Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Logic and
Algebraic Programming 67 (2006) 198–225

THE JOURNAL OF
LOGIC AND
ALGEBRAIC
PROGRAMMING

www.elsevier.com/locate/jlap

Petri net modules in the transformation-based component framework

Julia Padberg*, Hartmut Ehrig

Technische Universität Berlin, Fakultät IV—Informatik und Elektrotechnik, Franklinstr. 28/29, D-10587 Berlin, Germany

Abstract

Component-based software engineering needs to be backed by thorough formal concepts and modeling techniques. This paper combines two concepts introduced independently by the two authors in previous papers. On one hand, the concept of Petri net modules introduced at IDPT 2002 in Padberg [J. Padberg, Petri net modules, Journal on Integrated Design and Process Technology 6 (4) (2002) 105–120], and on the other hand a generic component framework for system modeling introduced at FASE 2002 in Ehrig et al. [H. Ehrig, F. Orejas, B. Braatz, M. Klein, M. Piirainen, A generic component concept for system modeling, in: Proceedings of FASE ’02, Lecture Notes in Computer Science, vol. 2306, Springer, 2002]. First we develop a categorical formalization of the transformation based approach to components that is based on pushouts. This is the frame in which we show that Petri net modules can be considered as an instantiation of the generic component framework. This allows applying the transformation based semantics and compositionality result of the generic framework to Petri net modules. In addition to general Petri net modules we introduce Petri net modules preserving safety properties which can be considered as another instantiation of pushout based formalization of the generic framework.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Components; Petri nets; Transformation based approach to components; Petri net modules; Category theory

1. Introduction

In order to build up large software systems from smaller parts, a flexible component concept for software systems and infrastructures is highly important (see e.g. [29,19,15]).

* Corresponding author.
E-mail addresses: padberg@cs.tu-berlin.de (J. Padberg), ehrig@cs.tu-berlin.de (H. Ehrig).

Software components are a useful and widely accepted abstraction mechanism. Components are deployed during the entire software life cycle, from analysis to maintenance. Although there are many approaches available, only few are general enough to be used for different specification techniques. To achieve a generic concept the focus has to be on the fundamental issues of components and component-based systems. These are the interfaces, the compositionality of components and its embedding into the environment.

The transformation-based component framework for generic components has been first presented at FASE 2002 in [11]. The main concepts are a self-contained semantics and composition of components, based on a generic transformation concept. Here we present a categorical formalization where we use pushouts to characterize the main construction. We achieve the desired properties as proposed in [11] using properties of the pushout construction.

In the transformation-based component framework a component consists of an import, an export and the body. The import states the prerequisites the modules assumes. The body represents the internal functionality. The export gives an abstraction of the body that can be used by the environment. These modules conform with the basic concepts of components and component-based systems of *Continuous Software Engineering (CSE)* [30].

In [22,21] Petri net modules have been introduced independently of the general framework discussed above. Similar to components they 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. In [23] this approach is extended to include safety properties. In this paper we show as a main result that Petri net modules as well as Petri net modules preserving safety properties can be considered as an instantiation of the generic component framework.

For this purpose we present a categorical formalization of the concepts of the transformation-based approach using specific kinds of pushout properties. We show that Petri net modules of both kinds satisfy these properties. Hence they are an instantiation of the transformation-based approach. In our main results of this paper we transfer the transformation based composition, semantics and compositionality result of the generic framework to Petri net modules of both kinds.

The paper is organized as follows: The introduction is continued discussing the relation to algebraic development techniques and related work. In the first part of Section 2 the transformation-based approach to components is reviewed. In the second part we develop a categorical formalization leading to a transformation-based approach based on pushouts. Section 3 treats Petri net modules. We show that they are an instantiation of the transformation-based approach. This allows transferring the general results to Petri net modules. In Section 4 we repeat this procedure for Petri net modules with safety. Then in Section 5 we discuss the relation to Petri net components based on high-level replacement systems. Moreover we discuss the main insights of a case study in [24]. We conclude this paper with a summary and an outlook to future work.

1.1. Relation to algebraic development techniques

Both, the generic component concept and the Petri net modules have been motivated at least by the algebraic module specification concept of Ehrig and Mahr [9]. The main idea of this algebraic module specification concept is to have interface specifications IMP and *EXP* for import and export, and body specification *BOD*, where these specifications are connected by specification morphisms from *IMP* to *BOD* and from *EXP* to *BOD*. In our approaches the algebraic specifications are replaced by generic specifications in the generic approach and by Petri nets in the case of Petri net modules. Moreover, we distinguish between two different kinds of morphisms: Transformations respectively substitution morphisms between *EXP* and *BOD*, and embeddings respectively injective plain morphisms between *IMP* and *BOD*. The main difference, however, is the model-theoretic semantics of algebraic module specification in contrast to a transformation-based semantics in our approaches. The model-theoretic semantics is given by a functor from *IMP*-algebras to *EXP*-algebras, while the transformation-based semantics is given by a function from transformations of *IMP* to transformations of *EXP*. The concept of transformations is well known in the area of algebraic development techniques, especially rule-based transformations in the sense of string-, tree-, and graph rewriting or double pushout transformations in the sense of high-level replacement systems. The algebraic treatment of Petri nets has been initiated by Meseguer and Montanari [20], where Petri nets are considered as algebraic objects and Petri net morphisms are introduced leading to an algebraic theory for the category of Petri nets. Our plain morphisms of Petri nets are a special case of Meseguer and Montanari concerning the mapping of places. Our substitution morphisms, however, allow a more general mapping of transitions and they are not restricted to preserve the firing behavior. Moreover, we consider more specific substitution morphisms, which are safety property preserving, leading to the new concept of Petri net modules preserving safety properties.

Finally, the composition of Petri net modules is defined using the well-known concept of pushouts, which is also used for the composition of algebraic module specifications.

1.2. Related Work

As pointed out above Petri net modules and the transformation-based component are both related to algebraic specification modules as defined by [9]. The transfer of these concepts to process description techniques is a recent development. It has been started in [28] 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 (see [8]) is presented in [10] including different process modeling techniques like graph transformation systems, Petri nets and related techniques. The component concepts in [10,11] are closely related. The semantical concepts, however, are different. In the first case the semantics is constructor based, e.g. free constructions in the case of algebraic specifications, and in the second case transformation based, as shown in Section 2 below.

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. There are hierarchical concepts (e.g. [17,4,16,13]) as well as a variety concepts for connector mechanisms as communication, coordination or cooperation (e.g. [5,27,7,6]). In other approaches places and transitions of modules are merged by well-defined operations (e.g. [18,2,1,3]).

2. Transformation-based component framework

First we give a short review of the transformation-based component framework as given in [11]. In Section 2.4 we discuss the formalization of this approach based on pushouts.

There, the main concept of extension is described using pushouts. Subsequently we show that the results proposed in [11] are achieved with this formalization.

2.1. A generic component concept

In this section we sketch the basic concepts of our generic component concept as given in [11]. Components are self-contained modeling units with a clear separation between the interface of the component and the body. Moreover, the interface can be divided into two parts: the import interface, describing what the component assumes about the environment, and the export interface, describing the services provided by the component itself.

We assume to have a generic modeling technique for describing systems which includes model specifications and suitable connections between them. Given such a generic modeling technique we are able to define our *generic component concept*. A *component specification,* in short *component*, $COMP = (IMP, EXP, BOD, \text{imp}, exp)$ consists of model specifications and connections:

- IMP, called *import interface*,
- EXP, called *export interface*,
- BOD, called *body*,
- $imp : IMP \to BOD$, called *import connection*,
- $exp : EXP \Longrightarrow BOD$, called *export connection*.

We require that each export connection $exp : EXP \Longrightarrow BOD$ uniquely defines a transformation of model specifications (see Section 2.2). $exp : EXP \Longrightarrow BOD$ is called export transformation. Intuitively, it can be considered a *refinement* describing how the elements presented in the export interface are implemented by the body. Moreover, if the model specifications have an informal or formal semantics, we assume that the semantics of the body is a refinement of the semantics of the export via the export transformation $exp : EXP \Rightarrow BOD$.

With respect to the import connection we require the body of a component to be an extension of the import interface. For the sake of simplicity, we assume that each import connection, $imp : IMP \to BOD$, defines an embedding $imp : IMP \to BOD$ of the corresponding specifications.

2.2. A generic transformation concept

In view of our generic component concept we assume that a transformation framework $\mathcal{T}$ consists of a class of transformations, which includes identical transformations, is closed under composition and satisfies the following *extension property*: For each transformation $trafo : SPEC_1 \Longrightarrow SPEC_2$ and each embedding $i_1 : SPEC_1 \to SPEC_1'$ there is a selected transformation $trafo' : SPEC_1' \Longrightarrow SPEC_2'$ with embedding $i_2 : SPEC_2 \to SPEC_2'$, called the *extension* of *trafo* with respect to i_1, leading to the extension diagram in Fig. 1. Intuitively, each refinement from $SPEC_1$ to $SPEC_2$ via *trafo* can be extended to a refinement from $SPEC_1'$ to $SPEC_2'$ via $trafo'$.

It must be pointed out that, in a given framework, given *trafo* and i_1 as above, there may be several $trafo'$ and i_2, that could satisfy this extension property. However, our assumption means that, in the given framework $\mathcal{T}$ only one such $trafo'$ and one embedding i_2 are chosen, in some well-defined way, as the extension of *trafo* with respect to i_1.

$$\begin{tikzcd}
SPEC_1 \arrow[r, "i_1"] \arrow[d, Rightarrow, "trafo"'] & SPEC'_1 \arrow[d, Rightarrow, "trafo'"] \\
SPEC_2 \arrow[r, "i_2"] & SPEC'_2
\end{tikzcd}$$

Fig. 1. Extension diagram for the extension property.

2.3. *Transformation semantics of components*

According to the general requirements, components are self-contained units, with respect to syntax and semantics. Hence, it is necessary to have a semantics for each single component.

In contrast to a model theoretic or functional semantics as considered for algebraic module specifications in [9] we propose a transformation semantics.

The main idea proposed in [11] is a semantics that takes into account the environment of a component, in a similar way as the continuation semantics of a programming language assigns the meaning of a program statement in terms of the environment of the statement. Here, the idea is to think that, what characterizes the import interface of a component is not its class of models, but the possible refinements or transformations of this interface that we can find in the environment of the component. In this sense, it is natural to consider that the semantical effect of a component is the combination of each possible import transformation, $trafo : IMP \Longrightarrow SPEC$ with the export transformation $exp : EXP \Longrightarrow BOD$ of the component. Since *IMP* is included in *BOD*, we have to extend the import transformation from *IMP* to *BOD* in order to be able to compose both transformations. Due to the extension property for transformations, we obtain $trafo' : BOD \Longrightarrow SPEC'$, as shown in Fig. 2.

Let us call the class of all transformations *trafo* from *IMP* to some specification *SPEC* the *transformation semantics* of *IMP*, denoted by $Trafo(IMP)$, and similar for *EXP*. According to Fig. 2 the *transformation semantics* of the component *COMP* can be considered as a function

$$TrafoSem(COMP) : Trafo(IMP) \to Trafo(EXP)$$

defined for all $trafo \in Trafo(IMP)$, by

$$TrafoSem(COMP)(trafo) = trafo' \circ exp \in Trafo(EXP),$$

where $trafo'$ is defined by the extension diagram in Fig. 2.

2.4. *Transformation-based component framework based on pushouts*

In the following subsections we present one possible formalization of the generic component concept. There, the extension property corresponds to the existence of specific

$$\begin{tikzcd}
 & EXP \arrow[d, Rightarrow, "exp"] \\
IMP \arrow[r, "imp"] \arrow[d, Rightarrow, "trafo"'] & BOD \arrow[d, Rightarrow, "trafo'"] \\
SPEC \arrow[r, "imp'"] & SPEC'
\end{tikzcd}$$

Fig. 2. Transformation semantics.

pushouts in a categorical framework. This framework includes definition, composition, and compositional semantics of components that have been considered in [11] using only the extension property. For the purpose of this paper however, it is sufficient to consider only those extension diagrams that are defined by pushouts. In fact this is the case for both kinds of Petri net modules considered as instantiations in the subsequent sections.

Definition 1 (*Transformation framework* $\mathcal{T}$). A transformation framework $\mathcal{T} = (\mathbf{Cat}, \mathcal{I}, \mathcal{E})$ consists of an arbitrary category and two classes of morphisms $\mathcal{I}$ and $\mathcal{E}$ in **Cat**, called import and export morphisms, that are closed under composition.

Moreover the following *extension conditions* have to hold:

(1) $\mathcal{E}$–$\mathcal{I}$-pushout condition:
Given the morphisms $A \xrightarrow{e} B$ with $e \in \mathcal{E}$ and $A \xrightarrow{i} C$ with $i \in \mathcal{I}$, then there exists the pushout D in **Cat** with morphisms $B \xrightarrow{i'} D$ and $C \xrightarrow{e'} D$ as depicted below.

$$\begin{tikzcd} A \arrow[r, "e"] \arrow[d, "i"'] \arrow[dr, phantom, "(\mathbf{1})"] & B \arrow[d, "i'"] \\ C \arrow[r, "e'"'] & D \end{tikzcd}$$

(2) $\mathcal{E}$ and $\mathcal{I}$ are stable under pushouts:
Given a $\mathcal{E}$–$\mathcal{I}$-pushout as (1) above, then we have $i' \in \mathcal{I}$ and $e' \in \mathcal{E}$ as well.

Accordingly we have to require for a component that the import and export connection are of the right class of morphisms.

Definition 2 (*Component*). A component $COMP = (IMP, EXP, BOD, imp, exp)$ is given by objects IMP, EXP, and BOD in **Cat** and by morphisms $exp : EXP \to BOD$ and $imp : IMP \to BOD$, so that $exp \in \mathcal{E}$ and $imp \in \mathcal{I}$.

2.5. Composition of components

Several different operations on components can be considered in our generic framework. For the sake of simplicity we subsequently consider merely one basic operation that allows composing components $COMP_1$ and $COMP_2$. It provides a *connector*, $connect : IMP_1 \to EXP_2$ from the import interface IMP_1 of $COMP_1$ to the export interface EXP_2 of $COMP_2$. Similar to an export connection we require the connector to define a transformation $connect : IMP_1 \to EXP_2$ uniquely. Now we are able to define the composition $COMP_3 = COMP_1 \circ_{connect} COMP_2$ as follows.

Definition 3 (*Composition*). Given components $COMP_i = (IMP_i, EXP_i, BOD_i, imp_i, exp_i)$ for $i \in \{1, 2\}$ and a morphism $connect : IMP_1 \Longrightarrow EXP_2$ in $\mathcal{E}$ the composition $COMP_3$ of $COMP_1$ and $COMP_2$ via $connect$ is defined by

$$COMP_3 = (IMP_3, EXP_3, BOD_3, imp_3, exp_3)$$

with $imp_3 = imp_1' \circ imp_2$ and $exp_3 = xconnect' \circ exp_1$ as depicted below, where **(1)** is pushout diagram:

$$\begin{tikzcd}
 & & EXP_3 = EXP_1 \arrow[d, Rightarrow, "exp_1"] \\
 & IMP_1 \arrow[r, "imp_1"] \arrow[d, Rightarrow, "connect"'] & BOD_1 \arrow[dd, Rightarrow, "xconnect'"] \\
 & EXP_2 \arrow[d, Rightarrow, "exp_2"'] \arrow[r, phantom, "(\mathbf{1})"] & \\
IMP_3 = IMP_2 \arrow[r, "imp_2"] & BOD_2 \arrow[r, "imp'_1"] & BOD_3
\end{tikzcd}$$

The composition is denoted by $COMP_3 = COMP_1 \circ_{connect} COMP_2$.

Since we have $IMP_3 = IMP_2$ and $EXP_3 = EXP_1$, this means especially that the result of the composition concerning the interfaces is independent of the body parts.

Here, we merely give the composition of one import to one export via the transformation $connect$. A simple way to obtain different imports related to different exports employs the disjoint union of components. Another possibility is using connector architectures as suggested in [12].

Lemma 1 (Composition). *$COMP_3$ is a well-defined component in the sense of Definition 2.*

Proof. The pushout construction **(1)** above exists due to the $\mathscr{E}$–$\mathscr{I}$-pushout condition in Definition 1. As both exp_2 and $connect$ are in $\mathscr{E}$, so is the composition $exp_2 \circ connect$. Stability of pushouts under $\mathscr{E}$ and $\mathscr{I}$ implies $xconnect' \in \mathscr{E}$ and $imp'_1 \in \mathscr{I}$. Since $\mathscr{E}$ and $\mathscr{I}$ are closed under composition we obtain $exp_3 = xconnect' \circ exp_1 \in \mathscr{E}$ and $imp_3 = imp'_1 \circ imp_2 \in \mathscr{I}$. □

Note that each connector $connect : IMP_1 \to EXP_2$ can also be considered as a separate component $COMP_{12}$ with $exp_{12} = connect$ and $imp_{12} = id_{EXP_2}$. This allows to consider $COMP_3$ in Definition 3 as the composition of three components $COMP_1$, $COMP_{12}$ and $COMP_2$, where all connectors are identities. Of course this requires that the corresponding identities are in $\mathscr{I}$ and $\mathscr{E}$.

2.6. Compositional transformation semantics

Compositional semantics means that the semantics of the composition of two components can be obtained by composing the semantics of the simple components. This is a most important property for a component concept.

Stated informally, we have for a connector, $connect : IMP_1 \to EXP_2$, between IMP_1 of $COMP_1$ and EXP_2 of $COMP_2$, that the composition $COMP_3$ of these components via $connect$ is well defined. As motivated in Section 2.4 let us rephrase the transformation semantics of components in our categorical framework using pushouts.

Definition 4 (*Transformation semantics of components*). Given a component $COMP = (IMP, EXP, BOD, imp, exp)$ its transformation semantics is given by the function $TrafoSem(COMP) : Trafo(IMP) \to Trafo(EXP)$ defined for all $trafo \in Trafo(IMP)$ by

$$\begin{tikzcd}
 & & EXP_3 = EXP_1 \arrow[d, Rightarrow, "exp_1"] \\
 & IMP_1 \arrow[r, "imp_1"] \arrow[d, Rightarrow, "connect"'] & BOD_1 \arrow[dd, Rightarrow, "xconnect'"] \\
 & EXP_2 \arrow[d, Rightarrow, "exp_2"'] \arrow[r, phantom, "(\mathbf{3})"] & \\
IMP_3 = IMP_2 \arrow[r, "imp_2"] \arrow[d, Rightarrow, "trafo_1"'] \arrow[dr, phantom, "(\mathbf{1})"] & BOD_2 \arrow[r, "imp'_1"] \arrow[d, Rightarrow, "trafo_2"'] \arrow[dr, phantom, "(\mathbf{2})"] & BOD_3 \arrow[d, Rightarrow, "trafo_3"] \\
SPEC_1 \arrow[r] & SPEC_2 \arrow[r] & SPEC_3
\end{tikzcd}$$

Fig. 3. Compositional Transformation Semantics.

$$TrafoSem(COMP)(trafo) = trafo' \circ exp \in Trafo(EXP),$$

where *trafo*$'$ is defined by the pushout diagram **(1)** below:

$$\begin{tikzcd}
 & EXP \arrow[d, Rightarrow, "exp"] \\
IMP \arrow[r, "imp"] \arrow[d, Rightarrow, "trafo"'] \arrow[dr, phantom, "(\mathbf{1})"] & BOD \arrow[d, Rightarrow, "trafo'"] \\
SPEC \arrow[r, "imp'"] & SPEC'
\end{tikzcd}$$

(1) exists due to Definition 1 with $trafo' \in \mathcal{E}$ and hence $trafo' \circ exp \in \mathcal{E}$ as well.

We now obtain the following compositionality result.

The transformation semantics of the composition can be obtained by functional composition of the transformation semantics of $COMP_1$ and $COMP_2$ with a most simple intermediate function

$$Trafo(connect) : Trafo(EXP_2) \to Trafo(IMP_1),$$

where $Trafo(connect)(trafo) = trafo \circ connect$.

More precisely we have the following.

Theorem 1 (Compositional transformation semantics). *Given a composition of components $COMP_3 = COMP_1 \circ_{connect} COMP_2$ as shown in Definition 3 then we have the following compositionality property*:

$$\begin{aligned} & TrafoSem(COMP_3) \\ & \quad = TrafoSem(COMP_1) \circ Trafo(connect) \circ TrafoSem(COMP_2). \end{aligned}$$

Proof. Given a transformation $trafo_1 : IMP_3 \Longrightarrow SPEC_1$ we consider Fig. 3, where diagram **(3)** is a pushout by definition of $COMP_3$ in Definition 3. **(1)** and **(2)** are constructed as pushouts according to Definition 1.

Due to the composition property of pushouts we have that also the horizontal composition **(1)**+**(2)** and the vertical composition **(2)**+**(3)** are pushouts.

We have to show

$$\begin{aligned}&TrafoSem(COMP_3)(trafo_1)\\&\quad = TrafoSem(COMP_1)\circ Trafo(connect)\circ TrafoSem(COMP_2)(trafo_1).\end{aligned}\tag{4}$$

Using $imp_3 = imp'_1 \circ imp_2$, $exp_3 = xconnect' \circ exp_1$ and diagram **(1)**+**(2)** we have

$$TrafoSem(COMP_3)(trafo_1) = trafo_3 \circ xconnect' \circ exp_1.\tag{5}$$

Concerning the right-hand side of (4) we have $TrafoSem(COMP_2)(trafo_1) = trafo_2 \circ exp_2$ and hence

$$\begin{aligned}&TrafoSem(COMP_1)\circ Trafo(connect)\circ TrafoSem(COMP_2)(trafo_1)\\&\quad = TrafoSem(COMP_1)\circ Trafo(connect)(trafo_2\circ exp_2)\\&\quad = TrafoSem(COMP_1)(trafo_2\circ exp_2\circ connect)\\&\quad = trafo_3\circ xconnect'\circ exp_1,\end{aligned}\tag{6}$$

where the last step uses diagram **(2)**+**(3)**.

Now (4) follows immediately from (5) and (6). □

3. Petri net modules

In this section we show that Petri net modules can be considered as an instantiation of the transformation-based component approach.

First, we introduce Petri net modules as given in [21,22]. In Section 3.2 we then show, that Petri net modules give rise to a transformation framework $\mathcal{T}_{PN}$ of Petri net modules as an instantiation of the categorical framework. The main technical result of this section is that the conditions given for this categorical formulation of the transformation framework are satisfied (in Theorem 2). The main results *composition and compositional semantics* are obtained by instantiation of the results stated in the previous section. In Section 3.3 we illustrate the semantics of Petri net modules that has not been given in [21,22].

3.1. Basic ideas of Petri net modules

First we give a short intuition of the underlying basics. Here we use the algebraic notion of Petri nets as introduced in [20]. Hence a place/transition net is given by a set of transitions, a set of places and pre- and post-domain functions $N = T \overset{pre}{\underset{post}{\rightrightarrows}} P^{\oplus}$, where $P^{\oplus}$ is the free commutative monoid over P. $P^{\oplus}$ also can be considered as the set of finite multisets over P. The pre- (and post-) domain function maps each transition into the free commutative monoid over the set of places, representing the places and the arc weight of the arcs in the pre-domain (respectively in the post-domain). An element $w \in P^{\oplus}$ can be presented as a linear sum $w = \sum_{p\in P} \lambda_p \cdot p$ or as a function $w : P \to \mathbb{N}$. We extend the usual operations and relations on natural numbers to operations and relations on linear sums namely $\oplus$, $\ominus$, $\le$, and so on. Moreover, we need to state how often a basic element is given within an element of the free commutative monoid. We define this for an element $p \in P$ and a word $w \in P^{\oplus}$ with $w_{|p} = \lambda_p \cdot p \in P^{\oplus}$. For $P' \subseteq P$ we define $w_{|P'} = \sum_{p\in P'} \lambda_p \cdot p$ and

$w_{|f_P(P)}$ we abbreviate to $w_{|f_P}$. The preset of a transition $\bullet t = \{p | pre(t)_{|p} \neq 0\}$ and the postset $t\bullet = \{p | post(t)_{|p} \neq 0\}$ are defined as usual. The firing of a transition $M[t > M'$ is computed by $M' = (M \ominus pre(t)) \oplus post(t)$ provided the transition is enabled, that is $M \geq pre(t)$. The set of all markings reachable from M is given by $[M >= \{M' \mid M[t_1 > \cdots [t_n > M'\}$.

Definition 5 (*Place/transition nets*). A place/transition net is given by $N = (P, T, pre, post, \widehat{M})$ with P the set of places, T the set of transitions, $pre, post : T \to P^{\oplus}$ the pre- and post-domain of transitions. $\widehat{M} \in P^{\oplus}$ describes the initial marking.

Morphisms are the basic entity in category theory; they relate the objects of the category and hence present its internal structure. 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 in [20] we use simple homomorphisms that are generated over the set of places. These morphisms map places to places and transitions to transitions. Morphisms are essential for the notion of modules and the definition of module operations. Two Petri net morphisms $m : IMP \to BOD$ and $r : EXP \rightsquigarrow BOD$ connect the interfaces to the body. The import morphism m is a *plain morphism* and describes how and where the resources in the import interface are used in the body. This morphism maps each place and transition in the import interface to its counterpart in the body. The initial marking of the source net needs to be compatible with the initial marking of the target net. Plain morphisms are presented by an arrow $\to$.

Definition 6 (*Plain morphisms*). A plain morphism $f : N_1 \to N_2$ between two place/transition nets $N_i = (P_i, T_i, pre_i, post_i, \widehat{M}_i)$ for $i = 1, 2$ is given by $f = (f_P, f_T)$ with $f_P : P_1 \to P_2$ and $f_T : T_1 \to T_2$ so that:

$pre_2 \circ f_T = f_P^{\oplus} \circ pre_1$ and
$post_2 \circ f_T = f_P^{\oplus} \circ post_1$

Moreover, for the initial marking we require for all $p \in P_1$:

$$\widehat{M}_1(p) \leq \widehat{M}_2(f_P(p)).$$

This yields the category **PT** consisting of place/transition nets and plain morphisms.

It is well known that plain morphisms $f : N_1 \to N_2$ preserve the firing behavior of nets, i.e. for each firing step $M_1[t_1 > M_1'$ in N_1 we have a corresponding firing step $M_2[t_2 > M_2'$ where M_2, M_2', t_2 are the translations of M_1, M_1', t_1 via f respectively. This means that the import morphism $m : IMP \to BOD$ preserves the firing behavior of the import *IMP*, but the Petri net *BOD* has in general much more places and transitions such that the firing behavior of *BOD* extends that of *IMP*.

The idea of the relationship between export *EXP* and body *BOD* presented by an export morphism $r : EXP \rightsquigarrow BOD$ is different. The intuitive idea of r is a refinement from the export *EXP* to the body *BOD*. This means that the functionality and the firing behavior of *EXP* is an abstraction of functionality and firing behavior of *BOD*. On purpose we do not require any explicit correspondence between the firing behaviors of *EXP* and *BOD* in order to allow the designer of a Petri net module enough freedom to express a refinement relationship which is adequate for his application area. The export morphism r is a *substitution morphism* and describes how the functionality provided by the export interface is realized in the body. This is done by mapping each part of the export interface that represents a certain functionality to the part of the body by which the functionality is realized. Substitution morphisms map places to places as well. But they can map a single transition to a whole subnet. So first we need to make precise what a subnet is.

Definition 7 (*Subnets*). Given a place/transition net $N = (P, T, pre, post, \widehat{M})$ then a subnet of N is given by $N' = (P', T', pre', post')$—written $N' \subseteq N$—if and only if $P' \subseteq P$ and $T' \subseteq T$ as well as for pre- and post-domain functions $pre'(t) = pre(t)_{|P'}$ and $post'(t) = post(t)_{|P'}$ for all $t \in T'$.

The set of all subnets of N is given by

$$\mathcal{P}(N) := \{N' | N' \subseteq N\}.$$

Note that we omit the initial marking of the subnet, since we use subnets always in relation to the given net N.

The basic idea is that substitution morphisms *substitute* a transition by a net. These morphisms capture a very broad idea of refinement and hence are adequate for the relation between the export net and the body net. The initial marking has to satisfy the same condition as for plain morphisms. Subsequently substitution morphisms are presented by an undulate arrow $\rightsquigarrow$.

We are convinced that our notion of substitution morphism, defined below, is flexible enough to express different kinds of refinement relationships between export and body as needed in different application areas. In this sense we assume that a substitution morphism $r : EXP \rightsquigarrow BOD$ defines a refinement of the firing behavior of *EXP* by that of *BOD*.

Definition 8 (*Substitution morphism*). A substitution morphism $f : N_1 \rightsquigarrow N_2$ between two place/transition nets N_1 and N_2 is given by $f = (f_P, f_T)$ with $f_P : P_1 \to P_2$ and $f_T : T_1 \to \mathcal{P}(N_2)$ where for each $t \in T_1$ there is $f_T(t) := N_2^t \subseteq N_2$ with $N_2^t = (P_2^t, T_2^t, pre_2^t, post_2^t)$ so that
$f_P(^\bullet t) \subseteq P_2^t$ and
$f_P(t^\bullet) \subseteq P_2^t$
Moreover, for the initial marking we require $\widehat{M}_1(p) \leq \widehat{M}_2(f_P(p))$ for all $p \in P_1$.

The composition of substitution morphisms $f : N_1 \rightsquigarrow N_2$ and $g : N_2 \rightsquigarrow N_3$ is given by $g \circ f := (g_P \circ f_P, g_T \circ f_T)$,

where $g_T \circ f_T : T_1 \to \mathcal{P}(N_3)$ is defined by $g_T \circ f_T(t) := \bigcup_{x \in T_2^t} N_3^x$

for $f_T = N_2^t = (P_2^t, T_2^t, pre_2^t, post_2^t)$ and $g_T(x) = N_3^x$ for $x \in T_2^t$.

Since all $N_3^x \subseteq N_3$ this construction is well defined.

This yields the category **PTS** consisting of place/transition nets and substitution morphisms.

This definition merely requires that the pre- and post-set of the mapped transition have to be part of the targeted subnet through the mapping f_P. It allows for example morphisms that are partial on the transitions ($T_2^t = \emptyset$), plain morphisms ($|T_2^t| = 1$), or morphisms, where each transition is mapped to the entire target net ($f_T(t) = N_2$).

An example of a substitution morphism can be found in Fig. 5, where the morphism $EXP \rightsquigarrow BOD$ is a substitution morphism. It maps places a and b in *EXP* to the places a and b in *BOD*. The transition z in *EXP* is mapped to the subnet of *BOD* consisting of places a, b, c, and d and the transitions v, w, and x. The transition y in *EXP* is mapped to the subnet of *BOD* consisting of places a and b and the transition u.

$$\begin{tikzcd} & EXP \arrow[d, squiggly, "r"'] \\ IMP \arrow[r, "m"] & BOD \end{tikzcd}$$

Fig. 4. Petri net module.

Next we relate plain and substitution morphisms. Therefore we define the net of a transition as the net surrounding t given by t and its adjacent places.

Definition 9 (*Net of a transition*). Given a transition $t \in T$ for some net $N = (P, T, pre, post, \widehat{M})$, then $net(t)$ the net surrounding t is given by:

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

with
$P^t = \bullet t \cup t\bullet$,
$T^t = \{t\}$, and
$pre^t : T^t \to P^{t\oplus}$ with $pre^t(t) = pre(t)$, and
$post^t : T^t \to P^{t\oplus}$ with $post^t(t) = post(t)$.

The subsequent lemma states that plain morphisms can be considered as a special case of substitution morphisms, that substitute a transition t by a subnet containing exactly one transition. The subnet is given by the net of the target transition $net(f_T(t))$.

Lemma 2 (Plain morphisms and substitution morphisms). *Each plain morphism $f : N_1 \to N_2$ given by $f = (f_P, f_T)$ can be expressed as a substitution morphism $f' : N_1 \rightsquigarrow N_2$ given by $f' = (f_P, f'_T)$ where $f'_T(t) = net(f_T(t))$, and we have the inclusion functor I :* **PT** $\to$ **PTS**, *with $I(f) := f'$.*

Moreover, if a substitution morphism $f : N_1 \rightsquigarrow N_2$ given by $f = (f_P, f_T)$ has for all $t \in T_1$ $f_T(t) = net(t')$ 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. □

Note that we omit the inclusion functor when plain morphisms in **PTS** are used.

Next we define Petri net modules as in [21,22]. We use this name as this notion of module can easily be transferred to any variant of Petri nets. In order to conform with the transformation-based component concept we restrict the plain morphism $m : IMP \to BOD$ to be injective.[1]

Definition 10 (*Petri net module*). A Petri net module $MOD = (IMP, EXP, BOD, m, r)$ is given by three place/transition nets *IMP*, *EXP*, and *BOD* that are related by morphisms; a plain injective morphism $m : IMP \to BOD$, and a substitution morphism $r : EXP \rightsquigarrow BOD$ as depicted in Fig. 4.

In the black box view of a Petri net module $MOD = (IMP, EXP, BOD, m, r)$ only the import net *IMP* and the export net *EXP* are visible, while the net *BOD* and the morphisms

[1] Here we require m to be injective, in [21,22] we use arbitrary plain morphisms in order to obtain union of modules as well.

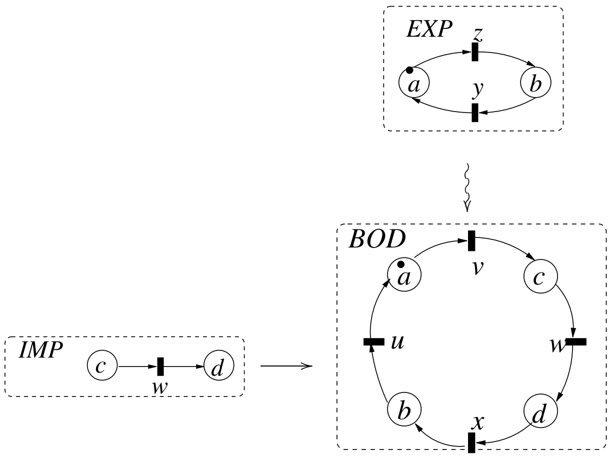

Fig. 5. Example of a simple module.

m and r are hidden. This black box view is sufficient to connect different Petri net modules via composition as defined below. The designer of a Petri net module MOD, however, must have a white box view of MOD, i.e. all parts are visible, in order to define explicitly the body net *BOD* and the morphisms m and r. For the semantics of MOD considered below we need a grey box view of MOD, i.e. only *IMP* and *EXP* are visible explicitly, but we need *BOD*, m an r in order to define the transformation semantics in the sense of Definition 4. The distinction between black, white and grey box view of modules or components is well known for software components (see [19,29]).

Example 1 (*Simple module MOD*). Fig. 5 illustrates a very simple net module. The import describes a single transition and the plain morphism $IMP \to BOD$ is an inclusion. The export of the module presents cyclic runs. The morphism $EXP \rightsquigarrow BOD$ abstracts from the more complex behavior of the body. The places a and b in *EXP* are mapped to a and b in *BOD*. Transition z is mapped to the subnet including the places a, b, c, d and the transitions v, w, x in *BOD*. Transition y is mapped to transition u. Hence, the export is an abstraction of the body. The slightly more complex structure of the cycle is hidden in the body.

The composition of modules is one of the module operations defined in [21,22]. From the practical point of view it is the most important one. The composition describes the import of a module into another module. Composition will be treated formally in the following subsection. Here, we merely give an example to illustrate this operation.

Example 2 (*Module composition*). We illustrate the composition of modules in Fig. 6 using the module $MOD = (IMP, EXP, BOD)$ from Example 1 and another module $MOD' = (IMP', EXP', BOD', m', r')$. The export of this module is mapped to the body by the substitution morphism mapping the transition p to the subnet consisting of the places n, o, and m and the intermediate transitions. The import is empty and so is the import morphism.

The connecting morphism $IMP \rightsquigarrow EXP'$ is an isomorphism.

The resulting module $MOD'' = MOD \odot MOD' = (IMP', EXP, BOD'')$ is constructed subsequently. The new body BOD'' of the resulting module is the net *BOD*, where the

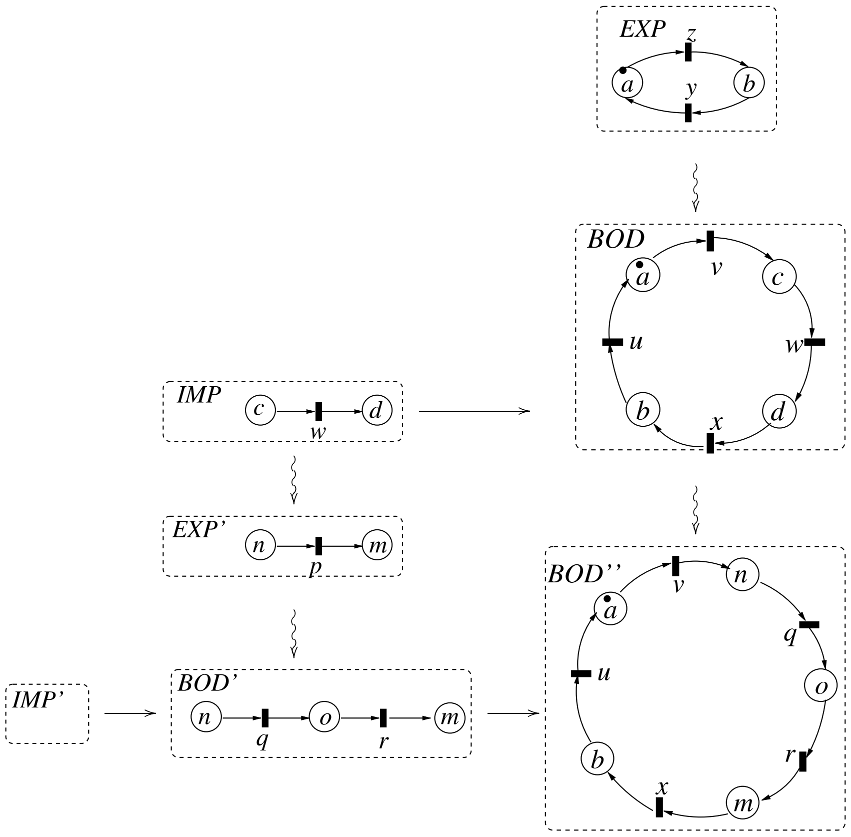

Fig. 6. Composition of modules.

transition between the places c and d is replaced by the net between the places n and m in the net BOD'. So, BOD'' can be considered as the gluing of BOD and BOD' along the net IMP.

Multiple interfaces could be obtained by having unconnected subnets in the import as well as in the export. This requires merely the disjoint union of nets, i.e. the coproduct in the category **PTS**. Another possibility is the explicit treatment of multiple interfaces as introduced for connector architectures in [12].

3.2. Transformation framework $\mathcal{T}_{PN}$ of Petri net modules

To define the transformation framework $\mathcal{T}_{PN}$ we need a class of transformations, which includes identical transformations, is closed under composition and satisfies the *extension property*. Substitution morphisms represent such a transformation. Obviously there are identities and composition. The extension property holds as there are pushouts of plain, injective morphisms and substitution morphisms.

We use the categorical version as given in Section 2.4.

Definition 11 (*Classes $\mathscr{I}$ and $\mathscr{E}$*). We have the class $\mathscr{I}$ consisting of plain, injective morphisms and the class $\mathscr{E}$ consisting of substitution morphisms, i.e. $\mathscr{I} = I(Mono_{\mathbf{PT}})$ and $\mathscr{E} = Mor_{\mathbf{PTS}}$.

Lemma 3 (Pushouts of plain, injective and substitution morphisms). *In the category* **PTS** *we have pushouts for plain, injective morphisms $f : N_0 \to N_1$ and substitution morphisms $g : N_0 \rightsquigarrow N_2$.*

Proof. Given an injective, plain morphism $f : N_0 \to N_1$ and a substitution morphism $g : N_0 \rightsquigarrow N_2$ then we have the pushout (N_3, f', g')

$$\begin{tikzcd}
N_0 \arrow[r, "f"] \arrow[d, squiggly, "g"'] \arrow[dr, phantom, "(1)"] & N_1 \arrow[d, squiggly, "g'"] \arrow[ddr, squiggly, bend left, "g''"] & \\
N_2 \arrow[r, "f'"] \arrow[drr, squiggly, bend right, "f''"'] & N_3 \arrow[dr, squiggly, "h"] & \\
& & N_4
\end{tikzcd}$$

$N_3 := (P_3, t_3, pre_3, post_3, \widehat{M}_3)$ is given by
- $P_3 = P_1 +_{P_0} P_2$ is a pushout in **Set**,
- $T_3 := (T_1 \setminus f_T(T_0)) \uplus T_2$,
  hence we have: $t_3 \in T_3$ implies $t_3 \in T_1$ xor $t_3 \in T_2$, (*)
- $pre_3 = \begin{cases} g'_P(pre_1(t_3)); & t_3 \in T_1 \\ f'_P(pre_2(t_3)); & t_3 \in T_2 \end{cases}$ $post_3$ is defined analogously.
- The initial marking $\widehat{M}_3$ is defined by
  $\widehat{M}_3 = \sum_{p \in P_3} \widehat{M}_{3|p}$. And $\widehat{M}_{3|p}$ is given by

$$\widehat{M}_{3|p} = \begin{cases} {f'_P}^{\oplus}(\widehat{M}_{2|p_2}); & p = f'_P(p_2) \text{ and } p \notin g'_P(P_1) \\ {g'_P}^{\oplus}(\widehat{M}_{1|p_1}); & p = g'(p_1) \text{ and } p \notin g(P_2) \\ max({f'_P}^{\oplus}(\widehat{M}_{2|p_2}), {g'_P}^{\oplus}(\widehat{M}_{1|p_1})); & p = g'(p_1) = f_P{}'(p_2) \end{cases}$$

  with $max(\lambda_1 p, \lambda_2 p) = max(\lambda_1, \lambda_2) \cdot p$.

The morphisms are given by $f' = (f'_P, f'_T)$ and $g' = (g'_P, g'_T)$.
f'_P and g'_P are defined by the pushout P_3.
f'_T and g'_T are given below:

$f'_T : T_2 \to \mathscr{P}(N_3)$ with
$f'_T(t) = (f'_P, inc_{T_2})(net(t))$
So f' is plain (see Lemma 2)
and can be considered to be $f' = (f'_P, inc_{T_2})$. (**)

$g'_T : T_1 \to \mathscr{P}(N_3)$ with

$$g'_T(t) = \begin{cases} f'(g_T(t_0)); & t = f_T(t_0), \\ (g'_P, inc_{T_1})(net(t)); & t \notin f_T(T_0), \end{cases}$$

f' and g' are well-defined due to the definition of pre_3 and $post_3$.

It remains to prove the universal property:
Given $f''_T : N_2 \rightsquigarrow N_4$ and $g''_T : N_1 \rightsquigarrow N_4$ such that $g''_T \circ f = f''_T \circ g$.
Then we define the unique $h : N_3 \rightsquigarrow N_4$ where $h_P : P_3 \to P_4$ is uniquely induced by the pushout P_3 in **Set**.

And we define $h : T_3 \to \mathcal{P}(N_4)$ with

$$h(t) = \begin{cases} f''_T(t); & t \in T_2, \\ g''_T(t); & t \in T_1, \end{cases}$$

h is well defined due to $(*)$ and because:

$$\text{for } t \in T_1 \text{ we have:} \quad h_P({}^\bullet t) = h_P \circ g'_p({}^\bullet t) = g''_p({}^\bullet t) \in P^t_4;$$

$$\text{for } t \in T_2 \text{ we have:} \quad h_P({}^\bullet t) = h_P \circ f'_p({}^\bullet t) = f''_p({}^\bullet t) \in P^t_4.$$

The next lemma states that plain, injective morphisms are preserved under pushouts. Trivially, so are substitution morphisms.

Lemma 4 (Pushout-stable morphism classes $\mathcal{I}$ and $\mathcal{E}$). *Given a pushout (N_3, f', g') of an injective, plain morphism $f : N_0 \to N_1$ and a substitution morphism $g : N_0 \rightsquigarrow N_2$ then we have $f' \in \mathcal{I}$ and $g' \in \mathcal{E}$:*

$$\begin{tikzcd} N_0 \arrow[r, "f"] \arrow[d, rightsquigarrow, "g"'] \arrow[dr, phantom, "(1)"] & N_1 \arrow[d, rightsquigarrow, "g'"] \\ N_2 \arrow[r, "f'"] & N_3 \end{tikzcd}$$

Proof. Follows directly for morphisms in class $\mathcal{I}$ from $(**)$ in proof of Lemma 3 and for morphisms in $\mathcal{E}$ from $\mathcal{E} = Mor_{PTS}$. □

Theorem 2 (Transformation framework $\mathcal{T}_{PN}$). *Given*
- **PTS** *the category of place/transition nets with substitution morphisms,*
- $\mathcal{I} = I(Mono_{\mathbf{PT}})$ *the class of plain, injective morphisms, and*
- $\mathcal{E} = Mor_{\mathbf{PTS}}$ *the class of substitution morphisms.*

then we have a transformation framework as in Definition 1, called transformation framework $\mathcal{T}_{PN} = (\mathbf{PT}, \mathcal{I}, \mathcal{E})$ of Petri net modules.

Proof

(1) $\mathcal{E}$-$\mathcal{I}$-Pushout condition holds due to Lemma 3.

(2) $\mathcal{E}$ and $\mathcal{I}$ are stable under pushouts due to Lemma 4. □

By instantiation of the general theory in Sections 2.4–2.6 we obtain the subsequent results.

Results 1 (*Composition and compositional semantics of Petri net modules*). For the transformation framework $\mathcal{T}_{PN}$ of Petri net modules we have the following results:
- Composition of Petri net modules as given by instantiation of Definition 3 and Lemma 1. The composition has been illustrated in Example 2.
- Compositional semantics of Petri net modules as given by instantiation of Definition 4 and Theorem 1. Examples are discussed in the subsequent section.

3.3. Interpretation and examples for the semantics of Petri net modules in the transformation-based framework

The transformation framework $\mathcal{T}_{PN}$ of Petri net modules in the previous section provides Petri net modules with a transformation-based semantics, while semantics has not been yet defined in [21,22]. So in the subsequent subsection we employ the semantics of the transformation-based framework.

According to Fig. 2 the transformation semantics of the component *COMP* is a function $TrafoSem(COMP)\colon Trafo(IMP) \to Trafo(EXP)$ defined for all $trafo \in Trafo(IMP)$, by $TrafoSem(COMP)(trafo) = trafo' \circ exp \in Trafo(EXP)$.

This means that the transformation-based semantics of a Petri net module $MOD = (IMP, EXP, BOD, m, r)$ is given by a fuction $TrafoSem(MOD)$ which maps each import substitution morphism $trafo : IMP \rightsquigarrow N$ to a corresponding export substitution morphism $trafo' \circ r : IMP \rightsquigarrow N'$. According to Section 3.1 the idea of a substitution morphism $f : N_1 \rightsquigarrow N_2$ is a refinement of the firing behavior of N_1 by that of N_2. This means that $TrafoSem(MOD)$ maps each refinement of the firing behavior of the import net *IMP* given by $trafo : IMP \rightsquigarrow N$ to a corresponding refinement of the firing behavior of the export net *EXP* given by $trafo' \circ r : EXP \rightsquigarrow N'$. In fact, if MOD is composed with MOD' via a substitution morphism $connect : IMP \rightsquigarrow EXP'$ we obtain a substitution morphism $r' \circ connect : IMP \rightsquigarrow BOD'$. This means composition of MOD leads in a natural way to different substitution morphisms $trafo : IMP \rightsquigarrow N$. The transformation-based semantics of MOD reflects how each refinement $trafo : IMP \rightsquigarrow N$ of the firing behavior of *IMP* leads to a corresponding refinement of the firing behavior of *EXP* given by $trafo' \circ r : EXP \rightsquigarrow N'$. The following examples illustrate these semantical concepts.

Example 3 (*Transformation semantics of the simple module MOD*). The transformation semantics of the simple module MOD is given by:

$$TrafoSem(MOD)\colon Trafo(IMP) \to Trafo(EXP)$$

where $Trafo(IMP)$ is the set of substitution morphisms from *IMP* to some refined net N and accordingly $Trafo(EXP)$ the set of substitution morphisms from *EXP* to some refined net N'.

This semantics maps substitution morphisms from the import net to corresponding substitution morphisms from the export net as defined by $TrafoSem(MOD)$.

Below you find two examples:

- On the left-hand side of the Fig. 7 we have the substitution morphism from the import *IMP* of the simple module MOD as given in Fig. 5 to the net $N1$.
  It maps the transition w to the subnet consisting of the places c, d, e and the transitions t, u. This morphism is mapped to the substitution morphism on the right-hand side of the Fig. 7 and has been achieved by the construction of pushout **(1)** to the right.

$$\begin{tikzcd}
 & EXP \arrow[d, squiggly] \\
IMP \arrow[r] \arrow[d, squiggly] \arrow[dr, phantom, "(\mathbf{1})"] & BOD \arrow[d, squiggly] \\
N1 \arrow[r] & N1'
\end{tikzcd}$$

- On the left-hand side of the Fig. 8 we have the substitution morphism from the import *IMP* of the simple module MOD as given in Fig. 5 to the net $N2$.

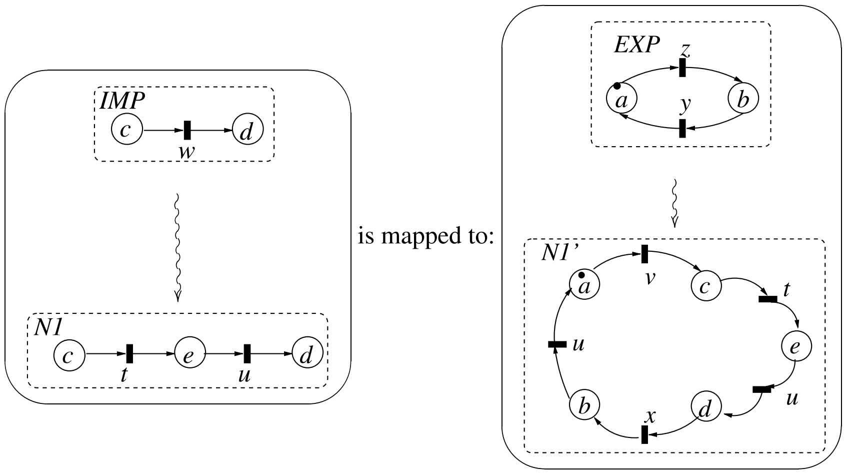

Fig. 7. Mapping of transformation semantics.

It maps the transition w to the subnet consisting of the places c, d and the transitions r, s. This morphism is mapped to the substitution morphism on the right-hand side of the Fig. 8 and has been achieved by the construction of pushout **(2)** to the right.

$$\begin{tikzcd}
 & EXP \arrow[d, squiggly] \\
IMP \arrow[r] \arrow[d, squiggly] \arrow[rd, phantom, "(\mathbf{2})"] & BOD \arrow[d, squiggly] \\
N2 \arrow[r] & N2'
\end{tikzcd}$$

Examples for the compositional semantics look similar, but have been omitted due to space limitations.

4. Petri net modules preserving safety properties

In this section we investigate Petri net modules with safety properties as introduced in [23]. Here we give a concise review and relate them to the transformation-based approach to components. These conform as well with the transformation-based approach to components and in Theorem 3 we show that the conditions given for this categorical formulation of the transformation framework are satisfied. The main results concerning composition and a compositional semantics are again obtained by instantiation of the results stated in Section 2.

4.1. Safety properties and modules

First we formalize safety properties in order to formulate our theorems concerning their preservation. We recall formulas over markings and their translations via morphisms. An axiomatic expression is λp, denoting that $\lambda \in \mathbb{N}$ tokens are on place p. We then can build

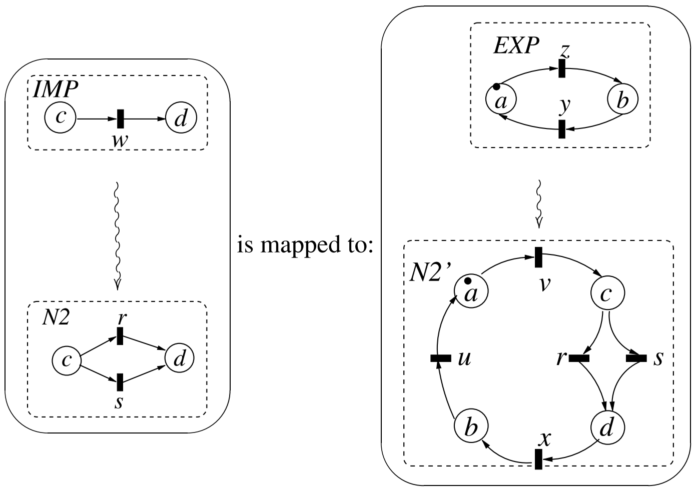

Fig. 8. Mapping of transformation semantics.

logic formulae over such axioms, e.g. $4a \Longrightarrow 2b$ formalizes the statement that four tokens on place a imply two tokens on place b. Safety properties describe invariants of the net behavior. Hence we use the henceforth operator $\Box$ to express that a formula shall hold for all reachable markings. The formula $\Box a \vee b$ in the export in Fig. 9 states that one token is always either on place a or on place b.

Definition 12 (*Formulas, safety properties, translations*). Given a place/transition net $N = (P, T, pre, post, \widehat{M})$ then λp is a static formula for $\lambda \in \mathbb{N}$ and $p \in P$, static formulas are built up using the logical operators $\wedge$ and $\neg$, deriving the other operators $\vee$, $\Rightarrow$ etc. in the standard way. Let φ be a static formula over N. Then $\Box\varphi$ is called a safety property.

The validity of formulas is given w.r.t. the marking of a net. Let $M \in P^{\oplus}$ be an arbitrary marking of N then the formula λp holds in N under M written as $M \models_N \lambda p$ if and only if $\lambda p \leq M$ in terms of linear sums. For $M \models_N \neg\varphi_1$ if and only if $\neg(M \models_N \varphi_1)$ and $M \models_N \varphi_1 \wedge \varphi_2$ if and only if $(M \models_N \varphi_1) \wedge (M \models_N \varphi_2)$.

The safety property $\Box\varphi$ holds in N under M if and only if φ holds in all states reachable from M: $M \models_N \Box\varphi$ if and only if $\forall M' \in [M\rangle : M' \models_N \varphi$. We also write $N \models \Box\varphi$ instead of $\widehat{M} \models_N \Box\varphi$ if $\widehat{M}$ is the initial marking.

The translation of formulas $\mathbb{T}_f$ over N_1 along a morphism $f = (f_P, f_T) : N_1 \to N_2$ to formulas over N_2 is given for atoms by $\mathbb{T}_f(\lambda p) = \lambda f_P(p)$. The translation of formulas is given recursively by $\mathbb{T}_f(\neg\varphi) = \neg\mathbb{T}_f(\varphi)$, and $\mathbb{T}_f(\varphi_1 \wedge \varphi_2) = \mathbb{T}_f(\varphi_1) \wedge \mathbb{T}_f(\varphi_2)$, and $\mathbb{T}_f(\Box\varphi) = \Box\mathbb{T}_f(\varphi)$.

We now have to ensure specific conditions that guarantee morphisms preserving safety properties. Intuitively, the next definition requires substitution morphism to be place preserving: Any transition of the target net that has a place of the source net in its pre- or post-domain (i.e. there is an ingoing respectively an outgoing arc between the transition and the place) needs to have a source transition in the source net, so that the pre-domain and post-domain of the transition is preserved. In other words this definition ensures that

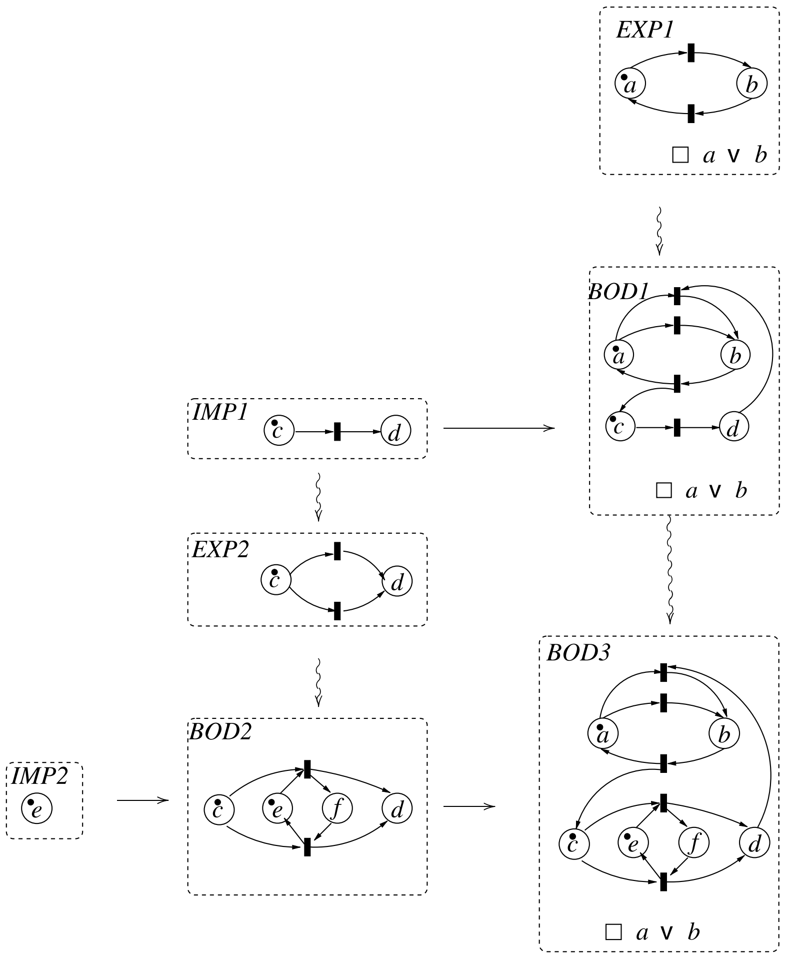

Fig. 9. Example for composition.

neither arcs may be deleted nor "new" arcs to "old" places are allowed. Hence we have chosen the term place preserving.

Definition 13 (*Place preserving substitution morphism*). A substitution morphism $f : N_1 \rightsquigarrow N_2$ is called place preserving if for all $t_2 \in T_2$ with $pre_2(t_2)_{|f_P} \oplus post_2(t_2)_{|f_P} \neq \epsilon$ we have:
There is some $t_1 \in T_1$, so that
$t_2 \in T_2^{t_1}$, where $T_2^{t_1}$ are the transitions of $f_T(t_1) = N_2^{t_1} \subseteq N_2$ and
$f_P^{\oplus}(pre_1(t_1)) = pre_2(t_2)_{|f_P}$, and
$f_P^{\oplus}(post_1(t_1)) = post_2(t_2)_{|f_P}$.

Clearly, to preserve safety properties the marking on the mapped places needs to stay the same. Places that are not in the image of the morphism may be marked arbitrarily.

Definition 14 (*Marking strict substitution morphism*). A substitution morphism $f : N_1 \rightarrow N_2$ is called marking strict if

f_P is injective, and
$\widehat{M}_1(p) = \widehat{M}_2(f_P(p))$

An example of a substitution morphism that is marking strict and place-preserving can be found in Fig. 9. The morphism $EXP1 \rightsquigarrow BOD1$ is injective on the places, it preserves the marking, and no "new" transitions are adjacent to "old" places. The morphism $EXP \rightsquigarrow BOD$ in Fig. 5 is not place-preserving, but then it does not preserve safety properties either.

The notion of place-preserving morphisms is quite restrictive. But in order to preserve any safety property, we need to ensure that each transition of the target net has no effect or the same effect as one of the original transitions on the places mapped trough f_P.

A substitution morphism that is place-preserving and marking strict preserves safety properties up to the renaming $\mathbb{T}_f$ induced by the morphism.

Lemma 5 (Safety property preserving morphism). *Given a safety property $\Box\varphi$ and a substitution morphism $f : N_1 \rightsquigarrow N_2$ that is place preserving and marking strict then the following holds*:

$$N_1 \models \Box\varphi \; implies \; N_2 \models \Box\mathbb{T}_f(\varphi).$$

Hence, such a morphism is called safety property preserving morphism. Moreover, safety property preserving morphism are closed under composition.

The proof is analogous to the proof of Fact 4.16 in [14]. Although the underlying morphism is different we can use the same argument, since the conditions we use in Definition 13 correspond to Fact 4.13 in [14].

Proof Sketch. All transitions that are adjacent to a place being mapped from N_1 to N_2 have a pre-image in N_1 with the same arc weight to that place (Definition 13 and Fact 4.13 in [14]). Hence we can prove inductively that these morphisms reflect reachability (Fact 4.14 in [14]). By induction over the structure of a static formula (without any temporal quantor) we show that place preserving and marking strict preserve and reflect reachability (Fact 4.15 in [14]). Proving indirectly that whenever N_2 violates a safety property so does N_1 (Fact 4.15 in [14]) concludes the proof. □

This allows the definition of Petri net modules that preserve safety properties from the export net to the body net.[2] As we desire a treatment of properties that is independent of the body net, we can use safety property preserving morphisms to relate the export net to the body net. If we require $r : EXP \to BOD$ to be safety property preserving, then any safety property holding in EXP will be preserved.

Definition 15 (*Petri net modules with safety*). A module $MOD = (IMP, EXP, BOD, m, r)$ where $r : EXP \to BOD$ is safety property preserving is a called a Petri net module with safety.

[2] In [23] we have distinguished between implicit and explicit safety properties. Since only Petri net modules with implicit safety properties conform with the transformation-based component approach, we drop that distinction here.

Using the result of Lemma 5 we can now conclude that those safety properties $\Box\varphi$ holding in the export net *EXP* must also hold in *BOD*. The safety properties $\Box\mathbb{T}_r(\varphi)$ in *BOD* are translated along the morphism $r : EXP \to BOD$.

Corollary 1 (Translated safety properties). *Given a Petri net module* $MOD = (IMP, EXP, BOD)$ *with safety then for any safety property holding in the export net* $EXP \models \Box\varphi$, *the translated safety property holds in the body* $BOD \models \Box\mathbb{T}_r(\varphi)$.

We now illustrate what happens to safety properties when we compose modules. The main intention of this work is to simplify compositional reasoning by preserving safety properties throughout the construction of modules. First we give an example of a composition of modules with safety properties. Composition can also be obtained by instantiation of the transformation-based approach in the subsequent subsection.

Example 4 (*Composition of modules*). An example of a composition is illustrated in Fig. 9. There is a module $MOD1 = (IMP1, EXP1, BOD1, m1, r1)$ with safety. The safety property $\Box a \vee b$ is stated explicitly in the export of module $MOD1$. It states the fact that the token is on place a or on place b. This can be seen immediately. In fact, the property could be formulated even stronger using an "exclusive or".

The morphism $EXP1 \rightsquigarrow BOD1$ is safety property preserving: It is injective on the places and marking-strict because on the places a and b are no new tokens. And it is place-preserving as there are only transitions adjacent to the places a and b in *BOD*1 that are adjacent to the places a and b in *EXP*1 as well.

The module $MOD2 = (IMP2, EXP2, BOD2)$ is a module with safety as well. The resulting module $MOD3 = MOD1 \odot MOD2 = (IMP2, EXP1, BOD3)$ is constructed subsequently.

The new body *BOD*3 of the resulting modules is the net *BOD*1, where the transition between the places c and d is replaced by the net between the places c and d in the net *BOD*2. So, *BOD*3 can be considered as the gluing of *BOD*1 and *BOD*2 along the net *IMP*1. The resulting module $MOD3 = MOD2 \odot MOD1 = (IMP2, EXP1, BOD3)$ is a module with safety because the morphism $EXP1 \rightsquigarrow BOD3$ is safety property preserving.

4.2. Transformation framework $\mathcal{T}_{PN_{Safe}}$ of Petri net modules with safety

To define the transformation framework $\mathcal{T}_{PN_{Safe}}$ we again have the class $\mathcal{I}$ consisting of plain, injective morphisms. In this case the class $\mathcal{E}$ consists of safety property preserving morphisms.

Lemma 6 (Pushouts are stable under safety property preserving morphisms). *Given a pushout as in Lemma 3. Then we have*:

g is safety property preserving implies that g' is safety property preserving.

Proof. We have to show that g' is safety property preserving if g is safety property preserving:

(1) g' is place preserving:
Let $\epsilon \neq pre_3(t) \geq p_3$ for some $p_3 \in g'_P(P_1)$.
Then there are two cases as $t \in T_1$ xor $t \in T_2$.
(a) $t \in T_1$:
Since $T_3 := (T_1 \setminus f_T(T_0)) \uplus T_2$ we conclude $t \notin f_T(T_0)$.
Due to the definition of $g'_T : T_1 \to T_2$ we have $g'_T(t) = (g'_P, id_{T_1})(net(t))$.
Due to the definition of $net(t)$ we conclude:
$pre_3(t) = g'^{\oplus}_P(pre_1(t))$ and $post_3(t) = g'^{\oplus}_P(post_1(t))$.
And hence:
$pre_3(t)_{|g'_P} = g'^{\oplus}_P(pre_1(t))$ and $post_3(t)_{|g'_P} = g'^{\oplus}_P(post_1(t))$.

(b) $t_3 \in T_2$:
This implies some $p_2 \in P_2$ and some $p_1 \in P_1$ with $f'_P(p_2) = p_3 = g'_P(p_1)$.
Due to the pushout properties of P_3 there is some $p_0 \in P_0$ with $f_P(p_0) = p_1$ and $g_P(p_0) = p_2$.
As g_P is place-preserving we know there is some $t_0 \in T_0$ with $t \in T_2^{t_0}$ so that

$$pre_2(t)_{|g_P} = g_P^{\oplus}(pre_0(t_0)), \text{ and}$$
$$post_2(t)_{|g_P} = g_P^{\oplus}(post_0(t_0)).$$

By definition of g'_T with $g'_T(f_T(t_0)) = f'_T \circ g_T(t_0)$ there is $f_T(t_0) \in T_1$ with $t \in T_3^{f_T(t_0)}$.
We now conclude:

$$\begin{aligned} pre_3(t)_{|g'_P} &== f'^{\oplus}_P(pre_2(t))_{|g'_P} \\ &= f'^{\oplus}_P(pre_2(t)_{|g_P}) \\ &= f'^{\oplus}_P(g_P^{\oplus}(pre_0(t_0)) \\ &= g'^{\oplus}_P(f_P^{\oplus}(pre_0(t_0)) \\ &= g'^{\oplus}_P(pre_1(f_T(t_0)) \end{aligned}$$

and the same for:

$$\begin{aligned} post_3(t)_{|g'_P} &== f'^{\oplus}_P(post_2(t))_{|g'_P} \\ &= f'^{\oplus}_P(post_2(t)_{|g_P}) \\ &= f'^{\oplus}_P(g_P^{\oplus}(post_0(t_0)) \\ &= g'^{\oplus}_P(f_P^{\oplus}(post_0(t_0)) \\ &= g'^{\oplus}_P(post_1(f_T(t_0)) \end{aligned}$$

(2) g' is marking strict:
g'_P is injective, as g_P is injective and pushouts in **Set** preserve injective morphisms.
Moreover, we need to show $\widehat{M}_{3|g'_P} = g'^{\oplus}_P(\widehat{M}_1)$:
We only need to investigate $p_3 \in g'_P(P_1)$.
We then have two cases:
(a) $\widehat{M}_{3|g'_P} = g'^{\oplus}_P(\widehat{M}_1)$ for $p_3 \in P_3 \setminus f'_P(P_2)$,
(b) and for $p_3 = g'_P(p_1) = f'_P(p_2)$ we have:
$\widehat{M}_{3|g'_P} = max(f'^{\oplus}_P(\widehat{M}_{2|p_2}), g'^{\oplus}_P(\widehat{M}_{1|p_1})) = g'^{\oplus}_P(\widehat{M}_1)$
due to the following estimation:

$$\begin{aligned} g'^{\oplus}_P(\widehat{M}_{1|f_P(p_0)}) &= \geq g'^{\oplus}_P \circ f_P^{\oplus}(\widehat{M}_{0|p_0}) \\ &= f'^{\oplus}_P \circ g_P^{\oplus}(\widehat{M}_{0|p_0}) \\ &= f'^{\oplus}_P(\widehat{M}_{2|_2}) \end{aligned}$$

The following theorem relates the results from [23] to those in [11] and yields the new ones as stated in Results 2 below.

Theorem 3 (Transformation framework $\mathcal{T}_{PN_{Safe}}$). *Given*

- **PTS** *the category of place/transition nets with substitution morphisms*,
- $\mathcal{I}$ *the class of plain*, *injective morphisms*, *and*
- $\mathcal{E}$ *the class of safety property preserving morphisms*

then we have a transformation framework, called the transformation framework $\mathcal{T}_{PN_{Safe}} = (\mathbf{PTS}, \mathcal{I}, \mathcal{E})$ *of Petri net modules with safety*.

Proof

(1) $\mathcal{E}$-$\mathcal{I}$-Pushout condition holds due to Lemma 3.
(2) $\mathcal{E}$ and $\mathcal{I}$ are stable under pushouts due to Lemmas 6 and 4. □

By instantiation of the general theory in Sections 2.4–2.6 we have the following results.

Results 2 (*Composition and semantics of Petri net modules with safety*). For the transformation framework $\mathcal{T}_{PN_{Safe}}$ of Petri net modules with safety we have the following results:

- Composition of Petri net modules with safety as given by instantiation of Definition 3 and Lemma 1. The composition has been illustrated in Example 4.
- Compositional semantics of Petri net modules with safety as given by instantiation of Definition 4 and Theorem 1.

5. Discussions

To round up this paper we continue with a discussion of the relation to components based on Petri nets and net transformations in [11]. Subsequently, we present our case study on Petri net modules in [24] and discuss the main insights.

5.1. Relation to high-level replacement systems approach

An alternative approach to the Petri net modules in this paper has been presented in [11] where Petri net transformations have been considered as instantiations of high-level replacement systems, short HLR-systems. HLR-systems have been introduced in [8] as an abstraction of graph transformation systems. This abstraction is obtained by defining HLR-systems for any category **Cat** using double-pushout transformations in **Cat** instead of the category of graphs. In this approach a rule consists of three objects and two morphisms $L \leftarrow K \rightarrow R$. A direct transformation of an object N according to a rule is given by a context object C and a morphism $K \rightarrow C$, such that M becomes a pushout object for diagram **(1)** in Fig. 10. This means that N can be obtained by gluing C and L over K. The result of the direct transformation is then given by M is a pushout object for diagram **(2)**.

This HLR-approach has been applied to Petri nets in our paper [25] leading to the notion of net transformation systems. Instantiated to Petri nets, Fig. 10 defines a direct net transformation from net $N \rightarrow M$ via a net rule $p : (L \leftarrow K \rightarrow R)$. A net transformation is defined to be sequence of direct net transformations via rules $p_1, \ldots, p_2$. This makes sure

$$\begin{tikzcd}
L \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dl, phantom, "(\mathbf{1})"] \arrow[dr, phantom, "(\mathbf{2})"] & R \arrow[d] \\
N & C \arrow[l] \arrow[r] & M
\end{tikzcd}$$

Fig. 10. Double-pushout transformation.

that net transformations are closed under composition. Embeddings $i : N \to N'$ in the sense of the generic transformation concept presented in Section 2.2 are defined as inclusion morphisms.

The main question in view of the generic transformation concept is now the validity of the extension property. In fact, the extension diagram corresponds to the Embedding Theorem well known in the theory of graph transformation and HLR-systems. Actually, the Embedding Theorem allows the extension of a transformation along an embedding, but it requires that the boundary points of the embedding are preserved by the transformation. The boundary points of a graph embedding $f : G \to G'$ are all those nodes v in G such that $f(v)$ is source or target of an edge $e \in G' - f(G)$. This means that the extension property for HLR-systems and hence for net transformation systems is not satisfied in the strict form stated in Section 2.2 but only in a weaker form, where the embedding is consistent with the transformation as discussed above. This case is studied in detail in [11] mentioned above. Especially this weaker version of an extension diagram is in general not a pushout as required in our categorical version of the transformation framework in this paper. Hence the component framework based on HLR-systems instantiated to Petri nets is not a special case of the Petri net modules in this paper.

Vice versa the substitution morphisms of Petri net modules in this paper can be considered as transformations, where the rules might correspond to substitutions of single transitions. In some examples this is the case (see the substitution morphism in Fig. 5), but in general substitution morphisms cannot be considered rule-based transformations.

The advantage of Petri net modules based on substitution morphisms in contrast to Petri net components based on net transformations is the fact that the extension property holds without additional consistency condition. This implies that composition of Petri net modules is always well defined, while in the case of Petri net components a consistency condition has to be checked. Moreover, Petri net modules are defined for marked Petri nets, while the HLR-approach has been instantiated to unmarked Petri nets only. Vice versa the HLR-approach has been considered for low-level and high-level Petri nets already, but it is still open to generalize the approach in this paper to high-level Petri nets.

Finally let us note that the preservation of safety properties has been considered in both approaches based on place preserving morphisms. Moreover, in the HLR-approach also the preservation of liveness properties has been studied.

5.2. Case study

Let us present the main insights of a case study presented in [24]. The case study models a simple version of a fully automated call center of a phone company featuring basic services for enquiring about telephone numbers of other telephone subscribers as well as for recording and delivering messages to a given phone number at a time specified by the customer. The customer may choose from a selection of modes for payment (like paying by credit card, by telephone bill etc.) and he can query his balance if he has an account with the operator company of the call center. The services of the call center are only available in a specific area (a city, a country etc.). The main focus of this case study has been the question whether the new structuring technique of Petri net modules can be applied reasonably to a larger example. Since the emphasis of this case study is the structuring of the system with Petri net modules and not in the realistic and accurate modeling of the call center, place/transition nets are used instead of some more expressive high-level Petri net type. Moreover the case study is limited to the user/system-interface of the call center and neglects the underlying technical details of the call center.

In the case study as given in [24] the modules comprising the telecom service center are presented. The order of presentation is roughly top-down, beginning with the overall system and ending with the modules that provide basic functions such as announcing system messages to the user. Here we can merely give a short summary focusing the topmost level and its construction.

The development of the case study has clearly shown that the concept of Petri net modules [21,22] is applicable for structuring large and complex net models. The main advantages of this approach are:

- The 1-to-1 correspondence to component concepts in the sense of [30,19]:
  As the underlying paradigms are essentially the same Petri net modules can directly be used to model the process view (operational behavior) of a component.
- The expressiveness of the interfaces:
  The interfaces introduced in our approach are Petri nets, and not only nodes of a net. Hence the export allows presenting an abstraction of the modules behavior. And the import allows requiring a specific behavior of the modules to be imported.
- The openness of the interfaces:
  The import specifies what must be satisfied by the export interface of an imported module. But it does not specify specific modules to be imported. Hence every module is formally completely unrelated to other modules. So it can be easily exchanged by another module as long as the export specifications are compatible. The actual relations between the modules are established using the module operations.

6. Conclusion

In this paper we have established the connection between the transformation-based approach to components [11] and Petri net modules [22,23]. First we have developed a formalization where we use the pushout construction to describe the extension property. We require two classes of morphisms for export and import that are closed under composition. The category where we instantiate the transformation-based approach has to have pushouts of these morphisms. We have used for the instantiation of the transformation-based approach to components the category of place/transition nets with substitution morphisms. There we use the class of plain injections and the class of all substitution morphisms.

For Petri net modules with safety we have instantiated the transformation-based approach in the same category but with different morphism classes: again the class of plain, injective morphisms, but then the class of safety property preserving substitution morphisms. In both cases the instantiation yields composition of modules and a compositional semantics. This semantics is the transformation semantics relating each refinement of the import to a corresponding refinement of the export via the body.

Future work comprises the following possibilities:

- Algebraic high-level net components:
  Algebraic high-level nets can be considered to be place/transition net with an additional data type description in terms of algebraic specifications. The algebra is used to describe the token of a net and the terms over the signature are used for the arc inscriptions. Plain morphisms in this paper correspond to morphisms between algebraic high-level net except that those treat additionally the algebraic specification. The concept of substitution morphisms can be easily transferred to algebraic high-level nets because the main features of this morphisms concern the net structure and not the arc inscriptions. It seems to be straightforward to prove the conditions for the categorical formulation of the transformation-based approach. Accordingly we can define algebraic high-level net components, composition as well as a compositional transformation semantics.
- Further system properties to be preserved:
  Especially liveness properties are of interest. For Petri net transformations we have already been coping with morphisms that preserve liveness. In [26] a survey of those results is given comprising different kinds of morphisms that preserve liveness and their properties concerning pushouts. These notions of liveness preserving morphisms are likely to be suitable for the export morphism. The resulting components should then preserve liveness in the sense of modules with safety as introduced in this paper.

References

[1] E. Battiston, F.D. Cindio, G. Mauri, OBJSA nets: a class of high-level nets having objects as domains, in: G. Rozenberg (Ed.), Advances in Petri Nets, Lecture Notes in Computer Science, vol. 340, Springer, 1988, pp. 20–43.

[2] E. Battiston, F.D. Cindio, G. Mauri, L. Rapanotti, Morphisms and minimal models for OBJSA nets, in: 12th International Conference on Application and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 524, Springer, 1991, pp. 455–476.

[3] M. Broy, T. Streicher, Modular functional modelling of Petri nets with individual tokens, Advances in Petri Nets, Lecture Notes in Computer Science, vol. 609, Springer, 1992.

[4] P. Buchholz, Hierarchical high level Petri nets for complex system analysis, Application and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 815, Springer, 1994, pp. 119–138.

[5] S. Christinsen, N. Hansen, Coloured Petri nets extended with channels for synchronous communication, Application and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 815, Springer, 1994, pp. 159–178.

[6] W. Deiters, V. Gruhn, The FUNSOFT net approach to software process management, International Journal on Software Engineering and Knowledge Engineering 4 (2) (1994) 229–256.

[7] J. Desel, G. Juhás, R. Lorenz, Process semantics of Petri nets over partial algebra, Proceedings of the XXI International Conference on Applications and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 1825, Springer, 2000, pp. 146–165.

[8] H. Ehrig, A. Habel, H.-J. Kreowski, F. Parisi-Presicce, From graph grammars to high level replacement systems, Lecture Notes in Computer Science, vol. 532, Springer-Verlag, 1991, pp. 269–291.

[9] H. Ehrig, B. Mahr, Fundamentals of algebraic specification. 2. Module specifications and constraints, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1990.

[10] H. Ehrig, F. Orejas, 2001. A generic component concept for integrated data type and process specification techniques, Tech. Rep. 2001/12, Technische Universität Berlin, FB Informatik.

[11] H. Ehrig, F. Orejas, B. Braatz, M. Klein, M. Piirainen, A generic component concept for system modeling, in: Proceedings of FASE '02, Lecture Notes in Computer Science, vol. 2306, Springer, 2002.

[12] H. Ehrig, J. Padberg, F. Orejas, B. Braatz, M. Klein, S. Perez, E. Pino, A generic framework for connector architectures based on components and transformations, in: Proceedings of FESCA 2004, Satellite Event of ETAPS 2004, 2004.

[13] R. Fehling, A concept of hierarchical Petri nets with building blocks, Advances in Petri Nets'93, Lecture Notes in Computer Science, vol. 674, Springer, 1993, pp. 148–168.
[14] M. Gajewsky, K. Hoffmann, J. Padberg, Place preserving and transition gluing morphisms in rule-based refinement of place/transition systems, Tech. Rep. 99-14, Technical University Berlin, 1999.
[15] V. Gruhn, A. Thiel, Komponentenmodelle: DCOM, JavaBeans, EnterpriseJavaBeans, CORBA, Addison-Wesley, 2000.
[16] X. He, A formal definition of hierarchical predicate transition nets, Application and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 1091, Springer, 1996, pp. 212–229.
[17] K. Jensen, Coloured petri nets. Basic concepts, analysis methods and practical use, EATCS Monographs in theoretical computer science edition, Basic Concepts, Springer-Verlag, 1992.
[18] E. Kindler, Modularer Entwurf verteilter Systeme mit Petrinetzen, Ph.D. thesis, Technische Universität München, Institut für Informatik, 1995.
[19] S. Mann, B. Borusan, H. Ehrig, M. Große-Rhode, R. Mackenthun, A. Sünbül, H. Weber, Towards a component concept for continuous software engineering, Tech. Rep. 55/00, FhG-ISST, 2000.
[20] J. Meseguer, U. Montanari, Petri nets are monoids, Information and Computation 88 (2) (1990) 105–155.
[21] J. Padberg, Place/transition net modules: transfer from algebraic specification modules, Tech. Rep. TR 01-3, Technical University Berlin, 2001.
[22] J. Padberg, Petri net modules, Journal on Integrated Design and Process Technology 6 (4) (2002) 105–120.
[23] J. Padberg, Safety properties in Petri net modules, in: Proceedings of 7th World Conference on Integrated Design and Process Technology (IDPT 2003), Society for Process Technology, CD-ROM, 2003.
[24] J. Padberg, M. Buder, Structuring with Petri net modules: a case study, Tech. Rep. TR 01-4, Technical University Berlin, 2001.
[25] J. Padberg, H. Ehrig, L. Ribeiro, Algebraic high-level net transformation systems, Mathematical Structures in Computer Science 5 (1995) 217–256.
[26] J. Padberg, M. Urbášek, Rule-based refinement of Petri nets: a survey, in: H. Ehrig, W. Reisig, G. Rozenberg, H. Weber (Eds.), Advances in Petri Nets: Petri Net Technologies for Modeling Communication Based Systems, Lecture Notes in Computer Science, vol. 2472, Springer, 2002.
[27] C. Sibertin-Blanc, Cooperative nets, Application and Theory of Petri Nets'94, Lecture Notes in Computer Science, vol. 815, Springer, 1994, pp. 471–490.
[28] M. Simeoni, A categorical approach to modularization of graph transformation systems using refinements, Ph.D. thesis, Università Roma La Sapienza, 1999.
[29] C. Szyperski, Component Software—Beyond Object-Oriented Programming, Addison-Wesley, 1997.
[30] H. Weber, Continuous engineering of communication and software infrastructures, Lecture Notes in Computer Science, vol. 1577, Springer-Verlag, 1999, pp. 22–29.