD_0} MOD_1$$

due to properties of colimits

Distributivity

$$(MOD_1 +_{MOD_0} MOD_2) \odot (MOD'_1 +_{MOD'_0} MOD'_2)$$
$$\cong$$
$$(MOD_1 \odot MOD'_1) +_{(MOD_0 \odot_{h_0} MOD'_0} (MOD_2 \odot_{h_2} MOD'_2)$$

$$\biguplus_{i \in I} (MOD_i \odot MOD'_i) \cong \biguplus_{i \in I} MOD_i \odot \biguplus_{i \in I} MOD'_i$$

see Theorem 17 and Corollary 18

3.4. Relation to the Transformation-Based Component Framework

In (Ehrig *et al.*, 2002*a*) a very promising approach has been presented as an instantiation of a generic component approach. Main results concerning compositionality are given in (Ehrig *et al.*, 2002*b*) and show that semantics and correctness for a system can be inferred from that of its components. Our approach presented in this paper can be considered as an instance of this general framework as presented in (Ehrig *et al.*, 2002*c*) in this volume.

This abstract approach gives a general frame for the modularization of process description techniques. There a generic transformation $trafo : SPEC_1 \implies SPEC_2$ is introduced. For the general framework the extension properties is required. That is for each inclusion $i_1 : SPEC_1 \hookrightarrow SPEC'_1$ and each transformation $trafo : SPEC_1 \implies SPEC_2$ there are $i_2 : SPEC_2 \hookrightarrow SPEC'_2$ and a transformation $trafo' : SPEC'_1 \implies SPEC'_2$ so that the following diagram commutes:

$$\begin{array}{ccc} SPEC_1 & \xhookrightarrow{i_1} & SPEC'_1 \\ {\scriptstyle trafo}\Downarrow & & \Downarrow{\scriptstyle trafo'} \\ SPEC_2 & \xhookrightarrow[i_2]{} & SPEC'_2 \end{array}$$

If we use the substitution morphisms in Definition 2as transformations and injective plain morphisms as inclusions then the extension property is satisfied because of Lemma 9. Due to the satisfaction of this property we can obtain the transformation semantics as well. The composition is obviously based on the same concepts.

In section 4 in (Ehrig *et al.*, 2002*b*) an example based on algebraic high level nets and so-called double pushout transformations is given. Apart from the data type substitution morphisms introduced in Definition 2 are slightly more general than the double pushout transformations in (Ehrig *et al.*, 2002*c*).

4. Conclusion

In this paper we have presented Petri net modules as a formal modeling technique for component-based systems. We have investigated Petri net modules that have the same internal structure as components in the CSE approach. Moreover we have several module operations that allow putting modules together in a well-defined and consistent way. Hence Petri net modules present an adequate and powerful modeling technique for the process view of components. We have shown that they conform the basic principles of component-based software engineering and the component concept in the CSE approach.

In (Padberg and Buder, 2001) we have presented a case study using Petri net modules for the modeling of a telephone service center.
We have developed a Petri net model of an automated telephone service center with a variety of telephone services. The overall system consists of three major services: the telephone enquiry service, the message delivery service, and the payment service. All services start with an announcement concerning the selection of the corresponding service. The telephone enquiry service gives the customer information about telephone numbers. The customer is asked for the data; either the name, the postal code, the city or the street. The corresponding entry in the database is announced subsequently. The message delivery service forwards a message recorded by the customer to another recipient. Here the customer leaves a message, a telephone number and set a time for delivering the message. Then the order is entered into the database and carried out at the specified point of time. Finally the payment service allows changing the payment mode or checking the account balance. Our case study contains twelve basic modules that can be composed to an abstract description of the overall system. Subsequently we present four basic modules concerning the message delivery service.

The work we have presented here is a first step towards a full theory as well as towards a practical approach for modeling components.
Future work in this area comprises:

- Transfer of this module concept to other formal techniques used for the modeling of other views.
- Extension to further important module operations as renaming, refinement, or even recursion as in (Ehrig and Mahr, 1990).
- Explicit descriptions of dependencies and propagation of dependencies within and between modules (e.g. the relevance for an evolution of components has already be stated in (Große-Rhode *et al.*, 2000)).
- In order to transfer the module semantics of (Ehrig and Mahr, 1990) we first need to develop an loose semantics for Petri nets. This is an open issue for behavioral specifications in general.

References

Battiston, E., F. De Cindio and G. Mauri (1991*a*). OBJSA Nets: A Class of High-Level Nets Having Objects as Domains. In: *Advances in Petri Nets* (Rozenberg/Jensen, Ed.). Springer.

Battiston, E., F. De Cindio, G. Mauri and L. Rapanotti (1991*b*). Morphisms and Minimal Models for OBJSA Nets. In: *12^th International Conference on Application and Theory of Petri Nets*. Gjern, Denmark. pp. 455–476. extended version: Technical Report i. 4.26, Progretto Finalizzato Sistemi Informatici e Calcolo Parallelo. Consiglio Nazionale delle Ricerche (CNR), Italy, Jan, 1991.

Broy, M. and T. Streicher (1992). Modular functional modelling of petri nets with individual tokens. *Advances in Petri Nets*.

Buchholz, P. (1994). Hierachical high level Petri nets for complex system analysis. In: *Application and Theory of Petri Nets*. Vol. LNCS 815. Springer. pp. 119–138.

Christinsen, S. and N.D. Hansen (1994). Coloured petri nets extended with channels for synchronous communication. In: *Application and Theory of Petri Nets*. Vol. LNCS 815. Springer. pp. 159–178.

Deiters, W. and V. Gruhn (1994). The FUNSOFT Net Approach to Software Process Management. *International Journal on Software Engineering and Knowledge Engineering* **4**(2), 229–256.

Desel, J., G. Juhás and R. Lorenz (2000). Process semantics of Petri nets over partial algebra. In: *Proceedings of the XXI International Conference on Applications and Theory of Petri Nets* (M. Nielsen and D. Simpson, Eds.). Vol. LNCS 1825. Springer. pp. 146–165.

Ehrig, H., A. Habel, H.-J. Kreowski and F. Parisi-Presicce (1991). From graph grammars to high level replacement systems. In: *Lecture Notes in Computer Science 532*. Springer Verlag. pp. 269–291.

Ehrig, H. and B. Mahr (1990). *Fundamentals of Algebraic Specification 2: Module Specifications and Constraints*. Vol. 21 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag. Berlin.

Ehrig, H. and F. Orejas (2001). A Generic Component Concept for Integrated Data Type and Process Specification Techniques. Technical Report 2001/12. Technische Universität Berlin, FB Informatik.

Ehrig, H., F. Orejas, B. Braatz, M. Klein and M. Piirainen (2002*a*). A Generic Component Concept for System Modelling. In: *Proc. FASE 2002: Formal Aspects of Software Engineering*.

Ehrig, H., F. Orejas, B. Braatz, M. Klein and M. Piirainen (2002*b*). A Transformation-Based Component Framework for a Generic Integrated Modeling Technique. In: *Proc. of the Sixth World Conference on Integrated Design& Process Technology (IDPT'02)*. CD-ROM, 15 pages.

Ehrig, H., F. Orejas, B. Braatz, M. Klein and M. Piirainen (2002*c*). A transformation-based component framework for a generic integrated modeling technique. *Journal of Integrated Design and Process Science* **6**(3), 36–63.

Fehling, R. (1993). A concept of hierarchical Petri nets with building blocks. In: *Advances in Petri Nets'93*. Springer. pp. 148–168. Lecture Notes in Computer Science 674.

Große-Rhode, M., R.-D. Kutsche and F. Bübl (2000). Concepts for the evolution of component-based software systems. Technical Report 2000/11. TU Berlin.

He, X. (1996). A Formal Definition of Hierarchical Predicate Transition Nets. In: *Application and Theory of Petri Nets*. Vol. LNCS 1091. Springer. pp. 212–229.

Jensen, K. (1992). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Vol. 1: Basic Concepts.. EATCS Monographs in Theoretical Computer Science ed.. Springer Verlag.

Kindler, E. (1995). Modularer Entwurf verteilter Systeme mit Petrinetzen. PhD thesis. Technische Universität München, Institut für Informatik.

Meseguer, J. and U. Montanari (1990). Petri Nets are Monoids. *Information and Computation* **88**(2), 105–155.

Müller, H. and Weber, H., Eds.) (1998). *Continuous Engineering of Industrial Scale Software Systems*. IBFI, Schloß Dagstuhl. Dagstuhl Seminar Report #98092.

Padberg, J. (2001). Place/Transition Net Modules: Transfer from Algebraic Specification Modules. Technical Report TR 01-3. Technical University Berlin.

Padberg, J. and M. Buder (2001). Structuring with Petri Net Modules: A Case Study. Technical Report TR 01-4. Technical University Berlin.

Sibertin-Blanc, C. (1994). Cooperative Nets. In: *Application and Theory of Petri Nets'94*. pp. 471–490. Springer LNCS 815.

Simeoni, M. (1999). A Categorical Approach to Modularization of Graph Transformation Systems using Refinements. PhD thesis. Università Roma "La Sapienza".

Urbášek, M. and J. Padberg (2002). Preserving liveness with rule-based refinement of place/transition systems. In: *Proc. IDPT 2002: Sixth World Conference on Integrated Design and Process Technology, CD-ROM* (Society for Design and Process Science (SDPS), Eds.). p. 10.

Weber, H. (1999). Continuous Engineering of Communication and Software Infrastructures. Vol. 1577 of *Lecture Notes in Computer Science 1577*. pp. 22–29. Springer Verlag. Berlin, Heidelberg, New York.

Weber, H. and H. Ehrig (1988). Specification of concurrently executable modules and distributed modular systems. In: *Proc. IEEE Workshop on Future Trends of Distr. Comp. Systems in the 1990s, Hongkong*. IEEE. pp. 202–215.