Integration of Categorical Frameworks: Rule-Based Refinement and Hierarchical Composition for Components

Julia Padberg
Technische Universität Berlin
Fakultät IV – Informatik und Elektrotechnik
Franklinstr. 28-29
D-10587 Berlin padberg@cs.tu-berlin.de

September 10, 2007

Abstract

The integration of two important categorical frameworks – namely adhesive High-Level Replacement (HLR) systems and the generic component concept – yields a categorical approach to component transformation and refinement. The generic component concept is shown to be an adhesive HLR category, so rules and transformations as well as the corresponding results are available. Moreover, the compatibility with the hierarchical component composition is provided. The extension to rule-based refinement requires additional property-preserving morphisms and yields property-preserving rules and transformations, i.e. refinements where compatibility with the hierarchical component composition again is achieved.

The categorical framework is instantiated to typed algebraic high-level (AHL) nets and illustrated with an example of AHL net components.

1 Introduction

Categorical frameworks play an important role in theoretical computer science as efficient means for generalizing and transferring results as well as an abstract level for reasoning about basic concepts. Well-known categorical frameworks are specification frames and institutions as a generalization of abstract data types and logics, high-level replacement systems as a generalization of the algebraic approach to graph transformations, Cartesian closed categories as an generalization of functional programming languages, parameterised net classes as a generalization for high-level Petri nets and others. These frameworks enjoy the following advantages of category theory:

- Categorical frameworks enable an effective way for proving results, as the main proofs are given at the abstract level and then the instantiations yield a large amount of results for very little proving effort.
- Moreover, categorical frameworks help separating the levels of abstraction. In this paper we can distinguish three levels of abstraction: At the lowest level there are concrete specifications, namely the algebraic high-level nets in the example. At the next level there are specification formalisms, e.g. the algebraic high-level net formalism, automata theory, the double pushout approach to graph transformations. This is the level of the instantiations. At the most abstract level there are the abstract theories (or meta-theories) that assume certain categorical properties, but do not fix the formalism, e.g. high-level replacement systems, the generic component concept or the formal component technique.

- Category theory has been the basis for the integration of the generic component concept and adhesive high-level replacement systems in this paper. It is the mathematical basis for the uniform description of a component technique for many different specification techniques, since it allows the formulation of basic concepts independently of a specific formalism.

The generic component concept for system modeling [EOB+04, EBK+05, PE05] has been first introduced in [EOB+02] at a semi-formal level. Its categorical formulation can be found in [PE05] where pushouts characterize the main construction. The motivation is to describe components independently of a specific specification technique. The main concepts are a self-contained semantics and the composition of components based on a generic transformation concept. There have been quite different formal and semi-formal specification techniques used within this framework, such as process algebras, UML, automata, various types of graph transformations as well as various types of Petri nets.

Adhesive high-level replacements (HLR) systems [EEPT06] are an abstract theory for the transformation of objects of an arbitrary category in the style of the double pushout approach to graph transformations [CMR+97]. Basically the replacement is carried out in an arbitrary category and not in the category of graphs. The main characteristic is that rules are given as a span of morphisms and transformations as two pushouts in the chosen category. The theory of HLR systems has been developed as an abstract framework for various types of graphs, as hypergraphs, attributed and typed graphs, structures, algebraic specifications, various Petri net classes, elementary nets, place/transition nets, Colored Petri nets, or algebraic high-level nets, and more (see [EHKP91, EEPT06]).

One important result of this paper is the integration of both theories and new results for compatibility of hierarchical composition and transformation. In order to achieve transformations of components we have to make the approach in [PE05] more concrete, by relating the morphism classes used for adhesive HLR systems and generic components leading to an adhesive HLR framework for generic components. This leads to the main technical result that generic components form a weak adhesive HLR system. Hence there are rules, transformations as well as suitable notions for independence and then varied theorems hold: the Church-Rosser Theorem, the Parallelism Theorem and the Concurrency Theorem. These concern the sequential, parallel or concurrent application of rules. The transformation of components allows the change of a component by changing its interfaces and/or its body specification. The integration of the framework for generic components with adhesive HLR systems yields a solid formal foundation for the transformation of generic components. Once adhesive HLR system have been instantiated with generic components new questions of compatibility arise. The compatibility of component transformation with component composition ensures the consistent development of components.

Rule-based refinement transforms specifications using rules so that specific system properties are preserved Since the specification describes some desired system properties these need to be guaranteed after each development step. Verification of each intermediate step requires a lot of effort and hence is cost intensive. Preservation of system properties by a transformation is to be understood in the following way: If a specification has a certain system property then the transformed specification has the corresponding property as well. Rule-based refinement is an extension of transformations with an additional refinement morphism (see [Pad99]) that preserves specific properties in the underlying specification category. These morphisms are combined with rules and transformations so that those preserve the properties as well. So, each part of the component, import, export and body can be refined by applying rules that preserve properties. Again we obtain the compatibility result that allows the stepwise preservation of desirable component properties.

The application of the gained results in software engineering yields a formal component technique for component based software development. Software components are a useful and widely accepted abstraction mechanism during the entire software life cycle from analysis to maintenance. They need to be backed by thorough formal concepts and modeling techniques, because the high complexity of component-based systems often impedes its consistency. The high complexity is caused mainly by the non-deterministic and concurrent interaction of components. These also

lead to strong dependencies between a component and its environment. This is one main obstacle for the adaption of component-based systems to changing environments. So, a formal component technique as proposed in this paper, consisting of the component description, semantics, composition operations and refinement concepts, is required for the formal foundation of component-based engineering. To illustrate this categorical framework we instantiate it with algebraic high-level nets – a variant of high-level Petri nets with arc inscriptions and data token based on an algebraic specification.

Parts of this paper can be found in [Pad05a], where we only sketch the results and where the proofs have been omitted. Here we provide additional results concerning rule-based refinement. We investigate algebraic high-level nets as an example instantiation, whereas in [Pad05a] the instantiations are outlined for place/transition nets and deterministic input automata.

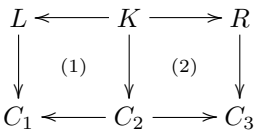
2 Categorical Frameworks in Computer Science

Now we review the both frameworks, namely *Adhesive HLR Systems* and the *Transformation-Based Framework for Generic Components*.

2.1 Summary of Adhesive HLR Systems

High-Level replacement (HLR) systems have been introduced in [EHKP91] as a generalization of the double pushout approach to graph transformations. Basically the replacement is carried out in an arbitrary category and not in the category of graphs. The basic notions remain more or less the same, but the notions of rules and transformations need an additional subclass $\mathcal{M}$ of monomorphisms.

Definition 2.1 (Rules and transformations)
A rule is given by $r = (L \leftarrow K \rightarrow R)$ where L and R are the left and right hand side objects, K is an intermediate object[1]*, and the morphisms $K \rightarrow L$ and $K \rightarrow R$ belong to $\mathcal{M}$ a subclass of monomorphisms. Given a rule r and a context object C_2 we use morphisms $K \rightarrow L, K \rightarrow R$ and $K \rightarrow C_2$ to express a transformation step as the pushout constructions (1) and (2) leading to a double pushout as depicted adjacently.*



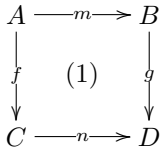
An application of a rule is called a transformation step and describes the change of an object C_1 to an object C_3 by applying that rule. A sequence of these rule applications yields a transformation.

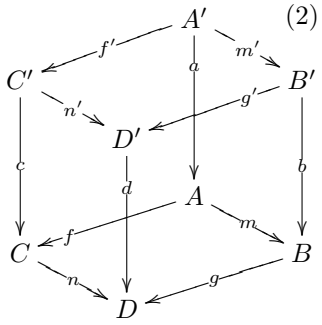
The theory of HLR systems has been developed as an abstract framework for different types of graph and Petri net transformation systems. HLR systems are instantiated with various types of graphs, as hyper-graphs, attributed and typed graphs, structures, algebraic specifications, various Petri net classes, elementary nets, place/transition nets, Colored Petri nets, or algebraic high-level nets, and more (see [EHKP91] and [EEPT06]). Adhesive categories have been introduced in [LS04] and have been combined with HLR categories and systems in [EHPP04] leading to the new concept of (weak) adhesive HLR categories and systems. The main reason why adhesive categories are important for the theory of graph transformation and its generalization to high-level replacement systems is the fact that most of the HLR conditions required in [EHKP91] are shown to be already valid in adhesive categories (see [LS04]). The fundamental construct for (weak) adhesive (HLR) categories and systems are (weak) van Kampen squares.

Definition 2.2 (Weak van Kampen square) *Given a class of monomorphisms $\mathcal{M}$, then a pushout (1) with $m \in \mathcal{M}$ is a weak van Kampen (VK) square*

if for any commutative cube (2) with

- *(1) in the bottom*
- *$f \in \mathcal{M}$ or $b, c, d \in \mathcal{M}$ and*
- *the back faces being pullbacks*





the following holds:

the top is pushout $\Leftrightarrow$ the front faces are pullbacks.

(Weak) adhesive HLR systems [EEPT06] can be considered as abstract graph transformation systems in the double pushout approach based on adhesive or weak adhesive HLR categories.

Definition 2.3 (Weak adhesive HLR category and system) *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; see Definition 2.2.*

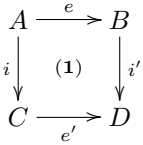
An adhesive HLR system $AHS = (\mathbf{Cat}, \mathcal{M}, P)$ consists of an adhesive HLR category $(\mathbf{Cat}, \mathcal{M})$ and a set of rules P.

2.2 Summary of the Transformation-Based Framework for Generic Components

We now present basic ideas concerning the generic concept of components in a categorical frame. In this framework a component consists of an import interface, an export interface 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. In [PE05] we presented a categorical formalization of the concepts of the generic framework using specific kinds of pushout properties which we use subsequently.

Definition 2.4 (Generic framework $\mathcal{T}$ for components) *A generic framework for components $\mathcal{T} = (\mathbf{Cat}, \mathcal{I}, \mathcal{E})$ consists of an arbitrary category* **Cat** *and two classes of morphisms $\mathcal{I}$, called import morphisms and $\mathcal{E}$, called export morphisms that are both closed under composition and isomorphisms.*
Moreover, they satisfy the following extension conditions*:*

1. *$\mathcal{E}$-$\mathcal{I}$-Pushout Condition:*
   Given the morphisms $A \xrightarrow{e} B$ with $e \in \mathcal{E}$ and $A \xrightarrow{i} C$ with $i \in \mathcal{I}$, then there exists the pushout D in **Cat** *with morphisms $B \xrightarrow{i'} D$ and $C \xrightarrow{e'} D$ as depicted adjacently.*
2. *$\mathcal{E}$ and $\mathcal{I}$ are stable under pushouts:*
   *Given a $\mathcal{E}$-$\mathcal{I}$-pushout as (**1**) above, then we have $i' \in \mathcal{I}$ and $e' \in \mathcal{E}$ as well.*

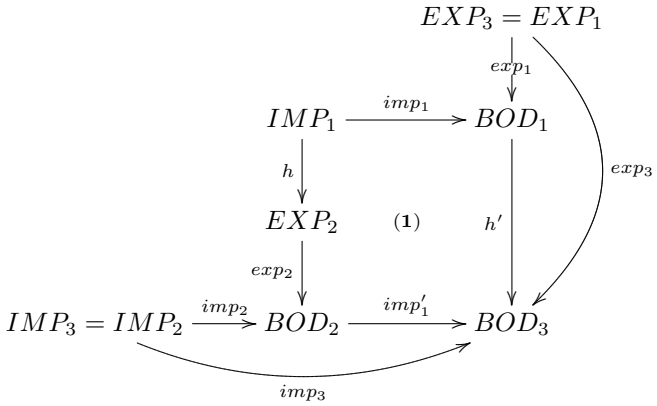


Accordingly, we have to require for a component that the import and export connections belong to the corresponding class of morphisms.

Definition 2.5 (Component) *A component $C = (IMP, EXP, BOD, imp, exp)$ is given by objects IMP, EXP, and BOD in* **Cat** *and by the export morphism $exp : EXP \to BOD$ and the import morphism $imp : IMP \to BOD$ with $exp \in \mathcal{E}$ and $imp \in \mathcal{I}$.*

Subsequently, hierarchical composition of components C_1 and C_2 is introduced. It takes a connection $h : IMP_1 \to EXP_2$ from the import interface IMP_1 of C_1 to the export interface EXP_2 of C_2. It is defined as follows.

Definition 2.6 (Hierarchical composition) *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 the components 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 a pushout diagram in the category* **Cat**.



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

3 Integration of Adhesive HLR Systems with the Transformation-Based Framework for Generic Components

In this section we integrate the two categorical frameworks. The benefit of this integration is the transfer of notions and results concerning rules and transformations to components. Basically, the transformations are carried out in each part of the component. For each – the export, the import and the body – there is a transformation in the underlying specification category. The specification category has to be weak adhesive HLR category.

As the definition of components involves the different classes of morphisms $\mathcal{I}$ and $\mathcal{E}$ these need to be taken into consideration as well as the class of monomorphisms $\mathcal{M}$ of the adhesive HLR category. We first investigate the properties of component categories in regard of the involved morphism classes. So, the difficulty to establish transformations for components directly depends on the chosen class of refinement morphisms. Therefore we extend the approach in [PE05] by relating the morphism classes used for the transformations and the components. This leads to the adhesive HLR framework for generic components.

Definition 3.1 (Adhesive HLR framework for generic components) *The adhesive HLR framework for generic components $\mathcal{A} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M})$ is given by:*

1. $\mathbf{Cat_p}$ *the category of specifications with plain morphisms.*
2. $\mathbf{Cat_r}$ *the category of specifications with refinement morphisms includes the category* $\mathbf{Cat_p}$ *with the functor* $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ *being an inclusion in the sense that* $Obj_{\mathbf{Cat_p}} = Obj_{\mathbf{Cat_r}}$.
3. $(\mathbf{Cat_p}, \mathcal{M})$ *is a weak adhesive HLR category.*

4. $\mathbf{Cat_r}$ *has pushouts if at least one of the given morphisms is in* $\mathsf{Inc}(Mor_{\mathbf{Cat_p}})$ *and* $\mathsf{Inc}(Mor_{\mathbf{Cat_p}})$ *is stable under pushouts.*

5. *The inclusion functor* $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ *preserves pushouts if at least one of the given morphisms is in* $\mathcal{M}$*.*

6. *The inclusion functor* $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ *preserves pullbacks if at least one of the given morphisms is in* $\mathcal{M}$*.*

First we relate the adhesive HLR framework for generic components $(\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M})$ to the generic framework given in Definition 2.4. We choose the import morphisms to be plain morphisms, i.e. $\mathcal{I} = \mathsf{Inc}(Mor_{\mathbf{Cat_p}})$, and the export morphisms to be refinement morphisms, i.e. $\mathcal{E} = Mor_{\mathbf{Cat_r}}$.

Fact 3.2 (Relation of frameworks) *For an adhesive HLR framework for generic components* $(\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M})$ *there is the framework for generic components* $\mathcal{T} = (\mathbf{Cat_r}, \mathsf{Inc}(Mor_{\mathbf{Cat_p}}), Mor_{\mathbf{Cat_r}})$.

Proof due to item 4 of Definition 3.1.
Subsequently, we show that the category of components **Comp** is a weak adhesive HLR category, provided that the underlying category of specifications with plain morphisms is a weak adhesive HLR category as well. We can define the category of components **Comp**, where we use plain morphisms at the specification level for the definition of component morphisms. These additionally have to be compatible with the corresponding import and export morphisms.

Definition 3.3 (Component category) *Component morphisms are defined by* $f : C_1 \to C_2$ *with* $f = (f_I, f_E, f_B)$ *s.t.*$f_I : IMP_1 \to IMP_2$
$f_E : EXP_1 \to EXP_2$
$f_B : BOD_1 \to BOD_2$ *so that*

1. $f_B \circ imp_1 = imp_2 \circ f_I$

2. $f_B \circ exp_1 = exp_2 \circ f_E$

Components and component morphisms constitute **Comp** *the category of components for* $f_I, f_E, f_B \in Mor(\mathbf{Cat_p})$ *and* $\mathbf{Comp_r}$ *for* $f_I, f_E, f_B \in Mor(\mathbf{Cat_r})$.

Theorem 3.4 ($(\mathbf{Comp}, \mathcal{M})$ is a weak adhesive HLR category) *An adhesive HLR framework for generic components* $\mathcal{A} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M}_{\mathbf{Cat_p}})$ *in Definition 3.1 yields that* $(\mathbf{Comp}, \mathcal{M})$ *with* $\mathcal{M} = \{f = (f_I, f_B, f_E) | f_I, f_B, f_E \in \mathcal{M}_{\mathbf{Cat_p}}\}$ *is a weak adhesive HLR category.*

In order to show Theorem 3.4 we first state the subsequent facts.

Fact 3.5 (Pushouts of at least one $\mathcal{M}$-morphism in Comp)

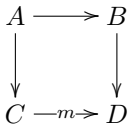
Given the span $B \xleftarrow{m} A \to C$ *in* **Comp** *with* $A = (IMP_A, EXP_A, BOD_A)$ *(the same for* B *and* C*) and the morphism* $m \in \mathcal{M}$ *then the pushout* $B \to D \leftarrow C$ *is constructed component-wise in* $\mathbf{Cat_p}$.

$$\begin{array}{ccc} A & \xrightarrow{m} & B \\ \downarrow & (1) & \downarrow \\ C & \longrightarrow & D \end{array}$$

Proof:
We have a component-wise construction, so we have the following pushouts in the category $\mathbf{Cat_p}$: $IMP_D = IMP_B +_{IMP_A} IMP_C$, $EXP_D = EXP_B +_{EXP_A} EXP_C$
and $BOD_D = BOD_B +_{BOD_A} BOD_C$
$imp_D : IMP_D \to BOD_D$ is the induced pushout morphism. We obtain – as Inc preserves pushouts – the pushout $EXP_D = EXP_B +_{EXP_A} EXP_C$ in $\mathbf{Cat_r}$ and the induced morphism $exp_D : EXP_P \to BOD_D$ in $\mathbf{Cat_r}$.
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. $\surd$

Fact 3.6 (Pullbacks with at least one $\mathcal{M}$-morphism in Comp)

Given cospan $C \xrightarrow{m} D \leftarrow B$ in **Comp** *with $m \in \mathcal{M}$ the pullback $B \leftarrow A \rightarrow C$ is constructed component-wise in* $\mathbf{Cat_p}$.



Proof analogously to the proof of Fact 3.5.

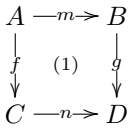
Proof of Theorem 3.4:

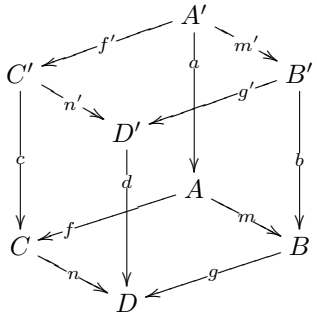
Given the pushout (1) with $m \in \mathcal{M}$ and the commutative cube (2) in **Comp** with

- (1) in the bottom
- $f \in \mathcal{M}$ or $b, c, d \in \mathcal{M}$ and
- the back faces being pullbacks

we need to show the VK property in **Comp**, i.e.

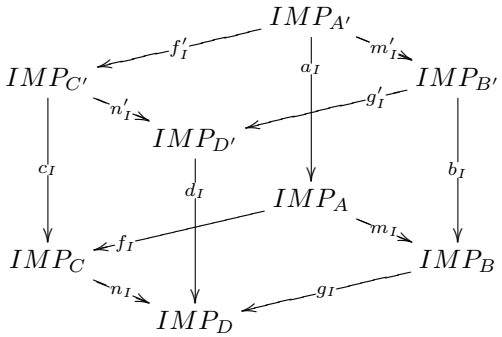
the top is pushout $\Leftrightarrow$ the front faces are pullbacks.





(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 square is a pushout, so we have to show that the front faces are pullbacks. This means that the components B' and C' with the corresponding morphisms are pullbacks in **Comp**:
Since there are the corresponding VK-diagrams in $\mathbf{Cat_p}$, we have $IMP_{C'}$, $BOD_{C'}$ and $EXP_{C'}$ are pullbacks in $\mathbf{Cat_p}$. Hence, we obtain the component $IMP_{C'} \rightarrow BOD_{C'} \leftarrow EXP_{C'}$. Due to the uniqueness of the induced morphisms we have $C' = IMP_{C'} \rightarrow BOD_{C'} \leftarrow EXP_{C'}$ and hence is pullback.

Analogously for B'.

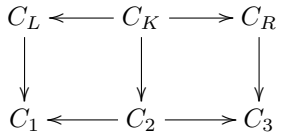
Part 2: (" $\Leftarrow$ ") The front faces are pullbacks and we need to show that component D' with the corresponding morphisms is pushout in **Comp**:
As we have the corresponding VK-diagram in $\mathbf{Cat_p}$, we have $IMP_{D'}$, $BOD_{D'}$ and $EXP_{D'}$ are pushouts in $\mathbf{Cat_p}$. Hence, we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

to the uniqueness of the induced morphisms we have $D' = IMP_{D'} \to BOD_{D'} \leftarrow EXP_{D'}$ and hence is pushout.

√

So, we directly obtain the following essential concepts and results for the transformation of components (see [EEPT06]) as the category of components **Comp** with the distinguished class $\mathcal{M}$ is a weak adhesive HLR category:

Rules and transformations A rule is given by $r = (C_L \leftarrow C_K \rightarrow C_R)$, where C_L and C_R are the left and right hand side components, C_K is an intermediate component of C_L and C_R and the morphisms are in $\mathcal{M}$. 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 define the transformation $C_1 \Longrightarrow C_3$ using the pushout constructions (1) and (2) as depicted below:



In Definition 4.1 we give these notions more precisely.

Parallelism results (chapter 5 in [EEPT06]) 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 neither necessary nor deleted in the second. The 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 [EEPT06]) 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 [EEPT06]) 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 strict confluence of critical pairs.

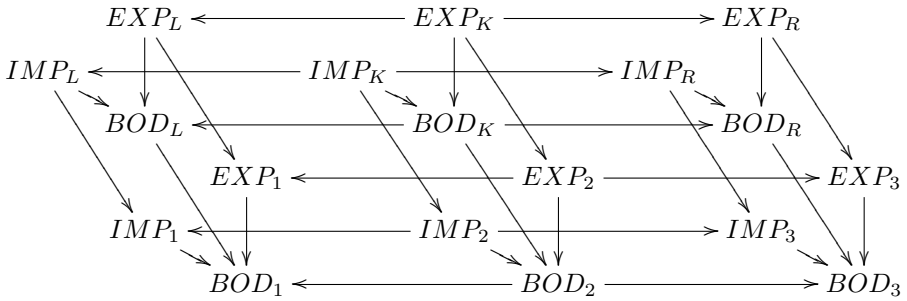
Up to now the instantiations of the HLR theory have been specification techniques as varied types of graph transformations, Petri nets, algebraic specifications, etc. Now we instantiate the HLR theory with the generic component approach presented in Section 2.2. Hence, new questions of compatibility emerge. Namely, the question arises whether the component operations are compatible with the transformation concept. In the subsequent section we characterize the conditions under which transformations and hierarchical composition are compatible.

4 Compatibility Results

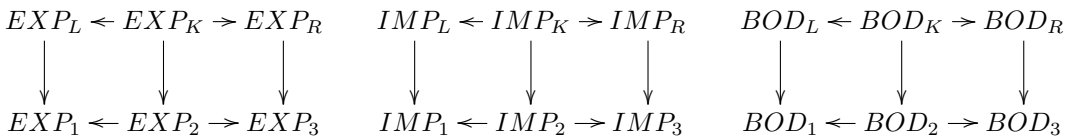
In this section we deal with the compatibility of hierarchical composition with rules, transformations and rule-based refinement. Composed rules are constructed by composing two given rules yielding one rule that combines the effects of both. Compatibility of composition and transformation (respectively refinements) is given with respect to the composed rule and requires that the connections for the rule composition are compatible with the connections of the component composition.

Definition 4.1 (Rules and transformations) *Based on an adhesive HLR framework for generic components* $\mathcal{A} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M}_{\mathbf{Cat_p}})$ *in the weak adhesive HLR category* $(\mathbf{Comp}, \mathcal{M})$ *a component rule* $r = (C_L \leftarrow C_K \rightarrow C_R)$ *is defined by the component morphisms* $C_K \rightarrow C_L$ *and* $C_K \rightarrow C_R$ *both in the class* $\mathcal{M}$ *of monomorphisms.*

Given a rule $r = (C_L \leftarrow C_K \rightarrow C_R)$ *the application of* r *yields the transformation step* $C_1 \overset{r}{\Longrightarrow} C_3$ *given by the following diagram in* $\mathbf{Cat_r}$*:*



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



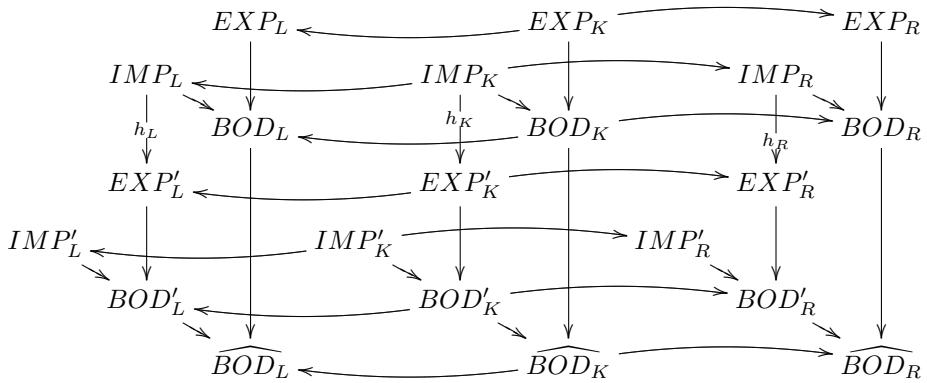
4.1 Compatibility of Hierarchical Composition and Transformations

If two rules change the component's interfaces and are applied together to the composed component, these rules need to composed as well. The following fact states the conditions ensuring that the composition is well-defined.

Fact 4.2 (Component-wise composition of rules) *Given the rules* $r = (C_L \leftarrow C_K \rightarrow C_R)$ *and* $r' = (C'_L \leftarrow C'_K \rightarrow C'_R)$ *and the morphisms* $h_L : IMP_L \rightarrow EXP'_L$*,* $h_K : IMP_K \rightarrow EXP'_K$*, and* $h_R : IMP_R \rightarrow EXP'_R$ *so that:*

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

as depicted in the following diagram



then we can compose component-wise $\widehat{r} := r \circ_h r' = (\widehat{C_L} \leftarrow \widehat{C_K} \rightarrow \widehat{C_R})$ *for* $h := (h_L, h_K, h_R)$ *where we have the following components:*

$\widehat{C_L} := C_L \circ_{h_L} C'_L = (IMP'_L \to \widehat{BOD_L} \leftarrow EXP_L)$,
$\widehat{C_K} := C_K \circ_{h_K} C'_K = (IMP'_K \to \widehat{BOD_K} \leftarrow EXP_K)$, *and*
$\widehat{C_R} := C_R \circ_{h_R} C'_R = (IMP'_R \to \widehat{BOD_R} \leftarrow EXP_R)$

Proof due to Condition 4 in Definition 3.1.

We have $C_1 \overset{r}{\Longrightarrow} C_3$ via the component C_2 and $C'_1 \overset{r'}{\Longrightarrow} C'_3$ via the component C'_2. Now independence ensures that applying the composed rule $r \circ r'$ to the composed component $C_1 \circ C'_1$ indeed results in the transformation $C_1 \circ C'_1 \overset{r \circ r'}{\Longrightarrow} C_3 \circ C'_3$ via the component $C_2 \circ C'_2$.
Figure 1 depicts the components as semi-transparent rectangles where the blockkarrows denote component morphisms.
The underlying diagram in the category $\mathbf{Cat_r}$ is required in Definition 4.3.

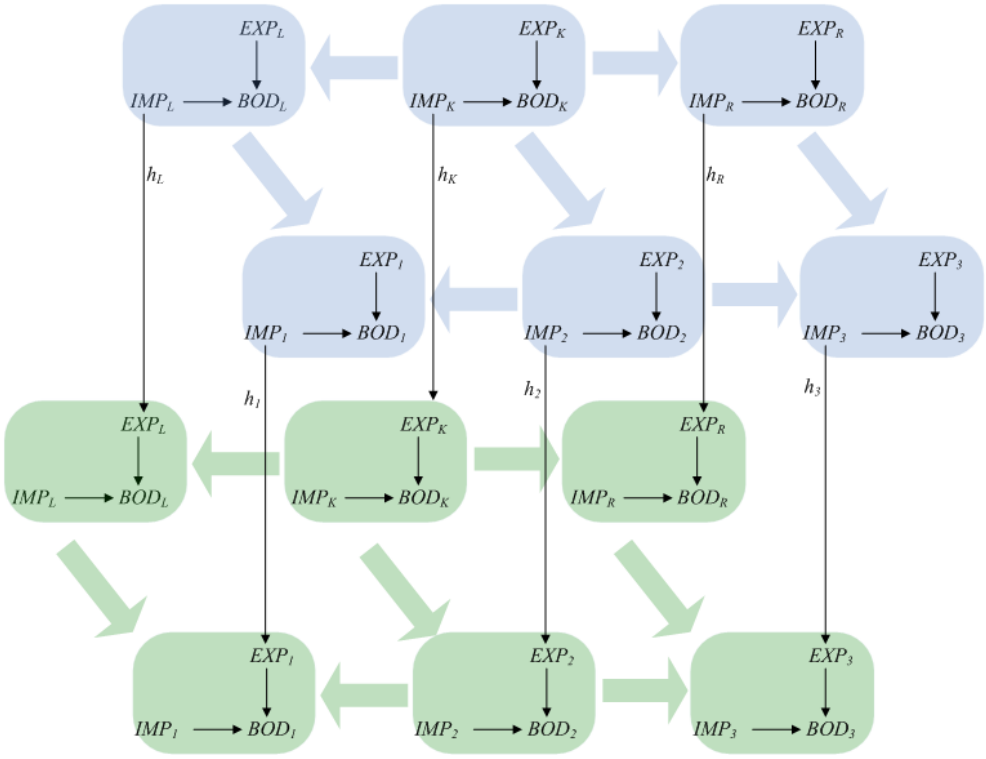


Figure 1: Diagram for independence

Definition 4.3 (Independence of transformation and composition) *Given the rules* $r = (C_L \leftarrow C_K \to C_R)$ *and* $r' = (C'_L \leftarrow C'_K \to C'_R)$ *with* $r \circ_h r'$ *for* $h := (h_L, h_K, h_R)$ *with the morphisms* $h_L : IMP_L \to EXP'_L$, $h_K : IMP_K \to EXP'_K$, *and* $h_R : IMP_R \to EXP'_R$ *then the composition* $C_1 \circ_{h_1} C'_1$ *is independent from* r *and* r' *if there is* $h_2 : IMP_2 \to EXP'_2$ *so that:*

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

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 4.4 (Compatibility Theorem for composition and transformation) *Based on an adhesive HLR framework for generic components* $\mathcal{A} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M}_{\mathbf{Cat_p}})$ *(see Definition 3.1) we have:*

Given the rules $r = (C_L \leftarrow C_K \rightarrow C_R)$ *and* $r' = (C'_L \leftarrow C'_K \rightarrow C'_R)$ *and*
– for $h := (h_L, h_K, h_R)$ *with* $h_L : IMP_L \rightarrow EXP'_L$, $h_K : IMP_K \rightarrow EXP'_K$, $h_R : IMP_R \rightarrow EXP'_R$
let their composition $r \circ_h r'$ *be independent of the composition* $C_1 \circ_{h_1} C'_1$,

then $C_1 \overset{r}{\Longrightarrow} C_3$ *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 is illustrated in the following diagram:

$$\begin{array}{ccccc} C_1 & \circ_{h_1} & C'_1 & = & \widehat{C_1} \\ {\scriptstyle r}\Downarrow & & {\scriptstyle r'}\Downarrow & & \Downarrow{\scriptstyle r \circ_h r'} \\ C_3 & \circ_{h_3} & C'_3 & = & \widehat{C_3} \end{array}$$

Proof:
First we construct the two transformations $C_1 \overset{r}{\Longrightarrow} C_3$ and $C'_1 \overset{r'}{\Longrightarrow} C'_3$.
Then we construct the compositions of the corresponding components $\widehat{C_i} = C_i \circ_{h_i} C'_i$ for $1 \leq i \leq 3$: h_1 and h_2 are given and we compose $\widehat{C_1} = C_1 \circ_{h_1} C'_1 = (IMP'_1, EXP_1, \widehat{BOD_1})$ and $\widehat{C_2} = C_2 \circ_{h_2} C'_2 = (IMP'_2, EXP_2, \widehat{BOD_2})$.
$h_3 : IMP_3 \rightarrow EXP'_3$ is obtained as the induced pushout morphism in $\mathbf{Cat_r}$. So there is the composition $\widehat{C_3} = C_3 \circ_{h_3} C'_3 = (IMP'_3, EXP_3, \widehat{BOD_3})$.

These compositions are well-defined as we have the corresponding pushouts with $IMP_i \rightarrow BOD_i$ in $\mathsf{Inc}(Mor_{\mathbf{Cat_p}})$ (due to Condition 4 in Definition 3.1).

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

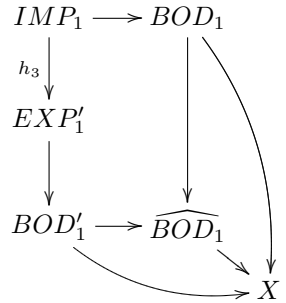
$$\begin{array}{ccccc} \widehat{C_L} & \longleftarrow & \widehat{C_K} & \longrightarrow & \widehat{C_R} \\ \downarrow & (\mathbf{A}) & \downarrow & (\mathbf{B}) & \downarrow \\ \widehat{C_1} & \longleftarrow & \widehat{C_2} & \longrightarrow & \widehat{C_3} \end{array} \qquad \begin{array}{ccccc} \widehat{BOD_L} & \longleftarrow & \widehat{BOD_K} & \longrightarrow & \widehat{BOD_R} \\ \downarrow & (\mathbf{A}') & \downarrow & (\mathbf{B}') & \downarrow \\ \widehat{BOD_1} & \longleftarrow & \widehat{BOD_2} & \longrightarrow & \widehat{BOD_3} \end{array}$$

As we have the transformations $C_1 \overset{r}{\Longrightarrow} C_3$ and $C'_1 \overset{r'}{\Longrightarrow} C'_3$ 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, that is the pushouts $\mathbf{A}'$ and $\mathbf{B}'$:
We construct $\widehat{BOD_L} \rightarrow \widehat{BOD_1}$ as the induced pushout morphism.

Next we show that the square ($\mathbf{A}'$) is a pushout:
Commutativity of ($\mathbf{A}'$) we obtain using the uniqueness of the induced pushout morphism of the pushout $\widehat{BOD_K}$ in $\mathbf{Cat_r}$.

To show the universal property of the square $\mathbf{A}'$ we assume some $X \in \mathbf{Cat_r}$, s.t. $\widehat{BOD_K} \rightarrow \widehat{BOD_2} \rightarrow X = \widehat{BOD_K} \rightarrow \widehat{BOD_L} \rightarrow X$. Then we can construct $BOD_1 \rightarrow X$ and $BOD'_1 \rightarrow X$ due to the universal property of the pushouts BOD_1 and BOD'_1.

So we have



Now we use the pushout $\widehat{BOD_1}$ to obtain $\widehat{BOD_1} \to X$.
Again we use the uniqueness property of induced pushout morphisms to show that $IMP_1 \to BOD_1 \to X$
$= IMP_1 \to BOD_1' \to\!\!\to X$.
We obtain the unique $\widehat{BOD_1} \to X$ with respect to $BOD_1' \to \widehat{BOD_1} \to X = BOD_1' \to X$ and $BOD_1 \to \widehat{BOD_1} \to X = BOD_1 \to X$.
The universal property of $\widehat{BOD_L}$ leads to $\widehat{BOD_L} \to \widehat{BOD_1} \to X = \widehat{BOD_L} \to X$ and the universal property of $\widehat{BOD_2}$ leads to $\widehat{BOD_2} \to \widehat{BOD_1} \to X = \widehat{BOD_2} \to X$.
Uniqueness of $\widehat{BOD_1} \to X$ is due to its construction.

Hence, we have the pushout $(\mathbf{A}')$ and the pushout $(\mathbf{B}')$ is constructed analogously.
We conclude: $\widehat{C_1} = C_1 \circ_{h_1} C_1' \overset{r \circ_h r'}{\Longrightarrow} C_3 \circ_{h_3} C_3' = \widehat{C_3}$ $\surd$

4.2 Compatibility of Hierarchical Composition and Rule-Based Refinement

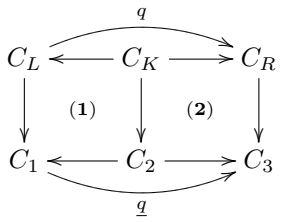
Preservation of system properties is of high interest in many applications as it allows omitting the tedious verification of system properties at different stages of the development. In [Pad99] a notion of refinement motivated by refinement concepts from Petri nets was introduced to the theory of high-level replacement systems. The refinement in high-level replacement is given by a morphism of a suitable category, relating the left with the right hand side of a rewriting rule. This category allows morphisms that are more complex than those given in the underlying HLR category. We require that this category includes the HLR category. On the one hand this allows the construction of transformations and on the other hand this allows the description of the refinement relation between the original and the refined parts.

Definition 4.5 (Adhesive HLR framework for rule-based component refinement) *The adhesive HLR framework for rule-based component refinement* $\mathcal{R} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M}, \mathcal{Q})$ *is given by*

1. $\mathbf{Cat_p}$ *the category of specifications with plain morphisms.*
2. $\mathbf{Cat_r}$ *the category of specifications with refinement morphisms includes the category* $\mathbf{Cat_p}$ *as the functor* $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ *is an inclusion in the sense that* $Obj_{\mathbf{Cat_p}} = Obj_{\mathbf{Cat_r}}$.
3. $(\mathbf{Cat_p}, \mathcal{M})$ *is a weak adhesive HLR category.*
4. $\mathbf{Cat_r}$ *has pushouts if at least one of the given morphisms is in* $\mathsf{Inc}(Mor_{\mathbf{Cat_p}})$.
5. *The inclusion functor* $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ *preserves pushouts if at least one of the given morphisms is in* $\mathcal{M}$.
6. *The inclusion functor* $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ *preserves pullbacks if at least one of the given morphisms is in* $\mathcal{M}$.
7. $\mathcal{Q}$ *is a class of morphisms in* $\mathbf{Cat_r}$ *that is closed under composition and isomorphisms.*
8. *The class* $\mathcal{Q}$ *in* $\mathbf{Cat_r}$ *is stable under pushouts and is closed under the construction of coproducts.*

The additional conditions have been introduced as $\mathcal{Q}$-Transformations in [Pad99]. The compatibility results that are already achieved for transformations are now extended to rule-based refinement. Thus, the theory of high-level replacement systems is expanded in a consistent way.

Definition 4.6 ($\mathcal{Q}$-morphisms, $\mathcal{Q}$-rule and $\mathcal{Q}$-transformations for components) *A $\mathcal{Q}$-morphism $q : C_L \to C_R$ is in $\mathbf{Comp_r}$ with $q_I, q_E, q_B \in \mathcal{Q}$.*
A $\mathcal{Q}$-rule (r, q) is given by a rule $r = C_L \leftarrow C_K \to C_R$ in $\mathbf{Comp}$ and a $\mathcal{Q}$-morphism $q : C_L \to C_R$, so that $C_K \to C_L \xrightarrow{q} C_R = C_K \to C_R$ in $\mathbf{Comp_r}$.
*Given a $\mathcal{Q}$-rule (r, q) and a transformation $C_1 \overset{r}{\Longrightarrow} C_3$ defined by the pushouts **(1)** and **(2)** in $\mathbf{Comp}$, there is a unique $\mathcal{Q}$-morphism $\underline{q} = (\underline{q}_I, \underline{q}_E, \underline{q}_B)$, such that $C_2 \to C_1 \xrightarrow{q} C_3 = C_2 \to C_3$ and $C_R \to C_3 \xleftarrow{\underline{q}} C_1$ is pushout of $C_1 \leftarrow C_L \xrightarrow{q} C_R$ in $\mathbf{Comp_r}$.*
The transformation $C_1 \overset{(r,q)}{\Longrightarrow} C_3$ is called $\mathcal{Q}$-transformation:



The existence of the pushout and the $\mathcal{Q}$-morphism $\underline{q}$ is a direct consequence of the conditions 7 and 8 in Definition 4.5.
In an instantiation where $\mathcal{Q}$ represents a class of property-preserving morphisms we obtain proof rules that allow the derivation of desired properties. The morphism $q : C_L \to C_R$ is a property-preserving morphism in some instantiation, provided it preserves theses properties in the body and the interfaces. Then we call (r, q) a property-preserving rule.
We directly have the extension of rules and transformations with additional morphisms, that preserve properties of the body and the interfaces of a component. So, each part of the component, import, export and body can be refined by applying property-preserving rules.
Proof rules are then obtained that allow the stepwise preservation of desirable 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 $C_1 \overset{(r,q)}{\Longrightarrow} C_3$ and a property-preserving rule (r, q) we have the following proof rule:

$$\frac{(r, q) \text{ is a property-preserving rule; } C_1 \text{ satisfies the corresponding property}}{C_3 \text{ satisfies this property, too}}$$

See Fact 5.4 for an example in case of the instantiation to algebraic high-level nets.
Moreover, the Compatibility Theorem 4.4 can be achieved for $\mathcal{Q}$-transformations as well. Again, we first define the hierarchical composition of rules and subsequently the independence of composition and transformation.

Fact 4.7 (Component-wise composition of $\mathcal{Q}$-rules) *Given the $\mathcal{Q}$-rules $r = (C_L \leftarrow C_K \to C_R, q : C_L \to C_R)$ with $q = (q_I, q_E, q_B)$ and $r' = (C'_L \leftarrow C'_K \to C'_R, q' : C'_L \to C'_R)$ with $q' = (q'_I, q'_E, q'_B)$ for $h_L : IMP_L \to EXP'_L$, $h_K : IMP_K \to EXP'_K$, and $h_R : IMP_R \to EXP'_R$ so that*

1. $IMP_K \to IMP_L \xrightarrow{h_L} EXP'_L = IMP_K \xrightarrow{h_K} EXP'_K \to EXP'_L$
2. $IMP_K \to IMP_R \xrightarrow{h_R} EXP'_R = IMP_K \xrightarrow{h_K} EXP'_K \to EXP'_R$
3. $IMP_L \xrightarrow{q_I} IMP_R \xrightarrow{h_R} EXP'_R = IMP_L \xrightarrow{h_L} EXP'_L \xrightarrow{q'_E} EXP'_R$
4. $BOD_L \xrightarrow{q_B} BOD_R \leftarrow IMP_R$ *is a pushout of* $BOD_L \leftarrow IMP_L \xrightarrow{q_I} IMP_R$

then we compose component-wise $\widehat{r} := (r \circ_h r', \widehat{q}) = (\widehat{C_L} \leftarrow \widehat{C_K} \rightarrow \widehat{C_R}, \widehat{q} : \widehat{C_L} \rightarrow \widehat{C_R})$
with the $\mathcal{Q}$*-morphism* $\widehat{q} = (q'_I, q_E, \widehat{q_B})$

Proof:
We compose component-wise $\widehat{r} := r \circ r' = (\widehat{C_L} \leftarrow \widehat{C_K} \rightarrow \widehat{C_R})$ where
$\widehat{C_L} := C_L \circ_{h_L} C'_L = (IMP'_L \rightarrow \widehat{BOD_L} \leftarrow EXP_L)$,
$\widehat{C_K} := C_K \circ_{h_K} C'_K = (IMP'_K \rightarrow \widehat{BOD_K} \leftarrow EXP_K)$, and
$\widehat{C_R} := C_R \circ_{h_R} C'_R = (IMP'_R \rightarrow \widehat{BOD_R} \leftarrow EXP_R)$
We merely need to construct $\widehat{q_B} : \widehat{BOD_L} \rightarrow \widehat{BOD_R}$ as morphism induced by the pushout $BOD'_L \rightarrow \widehat{BOD_L} \leftarrow BOD_L$ of $BOD'_L \leftarrow IMP_L \rightarrow BOD_L$.
We compose the pushouts (**1**) and (**2**) and decompose the pushout (**1** + **2**) into pushout (**3**) and the commutative square (**4**). As $IMP_L \xrightarrow{q_I} IMP_R \rightarrow BOD'_R = IMP_L \rightarrow DOB'_L \xrightarrow{q'_B} BOD'_R$ we have (**4**) is pushout, and with $q'_B \in \mathcal{Q}$ we have $\widehat{q_B} \in \mathcal{Q}$.

√

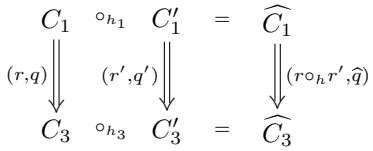
Next, the notion of independence of rules and composition is adapted for $\mathcal{Q}$-rules.

Definition 4.8 (Independence of rule-based refinement and composition) *Given the* $\mathcal{Q}$*-rules* $r = (C_L \leftarrow C_K \rightarrow C_R, q : C_L \rightarrow C_R)$ *with* $q = (q_I, q_E, q_B)$ *and* $r' = (C'_L \leftarrow C'_K \rightarrow C'_R, q' : C'_L \rightarrow C'_R)$ *with* $q' = (q'_I, q'_E, q'_B)$ *and their composition* $r \circ_h r'$ *with* $\widehat{q} = (q'_I, q_E, \widehat{q_B})$ *for the connection* $h = (h_L, h_K, h_R)$ *then we have that the composition* $C_1 \circ_{h_1} C'_1$ *is independent from* r *and* r' *if there is* $h_2 : IMP_2 \rightarrow EXP'_2$ *so that:*

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

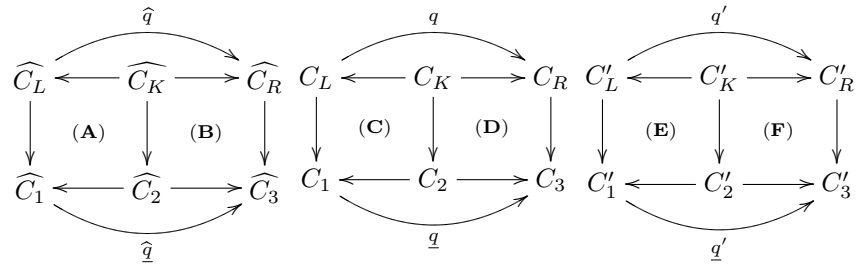
Theorem 4.9 (Compatibility Theorem for composition and rule-based refinement) *For an adhesive HLR framework for rule-based component refinement* $\mathcal{R} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M}, \mathcal{Q})$ *(see Definition 4.5) we have:*

Given the $\mathcal{Q}$*-rules* $r = (C_L \leftarrow C_K \rightarrow C_R, q : C_L \rightarrow C_R)$ *with* $q = (q_I, q_E, q_B)$ *and* $r' = (C'_L \leftarrow C'_K \rightarrow C'_R, q' : C'_L \rightarrow C'_R)$ *with* $q' = (q'_I, q'_E, q'_B)$ *and their composition* $r \circ_h r'$ *with* $\widehat{q} = (q'_I, q_E, \widehat{q_B})$ *for the connection* $h = (h_L, h_K, h_R)$ *so that the composition* $C_1 \circ_{h_1} C'_1$ *is independent from* r *and* r'*, then we have* $C_1 \overset{(r,q)}{\Longrightarrow} C_3$ *as well as* $C'_1 \overset{(r',q')}{\Longrightarrow} C'_3$ *and* $C_1 \circ C'_1 \overset{(r \circ_h r', \widehat{q})}{\Longrightarrow} C_3 \circ_{h_3} C'_3$.
This is illustrated by the following diagram:



Proof:
Analogously to the proof of the Composition Theorem 4.4 we have for the composed as well as for

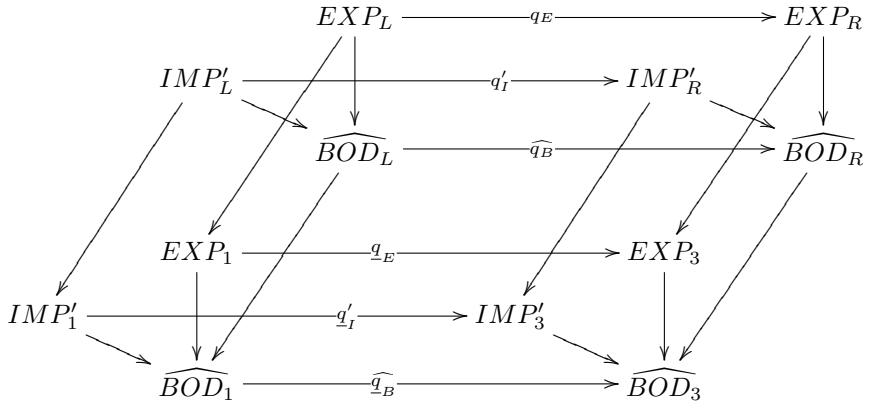
the simple $\mathcal{Q}$-transformations the following double-pushouts:



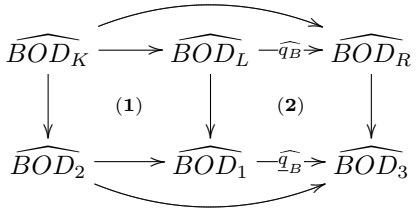
with

$\widehat{q} = (q'_I, q_E, \widehat{q_B})$ $\quad q = (q_I, q_E, q_B)$ $\quad q' = (q'_I, q'_E, q'_B)$
$\underline{\widehat{q}} = (\underline{q}'_I, \underline{q}_E, \underline{\widehat{q}_B})$ $\quad \underline{q} = (\underline{q}_I, \underline{q}_E, \underline{q}_B)$ $\quad \underline{q}' = (\underline{q}'_I, \underline{q}'_E, \underline{q}'_B)$

$\underline{\widehat{q}_B} : \widehat{BOD_1} \to \widehat{BOD_3}$ is given as the induced morphism of the pushout $BOD'_1 \to \widehat{BOD_1} \leftarrow BOD_1$ over $BOD'_1 \leftarrow IMP_1 \to BOD_1$. It remains to show that $(\mathbf{A} + \mathbf{B})$ along $\widehat{q}$ and $\underline{\widehat{q}}$ is a pushout as well.
At the level of specifications $(\mathbf{A} + \mathbf{B})$ is given in the category $\mathbf{Cat_r}$ by:



Due to $C_1 \overset{(r,q)}{\Longrightarrow} C_3$ and $C'_1 \overset{(r',q')}{\Longrightarrow} C'_3$ we have the following pushouts
$EXP_1 \xrightarrow{q_E} EXP_3 \leftarrow EXP_R$ over $EXP_1 \leftarrow EXP_L \xrightarrow{q_E} EXP_R$ and
$EXP_1 \xrightarrow{q_I} EXP_3 \leftarrow IMP_R$ over $IMP_1 \leftarrow IMP_L \xrightarrow{q_I} IMP_R$.
Pushout decomposition yields that (**2**) is pushout as well.



So, $(\mathbf{A} + \mathbf{B})$ is a pushout and we have $C_1 \circ_{h_1} C'_1 \overset{(r \circ_h r', \widehat{q_B})}{\Longrightarrow} C_3 \circ_{h_3} C'_3$ with the double pushout $(\mathbf{A}, \mathbf{B})$. $\surd$

5 The Application as a Formal Component Technique

Changing environments - commercial, technical or social - demand software systems that can be adapted to those changes with reasonable effort. In order to build up large software systems from

smaller parts, a flexible component concept for software systems and infrastructures is highly important (see e.g. [Szy97, MBE+00, GT00]). 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. In [MBE+00] 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 both, correctness and consistency can be ensured for the composed system. The presented formal description is the basis to make explicit the conditions under which a component can be changed or exchanged.

The compatibility of component transformation with component composition is a key issue of component transformation as it represents the core question of changing a component in some given context. The main results concern the conditions for transforming or refining both components and their interfaces in different ways while keeping the composition of these components intact. These results are given by the Compatibility Theorems stating that the result of first transforming respectively refining 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.

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 instantiation to algebraic high-level nets.

5.1 Instantiation to Algebraic High-Level Nets

We define algebraic high-level (AHL) nets and different notions of morphisms needed for safety preservation leading to rules and transformations that are safety preserving.

An algebraic high-level net consists – roughly speaking – of a Petri net with inscriptions of an algebraic specification $SPEC$ defining the data type part of the net. The pre- and post-domain of a transition is given by a multi-set of pairs of terms and places. Multi-sets can be considered as elements of a free commutative monoid. Morphisms between AHL nets are given by morphisms mapping places to places, transitions to transitions and mapping the data type to another one. In order to have suitable abstraction between the export and the body of an AHL net component the morphisms may map the transitions only partially. These are called t-partial AHL net morphisms.

Additionally, we define morphisms preserving safety properties of algebraic high-level nets. To be able to preserve safety properties (expressed via formulas on markings), we must take care that no new arcs are added to the context of mapped places by the morphism and no old (mapped) arcs are deleted from their context. Otherwise new transitions could add or delete tokens on "old" (mapped) places in an unpredictable way. We therefore call morphisms with these features *place-preserving*.

Definition 5.1 (Typed algebraic high-level nets) *A typed* **algebraic high-level net** $N = (SPEC, P, T, A, pre, post, type, \widehat{m})$ *consists of an algebraic specification* $SPEC = (S, OP, X, E)$*, a set of places* P*, a set of transitions* T *and a* $SPEC$ *algebra* A*. The two functions* $pre, post : T \longrightarrow (T_{OP}(X) \otimes P)^{\oplus}$ *with* $T_{OP}(X) \otimes P := \{(term, p) | term \in T_{OP,type(p)} \; ; \; p \in P\}$ *assign to each* $t \in T$ *an element of the free commutative monoid over the Cartesian product of terms* $T_{OP}(X)$ *with variables in* X *and the set* P *of places, so that the typing of the place is respected. The function* $type : P \longrightarrow S$ *assigns to each place its type, that is a sort of the specification, indicating the sort of the data elements allowed on that place. The function* $\widehat{m} : P \rightarrow A^{\oplus}_{type(p)}$ *assigns each place its marking as a sum of data elements of the corresponding sort.*

Given $N_i = (SPEC_i, P_i, T_i, A_i, pre_i, post_i, type_i, \widehat{m_i})$ *for* $1 \leq i \leq 2$ *there are the following morphism classes and categories:*

A ***t-partial AHL net morphism*** $f = (f_{SPEC}, f_P, f_T, f_A) : N_1 \rightarrow N_2$ *is given component-wise*

by the specification morphism $f_{SPEC} : SPEC_1 \to SPEC_2$, the function $f_P : P_1 \to P_2$ and the partial function $f_T : T_1 \to T_2$ and the isomorphism $f_A : A_1 \xrightarrow{\sim} V_{f_{SPEC}}(A_2)$ on the algebras such that:

- *f is persistent: $V_{f_{SPEC}}(T_{OP_2}(X_2)) \cong T_{OP_1}(X_1))$*
- *f is type preserving: $f_S \circ type_1 = type_2 \circ f_P$*
- *f is arc preserving: for all $t \in dom(F_T)$ we have*
  $(f^{\#}_{SPEC} \times f_P) \circ pre_1(t) \leq pre_2 \circ f_T(t)$ and
  $(f^{\#}_{SPEC} \times f_P) \circ post_1(t) \leq post_2 \circ f_T(t)$
  where $f^{\#}_{SPEC}$ is the extension of f_{SPEC} to terms and $(f^{\#}_{SPEC} \times f_A)$ is a generalised homomorphisms (see [EBO92]).
- *f is marking strict: $(f^{\#}_{SPEC} \times f_A) \circ \widehat{m_1} = \widehat{m_2} \circ f_P$*

AHL nets and t-partial morphisms comprise the category $\mathbf{AHL_{tp}}$.

*A **plain AHL net morphism** is a t-partial AHL net morphism such that additionally:*

- *f_T is a total mapping*
- *f is transition preserving: $(f^{\#}_{SPEC} \times f_P) \circ pre_1 = pre_2 \circ f_T$*

AHL nets and plain morphisms comprise the category $\mathbf{AHL_p}$.

*A **place-preserving AHL net morphism** is a t-partial AHL net morphism such that additionally:*

- *f_T is a total mapping*
- *f is place-preserving:*
  $\bullet(f_P(p)) = (f_{SPEC} \times f_T)^{\oplus}(\bullet p)$ and
  $(f_P(p))\bullet = (f_{SPEC} \times f_T)^{\oplus}(p\bullet)$ for all $p \in P_1$

  where the pre and post sets are defined as

$$\bullet p = \sum_{t \in T} post(t)(p) \cdot t \in (T_{OP} \times T)^{\oplus} \quad \text{and}$$
$$p\bullet = \sum_{t \in T} pre(t)(p) \cdot t \in (T_{OP} \times T)^{\oplus}$$

- *f_T, f_P and f_{SPEC} are injective.*

The class of all place-preserving morphisms is denoted by $\mathcal{Q}_{pp}$.

t-partial morphisms represent the abstraction of the component's body as given in its export interface. Place-preserving morphisms preserve safety properties (see Fact 5.4) and are hence used for the refinement along transformations.

Fact 5.2 (AHL nets as an adhesive HLR framework for rule-based component refinement) *AHL nets give rise to the following adhesive HLR framework for rule-based component refinement (see Definition 4.5) $\mathcal{R}_{AHL} = (\mathbf{AHL_p}, \mathbf{AHL_{tp}}, \mathcal{M}_{AHL}, \mathcal{Q}_{pp})$ as the following facts hold:*

1. *$\mathbf{AHL_p}$ is the category of AHL nets with plain morphisms.*
2. *$\mathbf{AHL_{tp}}$ is the category of AHL nets with t-partial morphisms and includes the category $\mathbf{Cat_p}$ as the functor $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ is an inclusion in the sense that $Obj_{\mathbf{AHL_p}} = Obj_{\mathbf{AHL_{tp}}}$.*
3. *$(\mathbf{AHL_p}, \mathcal{M}_{AHL})$ is a weak adhesive HLR category with $\mathcal{M}_{AHL}$ the class of plain strict, injective morphisms.*

4. $\mathbf{AHL_{tp}}$ *has pushouts if at least one of the given morphisms is in* $\mathsf{Inc}(Mor_{\mathbf{AHL_p}})$ *and* $\mathsf{Inc}(Mor_{\mathbf{AHL_p}})$ *is stable under pushouts.*

5. *The inclusion functor* $\mathsf{Inc} : \mathbf{AHL_p} \to \mathbf{AHL_{tp}}$ *preserves pushouts if at least one of the given morphisms is in* $\mathcal{M}_{AHL}$*.*

6. *The inclusion functor* $\mathsf{Inc} : \mathbf{AHL_p} \to \mathbf{AHL_{tp}}$ *preserves pullbacks if at least one of the given morphisms is in* $\mathcal{M}_{AHL}$*.*

7. $\mathcal{Q}_{pp}$ *is the class of place-preserving morphisms in* $\mathbf{AHL_{tp}}$ *and is closed under composition and isomorphisms.*

8. *The class* $\mathcal{Q}_{pp}$ *is stable under pushouts and closed under the construction of coproducts.*

Proof:
Item 1 and item 2 are trivial.
Item 3 can be shown similarly to Fact 4.21 in [EEPT06], where typed, guarded AHL-nets over one fixed specification are considered or it can be shown as in [PER95].
Item 4: $\mathbf{AHL_{tp}}$ has pushouts being constructed component-wise (see Fact A.1 in Appendix A).
Item 5 and 6: The inclusion functor $\mathsf{Inc} : \mathbf{AHL_p} \to \mathbf{AHL_{tp}}$ preserves limits and colimits as the corresponding universal constructions are compatible with partial morphisms and the inequation concerning the pre and post domains.
Item 7 and Item 8 are proved similarly to the proof of Theorem 4.2.1 in [PGE01]. $\surd$

Next, we give the formulas over the possible markings of an AHL net. Each atomic expression $\lambda(a, p)$ – with $p \in P$, $a \in A_{type(p)}$ and $\lambda \geq 1$ – states that at least λ tokens with value a are on the place p. The safety property $\Box\varphi$ states that φ holds for all reachable markings.

Definition 5.3 (Safety Properties, Formulas, Translations)

1. *The set of static formulas* $\mathcal{F}$ *is given inductively; we have the atoms* $\lambda(a, p) \in \mathcal{F}$ *for* $p \in P$ *and* $a \in A_{type(p)}$*. Furthermore* $(\varphi_1 \in \mathcal{F} \Longrightarrow \neg\varphi_1 \in \mathcal{F})$*, and* $(\varphi_1 \in \mathcal{F}, \varphi_2 \in \mathcal{F} \Longrightarrow \varphi_1 \wedge \varphi_2 \in \mathcal{F})$*.*
   The validity of formulas is given w. r. t. the marking of a net. Let $m : P \to A^{\oplus}_{type(p)}$ *be a marking of* N *then:* $m \models_N \lambda(a, p)$ *iff* $\lambda a \leq m(p)$*, and* $m \models_N \neg\varphi_1$ *iff* $\neg(m \models_N \varphi_1)$*, and* $m \models_N \varphi_1 \wedge \varphi_2$ *iff* $(m \models_N \varphi_1) \wedge (m \models_N \varphi_2)$*.*

2. *Let* φ *be a static formula over* N*. Then* $\Box\varphi$ *is a* **safety property***. The safety property* $\Box\varphi$ *holds in* N *under* m *iff* φ *holds in all states reachable from* m*:*
$$m \models_N \Box\varphi \Longleftrightarrow \forall m' \in [m\rangle : m' \models_N \varphi$$
   For the initial marking $\widehat{m}$ *we also write* $N \models \Box\varphi$ *instead of* $\widehat{m} \models_N \Box\varphi$*.*

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

Fact 5.4 (Place preserving morphism preserve safety properties (Thm 3.5 in [PGE01]))
A place-preserving morphism $q \in \mathcal{Q}_{pp}$ *with* $q : N \to N'$ *preserves safety properties, i.e. given a safety property* $\Box\varphi$ *we have*
$$N \models \Box\varphi \Longrightarrow N' \models \Box\mathcal{T}_q(\varphi)$$

.

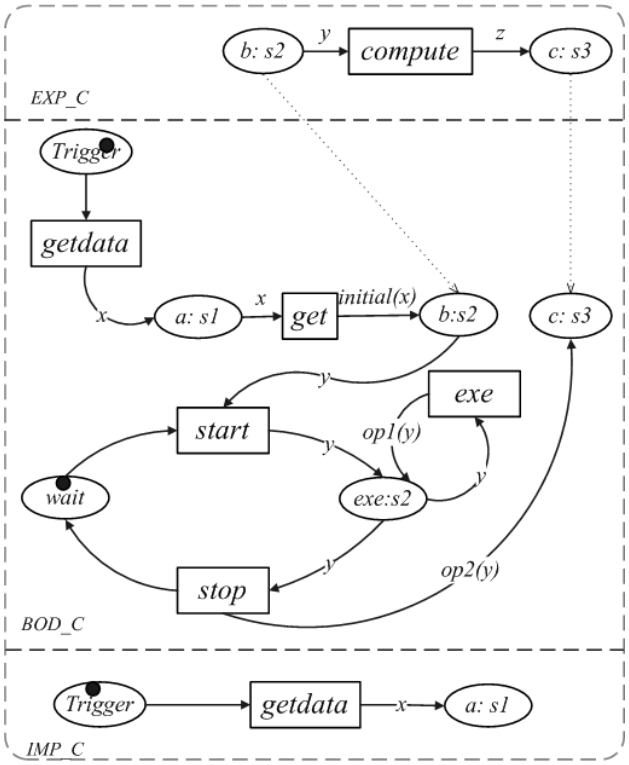
This fact can then be used to formulate the following proof rule: For the transformations $N_1 \overset{(r,q)}{\Longrightarrow} N_3$ – a $\mathcal{Q}_{pp}$-transformation – and a safety property-preserving rule (r, q) – a $\mathcal{Q}_{pp}$-rule – we now have this proof rule:

$$\frac{(r,q) \text{ is a place-preserving rule; } N_1 \text{ satisfies the safety properties } \Box\varphi}{N_3 \text{ satisfies the safety properties } \Box\mathcal{T}(\varphi) \text{ as well}}$$

This proof rule is valid as we can conclude from $N_1 \overset{(r,q)}{\Longrightarrow} N_3$ that there is $\underline{q} : N_1 \to N_3$ with $N_1 \models \Box\varphi \Longrightarrow N_3 \models \Box\mathcal{T}_{\underline{q}}(\varphi)$.

5.2 Example

The example describes two components modeling that something is done, i.e. data is computed by getting it from some other component, initialising it, executing something incrementally and so on. The illustration of an AHL net uses the following conventions: The typing of the places is depicted by $\_ : s$ where s is some sort of the corresponding specification. The pre- and post-domain of the transitions are adjacent arcs of a transition and the initial marking is given by tokens. In case of black tokens, i.e. data elements of an algebra of the specification **BLACK**, we denote with $\bullet$ the only data element and omit the arc inscriptions as well as the typing of the places. In Figure 2 an AHL net component with the corresponding specifications is given: It consists of



SP_EXP =
sorts: $s2, s3$
opns: ...
eqns: ...

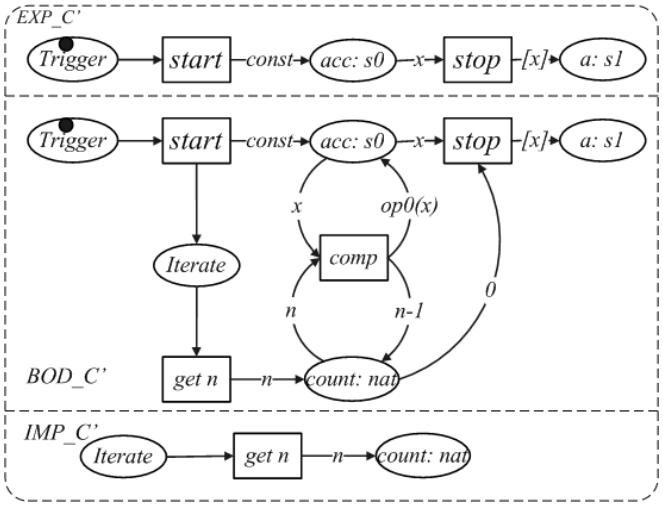
SP_BOD = **BLACK** +
sorts: $s1, s2, s3$
opns: $initial : s1 \to s2$
$op1 : s2 \to s2$
$op2 : s2 \to s3$
eqns: ...

SP_IMP = **BLACK** +
sorts: $s1$
opns: ...
eqns:

BLACK =
sorts: $token$
opns: $\bullet :\to token$
eqns: $t_1, t_2 \in X_{token}$
$t_1 = t_2$

Figure 2: AHL net component $C = (EXP\_C, IMP\_C, BOD\_C)$ with the corresponding algebraic specifications

the three AHL nets EXP_C, IMP_C and BOD_C. The export morphism $EXP\_C \to BOD\_C$ is an inclusion of the places and the undefined mapping of the transition. The import morphism $IMP\_C \to BOD\_C$ is an inclusion of the places and the transition. There is a safety property, i.e. a place invariant, in BOD_C, namely that there is always either a black token on place $wait$ or a token of sort $s2$ on place exe. This is denoted by $\varphi := \Box(\bullet, wait) xor (y, exe)$.



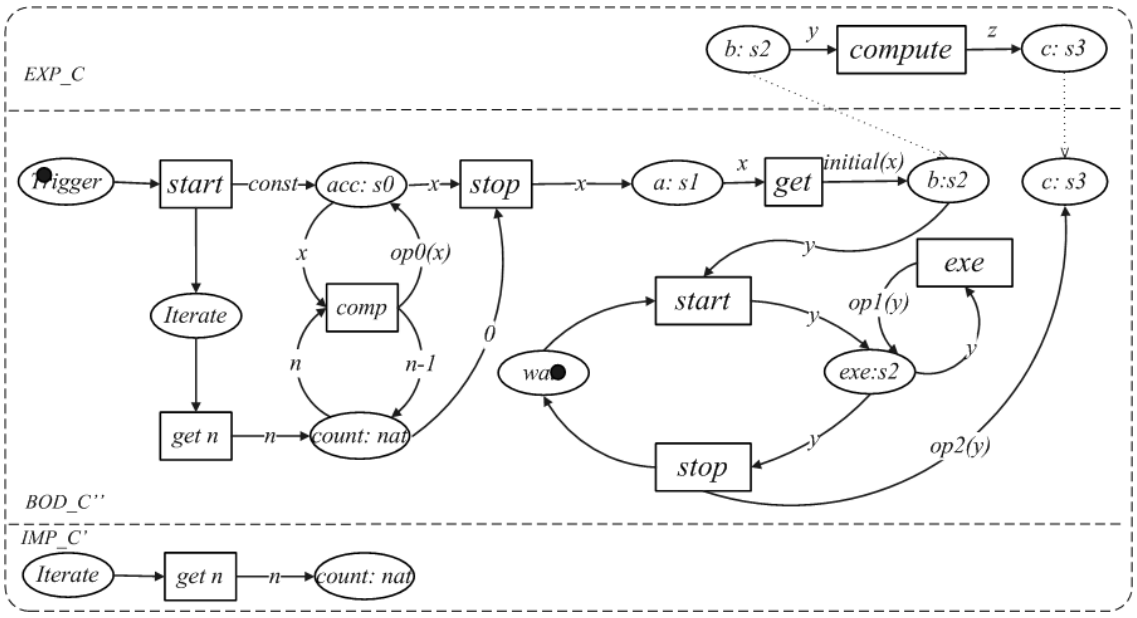
SP_EXP' =BLACK
sorts: $s0, s1$
opns: $const: \rightarrow s0$
$[\_]: s0 \rightarrow s1$
eqns: ...

SP_BOD' = SP_IMP' +
sorts: $s0, s1$
opns: $const: \rightarrow s0$
$[\_]: s0 \rightarrow s1$
$op0: s0 \rightarrow s0$
eqns: ...

SP_IMP' = BLACK + NAT

Figure 3: Second AHL net component $C' = (EXP\_C', IMP\_C', BOD\_C')$

The component $C' = (EXP\_C', IMP\_C', BOD\_C')$ models how the data is processed by taking a constant and applying an operation n times (see Figure 3).



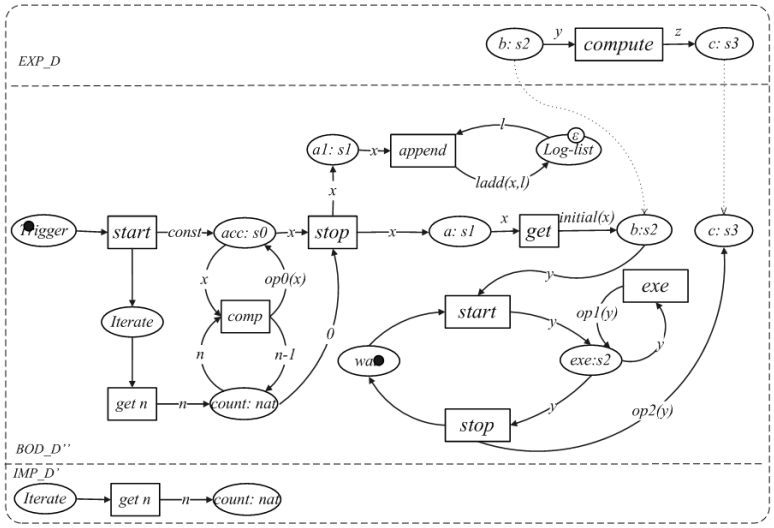
SP_BOD_C" = BLACK + NAT +
sorts: $s0, s1, s2, s3$
opns: $initial: s1 \rightarrow s2$
$op0: s0 \rightarrow s1$
$op1: s2 \rightarrow s2$
$op2: s2 \rightarrow s3$
eqns: ...

Figure 4: Composition $C'' = C \circ_h C'$ of AHL net components

The composition $C'' := C \circ_h C' = (EXP\_C, BOD\_C'', IMP\_C')$ uses a t-partial morphism $h : IMP\_C \rightarrow EXP\_C'$, an inclusion of places and a undefined mapping of the transition. The body BOD_C'' is a pushout over $BOD\_C \leftarrow IMP\_C \rightarrow EXP\_C' \rightarrow BOD\_C'$. The result of the

composition is given in Figure 4.

Next we give the place-preserving rules used for the refinement of the components. We have rule (r, q) and rule (r', q') depicted in Figure 6 and Figure 7, respectively. Rule (r', q') changes the body and the import by adding a place, to which the data token x is copied. Rule (r', q') extends the export and the body with an additional subnet, where the data token x is stored in a log list. The application of these rules yields the following refinements $C \overset{(r,q)}{\Longrightarrow} D$ (as depicted in Figure 8) and $C' \overset{(r',q')}{\Longrightarrow} D'$. Component D preserves the safety property $\varphi := \Box(\bullet, wait)xor(y, exe)$ as (r, q) is a place-preserving rule and q is an inclusion. Composing the components D and D' we obtain the component $D'' := D \circ D' = (EXP_D, IMP'_D, BOD''_D)$ depicted in Figure 5. The other way round, that is composing first the components $C'' = C \circ C'$ as well as the rules $(r'', q'') = (r, q) \circ (r', q')$, and then refining subsequently yields the same result, namely $C'' \overset{((r'',q'')}{\Longrightarrow} D''$.



SP_EXP =BLACK
sorts: $s0, s1$
opns: $[\_] : s0 \to s1$
eqns: ...

SP_BOD = SP_IMP' +
sorts: $s0, s1$
opns: $op0 : \quad s0 \to s1$
eqns: ...

SP_IMP' = BLACK + NAT

Figure 5: Result after composition and rule-based refinement: the component D''

5.3 Overview over other Instantiations

In this section we discuss examples of other specification formalisms fitting into this adhesive HLR framework for generic components.

Algebraic Specifications.

Algebraic specification modules as defined in [EM90] have been the starting point for Petri net modules and the transformation-based component concept. So they naturally fit into the presented approach. In [EM90] algebraic specification modules are given by a parameter specification, an import and an export specification that both are connected to the body specification by a specification morphisms. There are varied module operations as hierarchical composition, union, renaming, actualization and others. The semantics is proved to be compositional and is a model semantics based on the model semantics of algebraic specifications. Due to the generalization by specification frames [EBO92] this theory is available to other variants e.g. [CBEO99, JO99] as well.

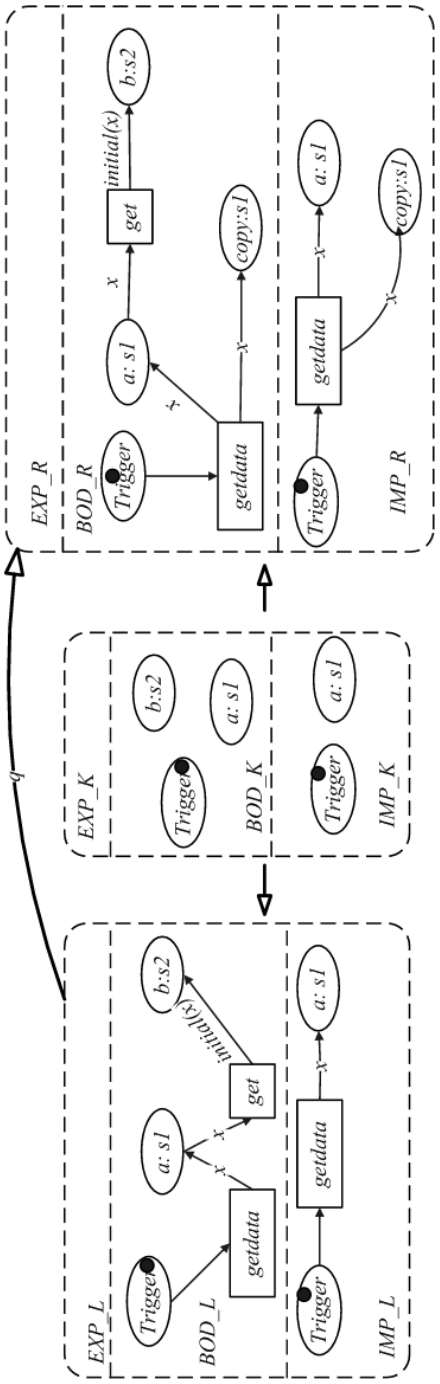


Figure 6: Rule r

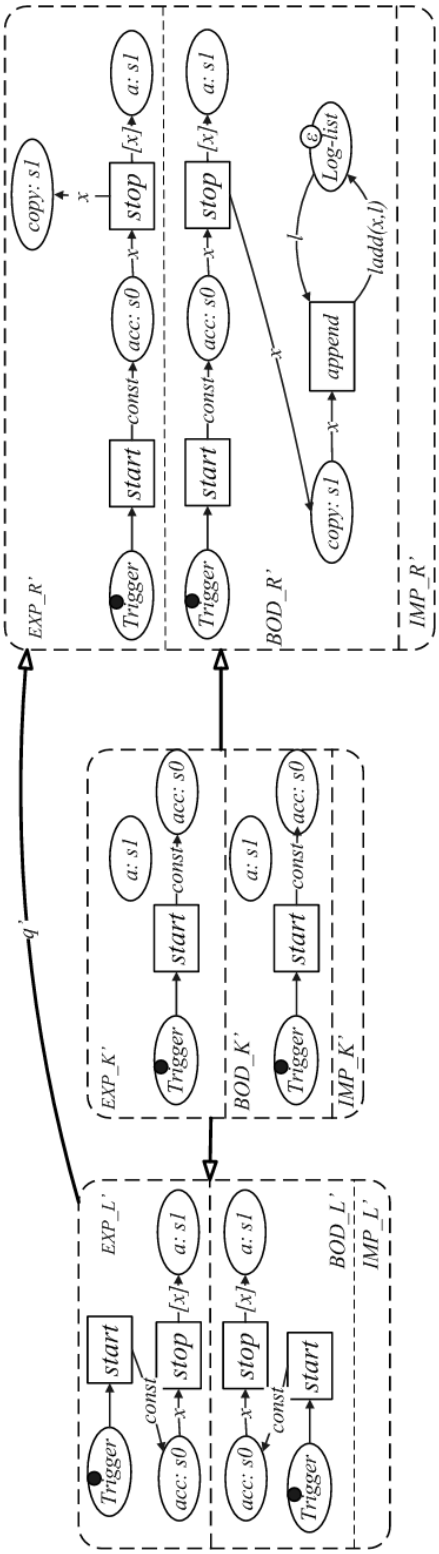


Figure 7: Rule r'

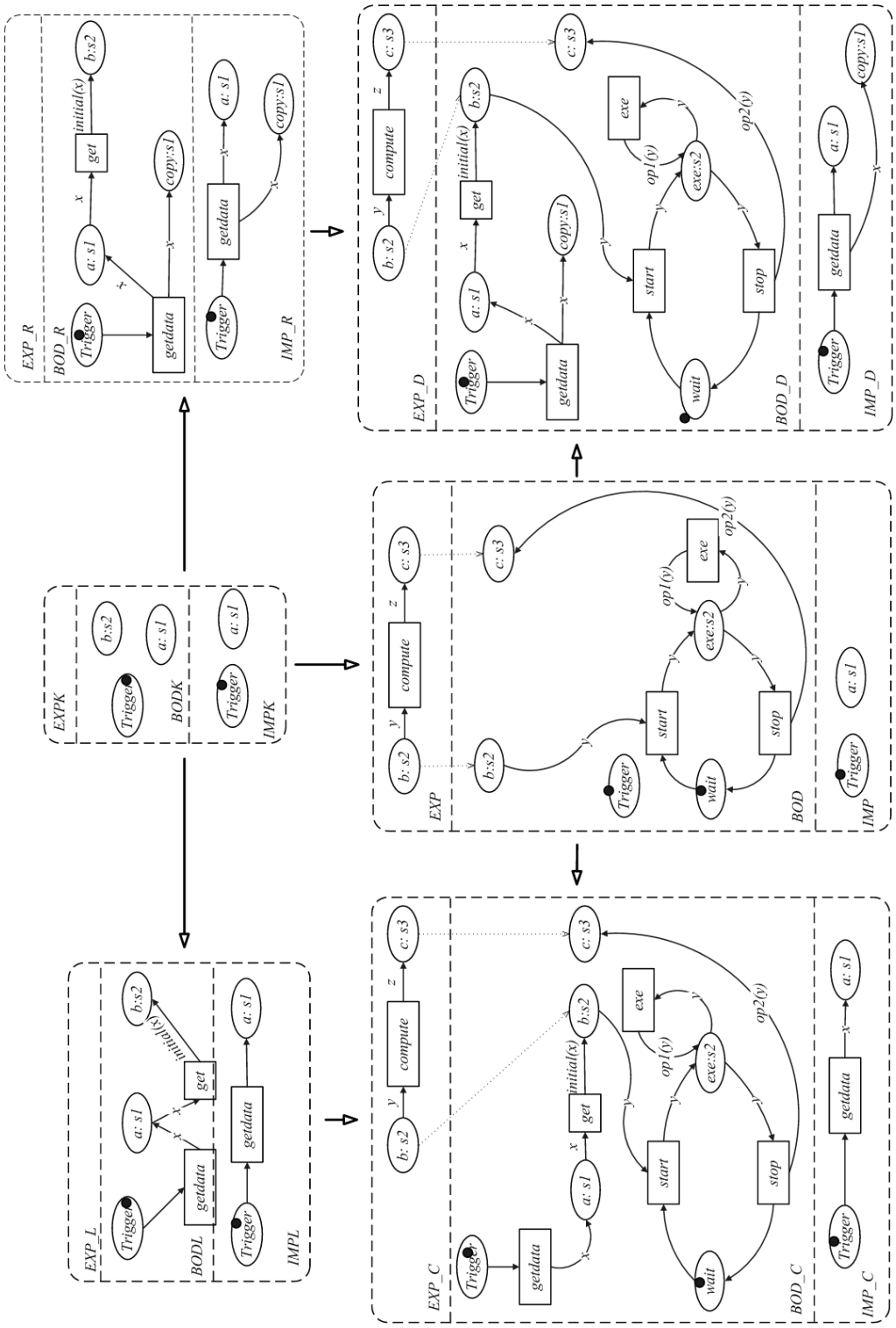


Figure 8: Rule-based refinement $C \stackrel{r,p}{\Longrightarrow} D$ of AHL net component

Graph Transformation Systems.

The transfer of algebraic specification modules as defined by [EM90] to process description techniques is a recent development. It has been started in [GPS99, Sim99, Sim02] where modules for typed graph transformation systems and local action systems have been investigated. A notion of *cat*-modules has been given that is closely related to the generic component concept. Plain morphisms as used for the mapping of the import to the body map the rules of the corresponding graph transformation system one by one. Refinement morphisms that map the export to the body allow mapping one rule to a combination of rules of the target graph transformation system. Without much doubt this approach can be transferred to other classes of graph transformations in the double pushout approach.

Automata.

In [Pad05a] we use a version of deterministic input automata where we skip the output function. 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 follower state s'. A plain morphism $f = (f_I, f_S) : A_1 \to A_2$ maps the alphabet $f_I : I_1 \to I_2$ and the set of states $f_S : S_1 \to S_2$. We use a comma category construction, namely $\mathbf{Aut_p} := (\mathsf{Prod} \downarrow \mathsf{ID})$, 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. It yields directly that $\mathbf{Aut_p}$ has pushouts and pullbacks. Refinement morphisms allow the refinement of a state by a sub-automaton of the codomain. The category $\mathbf{Aut_r}$ is given by automata and refinement morphisms. Automata components are motivated by the interface automata of [AH01]. There the automata are more complex, but the body automaton is missing. In our approach 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. The import morphism $imp : IMP \to BOD$ is a plain and the export morphism $exp : EXP \to BOD$ is a refinement morphism.

Place/Transition Nets

In [Pad02] Petri net modules - based on place/transition nets – have been introduced independently of the categorical framework discussed above. A PT net module $MOD = (IMP, EXP, BOD)$ consists of three place/transition nets, namely the import net IMP, the export net EXP and the body net BOD. Two net morphisms $m : IMP \to BOD$ and $r : EXP \to BOD$ connect the interfaces to the body. The import morphism m is a *plain morphism* and describes how and where the resources in the import interface are used in the body. 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 a generalization of plain morphisms, where a transition is mapped to a subnet.

6 Conclusion

High-level replacement systems are a categorical generalization of the algebraic approach to graph transformation systems with double pushouts. They allow formulating the same notions as for graph transformation systems, but not only for graphs but for objects of arbitrary categories. The generic concept of components is a categorical framework for modeling components, where a component consists of an import, an export and the body. The main results in this paper are the integration of both theories and results for the compatibility of hierarchical composition with transformation respectively rule-based refinement. To achieve transformations of components we have to integrate the different morphism classes used for adhesive HLR systems and generic components. This leads to an adhesive HLR framework for generic components that is a formal foundation for both component refinement and component evolution.

In [Pad05b] the connection from a component technique to the formal description of software architecture has been discussed. Most approaches used in practice have a component concept with multiple interfaces. The publications concerning the connector architectures based on the generic component approach [EPB+04, EBK+05, OP05] present of first step towards this goal. There the connectors have multiple import interfaces and the components have multiple export interfaces. The extension of the generic component technique to a family of import and a family of exports has been recently dealt with [Kle06, KPO06]. Based on components with multiple interfaces, the correspondence of the architecture graph and composition operations can be given by graph transformation of the architecture graph. The specification of software components with multiple require and provide interfaces allows multiple access to a single provide interface. Extending the component definition in this way requires the extension of the entire component technique, but rule-based transformation and refinement of components with multiple interfaces is still future work.

References

[AH01] 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.

[CBEO99] F. Cornelius, M. Baldamus, H. Ehrig, and F. Orejas. Abstract and behaviour module specifications. *Mathematical Structures in Computer Science*, 9:21–62, 1999.

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation I : Basic Concepts and Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, chapter 3. World Scientific, 1997.

[EBK+05] H. Ehrig, B. Braatz, M. Klein, F. Orejas, S. Pérez, and E. Pino. Object-oriented connector-component architectures. In *Proc. FESCA*, 2005.

[EBO92] H. Ehrig, M. Baldamus, and F. Orejas. New concepts for amalgamation and extension in the framework of specification logics. In *Proc. WADT-COMPASS-Workshop Dourdan, 1991*, pages 199–221. Springer LNCS 655, 1992.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[EHKP91] 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.

[EHPP04] H. Ehrig, A. Habel, J. Padberg, and U. Prange. Adhesive high-level replacement categories and systems. In F. Parisi-Presicce, P. Bottoni, and G. Engels, editors, *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04)*, volume 3256 of *LNCS*, pages 144–160, Rome, Italy, October 2004. Springer.

[EM90] 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.

[EOB+02] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A Generic Component Concept for System Modeling. In *Proc. FASE 2002: Formal Aspects of Software Engineering*, volume 2306 of *LNCS*, pages 32–48. Springer, 2002.

[EOB+04] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A component framework for system modeling based on high-level replacement systems. *Software and Systems Modeling*, pages 114–134, 3 2004.

[EPB+04] H. Ehrig, J. Padberg, B. Braatz, M. Klein, F. Orejas, S. Pérez, and E. 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, December 2004.

[GPS99] M. Große–Rhode, F. Parisi Presicce, and M. Simeoni. Refinements and Modules for Typed Graph Transformation Systems. In J. L. Fiadeiro, editor, *Workshop on Algebraic Development Techniques (WADT'98), at ETAPS'98, Lisbon, April 1998*, pages 137–151. Springer LNCS 1589, 1999.

[GT00] V. Gruhn and A. Thiel. *Komponentenmodelle: DCOM, JavaBeans, EnterpriseJavaBeans, CORBA*. Addison-Wesley, 2000.

[JO99] Rosa M. Jiménez and Fernando Orejas. An algebraic framework for higher-order modules. In *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems*, volume 1709 of *Lecture Notes in Computer Science*, pages 1778–1797. Springer, 1999.

[Kle06] M. Klein. *Transformation-Based Component Architectures General Framework, Instantiations and Case Study.* PhD thesis, Technische Universität Berlin, Fak. IV, 2006.

[KPO06] M. Klein, J. Padberg, and F. Orejas. Towards multiple access in generic component architectures. In *Proc. Formal Foundations of Embedded Software and Component-Based Software Architectures (FESCA 06), Satellite Event of the European Joint Conferences on Theory and Practice of Software (ETAPS)*, 2006.

[LS04] S. Lack and P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.

[MBE+00] 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.

[OP05] Fernando Orejas and Sonia Pérez. Towards architectural connectors for uml. In U. Kreowski, H.J.and Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors, *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 352–369. Springer, 2005.

[Pad99] J. Padberg. Categorical Approach to Horizontal Structuring and Refinement of High-Level Replacement Systems. *Applied Categorical Structures*, 7(4):371–403, December 1999.

[Pad02] J. Padberg. Petri net modules. *Journal on Integrated Design and Process Technology*, 6(4):121–137, 2002.

[Pad05a] Julia Padberg. Integration of the generic component concepts for system modeling with adhesive HLR systems. *EATCS Bulletin*, 2005.

[Pad05b] Julia Padberg. Specification and rule-based refinemt of software-components, 2005. Habiltiation Thesis, Technische Universität Berlin.

[PE05] J. Padberg and H. Ehrig. Petri net modules in the transformation-based component framework. *Journal of Logic and Algebraic Programming*, page 35, 2005. accepted.

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

[PGE01] J. Padberg, M. Gajewsky, and C. Ermel. Rule-based refinement of high-level nets preserving safety properties. *Science of Computer Programming*, 40:97–118, 2001. www.elsevier.nl/locate/scico.

[Sim99] M. Simeoni. *A Categorical Approach to Modularization of Graph Transformation Systems using Refinements*. PhD thesis, Università Roma *La Sapienza*, 1999.

[Sim02] M. Simeoni. An Abstract Module Concept for Graph Transformation Systems. *Electronic Notes of TCS*, 51, 2002. http://www.elsevier.nl/locate/entcs/volume51.html .

[Szy97] C. Szyperski. *Component Software – Beyond Object-Oriented Programming*. Addison-Wesley, 1997.

A Technical Details

Fact A.1 (Pushouts in $\mathbf{AHL_{tp}}$) *Given $N_i = (SPEC_i, P_i, T_i, A_i, pre_i, post_i, type_i)$ and $f_i : N_0 \to N_i$ for $0 \leq i \leq 2$, then there is a pushout $N_3 = (SPEC_3, P_3, T_3, A_3, pre_3, post_3, type_3)$ with $g_i : N_i \to N_3$ for $0 \leq i \leq 2$.*
Moreover, are plain morphisms stable under pushouts, i.e. f_1 plain implies g_2 plain.

Proof:
Given $N_i = (SPEC_i, P_i, T_i, A_i, pre_i, post_i, type_i)$ and $f_i : N_0 \to N_i$ for $0 \leq i \leq 2$, then the pushout $N_3 = (SPEC_3, P_3, T_3, A_3, pre_3, post_3, type_3)$ can be constructed by

- $(SPEC_1, A_1) \overset{(g_{1,SPEC}, g_{1,A})}{\longrightarrow} (SPEC_3, A_3) \overset{(g_{2,SPEC}, g_{2,A})}{\longleftarrow} (SPEC_2, A_2)$ is pushout of
  $(SPEC_1, A_1) \overset{(f_{1,SPEC}, f_{1,A})}{\longleftarrow} (SPEC_0, A_0) \overset{(f_{2,SPEC}, f_{2,A})}{\longrightarrow} (SPEC_2, A_2)$
  in the category of generalised algebras **GALG**.

- $P_1 \overset{g_{1,P}}{\longrightarrow} P_3 \overset{g_{2,P}}{\longleftarrow} P_2$ is pushout of $P_1 \overset{f_{1,P}}{\longleftarrow} P_0 \overset{f_{2,P}}{\longleftarrow} P_2$ in the category **Set** of sets with total mappings. m_3 and $Type_3$ are gives by the induced morphisms.

- $T_1 \overset{g_{1,T}}{\longrightarrow} T_3 \overset{g_{2,T}}{\longleftarrow} T_2$ is pushout of $T_1 \overset{f_{1,T}}{\longleftarrow} T_0 \overset{f_{2,T}}{\longleftarrow} T_2$ in the category **parSet** of sets with partial mappings.

- $pre_3 : T_3 \to (T_{OP_3}(X_3) \otimes P_3)^{\oplus}$ is given by $pre_3(t) =$
  $$\begin{cases} sup((g_{2,SPEC}, g_{2,P})^{\oplus}(pre_2(t_2)), (g_{1,SPEC}, g_{1,P})^{\oplus}(pre_1(t_1))) & ; \text{ for } t = g_{2,T}(t_2) = g_{1,T}(t_1) \\ (g_{1,SPEC}, g_{1,P})^{\oplus}(pre_1(t_1)) & ; \text{ for } t \notin g_{2,T}(T_2) \text{ and } t = g_{1,T}(t_1) \\ (g_{2,SPEC}, g_{2,P})^{\oplus}(pre_2(t_2)) & ; \text{ for } t \notin g_{1,T}(T_1) \text{ and } t = g_{2,T}(t_2) \end{cases}$$
  pre_3 is well-defined as $g_{1,T}$ and $g_{2,T}$ are jointly surjective. $post_3$ analogously.

$g_i = (g_{i,SPEC}, g_{i,P}, g_{i,T}, g_{i,A})$ are well-defined, as
$(g_{i,SPEC}, g_{i,P})^{\oplus}(pre_i(t_i)) \leq sup((g_{2,SPEC}, g_{2,P})^{\oplus}(pre_2(g_{2,P}(t_2))), (g_{1,SPEC}, g_{1,P})^{\oplus}(pre_1(g_{1,P}(t_1)))) = pre_3(g_{i,T}(t_i))$ for $1 \leq i \leq 2$.
Plain morphisms are preserved: Let $f_1 : N_0 \to N_1$ be plain, then we have for $t_2 \in T_2$ with $g_{2,T}(t_2) \notin g_{1,T}(T_1)$ directly $pre_3(g_{2,T}(t_2)) = (g_{2,SPEC}, g_{2,P})^{\oplus}(pre_2(t_2))$.
For $g_{2,T}(t_2) \in g_{1,T}(T_1)$ we have some $t_0 \in T_0$ with $f_{1,T}(t_0) = t_1$ and $f_{2,T}(t_0) = t_2$, and
$(g_{2,SPEC}, g_{2,P})^{\oplus} \circ pre_2(t_2) \geq (g_{2,SPEC}, g_{2,P})^{\oplus} \circ ((f_{2,SPEC}, f_{2,P})^{\oplus} \circ pre_0(t_0))$
$= (g_{1,SPEC}, g_{1,P})^{\oplus} \circ ((f_{1,SPEC}, f_{1,P})^{\oplus} \circ pre_0(t_0)) = (g_{1,SPEC}, g_{1,P})^{\oplus} \circ pre_1(t_1)$
Hence, we have: $pre_3(g_{2,T}(t_2)) = sup((g_{2,SPEC}, g_{2,P})^{\oplus}(pre_2(g_{2,P}(t_2))), (g_{1,SPEC}, g_{1,P})^{\oplus}(pre_1(g_{1,P}(t_1))))$
$= (g_{2,SPEC}, g_{2,P})^{\oplus} \circ pre_2(t_2)$
Moreover, g_2 is total because total morphisms are pushout stable in **parSet** $\surd$