High-Level Replacement Systems for Software Components

Julia Padberg

Bericht–Nr. 2005-07
ISSN-Nummer: 1436-9915

High-Level Replacement Systems for Software Components

Julia Padberg

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

padberg@cs.tu-berlin.de

11th September 2005

Contents

1 Introduction **2**

2 Examples of Rule-Based Refinement for Components **4**
2.1 Automata 4
2.2 Petri Nets 10

3 Transformations of Components **15**
3.1 The Transformation Framework 15
3.2 Rules, Transformations and Hierarchical Composition 20

4 Transformations of Automata Components **33**
4.1 Automata Components 33
4.2 Transformations for Automata Components 35

5 Transformations of Petri Net Modules **39**
5.1 Review of Petri Net Modules 39
5.2 Transformation of Petri Net Modules 40

6 Conclusion **44**

A Review of VK Squares and Weak AHLR Categories **49**

B Technical Background **50**

C Additional Diagrams **52**

1 Introduction

Changing environments - commercial, technical or social - demand software systems that can be adapted to those changes with reasonable effort. Component-based systems have been proposed as an adequate support for that task. Today there is no doubt on the importance of component-based systems. There are various specification for the description of component-based software architectures. In [28] a component concept for continuous software engineering is introduced that represents the informal basis of the generic approach adopted in this paper. Component-based software engineering needs to be backed by thorough formal concepts and modeling techniques, so that the correctness and consistency of the component and the component-based system increase. Formal specification of the component interface and the component specification allow the precise modeling and the verification of required functionality. Moreover, the composition of components can be given formally and correctness and consistency can be ensured even in the composed system. In our approach the formal description is the basis to make the conditions under which a component can be changed or exchanged explicitly. The compatibility of component transformation with component composition is an key issue of component transformation as it represents the core question of changing a component in some given context. The second main results concerns the conditions for the most complex case: transforming both components and their interfaces in different ways and keeping the composition of these components via their interfaces intact. This result is given by the Compatibility Theorem stating that the result of first transforming the two components and the composing them is the same (up to renaming) as composing the two components first and applying then the composed rule. Transformations of components arise whenever a component-based system is changed and new components are introduced, components are replaced by other ones, components are changed, or components are deleted. Here we present a formal frame for the description of these transformations.
As mentioned above the generic component framework including transformation can be applied to various formal and semi-formal specification techniques In this paper we concentrate on the new instantiation to automata. We use a very simple categorical notion of deterministic input automata. Additionally we show that the extension of the generic framework to transformations also holds for Petri nets.
In this report we concentrate on two formal specification techniques, automata and Petri nets. So we first give a cursory glance on the components concepts that use these.

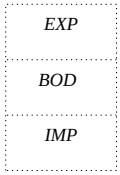
Automata The use of automata for the description of components and/or their interfaces is well established. In [9] the interfaces are modelled using input/output automata. The parallel composition of the interfaces is given and criteria for the compatibility are presented. but this approach merely concerns the interfaces. In [1] input/output automata are used as well. There are three abstraction levels, the structural description of the architecture, the modeling of the component behavior and a data type description of the component. but the the component is monolithic, consiting only of an interface. Parameterized contracts [29, 41] are used for the adequate component composition and architecture evaluation in practice. They make use of a component definition that is the motivation for the definition of automata components below. Provides and requires interfaces a given using deterministic automata, and they are related – making parameterization of contracts possible – by so called service effect automata (see [42]). these service effect automata represent the componet specification and allows the connection of the interfaces. The provides interface cyyresponds to the export interface, the requires interface to the import interface and the service effect automata correspond to the body specification.

Petri Nets 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. [24, 6, 23, 22]) as well as a variety concepts for connector mechanisms as communication, coordination or cooperation (e.g. [8, 43, 11, 10]). In other approaches places and transitions of modules are merged by well-defined operations (e.g. [26, 3, 2, 5]). Most attempts to Petri net modules (among others [7, 12, 25]) do not provide Petri nets as interfaces. For a not so recent survey see [4]. There are either places or transitions, but no full Petri nets in the interface. When modeling software components these notions of Petri net modules are not powerful enough, since they do not allow specifying behavior in the interfaces.
The transformation-based component framework for generic components has been first presented at FASE 2002 in [17]. The main concepts are a self-contained semantics and composition of components, based on a generic transformation concept. Here we have presented a categorical formalization where we use pushouts to characterize the main construction. We achieve the desired properties as proposed in [17] using properties of the pushout construction. There have been quite different formal and semi-formal specification techniques used within in this frame work, process algebras, Petri nets and UML. The idea of transferring the principles of algebraic graph transformation has been successfull. There are various specification techniques, where via high-level replacement systems the double-pushout transformations have been applied for different purposes starting with different types of Graphs, structures and place/transition nets in [15]. Transformations of algebraic specification have been given in [14], and to high-level Petri nets via abstract Petri nets in [31]. A recent survey can be found in chapter 5 of [21]. Hence, the main idea is to integrate two abstract descriptions: the transformation-based component framework and high-level replacement systems.

Overview

This report continues with two examples, that illustrate the transformation of components. In the first example in section 2.1 components are based on automata and in the second example in section 2.2 components are based on Petri nets. Then we present the general part of the theory starting with the transformation-based component framework. Subsequently we show that under specific assumptions we have transformations of components. Additionally we show that the hierarchical composition of components is compatible with transformations. This result is extended to the comaptibility of hierarchical composition with $\mathcal{Q}$-transformations. To achieve the formal foundation for the examples given in section 2 we have first the section 4 where we introduce automata components based on our formal framework. These automata components are the formal foundation for components used for parametrized contracts in [29]. We define components, their composition and how that the categorical assumptions for the transformations are satisfied. In section 5 we review Petri net modules as given in [34], that are components in the sense of section 3. Then we prove that the assumption for the transformation of components are satisfied. The conclusion summarizes the results (and the possible results) we have for the transformation of components for the given instantiations as well as for further instantiations.

2 Examples of Rule-Based Refinement for Components



In this section we present two small examples to motivate the use of transformations of components. The first is based on automata and the second is based on Petri nets. In the first case we use a notation, where the component is given as a box as depicted adjacently. EXP, IMP and BOD are then automata as in section 2.1. In case of Petri nets we could clearly use the same notation, but have used one that is closer to the formal basis and already has been used in [35]. There we give a components as an diagram. In section 1 we have already discussed the advantages of the generic approach and its instantiation to other specification techniques.

2.1 Automata

In this section we present a simplified version of an automatic teller machine, that consists of two components, that are composed hierarchically. Subsequently, both components get modified.
First we have a look at the two components. The component $ATM1$ consists of the export interface EXP_ATM1, the import interface IMP_ATM1, and the body BOD_ATM1. The export automaton models the allowed steps as the user may observe them. The body is a refinement in the sense that intermediate transitions as send_PIN, valid_PIN, and update are modeled additionally. The component imports those parts that deal with the connection to the corresponding bank.

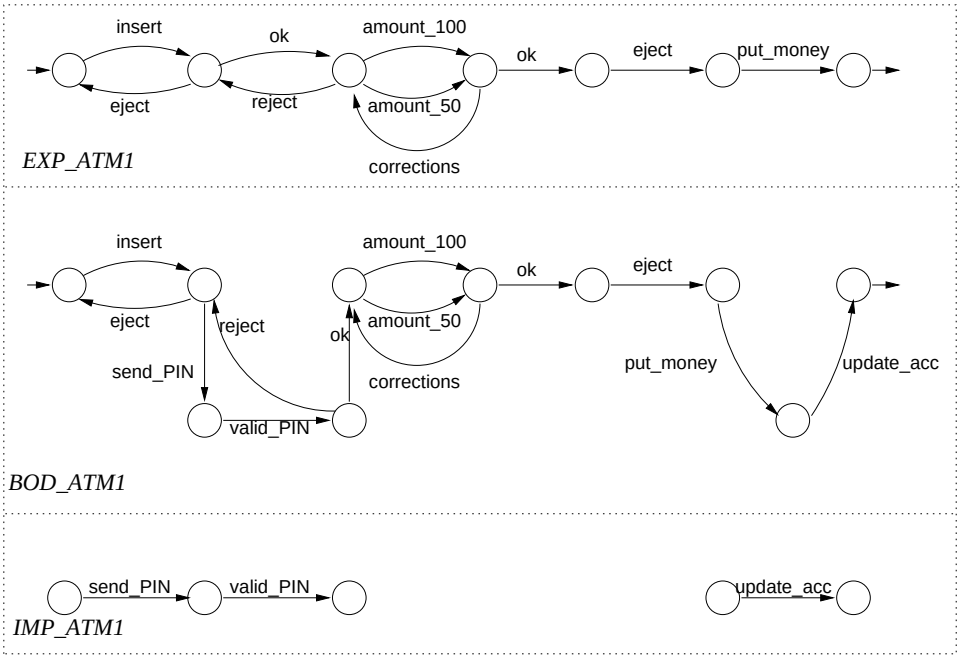


Figure 1: Component $ATM1$

In figure 2 we have the export that consists of two unconnected parts, that each abstract from the subautomaton that models the transactions concerning the bank, i.e. the sending and the validation of the PIN and the update of the users bank account. This is modeled in the body of the component. The import automaton

consists of two unconnected transitions, that each model the contact to the bank.

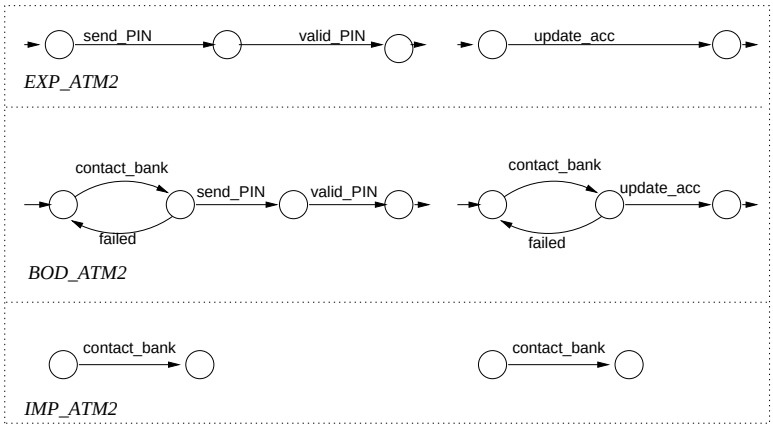


Figure 2: Component $ATM2$

The composition results in the following component $ATM = ATM1 \circ ATM2$, where the resulting component $ATM7$ is depicted in figure 3 and the construction diagram can be found in the appendix C in figure 21. $ATM1$ is the importing component and $ATM2$ the imported component. Hence, the export of $ATM7$ consists of the export of $ATM1$, and the import of ATM consists of the import of $ATM2$. The body of ATM is a well-defined gluing of the bodies of $AMT1$ and $ATM2$, see figure 21 in appendic C.

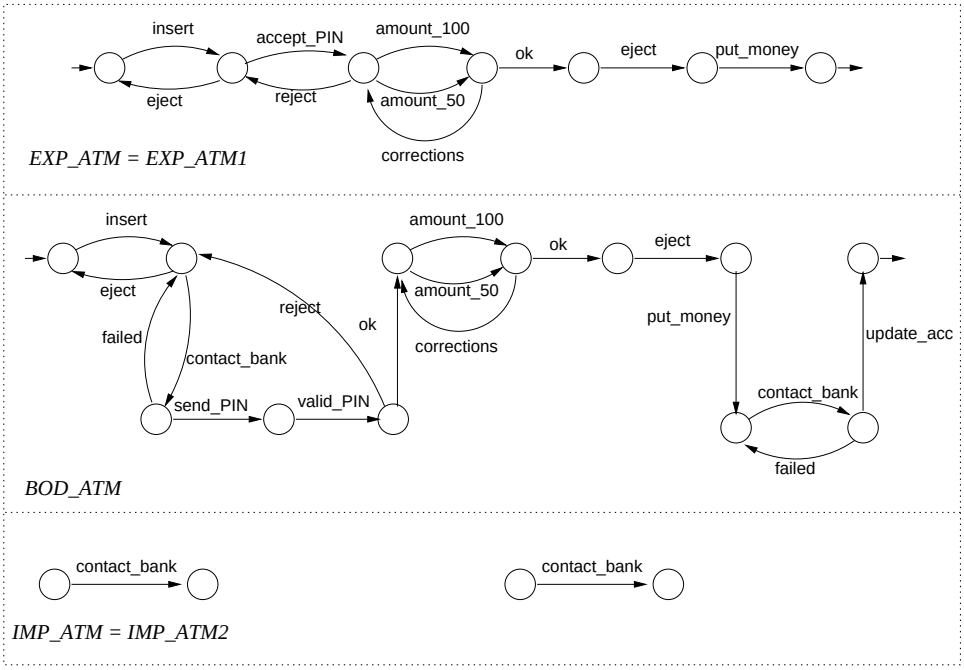


Figure 3: Component ATM

Transformations of the ATM

Let us imagine the following modifications: in $ATM2$ two times the bank is contacted, now we change this component, so that one a connection to the bank is opened and at the end of the transaction it is closed.

The rules and transformations we now illustrate are in the sense of the double pushout approach to graph transformation (e.g. in [21]). Rules are give by a span of morphisms where the left hand morphism indictes the part that is deleted or preserved, and the right hand morphisms indicates the part that is preserved or added. The application of rule leads to a transformation of the source usunig two pushouts in the corresponding category (hence double pushout approach).

The first rule depicted in figure 4 is a very basic one that merely changes the body without affecting the interfaces. The body automaton is changed as first the transitions send between $s1$ and $s2$ is deleted and subsequently replaced so that an additional state and the additional transition open_con and send are added. Here we give the states explicitly only to indicate the morphisms as inclusions. There are no automata in the export and import denoting the empty automata with the empty alphabet, the empty set of states and the empty transition function.

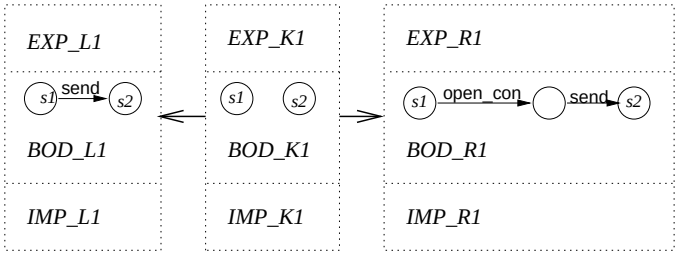


Figure 4: Rule atm_r1

The application of atm_r1 to the component $ATM2$ yields the transformation depicted in figure 5. This transformation consists of three transformations at the level of the automata, namely:

$$\begin{array}{ccccc} EXP\_L1 = \emptyset & \longleftarrow & EXP\_K1 = \emptyset & \longrightarrow & EXP\_R1 = \emptyset \\ \downarrow & & \downarrow & & \downarrow \\ EXP\_ATM2 & \longleftarrow & EXP\_ATM3 = EXP\_ATM2 & \longrightarrow & EXP\_ATM4 = EXP\_ATM2 \end{array}$$

$$\begin{array}{ccccc} IMP\_L1 = \emptyset & \longleftarrow & IMP\_K1 = \emptyset & \longrightarrow & IMP\_R1 = \emptyset \\ \downarrow & & \downarrow & & \downarrow \\ IMP\_ATM2 & \longleftarrow & IMP\_ATM3 = IMP\_ATM2 & \longrightarrow & IMP\_ATM4 = IMP\_ATM2 \end{array}$$

and

$$\begin{array}{ccccc} BOD\_L1 & \longleftarrow & BOD\_K1 & \longrightarrow & BOD\_R1 \\ \downarrow & & \downarrow & & \downarrow \\ BOD\_ATM2 & \longleftarrow & BOD\_ATM3 & \longrightarrow & BOD\_ATM4 \end{array}$$

The export and the import parts do net get changed, and in the body part first the transition send_PIN is deleted and subsequently its added together with the transition open_con.

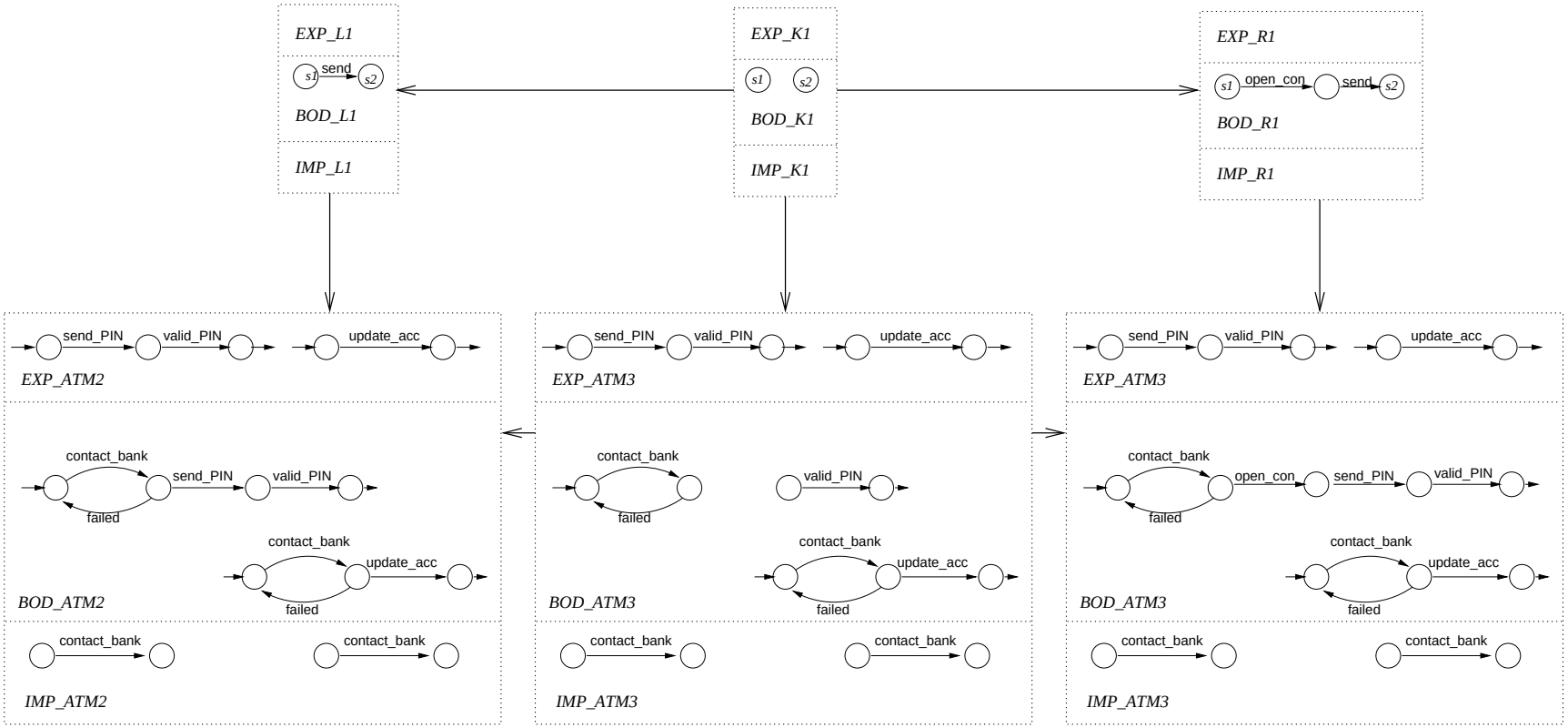


Figure 5: Transformation $ATM2 \stackrel{atm\_r1}{\Longrightarrow} ATM4$

The second rule depicted in figure 6 changes the second time the bank is contacted. There we change all three parts; export, import and body and replace in body and export the transition update_acc with the transitions update_acc and close_con. In the import the transition contact_bank is deleted. The application $ATM4 \overset{atm\_r2}{\Longrightarrow} ATM5$ to the component $ATM4$ yields the component $ATM5$ depicted in figure 7. There we have achieved that the bank has to be contacted only once and we open and close the corresponding connection.

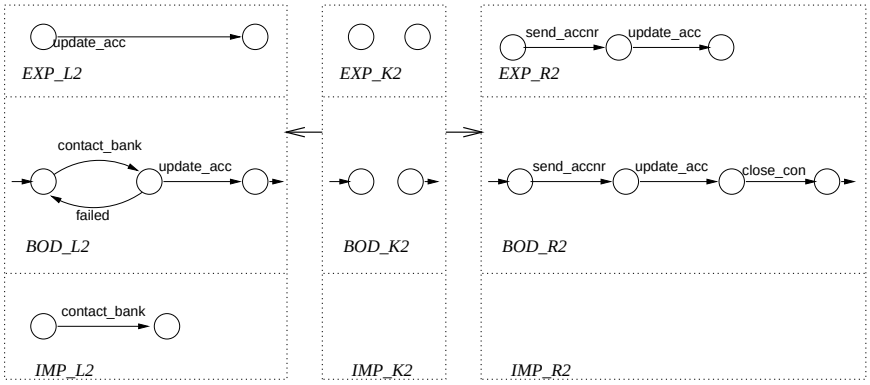


Figure 6: Rule atm_r2

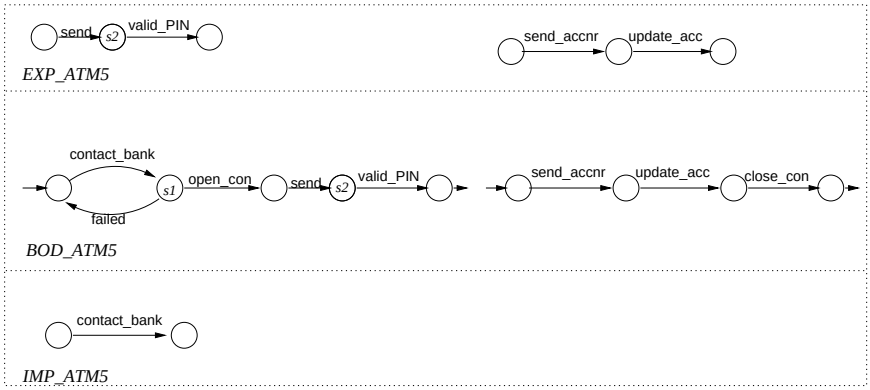


Figure 7: $ATM5$

Next we want to change the component $ATM1$ so that the import and the body is changed. We want to have an additional transition for counting the money before putting it to the withdrawal box for the user. This is modeled in rule atm_r3 in figure 8. In the body one state and the two adjacent transitions put_money and update_acc are deleted and two new state with two adjacent transitions count, put_money, and update_acc are added. In order to have a consistent interface component $Comp\_K3$ in the import also one state and the adjacent transition update_acc is deleted and added again. Similarly in the export the adjacent transition put_money is deleted and added again. The result of $ATM1 \overset{atm\_r3}{\Longrightarrow} ATM6$ is depicted in figure 9.

Now we can again compose $ATM5 \circ ATM6 = ATM7$. The composition diagram is given in the appendix in Figure 22 and the result $ATM7$ is given in figure 10.

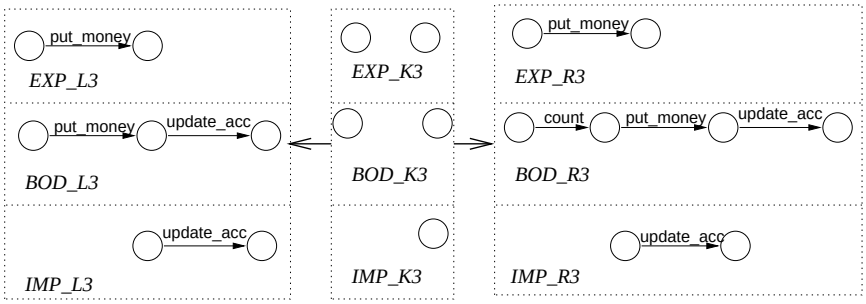


Figure 8: Rule atm_r3

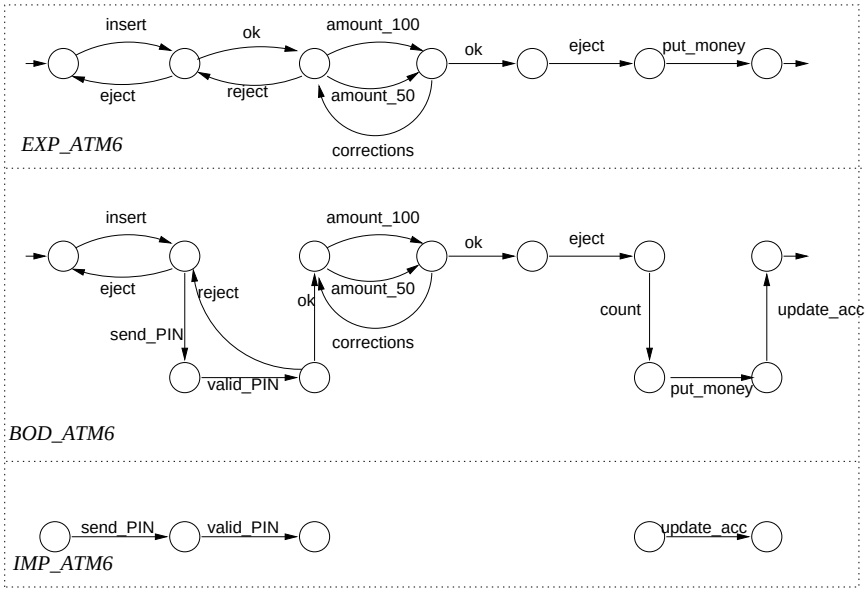


Figure 9: Component $ATM6$

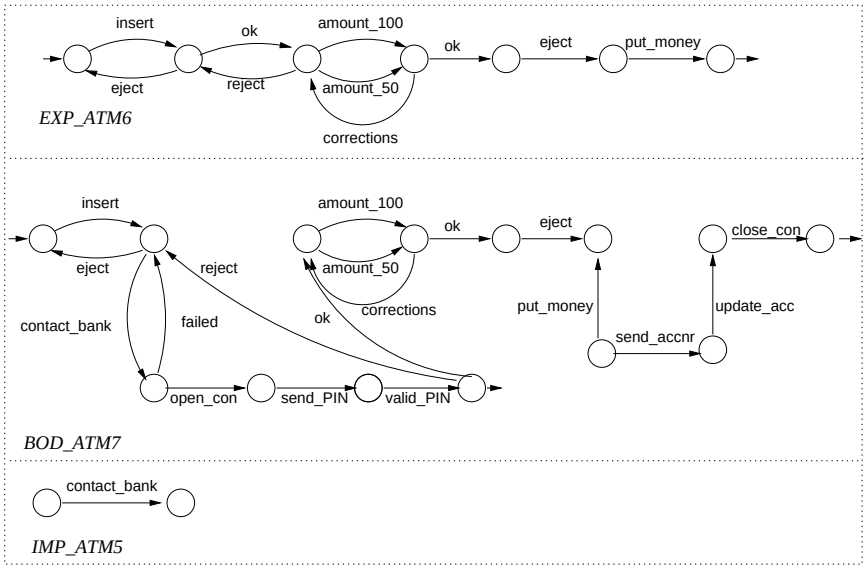


Figure 10: Component $ATM7$

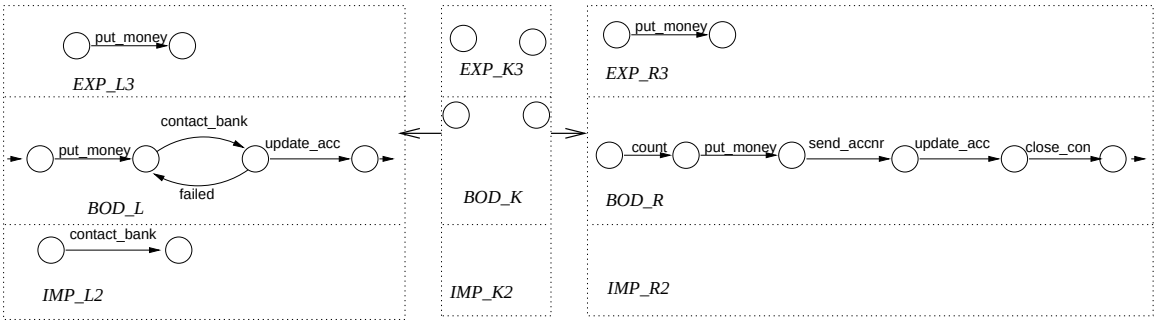
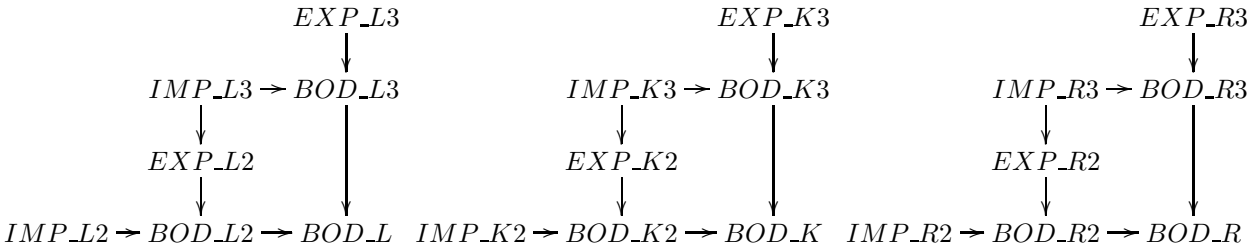


Figure 11: Rule $atm\_r = atm\_r3 \circ atm\_r2$

In fact 3.11 we state the conditions that a rule is independent of the composition. Obviously atm_r1 is independent of the composition $ATM1 \circ ATM2$, so we can conclude there is the derivation $ATM \overset{atm\_r1}{\Longrightarrow} ATM'$ where $ATM' \cong ATM1 \circ ATM2$. i.e. that ATM' and $ATM1 \circ ATM2$ are equal up to renaming.
One of the main results of this paper states in theorem 3.14 that the composition can be compatible with transformations, where even the intermediate interfaces are involved. Therefore we have to compose the rules atm_r3 and atm_r2 hierarchically. these means that we have to compose the left-hand side components of both rules, the intermediate components of both rules, and he right-hand side components of both rules. Hence we have the following three hierarchical compositions:



So we can directly transform $ATM' \overset{r}{\Longrightarrow} ATM''$ using the rule $r = atm\_r3 \circ atm\_r2$ as depicted in figure 11, where r is constructed by hierarchical composition with $atm\_r3 \circ atm\_r2$. And we then have due to theorem 3.14 that $ATM'' \cong ATM7$, i.e. that ATM'' and $ATM7$ are equal up to renaming.

2.2 Petri Nets

Here we use the same example as given in [35]: We have an example with three modules that describe the process of writing urging and offering letters. The module Mod_W given in Figure 12 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 13 we show the corresponding module and then in figure 15 we illustrate the composition of these modules. The module Mod_W in Figure 12 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 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. 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.

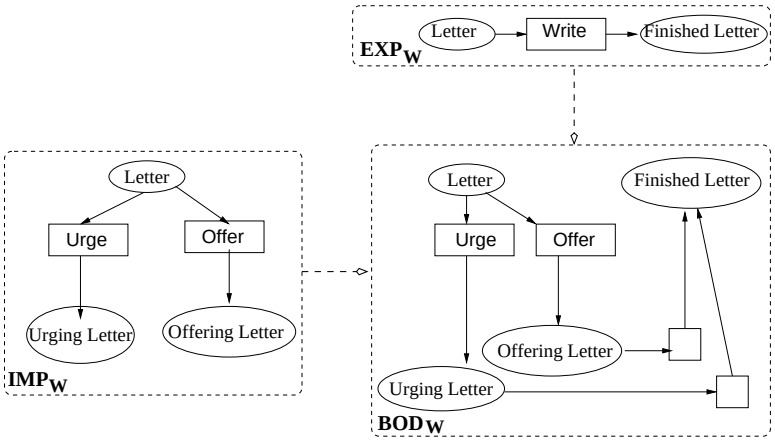


Figure 12: MOD_W

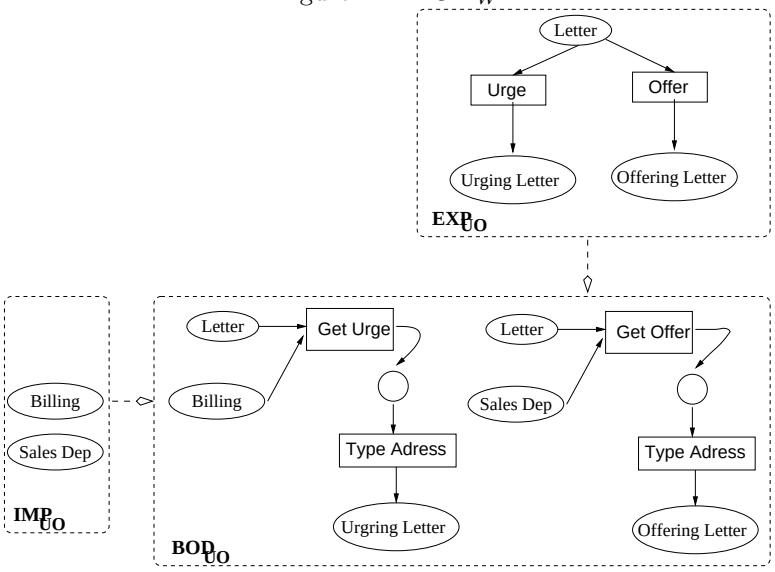


Figure 13: MOD_{UO}

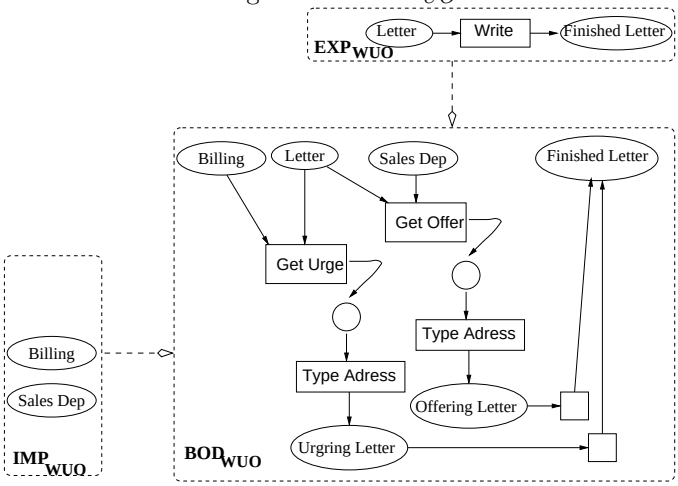


Figure 14: MOD_{WUO}

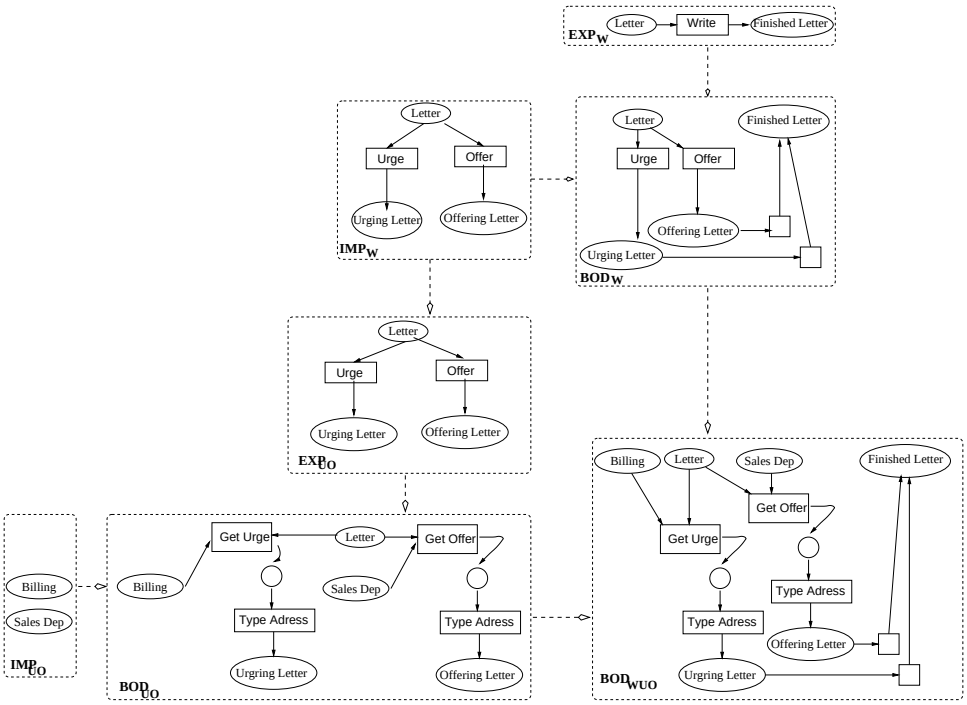


Figure 15: Composition of MOD_{UO} and MOD_W

The module MOD_{UO} is then composed with module MOD_W In Figure 15 we show the construction with the intermediate nets. BOD_{WUO} is the pushout of $BOD_{UO} +_{IMP_{UO}} BOD_W$. In Figure 14 we finally see the result of the composition. Note that $IMP_{WUO} = IMP_{UO}$ and $EXP_{WUO} = EXP_W$. The rule r_{UO} depicted in figure 16 transforms the Petri net modules mod_{UO} in the following way: The transition offer is deleted and replaced by the transtions GetOffer and GetSignature. This is done in the export as well as in the body. applying this rule to the module Mod_{UO}. The transformation $MOD_{UO} \overset{r_{UO}}{\Longrightarrow} MOD_2$ changes the module accordingly resulting in MOD_2 as given in Figure 17. To apply r_{UO} to the composed module MOD_{WUO} we need to compose it with the rule $r_W = (Mod_{RW} \leftarrow MOD_{RW} \rightarrow MOD_{RW})$, that has for the left-hand side, the intermediate and the right-hand side the same module, namely MOD_{RW}, see figure 18. This rule obviously does not change anything in MOD_W. The composition $r_W \circ r_{UO}$ is depicted in figure 19 and its application to MOD_{WUO} yields the transformation $MOD_{WUO} \overset{r_W \circ r_{UO}}{\Longrightarrow} MOD_3$. So, due to Theorem 3.14 we have

$$\begin{array}{ccccc} MOD_W & \circ & MOD_{UO} & = & MOD_{WUO} \\ \Big\Downarrow{\scriptstyle r_W} & & \Big\Downarrow{\scriptstyle r_{UO}} & & \Big\Downarrow{\scriptstyle r_W \circ r_{UO}} \\ MOD_W & \circ & MOD_2 & = & MOD_3 \end{array}$$

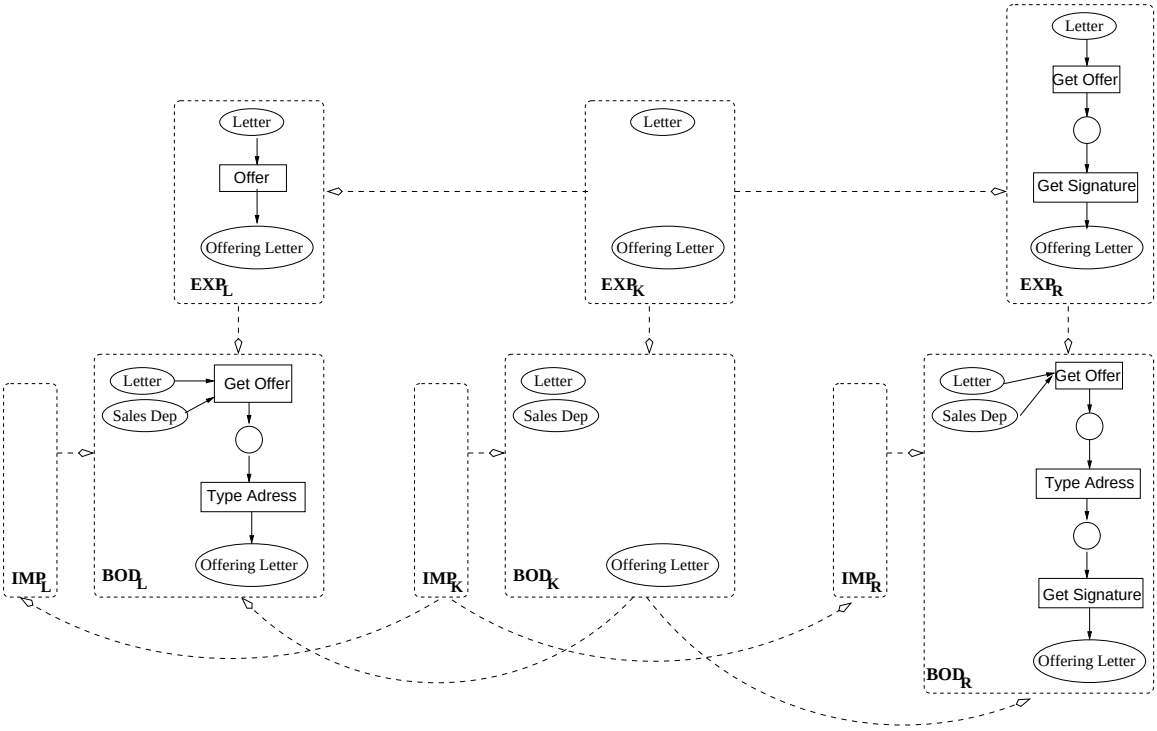


Figure 16: Rule r_{UO}

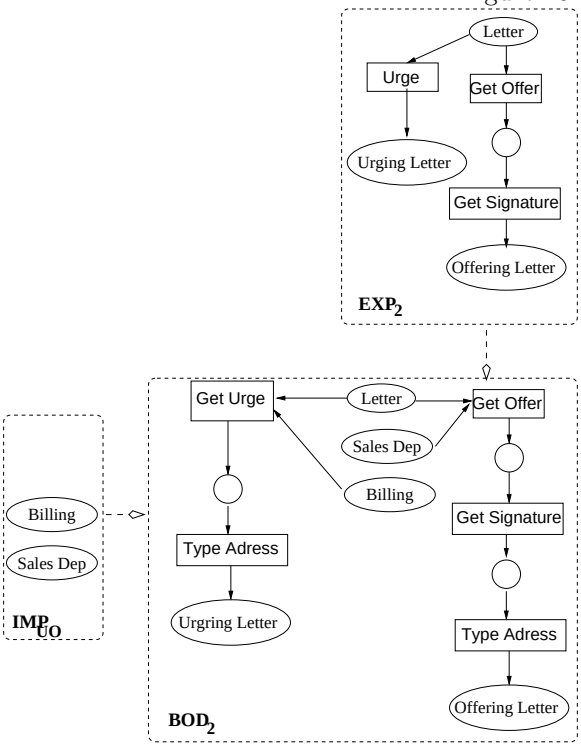


Figure 17: Module MOD_2

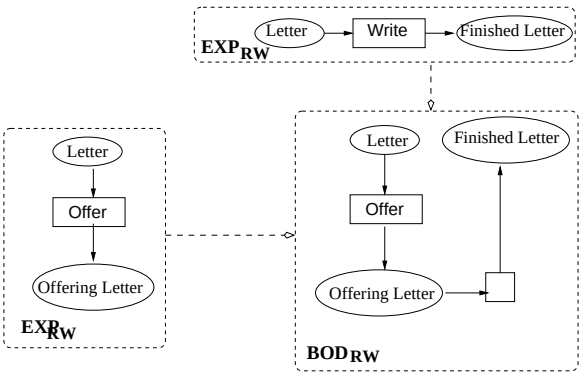


Figure 18: Module MOD_{RW}

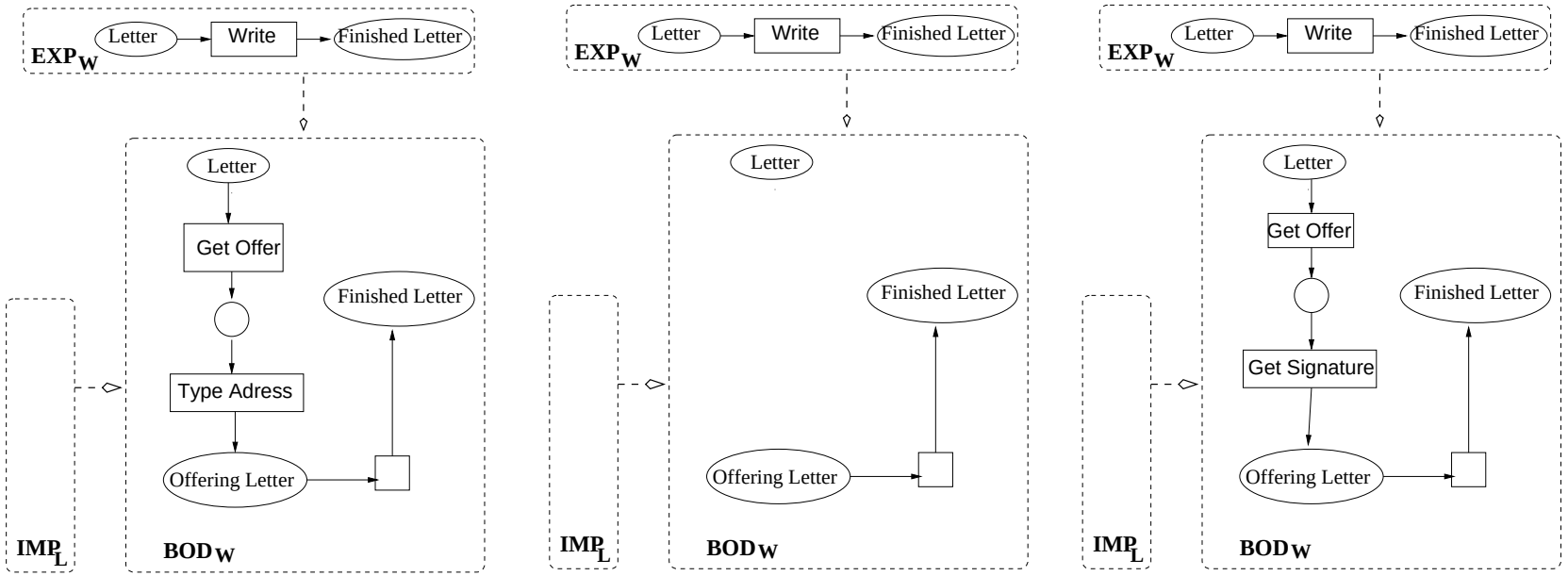


Figure 19: Rule $r_W \circ r_{UO}$

3 Transformations of Components

In this section we sketch how the notion of rule-based refinement can be carried over to components. Basically the idea is that the rule-based refinement of each part of the component, i.e. the export, the import, and the body, is a a rule-based refinement in the underlying specification category. Naturally, there the assumption need to be met, that is the specification category has to be weakly adhesive. As the definition of components involves different classes of morphisms these need to be taken into consideration. The difficulties to establish transformations of components is directly dependent from the class of refinement morphisms. So we first need to investigate the properties of component categories in regard of the involved morphism classes, as we have a more complicated situation as the class of refinement morphisms belongs to a supercategory of the morphisms used for the construction:

3.1 The Transformation Framework

In this section we present our work concerning the generic concept of components in a categorical frame.
In the transformation-based component framework a component consists of an import, an export and the body. The import states the prerequisites the component assumes. The body represents the internal functionality. The export gives an abstraction of the body that can be used by the environment. The abstract components conform with the basic concepts of components and component-based systems of *Continuous Software Engineering (CSE)* [46].
In [38] we present a categorical formalization of the concepts of the transformation-based approach using specific kinds of pushout properties. Hence they are an instantiation of the transformation-based approach. In order to achieve transformations of components we have to make the approach in [38] more concrete, by relating the morphism classes used for the transformation and the components.

Definition 3.1 (Categorical Assumptions)
The following assumptions

1. $\mathbf{Cat_p}$ category of specifications with plain morphisms
2. $\mathbf{Cat_r}$ category of specifications with refinement morphisms
3. the functor $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ is an inclusion in the sense that $Obj_{\mathbf{Cat_p}} = Obj_{\mathbf{Cat_r}}$.
4. $(\mathbf{Cat_p}, \mathcal{M}_{\mathbf{Cat_p}})$ is weakly adhesive HLR category
5. $\mathbf{Cat_r}$ has pushouts where at least one morphism is in $\mathsf{Inc}(Mor_{\mathbf{Cat_p}})$ and the pushout preserves $\mathsf{Inc}(Mor_{\mathbf{Cat_p}})$.
6. the inclusion functor $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ preserves pushouts where at least one morphism is in $\mathcal{M}_{\mathbf{Cat_p}}$.
7. the inclusion functor $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ preserves pullbacks where at least one morphism is in $\mathcal{M}_{\mathbf{Cat_p}}$.

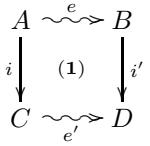
are supposed to hold throughout section 3. ◊

Definition 3.2 (Transformation Framework $\mathcal{T}$)
A transformation framework $\mathcal{T} = (\mathbf{Cat_r}, \mathcal{I}, \mathcal{E})$ consists of an arbitrary category and two classes of morphisms $\mathcal{I} = \mathsf{Inc}(Mor_{\mathbf{Cat_p}})$, called import morphisms and denoted by a simple arrow $\to$ and $\mathcal{E} = Mor_{\mathbf{Cat_r}}$, called export morphisms and denoted by an undulated arrow $\rightsquigarrow$. ◊

The following fact states that this transformation framework is in accordance with [38].

Fact 3.3 (Transformation Framework $\mathcal{T}$)
The following *extension conditions* hold:

1. $\mathcal{E}$-$\mathcal{I}$-Pushout Condition:
   Given the morphisms $A \overset{e}{\rightsquigarrow} B$ 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 $C \overset{e'}{\rightsquigarrow} D$ as depicted below.



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.

◊

Trivial, due to the assumption in definition 3.1 item 5.
Accordingly we have to require for a component that the import and export connection are of the right class of morphisms.

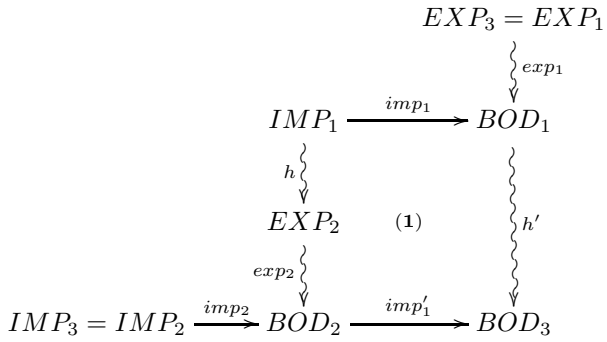
Definition 3.4 (Component [38])
A component $C = (IMP, EXP, BOD, imp, exp)$ is given by objects IMP, EXP, and BOD in **Cat** and by morphisms $exp : EXP \rightsquigarrow BOD$ and $imp : IMP \to BOD$, so that $exp \in \mathcal{E}$ and $imp \in \mathcal{I}$. ◊

Several different operations on components can be considered in our generic framework, eg. those that are given for algebraic module specification in [16]. For the sake of simplicity we subsequently consider merely one basic operation that allows composing components C_1 and C_2 hierarchically. It provides a connection, $h : IMP_1 \rightsquigarrow EXP_2$ from the import interface IMP_1 of C_1 to the export interface EXP_2 of C_2. Now we are able to define the composition $C_3 = C_1 \circ_h C_2$ as follows.

Definition 3.5 (Composition [38])
Given components $C_i = (IMP_i, EXP_i, BOD_i, imp_i, exp_i)$ for $i \in \{1, 2\}$ and a morphism $h : IMP_1 \to EXP_2$ in $\mathcal{E}$ the composition C_3 of C_1 and C_2 via h is defined by

$$C_3 = (IMP_3, EXP_3, BOD_3, imp_3, exp_3)$$

with $imp_3 = imp'_1 \circ imp_2$ and $exp_3 = h' \circ exp_1$ as depicted below, where (**1**) is pushout diagram :



The composition is denoted by $C_3 = C_1 \circ_h C_2$. ◊

Next we define the category of components **Comp**, where we use plain morphisms at the specification level for the definition of component morphisms, that have to be compatible with the corresponding import and export morphisms.

Definition 3.6 (Component Category)
Component morphisms are defined by $comp : comp_1 \to comp2$ with $comp = (comp_I, comp_B, comp_B)$ s.t.

$comp_I : IMP_1 \to IMP_2$
$comp_B : BOD_1 \to BOD_2$
$comp_E : EXP_1 \to EXP_2$

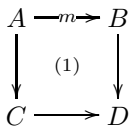
where $comp_I, comp_B, comp_B \in Mor(\mathbf{Cat_p})$

1. $comp_B \circ imp_1 = imp_2 \circ comp_I$
2. $comp_B \circ exp_1 = exp_2 \circ comp_E$

Components and component morphisms constitute **Comp** the category of components. ◊

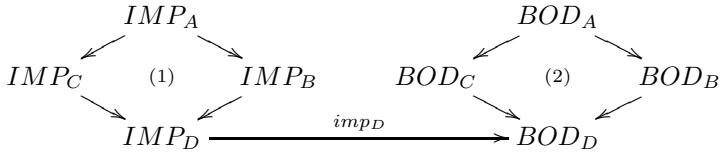
Subsequently we want to show that the category of components **Comp** is an adhesive HLR category, provided that the underlying category of specifications with plain morphisms is adhesive HLR category as well.
First we need the following facts.

Fact 3.7 (Pushouts with at least one $\mathcal{M}$-morphism in Comp)
Given the square (1) in **Comp** with $m \in \mathcal{M}$

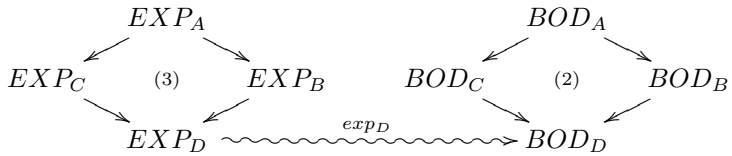


pushout $B \to D \leftarrow C$ is constructed componentwise in $\mathbf{Cat_p}$ and $\mathbf{Cat_r}$. ◊

Proof:
We have a component-wise construction, so we obtain the subsequent diagram

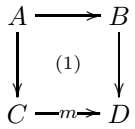


where (1) and (2) are pushouts in the category $\mathbf{Cat_p}$ and $imp_D : IMP_D \to BOD_D$ is the induced pushout morphism. Analogously, we obtain – as Inc preserves pushouts – from pushout (3) in $\mathbf{Cat_r}$ the induced morphism $exp_D : EXP_D \to BOD_D$:



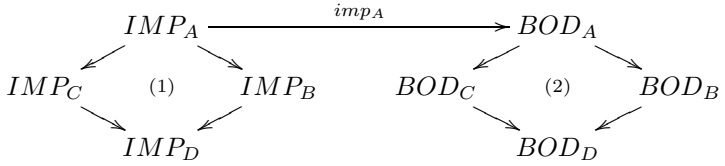
As IMP_D and BOD_D are pushouts in $\mathbf{Cat_p}$ and EXP_D is pushout in $\mathbf{Cat_r}$ commutativity and the universal property are inherited.
Hence we obtain $D = (EXP_D, IMP_D, BOD_D)$ as the pushout. √

Fact 3.8 (Pullbacks with at least one $\mathcal{M}$-morphism in Comp)
Given the square (1) in **Comp** with $m \in \mathcal{M}$

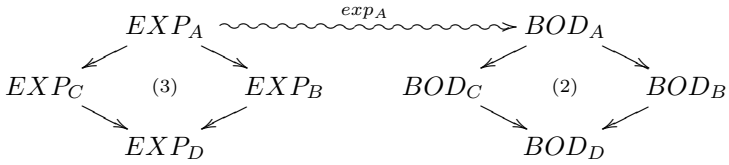


pullback $B \leftarrow A \rightarrow C$ is constructed componentwise in $\mathbf{Cat_p}$ and $\mathbf{Cat_r}$. $\Diamond$

Proof is analogously to proof of fact 3.7. **Proof:**
We have a component-wise construction, so we obtain the subsequent diagram



where (1) and (2) are pullbacks in the category $\mathbf{Cat_p}$ and $imp_A : IMP_A \rightarrow BOD_A$ is the induced pullback morphism. Analogously, we obtain – as Inc preserves pullbacks – from pullback (2) in $\mathbf{Cat_r}$ the induced morphism $exp_A : EXP_A \rightarrow BOD_A$:

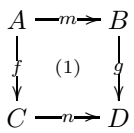


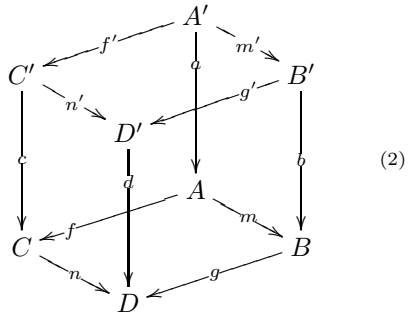
As IMP_D and BOD_D are pushouts in $\mathbf{Cat_p}$ and EXP_D is pushout in $\mathbf{Cat_r}$ commutativity and the universal property are inherited.
Hence we obtain $A = (EXP_A, IMP_A, BOD_A)$ as the pullback. $\surd$

Theorem 3.9 (Comp is weakly adhesive HLR category)
Given the assumptions in 3.1 then $(\mathbf{Comp}, \mathcal{M})$ is weakly adhesive HLR category with
$\mathcal{M} = \{comp = (comp_I, comp_B, comp_E) | comp_I, comp_B, comp_E \in \mathcal{M}_{\mathbf{Cat_p}}\}$ $\Diamond$

Proof:

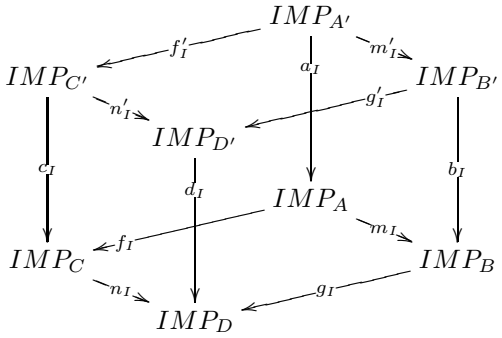
Given the pushout (1) and the commutative cube (2) in **Comp** we need to show that (2) is a VK square in **Comp**, i.e. the top is pushout $\Leftrightarrow$ the front faces are pullbacks.





(2)

We have for each part of the component an VK-diagram in $\mathbf{Cat_p}$, e.g. for the import part we have:



Part 1: (" $\Rightarrow$ ") the top is pushout, so we have to show that the components B' and C' are pullbacks $\in Comp$.
As we have the corresponding VK-diagram in $\mathbf{Cat_p}$, we have $IMP_{C'}$, $BOD_{C'}$ and $EXP_{C'}$ are pullbacks in $\mathbf{Cat_p}$. Hence, we obtain the component $IMP_{C'} \to BOD_{C'} \leftarrow EXP_{C'}$. Due to the uniqueness of the induced morphisms we have $C' = IMP_{C'} \to BOD_{C'} \leftarrow EXP_{C'}$ and hence is pullback.

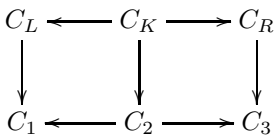
Analogously for B'.

Part 2: (" $\Leftarrow$ ") The front faces are pullbacks and we need to show that component D' is pushout in **Comp**.
As we have the corresponding VK-diagram in $\mathbf{Cat_p}$, we have $IMP_{D'}$, $BOD_{D'}$ and $EXP_{D'}$ are pushouts in $\mathbf{Cat_p}$. Hence, we obtain the component $IMP_{D'} \to BOD_{D'} \leftarrow EXP_{D'}$. Due to the uniqueness of the induced morphisms we have $D' = IMP_{D'} \to BOD_{D'} \leftarrow EXP_{D'}$ and hence is pushout.

$\surd$

So now we directly have the following nice concepts and results for the transformation of components (see [21]) as the category of components **Comp** with the distinguished class $\mathcal{M}$ is adhesive HLR category:

Rules and transformations These are the basic entities a rule is given by $r = (C_L \leftarrow C_K \to C_R)$, where C_L and C_R are the left and right hand side components and C_K is an intermediate component[1] of C_L and C_R. Given a rule $r = (C_L, C_K, C_R)$ and a context component C_2, we use morphisms $C_K \to C_L, K \to C_R$ and $C_K \to C_2$ to express a transformation as pushout constructions (1) and (2) leading to a double pushout as depicted below:



Below we give these notions more precisely.

Parallelism results (chapter 5 in [21]) The Church-Rosser Theorem states a local confluence in the sense of formal languages. The required condition of parallel independence means that the matches of both rules overlap only in parts that are not deleted. Sequential independence means that those parts created by the first transformation step are not deleted in the second. The

[1] In graph and net transformations K is often called the interface (of the rule) but in the context of components, this notion gets too confusing.

Parallelism Theorem states that sequential or parallel independent transformations can be carried out either in arbitrary sequential order or in parallel. In the context of step-by-step development these theorems are important as they provide conditions for the independent development of different parts or views of the system.

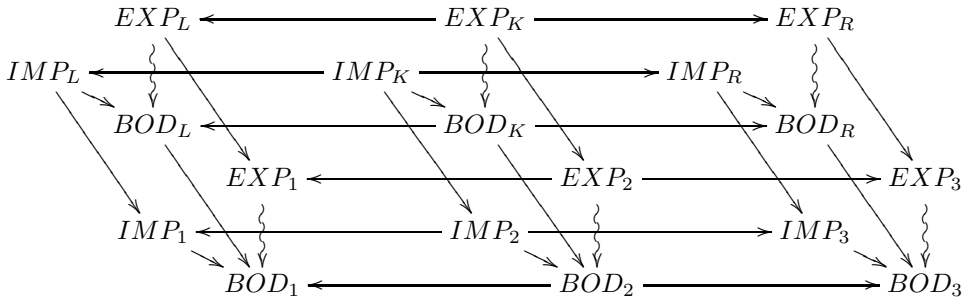
Concurrency and pair factorization (chapter 5 in [21]) The Concurrency Theorem handles general transformations, which may be non-sequentially independent. Roughly spoken, for a sequence there is a concurrent rule that allows the construction of a corresponding direct transformation.

Embedding and local confluence (chapter 6 in [21]) Further important results for transformation systems are the Embedding, Extension and the Local Confluence Theorems. The first two allow to embed transformations into larger contexts and with the third one we are able to show local confluence of transformation systems based on the confluence of critical pairs.

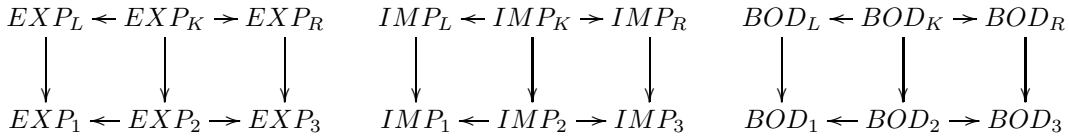
Up to now the instantiations of the HLR theory have been specification techniques as various types of graph transformations, Petri nets, algebraic specifications, etc. Now we instantiate with a gemeric component construction above these specification techniques, hence it is obvious, that new questions of compatibility arise. Namely, are the operations at the level of components compatible with the transformation concept. In the subsequent section we characterize the conditions under which transformations and hierarchical composition are compatible.

3.2 Rules, Transformations and Hierarchical Composition

Definition 3.10 (Rules and Transformations)
Given a rule in **Comp** with $r = (C_L \leftarrow C_K \rightarrow C_R)$ then the application of r yields the transformation $C_1 \stackrel{r}{\Longrightarrow} C_3$ given by the following diagram in $\mathbf{Cat_r}$:



with the following double pushouts in $\mathbf{Cat_p}$



$\Diamond$

Compatibility with Component Composition

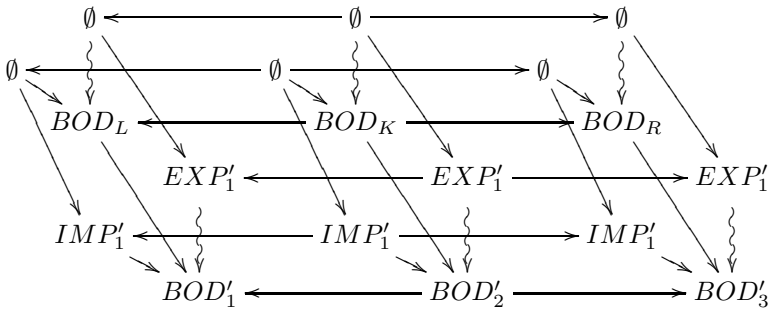
Obviously the tansformation of components in the context of hierarchical composition is quite complex and we can distinguish different cases, here we start with the more or less least complex and end with the most complex one. In principle, all the possible cases are subsumed by the most complex one.

- There is one rule changing only the body.
- There is one rule changing only the only the interface, that is not involved in the composition.
- There are two rules changing only the body or only the interfaces, that are not involved in the composition.
- There are two rules changing the involved interfaces as well.

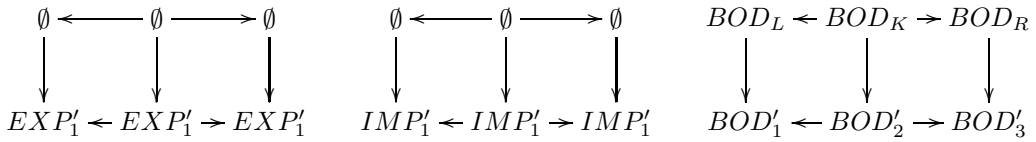
For the sake of simplicity and as we have the first case in our example in section 2.1 we treat the first case explicitly.

Fact 3.11 (Simple Compatibility)
Given the composition of $\widehat{C_1} = C_1 \circ_h C_1'$ and a rule $r = (C_L \leftarrow C_K \rightarrow C_R)$ where $IMP_L, IMP_K, IMP_R = \emptyset$ and $EXP_L, EXP_K, EXP_R = \emptyset$, where $\emptyset$ denotes the initial object, then we have
$C_1' \overset{r}{\Longrightarrow} C_3'$ iff $\widehat{C_1} \overset{r}{\Longrightarrow} \widehat{C_3} = C_1 \circ C_3'$ $\Diamond$

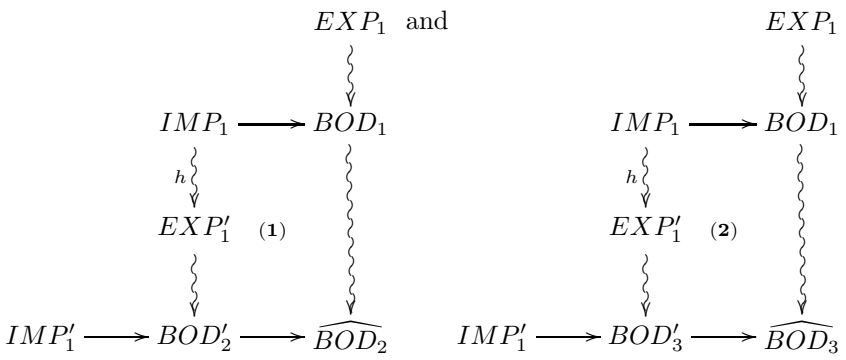
Proof:
Applying r to C_1' we have



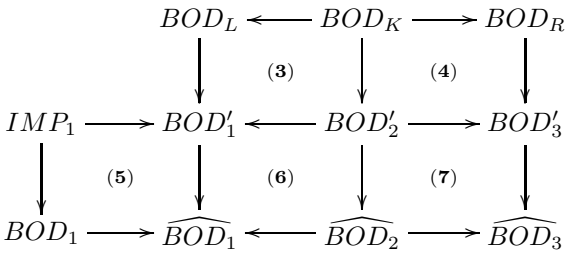
with the following double pushouts in $\mathbf{Cat_p}$



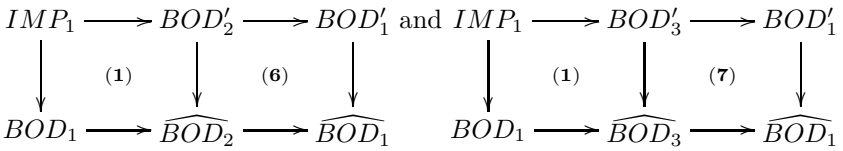
So we can construct $C_1 \circ_h C_2'$ and $C_1 \circ_h C_3'$ with the pushouts (**1**) and (**2**)



and it is easy to show that we obtain the same (up to isomorphism) result with $\widehat{C_1} \overset{r}{\Longrightarrow} \widehat{C_3}$:
We have the pushouts (**3**), (**4**) and (**5**)



Redrawing the above diagram we obtain the following pushout decomposition diagrams and hence have the pushouts (**6**) and (**7**) as well

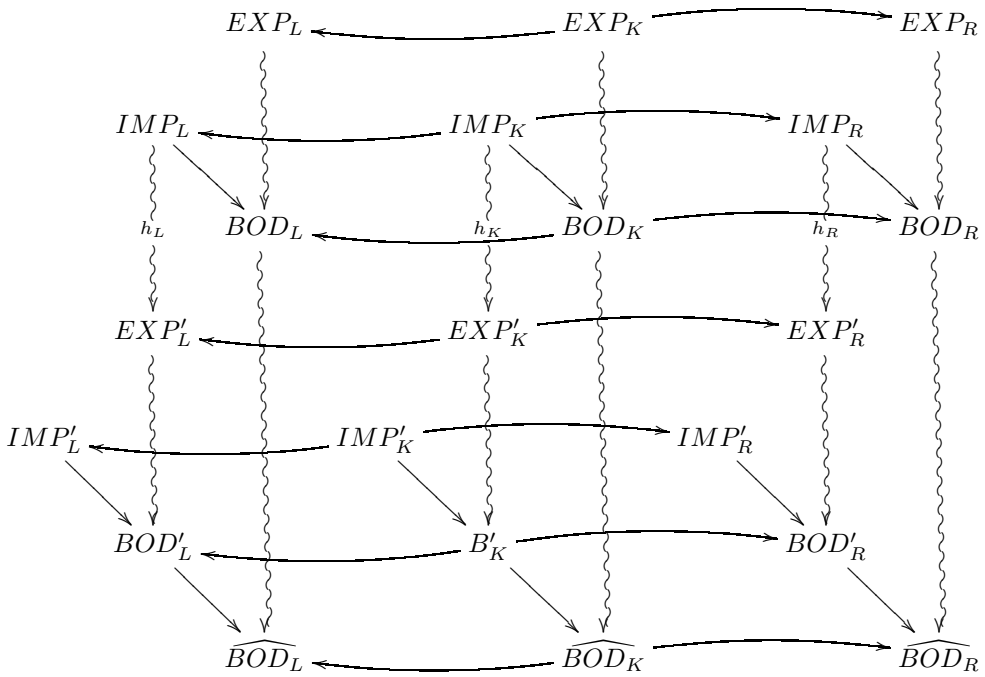


With pushout composition (**3** + **6**) and (**4** + **7**) we then have $\widehat{C_1} \overset{r}{\Longrightarrow} \widehat{C_3}$. $\surd$
If we have two rules changing the involved interfaces as well and want to apply them together to the composed component, we need to compose the rules as well. The following fact states the conditions, that ensure that the composition is well-defined.

Fact 3.12 (Componentwise composition of rules)
Given the rules $r = (C_L \leftarrow C_K \rightarrow C_R)$ and $r' = (C'_L \leftarrow C'_K \rightarrow C'_R)$ for $h_L : IMP_L \rightsquigarrow EXP'_L$, $h_K : IMP_K \rightsquigarrow EXP'_K$, and $h_R : IMP_R \rightsquigarrow EXP'_R$ so that

1. $IMP_K \rightarrow IMP_L \overset{h_L}{\rightsquigarrow} EXP'_L = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \rightarrow EXP'_L$
2. $IMP_K \rightarrow IMP_R \overset{h_R}{\rightsquigarrow} EXP'_R = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \rightarrow EXP'_R$

as depicted in the following diagram:



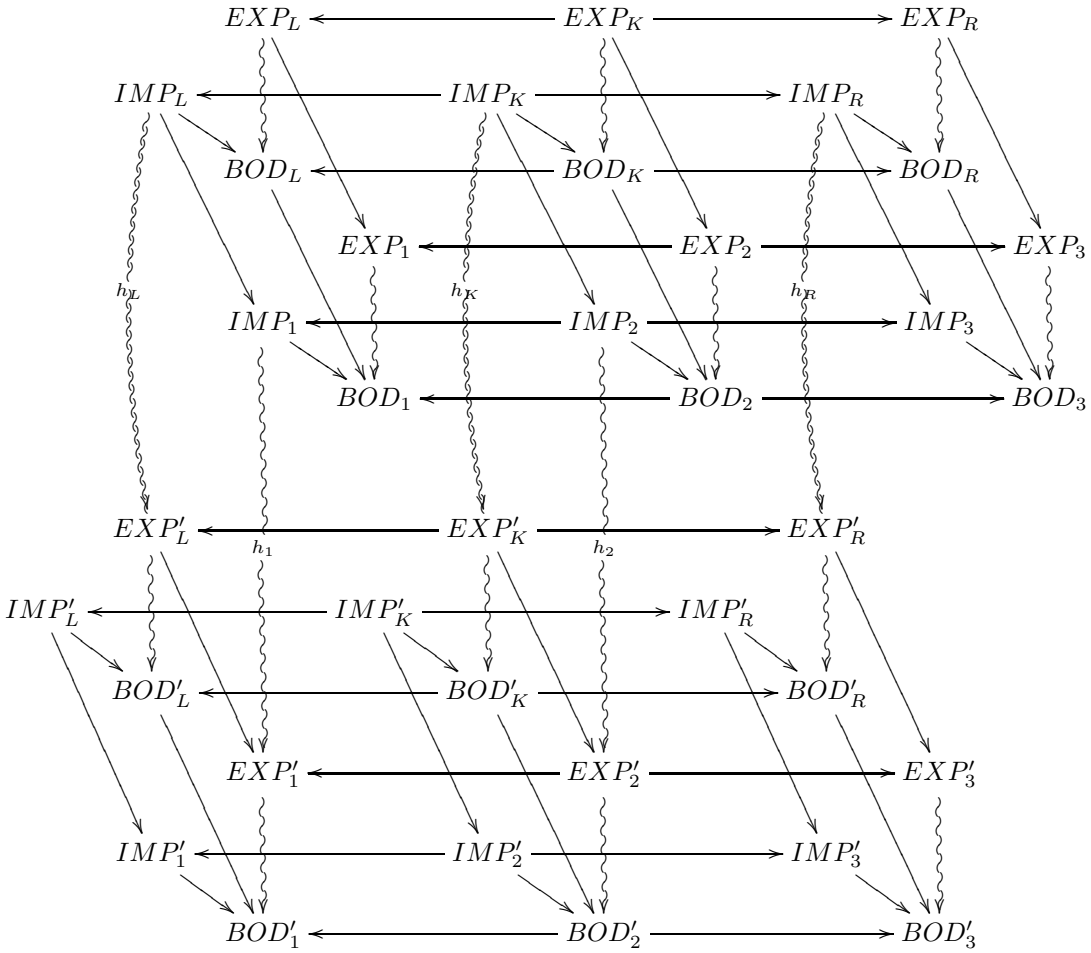
then we compose componentwise $\widehat{r} := r \circ r' = (\widehat{C_L} \leftarrow \widehat{C_K} \rightarrow \widehat{C_R})$ where
$\widehat{C_L} = (IMP'_L \rightarrow \widehat{BOD_L} \leftarrow EXP_L)$,
$\widehat{C_K} = (IMP'_K \rightarrow \widehat{BOD_K} \leftarrow EXP_K)$, and
$\widehat{C_R} = (IMP'_R \rightarrow \widehat{BOD_R} \leftarrow EXP_R)$ $\Diamond$

Due to assumption 4 in definition 3.1.

Definition 3.13 (Independence of transformation and composition)
Given the rules $r = (C_L \leftarrow C_K \rightarrow C_R)$ and $r' = (C'_L \leftarrow C'_K \rightarrow C'_R)$ with $r \circ r'$ for $h_L : IMP_L \rightsquigarrow EXP'_L$, $h_K : IMP_K \rightsquigarrow EXP'_K$, and $h_R : IMP_R \rightsquigarrow EXP'_R$ then the composition $C_1 \circ_{h_1} C'_1$ is independent from r and r' if there is $h_2 : IMP_2 \rightarrow EXP'_2$ s.t.

1. $IMP_2 \overset{h_2}{\rightsquigarrow} EXP'_2 \rightarrow EXP'_1 = IMP_2 \rightarrow IMP_1 \overset{h_1}{\rightsquigarrow} EXP'_1$
2. $IMP_L \rightarrow IMP_1 \overset{h_1}{\rightsquigarrow} EXP'_1 = IMP_L \overset{h_L}{\rightsquigarrow} EXP'_L \rightarrow EXP'_1$
3. $IMP_K \rightarrow IMP_2 \overset{h_2}{\rightsquigarrow} EXP'_2 = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \rightarrow EXP'_2$

as in the diagram below:



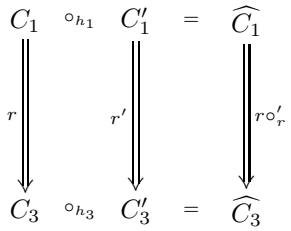
◊

If there are two components and two rules to be applied we can either compose two components and then use a composed rule to transform the component or we transform the two components independently and compose the results of the composition.
The following theorem states that under independence both ways result in the same component (up to isomorphism).

Theorem 3.14 (Composition Theorem)
Let the rules $r = (C_L \leftarrow C_K \rightarrow C_R)$ and $r' = (C'_L \leftarrow C'_K \rightarrow C'_R)$ with $r \circ_h r'$ for $h = (h_L, h_K, h_R)$ with $h_L : IMP_L \rightsquigarrow EXP'_L$, $h_K : IMP_K \rightsquigarrow EXP'_K$, and $h_R : IMP_R \rightsquigarrow EXP'_R$ be independ of the composition $C_1 \circ_{h_1} C'_1$, then we have

$C_1 \overset{r}{\Longrightarrow} C_3$ as well as $C'_1 \overset{r'}{\Longrightarrow} C'_3$ and $C_1 \circ_{h_1} C'_1 \overset{r \circ_h r'}{\Longrightarrow} C_3 \circ_{h_3} C'_3$

This can be illustrated in a diagram style by:

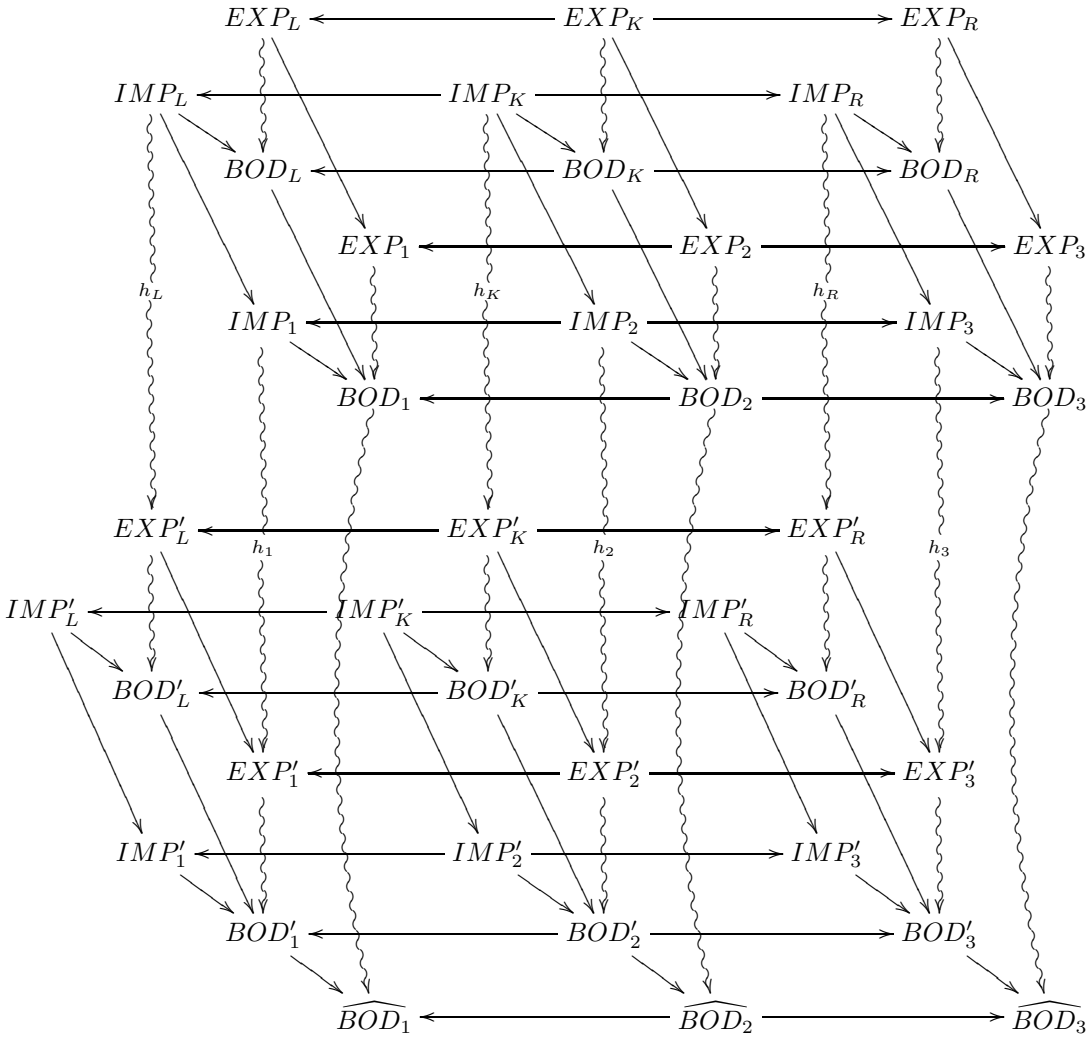


$\Diamond$

Proof:
First we construct the two derivations $C_1 \stackrel{r}{\Longrightarrow} C_3$ and $C'_1 \stackrel{r'}{\Longrightarrow} C'_3$ and then the composition $\widehat{C_3} = C_3 \circ_{h_3} C'_3$:
We obtain $h_3 : IMP_3 \rightsquigarrow EXP'_3$ as the induced pushout morphism in $\mathbf{Cat_r}$, because we have

$$
\begin{aligned}
&IMP_K \to IMP_R \overset{h_R}{\rightsquigarrow} EXP'_R \to EXP'_3 \\
&\quad = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \to EXP'_R \to EXP'_3 \text{ due to composition (item 2 of def. 3.17)} \\
&\quad = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \to EXP'_2 \to EXP'_3 \text{ due to rule } r', \text{ the second PO in export part} \\
&\quad = IMP_K \to IMP_2 \overset{h_2}{\rightsquigarrow} EXP'_2 \to EXP'_3 \quad \text{due to independence (item 3 of def. 3.13)}
\end{aligned}
$$

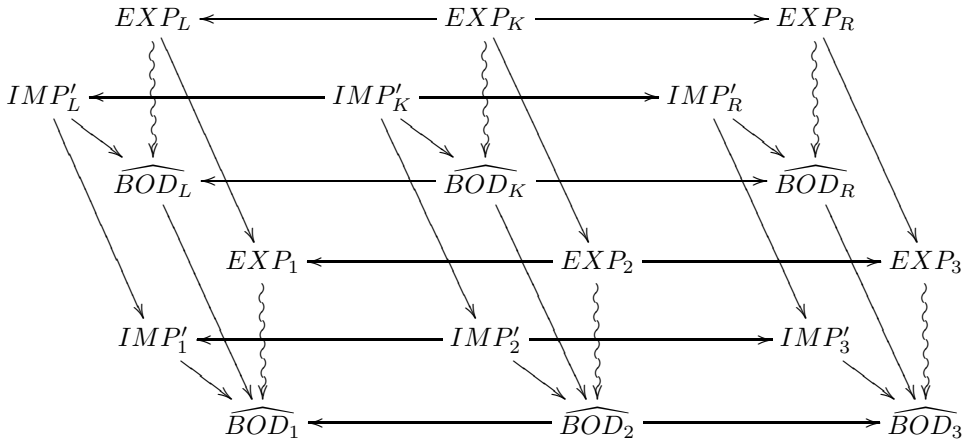
Now we can construct the pushout $BOD_3 \to \widehat{BOD_3} \leftarrow BOD'_3$ of $BOD_3 \leftarrow IMP_3 \overset{h_3}{\rightsquigarrow} EXP'_3 \to BOD'_3$, because of assumption 4 in definition 3.1 and obtain $\widehat{C_3} = C_3 \circ_{h_3} C'_3 = (IMP'_3, EXP_3, \widehat{BOD_3})$.
As h_1 and h_2 are given we obtain similarly $\widehat{C_1} = C_1 \circ_{h_1} C'_1 = (IMP'_1, EXP_1, \widehat{BOD_1})$ and $\widehat{C_2} = C_2 \circ_{h_2} C'_2 = (IMP'_2, EXP_2, \widehat{BOD_2})$.



Based on this construction it remains to show that (**A**) and (**B**) are pushouts in **Comp** :

$$\begin{array}{ccccc} \widehat{C_L} & \longleftarrow & \widehat{C_K} & \longrightarrow & \widehat{C_R} \\ \downarrow & & \downarrow & & \downarrow \\ & (\mathbf{A}) & & (\mathbf{B}) & \\ \downarrow & & \downarrow & & \downarrow \\ \widehat{C_1} & \longleftarrow & \widehat{C_2} & \longrightarrow & \widehat{C_3} \end{array}$$

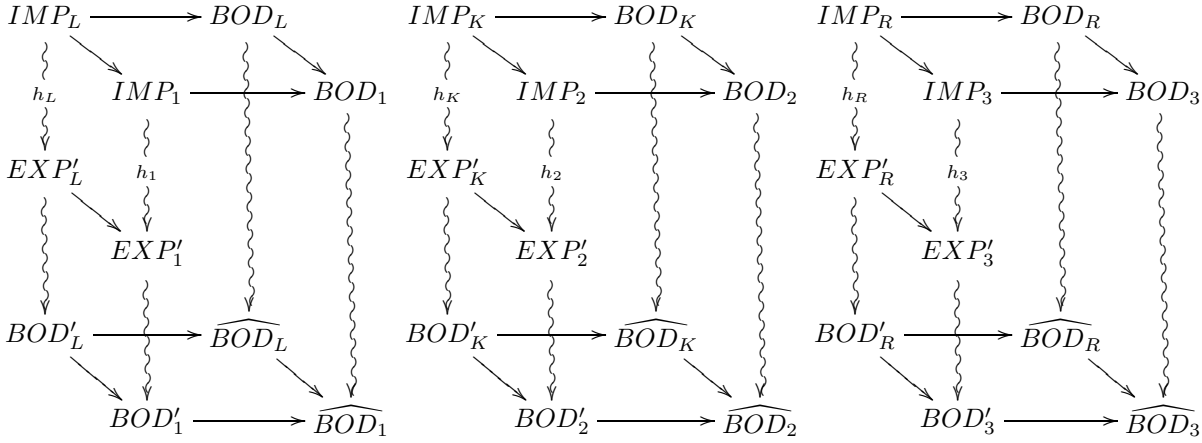
As we have the $C_1 \stackrel{r}{\Longrightarrow} C_3$ and $C'_1 \stackrel{r'}{\Longrightarrow} C'_3$ we already have the corresponding pushouts for the export and import part. In the category $\mathbf{Cat_r}$ we have to show the corresponding pushouts for the body part:



So, first we need to construct $\widehat{BOD_L} \to \widehat{BOD_1}$. We have :

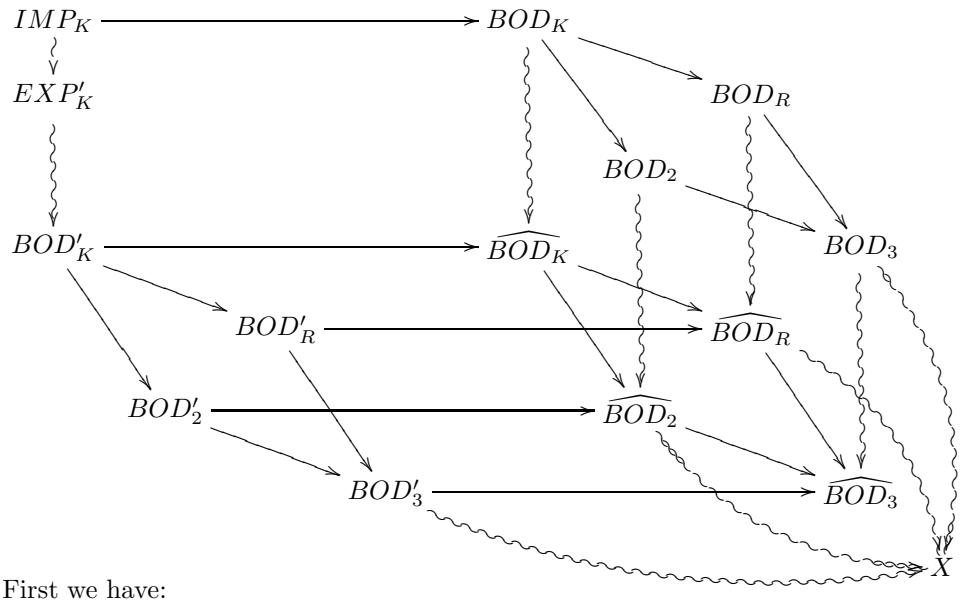
$$
\begin{aligned}
&IMP_L \to BOD_L \to BOD_1 \to \widehat{BOD_1} \\
&\quad = IMP_L \to IMP_1 \to BOD_1 \to \widehat{BOD_1} && \text{as } C_L \to C_1 \text{ is morphism in } \mathbf{Comp} \\
&\quad = IMP_L \to IMP_1 \overset{h_1}{\rightsquigarrow} EXP'_1 \to BOD'_1 \to \widehat{BOD_1} && \text{due to PO } \widehat{BOD_1} \\
&\quad = IMP_L \overset{h_L}{\rightsquigarrow} EXP'_L \to EXP'_1 \to BOD'_1 \to \widehat{BOD_1} && \text{due to independence (item 2 in def. 3.13)} \\
&\quad = IMP_L \overset{h_L}{\rightsquigarrow} EXP'_L \to BOD'_L \to BOD'_1 \to \widehat{BOD_1} && \text{as } C'_L \to C'_1 \text{ is morphism in } \mathbf{Comp}
\end{aligned}
$$

So we can construct $\widehat{BOD_L} \to \widehat{BOD_1}$ as the induced pushout morphisms and all squares in the left diagram below commute.
Analogously we obtain $\widehat{BOD_K} \to \widehat{BOD_2}$ and $\widehat{BOD_r} \to \widehat{BOD_3}$:



Next we show the body part of pushout (**B**), i.e. that $\widehat{BOD_R} \to \widehat{BOD_3} \leftarrow \widehat{BOD_2}$ is pushout of $\widehat{BOD_R} \leftarrow \widehat{BOD_K} \to \widehat{BOD_2}$:

First we show commutativity, i.e. $\widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} = \widehat{BOD_K} \to \widehat{BOD_R} \to \widehat{BOD_3}$ using the uniqueness of the induced pushout morphism of pushout $\widehat{BOD_K}$ in $\mathbf{Cat_r}$:



First we have:

$$
\begin{aligned}
&IMP_K \to BOD_K \to BOD_R \to BOD_3 \rightsquigarrow \widehat{BOD_3} \\
&\quad = IMP_K \to BOD_K \to BOD_2 \to BOD_3 \rightsquigarrow \widehat{BOD_3} \\
&\qquad\qquad \text{due to PO in } C_1 \overset{r}{\Longrightarrow} C_3 \\
&\quad = IMP_K \to BOD_K \to BOD_2 \rightsquigarrow \widehat{BOD_2} \to \widehat{BOD_3} \\
&\qquad\qquad \text{due to construction of } \widehat{BOD_2} \to \widehat{BOD_K} \\
&\quad = IMP_K \to BOD_K \rightsquigarrow \widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} \\
&\qquad\qquad \text{due to construction of } \widehat{BOD_K} \to \widehat{BOD_2} \\
&\quad = IMP_K \rightsquigarrow EXP'_K \rightsquigarrow BOD'_K \to \widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} \\
&\qquad\qquad \text{due to PO } \widehat{BOD_K} \\
&\quad = IMP_K \rightsquigarrow EXP'_K \rightsquigarrow BOD'_K \to BOD'_2 \to \widehat{BOD_2} \to \widehat{BOD_3} \\
&\qquad\qquad \text{due to construction of } \widehat{BOD_2} \to \widehat{BOD_K} \\
&\quad = IMP_K \rightsquigarrow EXP'_K \rightsquigarrow BOD'_K \to BOD'_2 \to BOD'_3 \to \widehat{BOD_3} \\
&\qquad\qquad \text{due to construction of } \widehat{BOD_2} \to \widehat{BOD_3}
\end{aligned}
$$

So there is unique $\widehat{BOD_K} \to \widehat{BOD_3}$. Secondly we have:

$$
\begin{aligned}
&BOD_K \rightsquigarrow \widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} \\
&\quad = BOD_K \to BOD_2 \rightsquigarrow \widehat{BOD_2} \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_K} \to \widehat{BOD_2} \\
&\quad = BOD_K \to BOD_2 \to BOD_3 \rightsquigarrow \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_2} \to \widehat{BOD_3} \\
&\quad = BOD_K \to BOD_R \to BOD_3 \rightsquigarrow \widehat{BOD_3} && \text{due to PO in } C_1 \overset{r}{\Longrightarrow} C_3 \\
&\quad = BOD_K \to BOD_R \rightsquigarrow \widehat{BOD_R} \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_R} \to \widehat{BOD_3} \\
&\quad = BOD_K \rightsquigarrow \widehat{BOD_K} \to \widehat{BOD_R} \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_K} \to \widehat{BOD_R}
\end{aligned}
$$

And third we have:

$$
\begin{aligned}
&BOD'_K \to \widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} \\
&\quad = BOD'_K \to BOD'_2 \to \widehat{BOD_2} \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_2} \to \widehat{BOD_K} \\
&\quad = BOD'_K \to BOD'_2 \to BOD'_3 \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_2} \to \widehat{BOD_3} \\
&\quad = BOD'_K \to BOD'_R \to BOD'_3 \to \widehat{BOD_3} && \text{due to PO in } C'_1 \overset{r'}{\Longrightarrow} C'_3 \\
&\quad = BOD'_K \to BOD'_R \to \widehat{BOD_R} \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_R} \to \widehat{BOD_3} \\
&\quad = BOD'_K \to \widehat{BOD_K} \to \widehat{BOD_R} \to \widehat{BOD_3} && \text{due to construction of } \widehat{BOD_K} \to \widehat{BOD_R}
\end{aligned}
$$

So both $\widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3}$ and $\widehat{BOD_K} \to \widehat{BOD_R} \to \widehat{BOD_3}$ are the unique pushout morphism and hence we conclude:

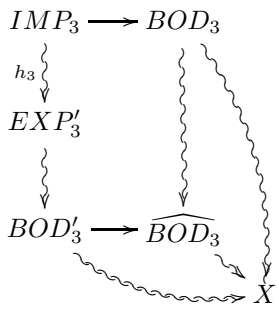
$$\widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} = \widehat{BOD_K} \to \widehat{BOD_R} \to \widehat{BOD_3}$$

Now we show the universal property of the square
$\widehat{BOD_K} \to \widehat{BOD_2} \to \widehat{BOD_3} = \widehat{BOD_K} \to \widehat{BOD_R} \to \widehat{BOD_3}$:

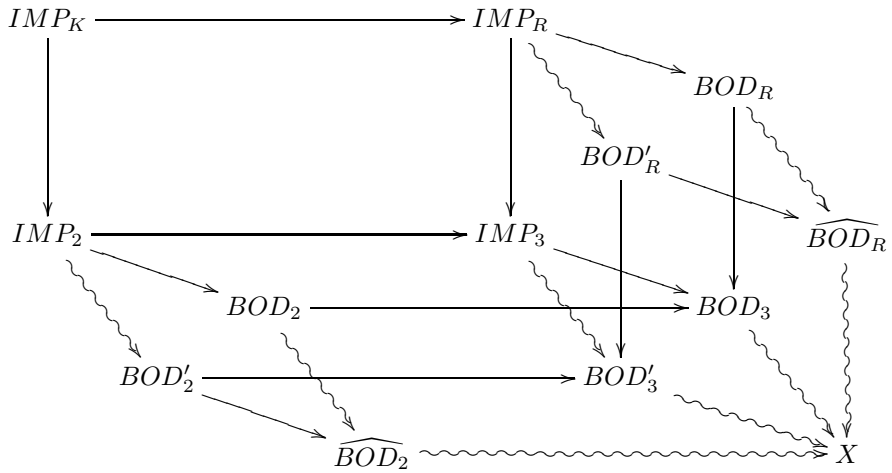
Given some $X \in \mathbf{Cat_r}$, s.t. $\widehat{BOD_K} \to \widehat{BOD_2} \rightsquigarrow X = \widehat{BOD_K} \to \widehat{BOD_R} \rightsquigarrow X$, then we can construct $BOD_3 \to X$ and $BOD'_3 \to X$ due to the universal property of the pushouts BOD_3 and BOD'_3 as we have:

$$\begin{aligned} &BOD_K \to BOD_R \to \widehat{BOD_R} \rightsquigarrow X \\ &\quad = BOD_K \rightsquigarrow \widehat{BOD_K} \to \widehat{BOD_R} \rightsquigarrow X \\ &\quad = BOD_K \rightsquigarrow \widehat{BOD_K} \to \widehat{BOD_2} \rightsquigarrow X \\ &\quad = BOD_K \to BOD_2 \rightsquigarrow \widehat{BOD_2} \rightsquigarrow X \end{aligned} \qquad \text{and} \qquad \begin{aligned} &BOD'_K \to BOD'_R \to \widehat{BOD_R} \rightsquigarrow X \\ &\quad = BOD'_K \to \widehat{BOD_K} \to \widehat{BOD_R} \rightsquigarrow X \\ &\quad = BOD'_K \to \widehat{BOD_K} \to \widehat{BOD_2} \rightsquigarrow X \\ &\quad = BOD'_K \to BOD'_2 \to \widehat{BOD_2} \rightsquigarrow X \end{aligned}$$

So we have



Now we use the pushout $\widehat{BOD_3}$ to obtain $\widehat{BOD_3} \rightsquigarrow X$. Therefore we have to show that
$IMP_3 \to BOD_3 \rightsquigarrow X$
$= IMP_3 \rightsquigarrow BOD'_3 \to \rightsquigarrow X$.
Again we use the uniqueness property of induced pushout morphisms.



First we have:

$$\begin{aligned} &IMP_K \to IMP_R \to BOD_R \rightsquigarrow \widehat{BOD_R} \rightsquigarrow X \\ &\quad = IMP_K \to IMP_R \rightsquigarrow BOD'_R \to \widehat{BOD_R} \rightsquigarrow X && \text{due to PO } \widehat{BOD_R} \\ &\quad = IMP_K \to IMP_R \rightsquigarrow BOD'_R \to BOD'_3 \rightsquigarrow X && \text{due to construction of } BOD'_3 \rightsquigarrow X \\ &\quad = IMP_K \to IMP_R \to IMP_3 \rightsquigarrow BOD'_3 \rightsquigarrow X && \text{due to composition of rules } r \circ r' \\ &\quad = IMP_K \to IMP_2 \to IMP_3 \rightsquigarrow BOD'_3 \rightsquigarrow X && \text{due to PO } IMP_3 \\ &\quad = IMP_K \to IMP_2 \rightsquigarrow BOD'_2 \to BOD'_3 \rightsquigarrow X && \text{due to construction of } h_3 \text{ and due to } r' \\ &\quad = IMP_K \to IMP_2 \rightsquigarrow BOD'_2 \to \widehat{BOD_2} \rightsquigarrow X && \text{due to construction of } BOD'_3 \rightsquigarrow X \end{aligned}$$

So there is a unique induced morphisms.
Second we have:

$$\begin{aligned} IMP_R \to IMP_3 \rightsquigarrow BOD'_3 \rightsquigarrow X \\ = IMP_R \rightsquigarrow BOD'_R \to BOD'_3 \rightsquigarrow X \\ = IMP_R \rightsquigarrow BOD'_R \to \widehat{BOD_R} \rightsquigarrow X \\ = IMP_R \rightsquigarrow BOD'_R \to BOD_3 \rightsquigarrow X \\ = IMP_R \to IMP_3 \to BOD_3 \rightsquigarrow X \end{aligned}$$

And thirdly we have:

$$\begin{aligned} IMP_2 \to IMP_3 \rightsquigarrow BOD'_3 \rightsquigarrow X \\ = IMP_2 \rightsquigarrow BOD'_2 \to BOD'_3 \rightsquigarrow X \\ = IMP_2 \rightsquigarrow BOD'_2 \to \widehat{BOD_2} \rightsquigarrow X \\ = IMP_2 \to BOD_2 \rightsquigarrow \widehat{BOD_2} \rightsquigarrow X \\ = IMP_2 \to BOD_2 \to BOD_3 \rightsquigarrow X \\ = IMP_2 \to IMP_3 \to BOD_3 \rightsquigarrow X \end{aligned}$$

So both morphisms $IMP_3 \to BOD_3 \rightsquigarrow X$ and $IMP_3 \rightsquigarrow BOD'_3 \rightsquigarrow X$ are unique induced morphisms, hence

$$IMP_3 \to BOD_3 \rightsquigarrow X = IMP_3 \rightsquigarrow BOD'_3 \rightsquigarrow X$$

And we obtain the unique $\widehat{BOD_3} \rightsquigarrow X$ with respect to $BOD'_3 \to \widehat{BOD_3} \rightsquigarrow X = BOD'_3 \rightsquigarrow X$ and $BOD_3 \rightsquigarrow \widehat{BOD_3} \rightsquigarrow X = BOD_3 \rightsquigarrow X$.
The universal property of $\widehat{BOD_R}$ leads to $\widehat{BOD_R} \to \widehat{BOD_3} \rightsquigarrow X = \widehat{BOD_R} \rightsquigarrow X$ and the universal property of $\widehat{BOD_2}$ leads to $\widehat{BOD_2} \to \widehat{BOD_3} \rightsquigarrow X = \widehat{BOD_2} \rightsquigarrow X$.
Uniqueness of $\widehat{BOD_3} \rightsquigarrow X$ is due to its construction.
Hence we have pushout (**B**) as required on page 26. Pushout (**A**) is constructed analogously. $\surd$

Compatibility with Component Composition for $\mathcal{Q}$-Transformations

Let us review the conditions for $\mathcal{Q}$-transformations:

Definition 3.15 ($\mathcal{Q}$-Transformations [32])
Let **QCat** be a category, so that **Cat** is a subcategory **Cat** $\subseteq$ **QCat** and $\mathcal{Q}$ a class of morphisms in **QCat**. We have the following $\mathcal{Q}$-conditions:

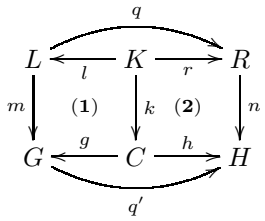
Preservation of Pushouts:
The inclusion functor I : **Cat** $\to$ **QCat** preserves pushouts.

Closedness: $\mathcal{Q}$ has to be closed under composition.

Inheritance of $\mathcal{Q}$-morphisms under Pushouts:
The class $\mathcal{Q}$ in **QCat** is closed under the construction of pushouts in **QCat** that is, if $C \xrightarrow{f'} D \xleftarrow{g'} B$ is a pushout of $B \xleftarrow{f} A \xrightarrow{g} C$ in **QCat**, then $f \in \mathcal{Q} \implies f' \in \mathcal{Q}$.

Inheritance of $\mathcal{Q}$-morphisms under Coproducts:
The class $\mathcal{Q}$ in **QCat** is closed under the construction of coproducts in **QCat** that is, for $A \xrightarrow{f} B$ and $A' \xrightarrow{f'} B'$ we have $f, f' \in \mathcal{Q} \implies f + f' \in \mathcal{Q}$ provided that the coproduct $A + A' \xrightarrow{f+f'} B + B'$ of f and f' exists in **QCat**.

The morphisms in $\mathcal{Q}$ are called $\mathcal{Q}$-morphisms, or refinement morphisms.
A $\mathcal{Q}$-rule (p, q) is given by a rule $p = L \xleftarrow{l} K \xrightarrow{r} R$ in **Cat** and a $\mathcal{Q}$-morphism $q : L \to R$, so that $K \xrightarrow{l} L \xrightarrow{q} R = K \xrightarrow{r} R$ in **QCat**.
Given a $\mathcal{Q}$-rule (p, q) and a transformation $G \overset{p}{\Longrightarrow} H$ in **Cat** defined by the pushouts **(1)** and **(2)**, there is a unique $q' \in \mathcal{Q}$, such that $q' \circ g = h$ and $q' \circ m = n \circ q$ in **QCat**. The transformation $(G \overset{p}{\Longrightarrow} H, q' : G \to H)$, or $G \overset{(p,q')}{\Longrightarrow} H$ for short, is called $\mathcal{Q}$-transformation



Moreover, $R \xrightarrow{n} H \xleftarrow{q'} G$ is pushout of $G \xleftarrow{m} L \xrightarrow{q} R$ in **QCat**. ◊

For the assumptions in 3.1 we need to clarify the relation of the morphism calss $\mathcal{Q}$ to the others.

Definition 3.16 ($\mathcal{Q}$-Rules for components)
Given the categories $\mathbf{Cat_p} \subseteq \mathbf{Cat_r} \subseteq \mathbf{Cat_q}$, so that assumptions 3.1 and the $\mathcal{Q}$-conditions in 3.15 hold for $\mathbf{Cat} = \mathbf{Cat_p}$ and $\mathbf{QCat} = \mathbf{Cat_q}$, then

a $\mathcal{Q}$-rule (r, p) is given by a rule $r = (L \leftarrow K \rightarrow R$ in $\mathbf{Cat_p}$ and a $\mathcal{Q}$-morphism $p : L \to R$, so that $K \to L \xrightarrow{q} R = K \to R$ in $\mathbf{Cat_q}$. ◊

So we directly have the extension of rules and transformations with additional morphisms. So, each part of the component, import, export and body can be refined by applying rules, that preserve or reflect properties. So, proof rules are obtained, that allow the stepwise preservation of desireable component properties. The proof rule states that given certain assumptions (stated above the line) the property (stated under the line) holds.
For the property preserving transformations $C_1 \Longrightarrow C_2$ and a property preserving rule (r, f) we have the following proof rule:

$$\frac{(r, f) \text{ is a property preserving rule; } C_1 \text{ satisfies the corresponding property}}{C_2 \text{ satisfies the property too}}$$

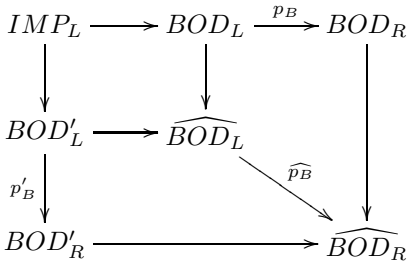
Moreover, the Composition Theorem 3.14 can be achieved for $\mathcal{Q}$-transfomations as well.

Fact 3.17 (Componentwise composition of $\mathcal{Q}$-rules)
Given the $\mathcal{Q}$-rules $r = (C_L \leftarrow C_K \rightarrow C_R, p : C_L \to C_R)$ with $p = (p_I, p_E, p_B)$ and $r' = (C'_L \leftarrow C'_K \rightarrow C'_R, p' : C'_L \to C'_R)$ with $p' = (p'_I, p'_E, p'_B)$ for $h_L : IMP_L \rightsquigarrow EXP'_L$, $h_K : IMP_K \rightsquigarrow EXP'_K$, and $h_R : IMP_R \rightsquigarrow EXP'_R$ so that

1. $IMP_K \to IMP_L \overset{h_L}{\rightsquigarrow} EXP'_L = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \to EXP'_L$
2. $IMP_K \to IMP_R \overset{h_R}{\rightsquigarrow} EXP'_R = IMP_K \overset{h_K}{\rightsquigarrow} EXP'_K \to EXP'_R$

then we compose componentwise $\widehat{r} := (r \circ_h r', \widehat{p}) = (\widehat{C_L} \leftarrow \widehat{C_K} \rightarrow \widehat{C_R}, \widehat{p} : \widehat{C_L} \to \widehat{C_R})$ with $\widehat{p} = (p'_I, p_E, \widehat{p_B})$ ◊

Proof:
We merely need to construct $\widehat{p_B} : \widehat{BOD_L} \to \widehat{BOD_R}$ as the induced pushout morphism in the diagram below.

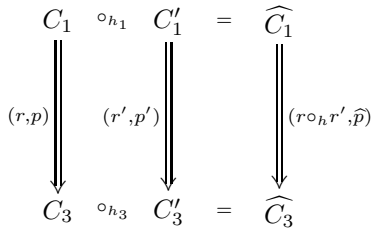


√

Independence is defined as in definition 3.13.

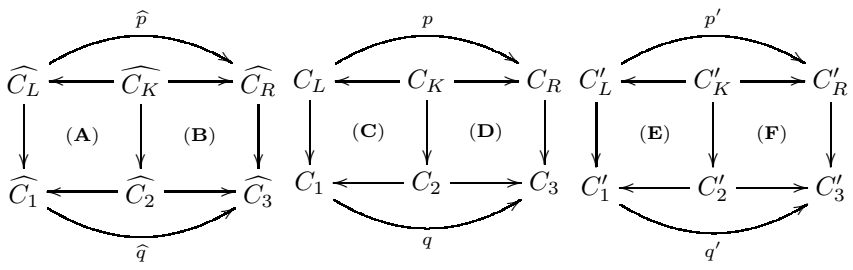
Theorem 3.18 (Composition Theorem for $\mathcal{Q}$-Transformations)
Let $\mathcal{Q}$-rules $r = (C_L \leftarrow C_K \rightarrow C_R, p : C_L \rightarrow C_R)$ with $p = (p_I, p_E, p_B)$ and $r' = (C'_L \leftarrow C'_K \rightarrow C'_R, p' : C'_L \rightarrow C'_R)$ with $p' = (p'_I, p'_E, p'_B)$ with $r \circ_h r'$ with $\widehat{p} = (p'_I, p_E, \widehat{p_B}$ for $h = (h_L, h_K, h_R)$ with $h_L : IMP_L \rightsquigarrow EXP'_L$, $h_K : IMP_K \rightsquigarrow EXP'_K$, and $h_R : IMP_R \rightsquigarrow EXP'_R$ be independ of the composition $C_1 \circ_{h_1} C'_1$, then we have

$C_1 \overset{(r,p)}{\Longrightarrow} C_3$ as well as $C'_1 \overset{(r',p')}{\Longrightarrow} C'_3$ and $C_1 \circ C'_1 \overset{(r \circ_h r', \widehat{p})}{\Longrightarrow} C_3 \circ_{h_3} C'_3$
This can be illustrated in a diagram style by:



◊

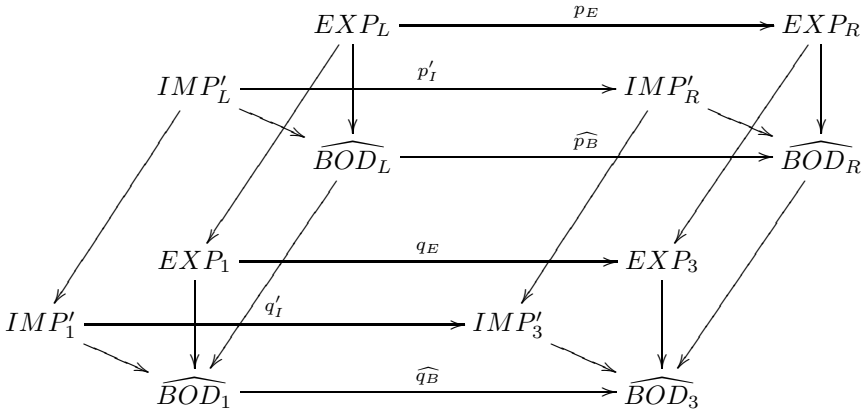
Proof:
Analogously to the proof of the Composition Theorem 3.14 we have for the composed as well as for the simple $\mathcal{Q}$-transformations the following double-pushouts



with

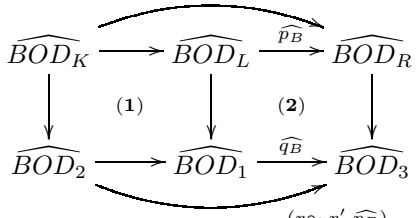
$\widehat{p} = (p'_I, p_E, \widehat{p_B})$ $\quad p = (p_I, p_E, p_B)$ $\quad p' = (p'_I, p'_E, p'_B)$
$\widehat{q} = (q'_I, q_E, \widehat{q_B})$ $\quad q = (q_I, q_E, q_B)$ $\quad q' = (q'_I, q'_E, q'_B)$

where $\widehat{q_B}$ is given as the induced pushout morphism of $BOD'_1 \to \widehat{BOD_1} \leftarrow BOD_1$ of $BOD'_1 \leftarrow IMP_1 \to BOD_1$.
And it remains to show that $(\mathbf{A} + \mathbf{B})$ is a pushout as well. At the level of specifications $(\mathbf{A} + \mathbf{B})$ is given in the category $\mathbf{Cat_r}$ by:



where we have the following pushouts due to $C_1 \overset{(r,p)}{\Longrightarrow} C_3$ and $C'_1 \overset{(r',p')}{\Longrightarrow} C'_3$

Due to pushout decomposition we obtain that $(\mathbf{2})$ is pushout as well.



So, $(\mathbf{A} + \mathbf{B})$ is a pushout and we have $C_1 \circ C'_1 \overset{(r \circ_h r', \widehat{p_B})}{\Longrightarrow} C_3 \circ_{h_3} C'_3$ is given by the double pushout $(\mathbf{A}, \mathbf{B})$. $\surd$

4 Transformations of Automata Components

4.1 Automata Components

Here we use a reduced version of the automata in [19]. We skip the output function as this is not needed for the automata we want to relate to in section 6. Basically, an automaton $A = (I, S, \delta : I \times S \rightharpoonup S)$ consists of the input alphabet I, the set of states S and the partial function $\delta : I \times S \rightharpoonup S$: For an element of the input alphabet $i \in I$ and a state $s \in S$ the transition function $\delta(i, s) = \perp$ is undefined or $\delta(i, s) = s'$ yields the followers state s'. A plain morphisms $f = (f_I, f_S) : A_1 \to A_2$ from Automaton A_1 to A_2 maps the alphabet to the alphabet $f_I : I_1 \to I_2$ and the set of states to the set of states $f_S : S_1 \to S_2$.

Definition 4.1 (Automata)
The category $\mathbf{Aut_p}$ is given as the comma category $\mathbf{Aut_p} := (\mathsf{Prod} \downarrow \mathsf{ID})$, where $\mathsf{Prod} : \mathbf{parSet} \times \mathbf{parSet} \to \mathbf{parSet}$ is the product functor and $\mathsf{ID} : \mathbf{parSet} \to \mathbf{parSet}$ is the identity on sets with partial functions. $\Diamond$

This definition states the category of automata with plain morphisms as discussed, nevertheless we use the comma category construction as it yields directly many important results.

Fact 4.2 (Properties of $\mathbf{Aut_p}$)
$\mathbf{Aut_p}$ has pushouts and pullbacks, as **parSet** has both, and Prod preserves pushouts and ID preserves pullbacks.
Moreover, $(\mathbf{Aut_p}, \mathcal{M})$ with $\mathcal{M}$ the class of injective morphisms is an adhesive HLR-category, as it is a comma category over **parSet**. ◊

See facts A.3 and B.2 and because of the construction of adhesive HLR-categories [21].
Note, that in the example we have starts symbols as well as terminal symbols. As these do not affect the morphisms (i.e. they need not to be mapped onto each other), we simply ignore them in the formal description.
Next we need to define refinement morphisms that allow the refinement of a state by automaton.

Definition 4.3 (State Refinement for Automata)
Given two automata $A_i = (I_i, S_i, \delta_i : I_i \times S_i \to S_i)$ for $i = 1, 2$, then $f : A_1 \to A_2$ is given by $f_I : I_1 \to I_2$ and $\mathfrak{f} : S_1 \to \mathcal{P}(A_2)$, such that for $\mathfrak{f}(s_1) = A_2' = (I_2', S_2', \delta_2) \subseteq A_2$ we have for all $i \in I_1$ with $\delta_1(i, s_1) = s_1'$:

$$\exists! s_2 \in S_2' : \ \delta_2(f_I(i), s_2) \in \mathfrak{f}(s_1')$$

The category $\mathbf{Aut_r}$ is given by automata and refinement morphisms. ◊

Figure 20 illustrates a state refinement as used in the example in section 2.1. The states $s1, s4, s5$ and $s7$ of the automaton EXP_ATM1 are mapped to the corresponding states (more precisely to the subautomaton consisting only of that state)in the automaton BOD_ATM1. The state $s2$ is mapped to the sub-auotmaton of BOD_ATM1 that consists of the states $s2.1, s2.2$ and $s2.3$ and the transitins in-between. Similarly the states $s3$ and $s6$ are mapped to the subautomata that are enclosed by the light green line.

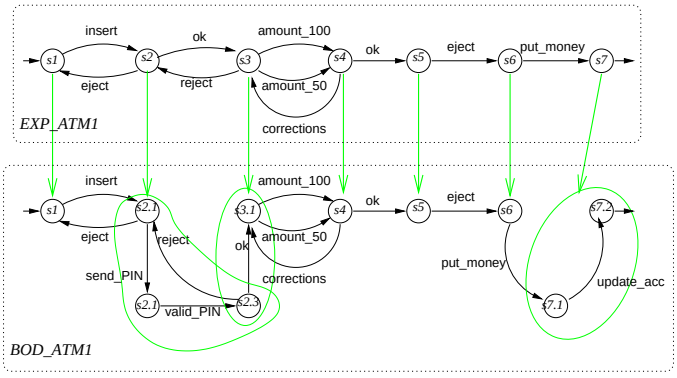
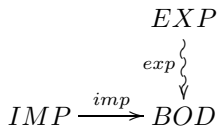


Figure 20: Refinement morphism from $EXP\_ATM1 \rightsquigarrow BOD\_ATM1$ in $ATM1$

Note that this definition is a syntactic notion of refinement. The two conditions merely ensure that the mapping does not delete transitions. Here we require the preservation of connectedness, but allow for instance that the target automata of a refined state may have unreachable states. For example in section 2.1 in transformation $ATM2 \overset{atm\_r1}{\Longrightarrow} ATM4$ in figure 5 we have as an intermediate situation component $ATM3$, where the mapping $EXP\_ATM3 \rightsquigarrow BOD\_ATM3$ yields an unreachable state due to the deletion of a transition. Nevertheless the right hand side of the rule closes this gap again. So, more complex refinement notions need a restriction the target automata of a refined state. They can be defined explicitly, but should not be integrated into the morphisms, because not in all situations the more complex refinement is desirable. The above given syntactic refinement can be easily extended to a proper refinement, e.g. in the following way: If we only use in the body an alphabet that is the codomain of the alphabet of the export and substitute all other elements by the silent tansition τ, then the language of the body automaton has to be equal to the language of the export automaton.

Definition 4.4 (Automata Component)
An automata component $AC = (IMP, EXP, BOD)$ consists of three automata, namely the import automaton IMP, the export automaton EXP and the body automaton BOD. Two morphisms $imp : IMP \to BOD$ and $exp : EXP \rightsquigarrow BOD$ connect the interfaces to the body with $imp \in \mathcal{I}$ and $exp \in \mathcal{E}$.



◊

Examples can be found in section 2.1.

4.2 Transformations for Automata Components

Fact 4.5 (Plain and Refinement Morphisms for Automata)
Given a plain morphism $f : A_1 \to A_2$ then there is the refinement morphism with $\mathfrak{f} : S_1 \to \mathcal{P}(A_2)$ where $\mathfrak{f}(s) = (\emptyset, \{f_s(s)\}, \emptyset)$ is the automaton that consists only of one state.
Hence, we have $Inc : \mathbf{Aut_p} \to \mathbf{Aut_r}$ ◊

Fact 4.6 (Automata component categories satisfy the assumptions 3.1)
The following holds:

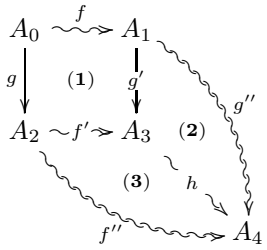
1. $\mathbf{Aut_p}$ category of specifications with plain morphisms
2. $\mathbf{Aut_r}$ category of specifications with refinement morphisms
3. we have the inclusion functor $\mathsf{Inc} : \mathbf{Aut_p} \to \mathbf{Aut_r}$ so that $Obj_{\mathbf{Aut_p}} = Obj_{\mathbf{Aut_r}}$.
4. $\mathbf{Aut_r}$ has pushouts where at least one morphism is in $\mathsf{Inc}(Mor_{\mathbf{Aut_p}})$ and the pushout preserves $\mathsf{Inc}(Mor_{\mathbf{Aut_p}})$.
5. $(\mathbf{Aut_p}, \mathcal{M})$ is weakly adhesive HLR category with $\mathcal{M}$ the class of injective morphisms
6. the inclusion functor $\mathsf{Inc} : \mathbf{Aut_p} \to \mathbf{Aut_r}$ preserves pushouts where at least one morphism is in $\mathcal{M}$.
7. the inclusion functor $\mathsf{Inc} : \mathbf{Aut_p} \to \mathbf{Aut_r}$ preserves pullbacks where at least one morphism is in $\mathcal{M}$.

◊

Proof:
Item 1-3 we have given above.
Let $A_i = (I_i, S_i, \delta_i : I_i \times S_i \rightharpoonup S_i)$ for $i = 1, 2, 3, ...$:

Proof of item 4

Given the span $A_2 \xleftarrow{g} A_0 \overset{f}{\rightsquigarrow} A_1$ then we can construct A_3 with $I_3 = I_2 +_{I_0} I_1$ the pushout of the alphabets, and



$S_3 = S_{1|\bar{E}} \cup S_2 \setminus g_S(S_0)$ with $\bar{E}$ the equivalence closure of $E = \{(s_1, s_1') | s_1 \in \mathfrak{f}(s_0), s_1' \in \mathfrak{f}(s_0') \text{ for } g(s_0) = g(s_0')\}$ and $\delta_3 : I_3 \times S_3 \rightharpoonup S_3$ with

$$\delta_3(i,s) = \begin{cases} [\delta_1(i_1, s_1)]\bar{E} & \text{for } i = g_I'(i_1), s = [s_1] \\ \delta_2(i_2, s_2) & \text{for } i = f_I'(i_2), s = s_2, \delta_2(i_2, s_2) \notin g_S(S_0) \\ [\delta_1(i_1, s_1)]\bar{E} & \text{for } i = f_I'(i_2), s = s_2, \delta_2(i_2, s_2) = g_S(s_0') \\ & \text{and there is unique } s_1 \in S_1 : \delta_1(f_I(i_0), s_1) \in \mathfrak{f}(s_0) \end{cases}$$

δ_3 is well-defined, as we have the following cases for $\delta_3(i, s)$, if

1. $i = g_I'(i_1), s = [s_1]$, due to the equivalence relation $\bar{E}$ and the corresponding equivalence classes and the PO I_3 this case is well-defined.
2. $i = g_I'(i_1), s = s_2]$ in this case we have due to PO I_3 that $i = f_I'(i_2)$, so it is well-defined due to item 3.
3. $i = f_I'(i_2), s = [s_1]$ this case we have due to PO I_3 that $i = g_I'(i_1)$, so it is well-defined due to item 1.
4. $i = f_I'(i_2), s = s_2$ there are again two possiblities: $\delta_2(i_2, s_2) \notin g_S(S_0)$ or $\delta_2(i_2, s_2) = g_S(s_0)$ for some s_0
   For $\delta_2(i_2, s_2) \notin g_S(S_0)$ we have that $\delta_2(i_2, s_2) \in S_3$ and hence it is well-defined.
   For $\delta_2(i_2, s_2) = g_S(s_0)$ we have due to the condition in 4.3 a unique $s_1 \in \mathfrak{f}(s_0)$ with $\delta_1(f_I(i_0), s_1) \in \mathfrak{f}(s_0')$ since there has to be $g_S(\delta_0(i_0, s_0)) = s_2$. Hence $\delta_1(i_1, s_1)$ is given and hence the corresponding equivalence class.

(**1**) commutes, due to the construction of S_3 and as I_3 is PO with $g_S' = [\,]_{\bar{E}}$ and $\mathfrak{f}'(s_2) = g'(\mathfrak{f}(s_0))$ for $g_S(s_0) = s_2$ and $\mathfrak{f}'(s_2) = (\emptyset, \{s_2\}, \emptyset) \subseteq A_3$ else.

Given some automata in $\mathbf{Aut_r}$ s.t. $g'' \circ f = f'' \circ g$, the $h : A_3 \to A_4$ is given by $h = (h_I, \mathfrak{h})$, where h_I is induced by PO I_3. $\mathfrak{h}$ is defined by

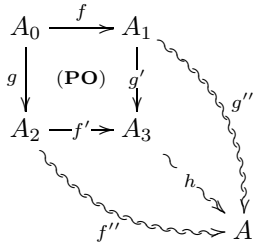
$$\mathfrak{h}(s) = \begin{cases} \bigcup_{x \in [s_1]} \mathfrak{g}''(x) & s = [s_1], s_1 \in S_1 \\ \mathfrak{f}''(s_2) & s = s_2 \in S_2 \setminus g_S(S_0) \\ \bigcup_{x \in \mathfrak{f}(s_0)} \mathfrak{g}''(x) & s = s_2 = g_S(s_0) \end{cases}$$

$\mathfrak{h}$ is well-defined and unique wrt. (**2**) and (**3**) due to construction.

Proof of item 5 due to fact 4.2.

Proof of item 6

Given the following pushout (**PO**) in $\mathbf{Aut_p}$ with $f'' : A_2 \rightsquigarrow A$ and $g'' : A_1 \rightsquigarrow A$ s.t $f'' \circ g = g'' \circ f$ in $\mathbf{Aut_r}$



As the square commutes in $\mathbf{Aut_r}$, it remains to show the universal property: Then we define $h = (h_I, \mathfrak{h}) : A_3 \to A$ with h_I as induced by the pushout I_3 in **Set**.
$\mathfrak{h} : S_3 \rightsquigarrow \mathcal{P}(A)$ is given by

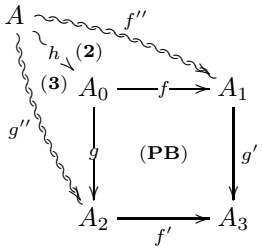
$$\mathfrak{h}(s) = \begin{cases} \mathfrak{g}''(s_1) & ; \text{if } s = g'_S(s_1) \text{ for some } s_1 \in S_1 \\ \mathfrak{f}''(s_2) & ; \text{if } s = f'_S(s_2) \text{ for some } s_2 \in S_2 \end{cases}$$

$\mathfrak{h}$ is well-defined because due to the pushout properties of S_3 in **Set** we have that f' and g' are jointly surjective and for $g'_S(s_1) = s = f'_S(s_2)$ we have some $s_0 \in S_0$ with $f(s_0) = s_1$ and $g(s_0) = s_2$ and hence $\mathfrak{g}'(s_1) = \mathfrak{g}' \circ f(s_0) = \mathfrak{f}' \circ g(s_0) = \mathfrak{f}'(s_2)$.

h is unique as h_I is unique wrt. g'_I and f'_I, and $\mathfrak{h}$ is unique wrt. $\mathfrak{g}'$ and $\mathfrak{f}'$ due to construction.

proof of item 7

Given the pullback (**PB**) in $\mathbf{Aut_p}$ with injective f', we show that (**PB**) is pullback in $\mathbf{Aut_r}$ as well.



As (**PB**) commutes in $\mathbf{Aut_r}$ it remains to show the universal property.

Given $f'' : A \rightsquigarrow A_1$ and $g'' : A \rightsquigarrow A_2$ so that:
$f' \circ g'' = g' \circ f'' \qquad (*)$
we define $h(h_I, \mathfrak{h}) : A \to A_0$ with h_I as induced by the pullback I_1 in **Set**.
$\mathfrak{h} : S \to \mathcal{P}(A_0)$ is defined for some $s \in S$ where
$\mathfrak{f}''(s) = A'_1 = (I'_1, S'_1, \delta_1 : I'_1 \times S'_1 \to S'_1)$ by
$\mathfrak{h}(s) = A'_0 = (I'_0, S'_0, \delta_0 : I'_0 \times S'_0 \to S'_0)$ with
$I'_0 = \{i \in I_0 \mid f_I(i) \in I'_1\}$ and $S'_0 = \{s \in S_0 \mid ; f_S(s) \in S'_0\}$.

$h : A \to A_0$ is well-defined:
Given some $i \in I$ then there is some $s_1 \in S_1$ with $\delta_1(f''_I(i), s_1) \in \mathfrak{f}''(\delta(i, s_1)$ as f'' is well-defined.

So, there is some $s_0 \in S'_0$ with $f_S(s_0) = s_1$ and we have
$f_S \circ \delta_0(h_I(i), s_0)) = \delta_1(f''_I(i), s_1) \in \mathfrak{f}''(\delta(i, s_1)) = f \circ \mathfrak{h}(\delta(i, s_1))$
and hence $\delta_0(h_I(i), s_0)) \in \mathfrak{h}(\delta(i, s))$ as f is injective as well.

$h : A \to A_0$ commutes uniquely (**2**) :
$f_I \circ h_I = f''_I$ due to pullback I_0 in SET, and
$f \circ \mathfrak{h} = \mathfrak{f}''$ due to construction of $\mathfrak{h}$.

$h : A \to A_0$ commutes uniquely (**3**) :
$g_I \circ h_I = g''_I$ due to pullback I_0 in SET.
For $s \in S$ we have $\mathfrak{g}''(s) = A'_2 = (I'_2, S'_2, \delta_2 : I'_2 \times S'_2 \to S'_2)$ and we need to show
$g_S(S'_0) = S'_2$ and $g_I(I'_0) = I'_2$: $\qquad (**)$

To show directly $g_S(S'_0) \subseteq S'_2$, let $s_0 \in S'_0$ then

$$
\begin{array}{lll}
s_0 \in S'_0 & & \\
\implies & f_S(s_0) \in S'_1 & \text{due to definition of } S'_0 \\
\implies & g'_S \circ f_S(s_0) \in g'(\mathfrak{f}''(s)) & \text{mapping via } g' \\
\implies & f'_S \circ g_S(s_0) \in f'(\mathfrak{g}''(s)) & \text{due to } (*) \\
\implies & g_S(s_0) \in \mathfrak{g}''(s) & \text{as } f \text{ is injective} \\
\implies & g_S(s_0) & \text{due to definition of } S'_2
\end{array}
$$

And to show directly $S'_2 \subseteq g_S(S'_1)$, let $s_2 \in S'_2$ then

$$
\begin{array}{lll}
s_2 \in S'_2 & & \\
\implies & s_2 \in \mathfrak{g}''(s) & \text{due to definition of } S'_2 \\
\implies & f'_S(s_2) \in f'(\mathfrak{g}''(s)) & \text{mapping via } f' \\
\implies & f'_S(s_2) \in g'(\mathfrak{f}''(s)) & \text{due to } (*) \\
\implies & \exists s_1 \in S'_1 \text{ with } f'_S(s_2) = g'_S(s_1) & \text{due to def. of refinement mor.} \\
\implies & \exists s_0 \in S'_0 \text{ with } f_S(s_0) = s_1 \text{ and } g_S(s_0) = s_2 & \text{due to PB } S_0 \text{ in } \mathbf{Set} \\
\implies & s_2 \in g_S(S'_0) & \text{due to definition of } S'_0
\end{array}
$$

Analogously we show $g_I(I'_0) = I'_2$.
Hence for $s \in S$ we have $g \circ \mathfrak{h}(s) = g(I'_0, S'_0, \delta_0) = (g_I(I'_0), g_S(S'_0), g_S \circ \delta_0) = (I'_2, S'_2, \delta_2) = \mathfrak{g}''(s)$.
$\mathfrak{h}$ is unique wrt. (**3**) due to $(**)$. $\surd$

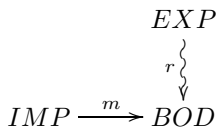
5 Transformations of Petri Net Modules

5.1 Review of Petri Net Modules

In [35] Petri net modules have been introduced independently of the categorical 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. 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. We motivate the notions and results of modules in terms of a larger example in the area of telephone services in [37], where we have developed a Petri net model of an automated telephone service center with a variety of telephone services. In [36] this approach is extended to include safety properties.

A Petri net module $MOD = (IMP, EXP, BOD)$ consists of three Petri nets, namely the import net IMP, the export net EXP and the body net BOD. Two Petri net morphisms $m : IMP \to BOD$ and $r : EXP \rightsquigarrow BOD$ connect the interfaces to the body.

The import interface specifies resources which are used in the construction of the body, while the export interface specifies the functionality available from the Petri net module to the outside world. The body implements the functionality specified in the export interface using the imported functionality. The import morphism m is a *plain morphism* and describes how and where the resources in the import interface are used in the body.



The export morphism r is a *substitution morphism* and describes how the functionality provided by the export interface is realized in the body. The class of substitution morphism is as generalization of plain morphisms, where a transition is replaced by a subnet. Nevertheless, the forgetful functor constructions can be given for substitution morphisms as well.

Morphisms of Petri Nets

First we give a short intuition of the underlying basics. The precise definitions can be found in [33]. Here we use the algebraic notion of Petri nets as introduced in [30]. 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.
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 [30] 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 $\rightarrow$.

Definition 5.1 (Plain Morphisms)
A plain morphism $f : N_1 \rightarrow N_2$ is given by $f = (f_P, f_T)$ with $f_P : P_1 \rightarrow P_2$ and $f_T : T_1 \rightarrow T_2$ so that

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

and post analogously.
These morphisms give rise to the category **PT**. $\Diamond$

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 5.2 (Substitution Morphisms)
A substitution morphism $f : N_1 \rightsquigarrow N_2$ is given by $f = (f_P, \mathfrak{f}_T)$ with $f_P : P_1 \rightarrow P_2$ and $\mathfrak{f}_T : T_1 \rightarrow \mathcal{P}(N_2)$ 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 $g_T \circ f_T : T_1 \rightarrow \mathcal{P}(N_3)$ 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. $\Diamond$

Note that these morphisms do not preserve properties. They can be restricted in order to preserve liveness (see for example [45]).

5.2 Transformation of Petri Net Modules

In the this section we briefly sketch the modifications and transformations for the various instantiations of our component concept. Rule-based Refinement of Petri net modules is achieved by proving the assumptions in definition 3.1.

Fact 5.3 (Petri net module categories satisfy the assumption 3.1)
The following holds:

1. **PN** category of specifications with plain morphisms
2. **SPN** category of specifications with refinement morphisms
3. we have the inclusion functor $\mathsf{Inc} : \mathbf{PN} \rightarrow \mathbf{SPN}$ so that $Obj_{\mathbf{PN}} = Obj_{\mathbf{SPN}}$.

4. **SPN** has pushouts where at least one morphism is in $\mathsf{Inc}(Mor_{\mathbf{PN}})$ and the pushout preserves $\mathsf{Inc}(Mor_{\mathbf{PN}})$.
5. $(\mathbf{PN}, \mathcal{M}_{\mathbf{PN}})$ is weakly adhesive HLR category
6. the inclusion functor $\mathsf{Inc} : \mathbf{PN} \to \mathbf{SPN}$ preserves POs where at least one morphism is in $\mathcal{M}_{\mathbf{PN}}$.
7. the inclusion functor $\mathsf{Inc} : \mathbf{PN} \to \mathbf{SPN}$ preserves PBs where at least one morphism is in $\mathcal{M}_{\mathbf{PN}}$.

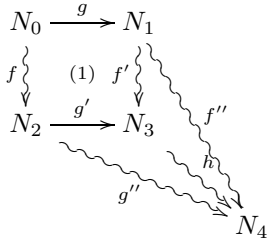
$\Diamond$

Note, that in [35] we have used specific conditions to ensure component-wise construction of pushouts with one plain morphism in **SPN**. In [38] we have used pushouts with one plain, injective morphism. Here, we have pushouts with one plain morphism in **SPN**, that can be constructed allways, but is less intuitive. We use more or less the same idea as for automata.

Proof:
In [35] we have already shown item 1-3 and 6. Item 5 is shown in [21]. Here we show item 4 and item 7.

proof of item 4

Given a plain morphism $f : N_0 \to N_1$ and a substitution morphism $g : N_0 \rightsquigarrow N_2$, then we have $N_3 := (P_3, t_3, pre_3, post_3)$ with



- $P_3 = (P_1 \uplus P_2)_{|\bar{E}}$ where $\bar{E}$ is the equivalence closure of
  $E = \{(g_P(p_0), f_P(p_0)) \,|p_p \in P_0\}$
  $\cup\{(x, y) \,|g_T(t_0) = g_T(t_0'), t_0 \neq t_0', \mathfrak{f}_T(t_0) \neq \mathfrak{f}_T(t_0'), x \in \mathfrak{f}_T(t_0), y \in \mathfrak{f}_T(t_0')\}$
  g'_P and f'_P are the coresspnding natural functions, and obviously we have $g'_P \circ f_P = f'_P \circ g_P$.

- $T_3 := T_1 \setminus g_T(T_0) \uplus T_2$,
  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 $(*)$.

Next we define $\mathfrak{f}'_T$ and g'_T:

$\mathfrak{g}'_T : T_2 \to \mathcal{P}(N_3)$ with
$\mathfrak{g}'_T(t) = (f'_P, id_{T_2})(net(t))$
So f is plain and we write g'_T.
$\mathfrak{f}'_T : T_1 \to \mathcal{P}(N_3)$ with

$$\mathfrak{f}'_T(t) = \begin{cases} g'_T(\mathfrak{f}_T(t_0)) & ; t = g_T(t_0) \\ (f'_P, id_{T_1})(net(t)) & ; else \end{cases}$$

$\mathfrak{g}'_T$ is well defined due to the construction of N_3.

It remains to prove the universal property:
Given $\mathfrak{f}''_T : N_2 \rightsquigarrow N_4$ and $\mathfrak{g}''_T : N_1 \rightsquigarrow N_4$ such that $\mathfrak{g}''_T \circ f = \mathfrak{f}''_T \circ g$, then we define the unique $h : N_3 \rightsquigarrow N_4$ where $h_P : P_3 \to P_4$ is uniquely dedfined by

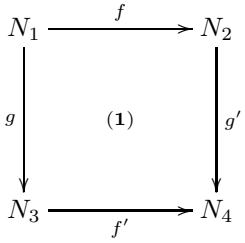
$$h_P(p_3) = \begin{cases} f''(p_2) & ; f'_P(p_2) = p_3 \\ g''(p_1) & ; g'_P(p_1) = p_3 \end{cases}$$

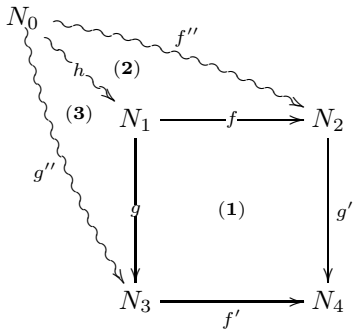
and well-defined due to the equivalence classes. And we define $\mathfrak{h}_T : T_3 \to \mathcal{P}(N_4)$ with

$$\mathfrak{h}_T(t) = \begin{cases} \mathfrak{f}''_T(t_3) & ; t_3 \in T_2 \\ \mathfrak{g}''_T(t_3) & ; t_3 \in T_1 \end{cases}$$

$\mathfrak{h}_T$ is well defined due to $(*)$.

proof of item 7 Given the pullback (1) in **PT** with injective f', we show that (1) is pullback in **SPN** as well.





Given $f'' : N_0 \rightsquigarrow N_2$ and $g'' : N_0 \rightsquigarrow N_3$ so that:
$f' \circ g'' = g' \circ f''$ $(*)$
Then we have $\mathfrak{f}''_T : T_0 \to \mathcal{P}(N_2)$ and for some $t_0 \in T_0$ we have $\mathfrak{f}''_T(t_0) = N'_2 = (P'_2, T'_2, pre_{2|P'_2}, post_{2|P'_2})$.

We define $h : N_0 \rightsquigarrow N_1$ by
$h_P : P_0 \to P_1$ by the induced pullback morphism in **Set** and
$\mathfrak{h}_T : T_0 \to \mathcal{P}(N_2)$ with $\mathfrak{h}_T(t_0) = N'_1 = (P'_1, T'_1, pre_{1|P'_1}, post_{1|P'_1})$ and
$P'_1 \subseteq P_1$ with $P'_1 = \{p \mid f_P(p) \in P'_2\}$ and
$T'_1 \subseteq T_1$ with $T'_1 = \{t \mid f_T(t) \in T'_2\}$.

$\mathfrak{h}_T : T_0 \to \mathcal{P}(N_2)$ is well-defined according to definition 5.2:
For $t_0 \in T_0$ we have $f_P \circ h_P({}^\bullet t_0) = f''_P({}^\bullet t_0)$ as h_P is induced by pullback in **Set** and hence $f_P \circ h_P({}^\bullet t_0) \in P'_2$. This implies $h_P({}^\bullet t_0) \in P'_1$.

Analogously we show $h_P(t_0^\bullet) \in P_1'$.

(**2**) commutes:
$f_p \circ h_P = f_P''$ as h_P is induced by pullback in **Set**.

For $f \circ \mathfrak{h}_T = \mathfrak{f}_T''$ we first have to show that $f_P(P_1') = P_2'$ and $f_T(T_1') = T_2'$.
Obviously $f_P(P_1') \subseteq P_2'$ and $f_T(T_1') \subseteq T_2'$ holds.
To show $P_2' \subseteq f_P(P_1')$ indirectly, let

$$p_2 \in P_2' \setminus f_P(P_1'). \qquad (**)$$

Then we have $g_P'(p_2) \in g' \circ \mathfrak{f}_T''(t_0)$ and because of $(*)$ we have $g_P'(p_2) \in f' \circ \mathfrak{g}_T''(t_0)$. It follows that $g_P'(p_2) \in f_P'(P_3)$ and as (**1**) is pullback in **PN** there is some $p_1 \in P_1$ with $f_P(p_1) = p_2$. This contradicts $(**)$.
Analogously we show $T_2' \subseteq f_T(T_1')$.

For $t_0 \in T_0$ we now have $f \circ \mathfrak{h}_T(t_0) =$
$f(N_1') =$
$(f_P(P_1'), f_T(T_1'), f_p^\oplus \circ pre_{1|P_1'}, f_p^\oplus \circ post_{1|P_1'}) =$
$(P_2', T_2', pre_{2|P_2'}, post_{2|P_2'}) =$
$N_2' =$
$\mathfrak{f}_T''(t_0)$

(**3**) commutes:
$g_p \circ g_P = g_P''$ as h_P is induced by pullback in **Set**.

To show $g \circ \mathfrak{h}_T = \mathfrak{g}_T''$:
For $t_0 \in T_0$ we have $\mathfrak{g}_T''(t_0) = N_3' = (P_3', T_3', pre_{3|P_3'}, post_{3|P_3'})$.
We first have to show that $g_P(P_1') = P_3'$ and $g_T(T_1') = T_3'$.

To show $P_3' \subseteq g_P(P_1')$ indirectly, let

$$p_3 \in P_3' \setminus g_P(P_1'). \qquad (***)$$

Then we have $f_P'(p_3) \in f' \circ \mathfrak{g}_T''(t_0)$ and because of $(*)$ we have $f_P'(p_3) \in g' \circ \mathfrak{f}_T''(t_0)$. It follows that $f_P'(p_3) \in g_P'(P_2)$ and as (**1**) is pullback in **PN** there is some $p_1 \in P_1$ with $f_P(p_1) = p_3$. This contradicts $(***)$.
Analogously we show $T_3' \subseteq f_T(T_1')$.

To show $g_P(P_1') \subseteq P_3'$ indirectly let

$$g_P(p_1) \in g_P(P_1') \setminus P_3'. \qquad (\circledast)$$

Then we have $f_P(p_1) \in P_2'$ and hence $g_P' \circ f_P(p_1) \in g' \circ \mathfrak{f}_T''(t_0)$. This implies $f_P' \circ g_P(p_1) \in f' \circ \mathfrak{g}_T''(t_0)$, because of $(*)$.
Hence we have $g_P(p_1) \in \mathfrak{g}_T''(t_0)$ contradicting $(\circledast)$
Analogously we show $f_T(T_1') \subseteq T_3'$.

We now have $g \circ \mathfrak{h}_T(t_0) =$
$g(N_1') =$
$(g_P(P_1'), g_T(T_1'), g_p^\oplus \circ pre_{1|P_1'}, g_p^\oplus \circ post_{1|P_1'}) =$
$(P_3', T_3, pre_{3|P_3'}, post_{3|P_3'}) =$
$N_3' =$
$\mathfrak{g}_T''(t_0)$

$h : N_0 \rightsquigarrow N_1$ is unique w.r.t. (**2**) and (**3**):
$h_P : P_0 \to P_1$ is the induced pullback morphism in **Set** and hence unique w.r.t. (**2**) and (**3**).
For $\mathfrak{h}_T' : T_0 \to \mathcal{P}(N_2)$ so that (**2**) and (**3**) commute and for $\mathfrak{h}_T'(t_0) = \widehat{(}N_1)$ we

have $f(\widehat{(}N_1) = N_2 = f(N_1')$ and as f is injective we conclude $\mathfrak{h}_T'(t_0) = \widehat{(}N_1) = N_1' = \mathfrak{h}_T(t_0)$. Hence $h : N_0 \rightsquigarrow N_1$ is unique w.r.t. (**2**) and (**3**).

$\surd$

6 Conclusion

In the conclusion we summarize the results and discuss future research. In table 2 we give a survey of the generic component concept and its instantiations. The various specification techniques we present have a component concept as used in this paper. nevertheless they have not been necessarily instantiated directly but have been developed independently, but are very closely related, e.g.. algebraic module specifications, graph transformation systems, or local action systems. The first column name th specification technique, the second refres to the existence of a component concept in the sense of [17], the third asks for the hierarchical composition and the last whether additional component operations are possible, e.g for algebraic module specifications there are union, renaming, actualization and others.

Spec. Tech.	component	hierarch comp.	other comp. ops
Det. input automata	yes	yes	
P/T nets	yes	yes	union [35]
AHL nets	yes [18]	yes [18]	
Graph transf. systems	yes [44]	yes [44]	union [44]
Local action systems	yes [44]	yes [44]	union [44]
Algebraic spec	yes, but with add. parameter [16]	yes [16]	yes [16]
CSP	as connector architectures in [20]	yes [20]	
UML [2]	as connector architectures in [13]	yes [13]	

Table 1: Results concerning component operations

Table 2 lists whether specification techniques have a transformation concept in the sense of the DPO approach, whether it is known that it forms an AHLR system, and whether component transformation is available.

Spec. Tech.	Trafo of Spec	Spec is AHLR	component trafo
det. input automata	yes	yes	yes
P/T nets	yes	yes	yes
AHL nets	yes	yes [21]	straight forward
Graph transf. systems	yes	yes [39, 40]	straight forward
Alg. spec	yes	yes	straightforward

Table 2: Results concerning component transformations

In this paper we have investigated deterministic input automata and place/transition nets, but other instantiations are easy to obtain. If the generic

[2] class diagrams, state machines, sequence diagrams

component concept can be used for a specific specification technique and the technique is an AHLR system, then component transformation can be achieved in a straightforward way.

References

[1] M. C. Bastarrica, S. F. Ochoa, and P. O. Rossel. Integrated notation for software architecture specification. In *Proc. of the XXIV International Conference of the SCCC*, 2004.

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

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

[4] L. Bernadinello and F. De Cindio. A survey of basic net models and modular net classes. *Lecture Notes in Computer Science; Advances in Petri Nets 1992*, 609:304–351, 1992.

[5] M. Broy and T. Streicher. Modular functional modelling of Petri nets with individual tokens. *Advances in Petri Nets*, LNCS 609, 1992.

[6] P. Buchholz. Hierarchical high level Petri nets for complex system analysis. In *Application and Theory of Petri Nets*, volume LNCS 815, pages 119–138. Springer, 1994.

[7] S. Christensen and L. Petrucci. Modular analysis of Petri nets. *Computer Journal*, 43(3):224–242, 2000.

[8] S. Christinsen and N.D. Hansen. Coloured Petri nets extended with channels for synchronous communication. In *Application and Theory of Petri Nets*, volume LNCS 815, pages 159–178. Springer, 1994.

[9] L. de Alfaro and T.A Henzinger. Interface automata. In *ESEC/FSE 01: Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 2001.

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

[11] J. Desel, G. Juhás, and R. Lorenz. Process semantics of Petri nets over partial algebra. In M. Nielsen and D. Simpson, editors, *Proceedings of the XXI International Conference on Applications and Theory of Petri Nets*, volume LNCS 1825, pages 146–165. Springer, 2000.

[12] J. Desel, G. Juhás, and R. Lorenz. Petri Nets over Partial Algebras. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, volume 2128 of *LNCS*. Springer, 2001.

[13] H. Ehrig, B. Braatz, M. Klein, F. Orejas, S. Pérez, and E. Pino. Object-oriented connector-component architectures. In *Proc. FESCA*, 2005.

[14] H. Ehrig, M. Gajewsky, and F. Parisi-Presicce. High-Level Replacement Systems with Applications to Algebraic Specifications and Petri Nets. In G. Rozenberg, U. Montanari, H. Ehrig, and H.-J. Kreowski, editors, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism, and Distribution*, chapter 6, pages 341–400. World Scientific, 1999.

[15] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Math. Struct. in Comp. Science*, 1:361–404, 1991.

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

[17] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A generic component concept for system modeling. In *Proc. FASE '02*, LNCS 2306. Springer, 2002.

[18] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A Transformation-Based Component Framework for a Generic Integrated Modeling Technique. *Journal of Integrated Design and Process Science*, 6(4):78–104, June 2003.

[19] H. Ehrig and Pfender, M. et al. *Kategorien und Automaten*. de Gruyter Lehrbuch, 1972.

[20] Hartmut Ehrig, Julia Padberg, Benjamin Braatz, Markus Klein, Fernando Orejas, Sonia Pérez, and Elvira Pino. A generic framework for connector architectures based on components and transformations. In *Proc. FESCA'04, satellite of ETAPS'04, Barcelona, ENTCS*, volume 108, pages 53–67, 2004.

[21] K. Ehrig, H.and Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2005. in preparation.

[22] R. Fehling. A concept of hierarchical Petri nets with building blocks. In *Advances in Petri Nets'93*, pages 148–168. Springer, 1993. LNCS674.

[23] X. He. A Formal Definition of Hierarchical Predicate Transition Nets. In *Application and Theory of Petri Nets*, volume LNCS 1091, pages 212–229. Springer, 1996.

[24] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts. Springer Verlag, EATCS Monographs in Theoretical Computer Science edition, 1992.

[25] G. Juhás and R. Lorenz. Modelling with Petri modules. In B. Caillaud, X. Xie, and L. Darondeau, Ph.and Lavagno, editors, *Synthesis and Control of Discrete Event Systems*, pages 125–138. Kluwer Academic Publishers, 2002.

[26] E. Kindler. *Modularer Entwurf verteilter Systeme mit Petrinetzen*. PhD thesis, Technische Universität München, Institut für Informatik, 1995.

[27] S. Lack and P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.

[28] S. Mann, B. Borusan, H. Ehrig, M. Große-Rhode, R. Mackenthun, A. Sünbül, and H. Weber. Towards a component concept for continuous software engineering. Technical Report 55/00, FhG-ISST, 2000.

[29] Jasminka Matevska-Meyer, Wilhelm Hasselbring, and Ralf Reussner. Software architecture description supporting component deployment and system runtime reconfiguration. In *Proceedings of Workshop on Component-Oriented Programming (WCOP 2004)*, 2004.

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

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

[32] J. Padberg. Categorical Approach to Horizontal Structuring and Refinement of High-Level Replacement Systems. *Applied Categorical Structures*, 7(4):371–403, December 1999.

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

[34] J. Padberg. Basic Ideas for Transformations of Specification Architectures. In R. Heckel, T. Mens, and Wermelinger M., editors, *Proc. Workshop on Software Evolution through Transformations (SET 02), Satellite Event of ICGT'02*, volume 74 of *Electronic Notes in Theoretical Computer Science (ENTCS)*, October 2002.

[35] J. Padberg. Petri net modules. *Journal on Integrated Design and Process Technology*, 6(4):121–137, 2002.

[36] J. Padberg. Safety properties in Petri net modules. *Journal on Integrated Design and Process Technology*, 2004.

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

[38] J. Padberg and H. Ehrig. Petri net modules in the transformation-based component framework. *Journal of Logic and Algebraic Programming*, page 35, 2005. accepted.

[39] F. Parisi-Presicce. Transformation of Graph Grammars. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS 1073*, 1996.

[40] F. Parisi-Presicce. On Modifying High-Level Replacement Systems. In H. Ehrig, J. Padberg, and C. Ermel, editors, *Proc. Workshop on Uniform Approaches to Graphical Process Specification Techniques (UniGra'01)*, volume 44. ENTCS, 2001.

[41] Ralf H. Reussner and H. W. Schmidt. Using Parameterised Contracts to Predict Properties of Component-Based Software Architectures. In I. Crnkovic, S. Larsson, and J. Stafford, editors, *Workshop on Component-Based Software Engineering*, 2002.

[42] Ralf Heinrich Reussner. *Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten*. PhD thesis, Universität Karlsruhe (Technische Hochschule), 2001.

[43] C. Sibertin-Blanc. Cooperative Nets. In *Application and Theory of Petri Nets'94*, pages 471–490. Springer LNCS 815, 1994.

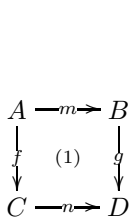
[44] M. Simeoni. An Abstract Module Concept for Graph Transformation Systems. *Electronic Notes of TCS*, 51, 2002. http://www.elsevier.nl/locate/entcs/volume51.html .

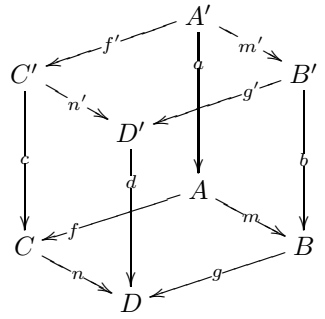
[45] M. Urbášek and J. Padberg. Preserving liveness with rule-based refinement of place/transition systems. In Society for Design and Process Science (SDPS), editors, *Proc. IDPT 2002: Sixth World Conference on Integrated Design and Process Technology, CD-ROM*, page 10, 2002.

[46] H. Weber. Continuous Engineering of Communication and Software Infrastructures. volume 1577 of *Lecture Notes in Computer Science 1577*, pages 22–29. Springer Verlag, Berlin, Heidelberg, New York, 1999.

A Review of VK Squares and Weak AHLR Categories

Definition A.1 (van Kampen square)
A pushout (1) is a van Kampen (VK) square, if for any commutative cube (2) with (1) in the bottom and back faces being pullbacks holds: the top is pushout $\Leftrightarrow$ the front faces are pullbacks.





(2) ◊

The main reason why adhesive categories are important for the theory of graph transformation and its generalization to high-level replacement systems (see [15]) is the fact that most of the HLR conditions required in [15] are shown to be valid already in adhesive categories (see [27]). On the other hand HLR categories in [15] are based on a class M of morphisms, which is restricted to the class of all monomorphisms in adhesive categories. This rules out several interesting examples. In order to avoid this problem we combine the two concepts leading to the notion of weak adhesive HLR categories in [21].

Definition A.2 (Weak adhesive HLR category and system [21])
A category **Cat** with a morphism class $\mathcal{M}$ is called weak adhesive HLR category $(\mathbf{Cat}, \mathcal{M})$, if

1. $\mathcal{M}$ is a class of monomorphisms closed under isomorphisms and closed under composition ($f : A \to B \in \mathcal{M},\ g : B \to C \in \mathcal{M} \Rightarrow g \circ f \in \mathcal{M}$) and decomposition ($g \circ f \in \mathcal{M},\ g \in M \Rightarrow f \in \mathcal{M}$),

2. **Cat** has pushouts and pullbacks along $\mathcal{M}$-morphisms and $\mathcal{M}$-morphisms are closed under pushouts and pullbacks,

3. pushouts in **Cat** along $\mathcal{M}$-morphisms are weak VK squares, i.e. the VK square property holds for all commutative cubes with $m \in \mathcal{M}$ and ($f \in \mathcal{M}$ or $b, c, d \in \mathcal{M}$) see definition A.1.

An adhesive HLR system $AHS = (\mathbf{Cat}, \mathcal{M}, P)$ consists of an adhesive HLR category $(\mathbf{Cat}, \mathcal{M})$ and a set of rules P. ◊

Fact A.3 ($(\mathbf{parSet}, \mathcal{M})$ is adhesive HLR category)
The category of set with partial functions **parSet** together with the class $\mathcal{M}$ of injective functions $(\mathbf{parSet}, \mathcal{M})$ is adhesive HLR category. ◊

Proof Sketch:
The category of sets with partial functions **parSet** is isomorphisc to the category of pointed sets which is the coslice category $\{\bullet\}\backslash\mathbf{Set}$, where $\{\bullet\}$ is some one-element set. As a coslice category adhesive HLR category if the underlying category is adhesive HLR category, we have the above result. √

B Technical Background

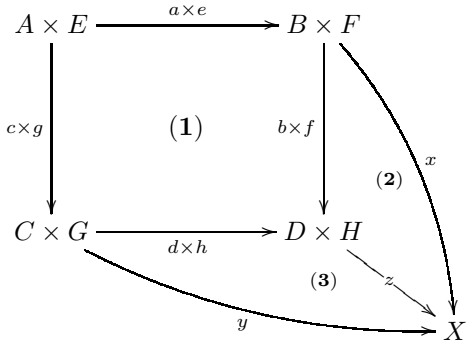
In this appendix we give for the sake of completeness all the definitions, lemmata and proofs required for the two main results. These notions are not neccessary for the understanding of the paper. They are only required for following the central and new proof. So we do not give additional comments.

Concerning Automata

Definition B.1 (Subautomata)
Given $A = (I, S, \delta : I \times S \rightharpoonup S)$ and $A' = (I', S', \delta : I' \times S' \rightharpoonup S')$ then $A' \subseteq A$ if and only if $I' \subseteq I$ and $S' \subseteq S$ and the transition function is obviously the same if defined, i.e $\delta'(i,s) = \begin{cases} \perp & ; \delta(i,s) \notin S' \\ \delta(i,s) & , else \end{cases}$
The set of all subautomata of A is given by $\mathcal{P}(A) := \{A' \subseteq A\}$. $\Diamond$

Fact B.2 (The functor Prod preserves POs) $\Diamond$

Proof Sketch:
Given $\mathsf{Prod} : \mathbf{parSet} \times \mathbf{parSet} \to \mathbf{parSet}$ with $\mathsf{Prod}(A,B) = A \times B$ and for the partial morphisms we have $\mathsf{Prod}(f,g) = f \times g$ with $f \times g(a,b) = \perp \Leftrightarrow (f(a) = \perp \vee g(b) = \perp)$
Given the POs $D = B +_A C$ and $H = F +_E G$ in **parSet** then $\mathsf{Prod}(D,H) \cong \mathsf{Prod}(B,F) +_{\mathsf{Prod}(A,E)} \mathsf{Prod}(C,G)$. obviously (**1**) commutes:



$z : D \times H \rightharpoonup X$ is defined by
$z(d,h) = \begin{cases} x(b_1, f_1) & ; \text{ if } b(b_1) = d \text{ and } f(f_1) = h \\ y(c,g)\delta(i,s) & ; \text{ if } c(c_1) = d \text{ and } g(g_1) = h \end{cases}$
z is well-defined and unique with respect to (**2**) and (**3**) due to the PO properties of D and H. $\surd$

Concerning Petri Nets

Definition B.3 (Commutative Free Monoids)
Given a set P then the free commuative monoid $(P^{\oplus}, \epsilon, \oplus)$ is constructed freely according to the following equations for $u, v, w \in P^{\oplus}$:

1. $w \oplus \epsilon = w$ neutrality
2. $w \oplus v = v \oplus w$ commutativity
3. $w \oplus (v \oplus u) = (w \oplus v) \oplus u$ associativity

Elements $w \in P^{\oplus}$ can be given by finite linear sums: $w = \sum_{p \in P} \lambda_p \cdot p$ with $\lambda_p \in \mathbb{N}$. Hence, the usual operations can be extended from natural numbers to linear sums. We use:

$w \oplus w'$	addition
$w \ominus w'$	substraction
$w \leq w'$	comparison

$\Diamond$

Definition B.4 (Restriction of a Linear Sum)
For $w = \sum_{p \in P} \lambda_p \cdot p$ we define the restriction to $p \in P$: $w_{|p} = \lambda_p \cdot p$ And for $P' \subseteq P$ we define: $w_{|P'} = \sum_{p \in P'} \lambda_p \cdot p$ For a mapping $f : P_1 \to P_2$ we use as a abbreviation:$w_{|f(P)} = w_{|f}$ $\Diamond$

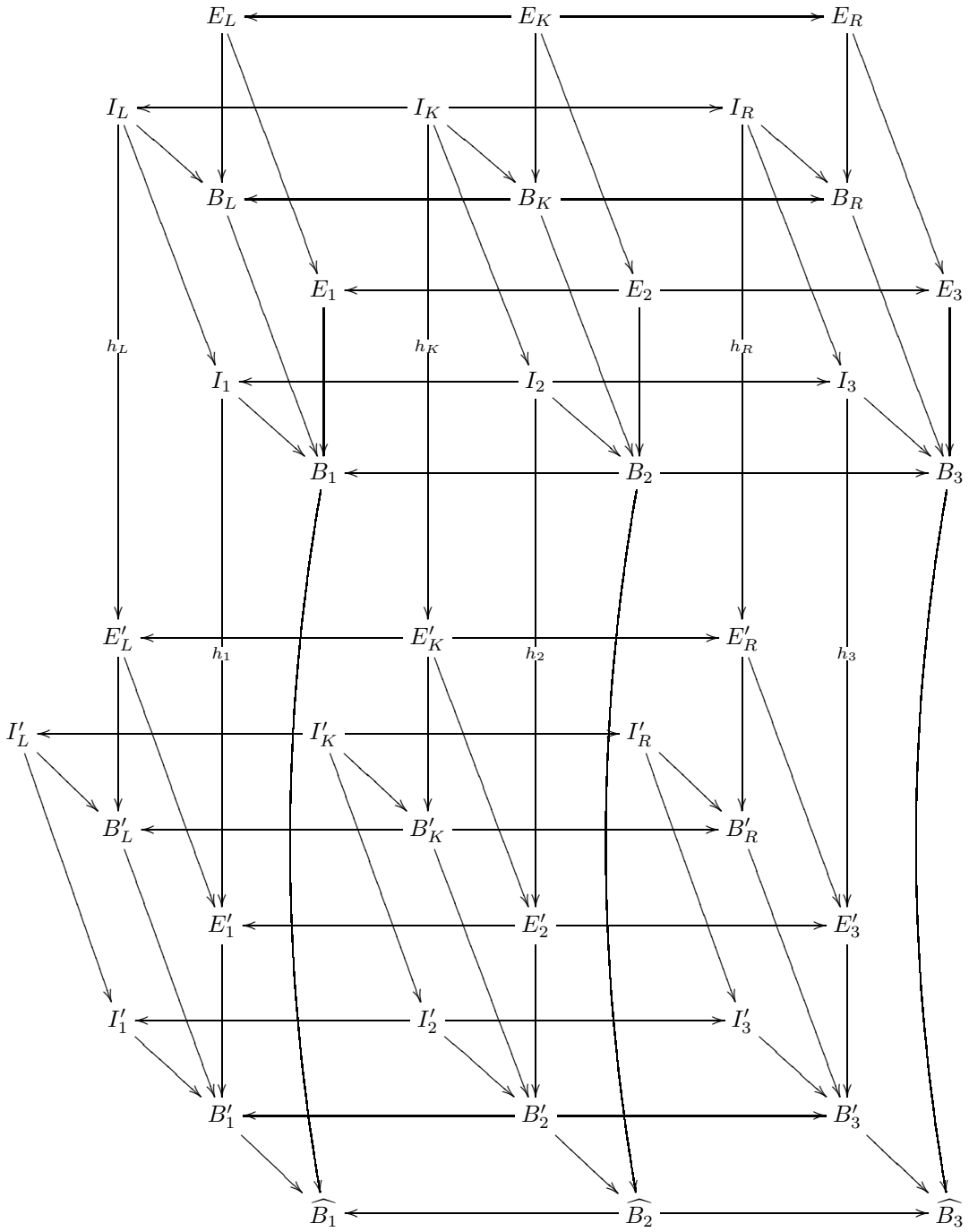
Definition B.5 (Pre-Set, Post-Set)
The pre-set of a net element is given by $\bullet t = \{p | pre(t)_{|p} \neq \epsilon\}$ or $\bullet p = \{t | post(t)_{|p} \neq \epsilon\}$, and analogously for the post-set. $\Diamond$

Definition B.6 (Subnets)
$N' \subseteq N$ if and only if $P' \subseteq P$ and $T' \subseteq T$ as well as for pre- and postdomain $pre'(t) = pre(t)_{|P'}$ for all $t \in T'$, $post$ analogously.
The set of all subnets of N is given by $\mathcal{P}(N) := \{N' \subseteq N\}$. $\Diamond$

Definition B.7 (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

- $P^t = \bullet t \cup t\bullet$,
- $T^t = \{t\}$, and
- $pre^t : T^t \to P^{t\oplus}$ with $pre^t(t) = pre_1(t)$, analogously $post^t$ $\Diamond$

C Additional Diagrams



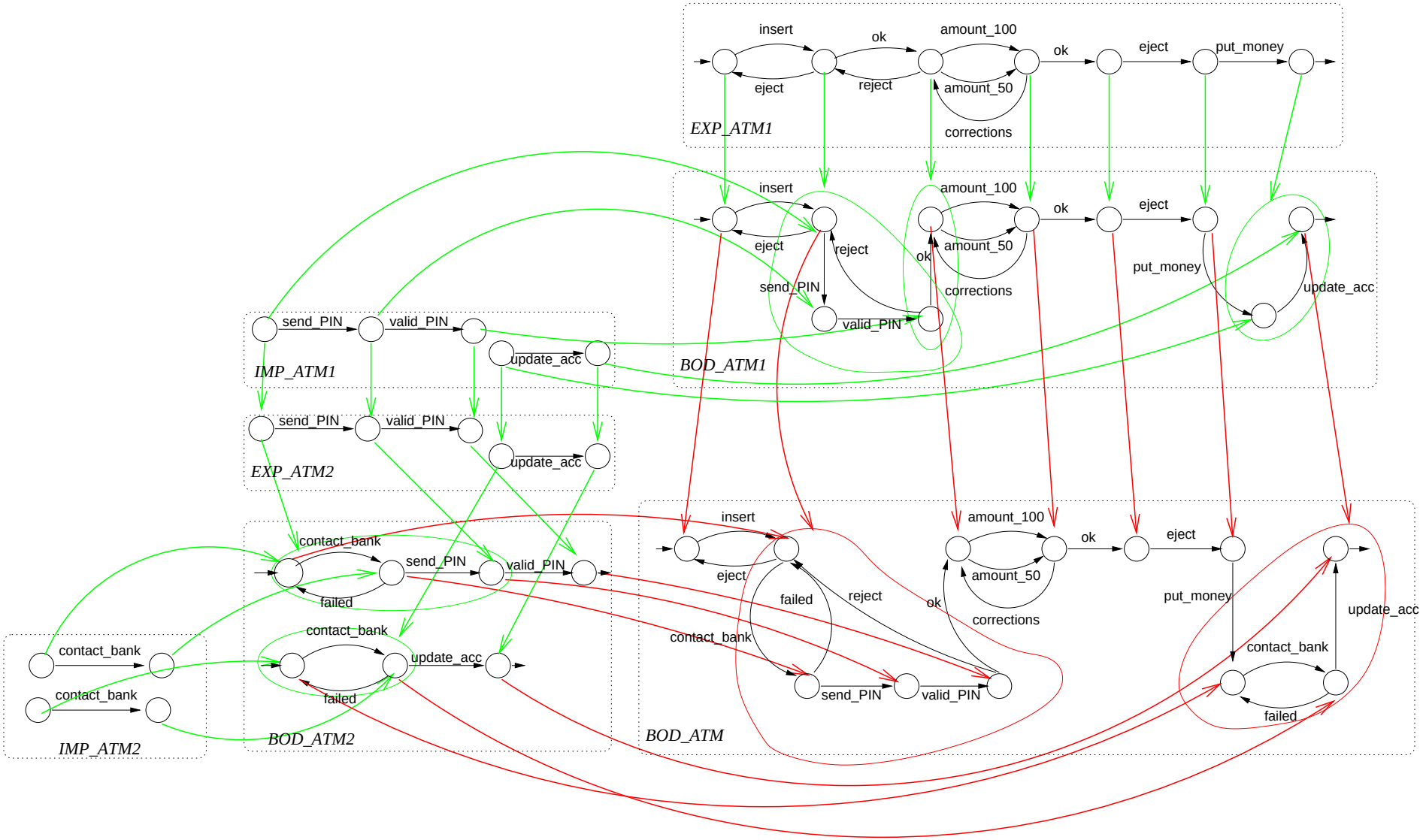


Figure 21: Composition $ATM = ATM1 \circ ATM2$

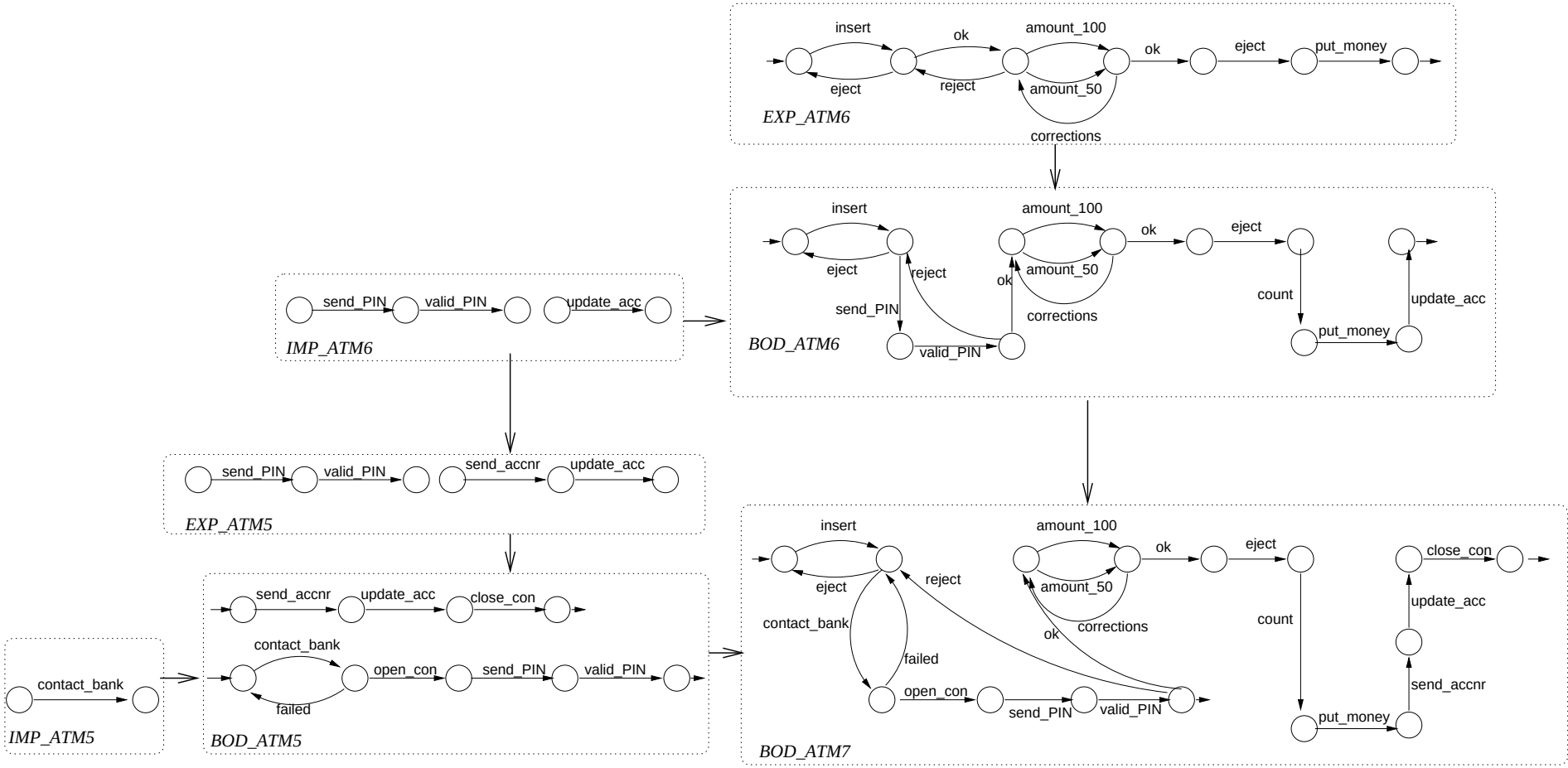


Figure 22: Composition $ATM7 = ATM6 \circ ATM5$