# Formal Specification and Rule-Based Refinement of Software Components

Kumulierte Habilitation an der Fakultät IV-
Elektrotechnik und Informatik
der Technischen Universität Berlin

Dr. Julia Padberg

Lehrgebiet:
**Theoretische Informatik**

Eröffnung des Verfahrens: 14. September 2005
Verleihung der Lehrbefähigung: 24. Januar 2006
Habiltationscolloquium: 15. Februar 2006
Aushändigung der Urkunde: 10. Mai 2006

Gutachter:
Prof. Dr. Hartmut Ehrig
(Technische Universität Berlin)

Prof. Dr. Grzegorz Rozenberg
(Universiteit Leiden, Niederlande)

Prof. Dr. Fernando Orejas
(Universitat Politècnica de Catalunya, Spanien)

**Berlin, 2006**
D 83

## Abstract

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, consisting of the component description, semantics, composition operations and refinement concepts, is required for the formal foundation of component-based engineering. Such a component techniques is presented in this thesis. For the precise description of a component a suitable syntax is required, for mathematical reasoning about components a formal semantics is needed. The formal specification of the component interfaces and the component specification allows the rigorous mathematical modeling and the verification of the desired functionality. Moreover, formal component operations and their compatibility with the semantics ensure correctness and consistency not only for basic components but for composed components as well as for the entire component-based system. Different composition operations increase the flexibility when building the system. Component-based software development and maintenance need refinement concepts as a formal foundation for changing the components in the component's environment. The compatibility of component transformation with component operations is a key question of changing components in a given context, so that inconsistencies can be avoided.

Starting points for this comprehensive, formal component technique are two well established theories: High-level replacement systems and the transformation-based framework for generic components. High-level replacements systems are an abstract theory for the transformation of objects in an arbitrary category. They are a generalization of the double pushout approach to graph transformations for arbitrary categories. The fundamental concepts in the transformation-based framework for generic components are the import and export interfaces, the compositionality of components and their semantics. In this thesis we present a formal component technique, that is based on the transformation-based framework for components, but that formalizes that framework by using category theory. So we have components with export and import interfaces as well as an explicit component specification, called the body. We supply two kinds of semantics, a model-based semantics and a transformation-based semantics. The first one is constructive but requires a model semantics of the underlying specification technique. If that is not available, there is the transformation-based semantics of component, that maps each transformation of the import to a transformation of the export. This semantics is functional, but it takes directly the environment into account. For the composition of components we have two different operations, namely union and hierarchical compostion. Union is a operation that glues components together at a defined subcomponent and the hierarchical composition describes the import of another component by an importing one. The compatibility of these operations is ensured. As category theory is the mathematical basis, now the level of abstraction is the same as for high-level replacement systems. Some underlying specification category together with a few assumptions is required, but the precise specification technique remains open. The combination of these theories allows the development of component transformations, rule-based refinement of components and the corresponding results. The basic idea is that the transformation of each part of the component, that is the export, the import and the body, is a transformation in the underlying specification category. The component technique can be instantiated with a large variety of different data type as well as process specification techniques. We discuss several instantiations, algebraic specifications, automata, Petri nets and graph transformation systems. Petri net transformations and Petri net modules are investigated more deeply, in order to show that the notions and results that stem from data type specifications are feasible for process specifications as well. Nevertheless, the notions and results are available for the other instantiations as well.

## Zusammenfassung

Software-Komponenten stellen einen allgemein anerkannten Abstraktionsmechanismus dar, der während des gesamten Lebenszykluses eines Software-Systems von der Analyse bis zur Wartung eingesetzt wird. Komponentenbasierte Software-Entwicklung benötigt formale Konzepte und Modellierungstechniken, da die hohe Komplexität solcher Systeme häufig die Konsistenz dieser beeinträchtigt. Diese Komplexität resultiert im Wesentlichen aus der nichtdeterminierten und nebenläufigen Interaktion der Komponenten und ist auch ein Haupthindernis für die Anpassung an sich ändernde Systemumgebungen. Also ist eine formale Komponententechnik, wie sie in dieser Schrift vorgestellt wird und die die formale Komponentenbeschreibung, ihre Semantik, Kompositionsoperationen sowie die Verfeinerungskonzepte umfasst, nötig für die formale Fundierung der komponentenbasierten Entwicklung. Für die präzise Komponentenbeschreibung ist eine passende Syntax und für die mathematische Argumentation eine formale Semantik erforderlich. Die formale Beschreibung der Komponentenschnittstellen sowie der Komponentenspezifikation erlaubt die gründliche mathematische Modellierung und die Verifikation der gewünschten Funktionalität. Darüber hinaus stellen Kompositionsoperatoren Korrektheit und Konsistenz sowohl für Basiskomponenten als auch für zusammengesetzte Komponenten sowie das gesamte komponentenbasierte System sicher. Unterschiedliche Kompositionsoperationen erhöhen die Flexibilität während der Entwicklung des Systems. Komponentenbasierte Softwareentwicklung und -wartung braucht Verfeinerungskonzepte als eine formale Basis für die Veränderung von Komponenten in ihrer Umgebung. Die Verträglichkeit der Komponentenverfeinerung mit den Komponentenoperationen ist eine zentrale Frage der Veränderung von Komponenten in einem gegebenen Umfeld, so dass Inkonsistenz vermieden wird.

Ausgangspunkte der umfassenden, formalen Komponententechnik, die in dieser Schrift beschrieben wird, sind zwei etablierte Theorien: HLR-Systeme (*High-Level Replacement*) und der transformationsbasierte Rahmen für generische Komponenten. HLR-Systeme sind eine abstrakte Theorie zur Beschreibung der Ersetzung von Objekten in beliebigen Kategorien und stellen eine Verallgemeinerung des *Double-Pushout*-Ansatzes der Graphtransformationssysteme für beliebige Kategorien dar. Die fundamentalen Konzepte des transformationsbasierten Rahmens sind Komponenten mit Export- und Importschnittstellen, die Komponentensemantik und die hierarchische Komposition. In dieser Schrift stellen wir eine formale Komponententechnik dar, die auf dem transformationsbasierten Rahmen beruht, diesen aber kategoriell formalisiert. Also, gibt es Komponenten mit Export- und Importschnittstellen und einer expliziten Komponentenspezifikation, dem so genannten *Body*. Wir liefern zwei Semantiken, eine modellbasierte und eine transformationsbasierte. Die erste ist konstruktiv, verlangt aber, dass die zugrunde liegende Spezifikationstechnik ebenfalls eine Modellsemantik hat. Liegt diese nicht vor, dann gibt es die transformationsbasierte Semantik, die jede Transformation des Imports auf eine Transformation des Exports abbildet. Diese Semantik ist jedoch nur funktional, bezieht aber dafür die Umgebung der Komponente mit ein. Die Komposition von Komponenten wird durch zwei Komponentenoperationen, nämlich Union und hierarchische Komposition, sichergestellt. Die Union beschreibt das Verkleben zweier Komponenten entlang einer gemeinsamen Unterkomponente. Die hierarchische Komposition beschreibt, wie eine Komponente eine andere, deren Export dem Import der ersten entspricht, benutzt. Die Verträglichkeit der Operationen ist sichergestellt. Die Kategorientheorie ist dafür die Grundlage und so hat diese Theorie den gleichen Abstraktionsgrad wie die der HLR-Systeme. In beiden Fällen gibt es eine zugrunde liegende Spezifikationskategorie zusammen mit einigen Annahmen über diese Kategorie, jedoch ohne die genaue Spezifikationstechnik festzulegen. Die Kombination dieser Theorien erlaubt die Entwicklung von Komponententransformationen, regelbasierter Verfeinerung für Komponenten und der entsprechenden Resultate. Grundsätzlich wird die Transformation von Komponenten durch die Transformation des Export, des Import und des Bodys in der zugrundeliegenden Spezifikationkategorie erreicht. Wir diskutieren unterschiedliche Instantiierungen, wie algebraische Spezifikationen, Automaten, Petrinetze und Graphtransformationssysteme. Petrinetztransformationen und Petrinetzmodule werden tiefergehend untersucht, um darzustellen, dass diese Konzepte, die aus der Datentypspezifikation stammen, auch für Prozessspezifikationen geeignet sind. Nichtsdestotrotz sind die Begriffe und Ergebnisse auch in den anderen Instantiierungen verfügbar.

# Contents

# 1 Introduction

It is common understanding to use all kinds of diagrammatic and textual notations in the specification phases of software development. Roughly we can distinguish between informal, semi-formal, and formal notations and techniques [EOP00]. An informal notation has an intuitive syntax and an intuitive semantics, but no formalization is given. A semi-formal notation provides a formal syntax, but usually no formal semantics. And a formal specification techniques is has both, a formal syntax and a formal semantics, that are based some mathematical theory, e.g. set theory, logic, algebra, or category theory. Informal notations usually are highly flexible and easy to comprehend but they lack precision. A formal technique allows the precise specification with a well-defined mathematical semantics. These are the basis for formal analysis of the specification, especially concerning safety and liveness properties. Structuring and refinement are important techniques for the development and the maintenance of complex systems. They need a formal foundation to ensure sufficient compatibility results. Without these it is not clear how structuring and refinement interact and inconsistencies in the software development process are easily encountered. Today there is little doubt on the fact that formal specification is needed for improving the quality of software systems, e.g. [Mon98, Gar03]. For a recent survey on formal specifications see [KMO+05]. For the practical use of formal specifications in software development it does not suffice to have syntax and a semantics, there is usually the need for structuring, for refinement, for analysis, for simulation, for verification and so on. Here we adopt the same interpretation of *techniques* as in [ERRW03]. So, a specification technique is a specification formalism (its syntax and semantic) together with additional operations and/or methods for the just mentioned tasks.

## 1.1 Formal Specification of Components

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. The formal specification of components is proposed for various different formal techniques [Gar03]. But there is no common understanding what a component specification is independent of the underlying modeling technique. This leads to the unsatisfactory situation that for each modeling technique there is at least one component concept. Although there are many approaches available, only few are general enough to be used for different specification techniques. An important insight for *Continuous Software Engineering (CSE)* in [Web99, MBE+00] has been that component concepts can be discussed and investigated without fixing the specification technique. The transformation-based framework for generic components for system modeling has been introduced and elaborated in a series of acknowledged papers [EOB+02b, EOB+03, EOB+04, EPB+04, EBK+05, PE06]. Their core 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. This framework has been instantiated with quite different formal and semi-formal specification techniques, such as process algebras, Petri nets, UML, and automata. Usually components are introduced together with some kind of composition, in most cases it is the hierarchical composition. In the development or the maintenance of a component-based system there are various tasks at hand beyond the structuring using components. So, similar to specification techniques, component techniques are needed that have a component concept and additionally provide a variety of operations, semantics, verification means, refinement concepts and so on.

In **[PE06]**[1] we present the formal component technique that is based on a categorical formalization of the transformation-based framework for generic components. We have adopted the basic concepts and have formalized them thoroughly. Components consist of three parts: the import $IMP$, the export $EXP$, and the body $BOD$ specification. The import states the prerequisites the component assumes. The body represents the internal functionality. The export gives an abstrac-

[1] The papers that are submitted as a part of this habilitation thesis are cited in bold face.

tion of the body that can be used by the environment. The specification formalism is characterized as some arbitrary specification category. The relation between import $IMP$ and body $BOD$ is given by a suitable class of plain morphisms and export $EXP$ and body $BOD$ are related by a a suitable class of refinement morphisms. So, a component is a diagram in a specification category that has to have pushouts of plain and refinement morphism. There are two semantics available. The transformation-based semantics [EOB$^+$02b] has the advantage, that it can be directly defined for component without the existence of a semantics of the specification formalism. The model semantics of a component [**PK05**] is based on a model semantics of the underlying specification formalism. It is a functorial construction that uses free and forgetful functors. Different component operations are required for the flexible composition of components. There are union and hierarchical composition. The first allows gluing components together at a defined subcomponent, this operation is a composition of two components at the same level. The hierarchical composition has one component that uses the other one, so the first requires a behavior, a functionality or else that the second component provides. Both semantics are compatible with the composition operations [EOB$^+$02b, PK05].

So, we have a theory that serves as a formal component technique, where components are diagrams in a specification category, the component operations are constructed using colimit constructions and the semantics can be either based on a model semantics of the specification or based on a transformation concept. Moreover, we enrich this technique with component transformations and rule-based refinement of components (see next subsection 1.2).

This component technique is generic in that sense that it can be instantiated with different specification formalisms. The various specification techniques we present in subsection 3.2 have a component concept in the sense of the formal component technique. 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, various Petri net classes and deterministic automata. The basic idea for the generic component concept stem from data type specification, precisely the algebraic module specifications [EM90]. It has been used for various related algebraic specification techniques as in [CBEO99, JO99]. The transfer of algebraic specification modules as defined by [EM90] to process description techniques has been started in [Sim99], where modules for graph transformation systems and local action systems have been investigated. In [**Pad05a**] deterministic automata have been shown to be an instantiation. In this thesis we consider the instantiation to Petri nets in detail. We present Petri net modules [**Pad02a**] that have been shown to be an instantiation of the component technique in [**Pad05a**] and give a short example that is pursued through the entire thesis. The Petri net modules with safety [**Pad04**] are an instantiation as well, but additionally they preserve safety properties of the export. This feature is compositional with respect to the hierarchical composition of modules. At last we discuss the model semantics of Petri nets as introduced in [**PK05**].This semantics is known mostly for datatype specifications and is now also defined for process specifications.

In [**Pad06**] we give a survey on formal specification formalism in the description of component-based software architectures. Based on that survey we argue for a unification of the approaches and present the connector architectures [EPB$^+$04], that also belong to the generic component concept, as a first step in that direction.

## 1.2 Rule-Based Refinement of Specifications and Components

The specification of a complex software system may reach a size that is difficult to handle and may compromise the advantages of the formal specification severely. One main counter measure is to develop the specification top-down so that the first specification is a very abstract view of the system and step by step more modeling details and functionality are added. Transformations in the sense of high-level replacement systems support the step-by-step development of a specification formally. High-level replacement 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. Rules describe the required changes of a specification and their application yields the transformations of the specification. Another area, where transformations are very useful, concerns variations in the development process. Often a development is not entirely unique, but variations of the same development process lead to variations in the desired specifications and resulting systems. These variations can be expressed by different rules yielding different transformations, that are used during the step-by-step development. Pure modification of specifications is often adequate as it preserves syntactical correctness and the models semantics may need to be changed. But there is also the case that it is not sufficient, since the specification already has some desired properties that have to be ensured during further development. Verification of each intermediate specification requires a lot of effort and hence is cost intensive. But refinement can be considered as the modification of specifications preserving desired properties. Hence the verification of properties is only required for the specification, where they can be first expressed. In this way properties are introduced into the development process and are preserved from then on. Rule-based refinement modifies specifications using rules and transformations so that specific system properties are preserved. The extensions of transformations to rule-based refinement has been presented in [**Pad99**] and uses an additional class of morphisms, that preserve certain properties in the underlying specification category. These morphisms are fused with the rules and tansformations so that those preserve the properties as well. Since the first formulation of high-level replacement systems [EHKP91] a large amount of divers specification formalisms have been instantiated to high-level replacement systems, here we merely discuss those we already have mentioned for the formal component technique. Petri net transformations and rule-based refinement of Petri nets are part of the main results of this thesis, where in a series of papers [**PGE01, PU03**] as well as [PGE98, EGP98, PGH99, PG00, PHG00, UP02, BEH$^+$02] the rule-based refinement of various Petri net classes, as elementary nets, place/transition nets, algebraic high-level nets or Colored Petri nets has been established.

Software systems have to be effectively adapted to changing environments and components together with component operations are an important technique for the modeling of complex systems. To combine the advantages of a component-based structuring with the advantages of step-by-step development techniques component transformations and rule-based refinement of components are required. The transformation of components allows the change of a component by changing its interfaces and/or its body specification. Using components transformations changes of components either during development or maintenance can be described formally. A formal description of change facilitates the complex task as there are precise conditions for the correct change, as the definition of tools with a precise semantics is possible, and the change itself can be modelled. The integration of the formal component technique with adhesive HLR systems (see [**Pad05a**]) yields a thorough and extensive theory for the transformation of generic components. This theory comprises component-based structuring techniques and ensures compatibility between these and the transformations. The theory allows complex component operations, where the independence of the components is not obvious. So, the formal foundation for the distributed development of component-based systems is given. Rule-based refinement is the result from integrating the formal component technique with adhesive HLR systems and again property preserving rules and transformations are compatible with hierarchical compostion (see [Pad05b]). Compatibility with union has already been given in [**Pad99**].

So, we have achieved as the main result of this thesis a formal component technique that comprises a generic component concept, two semantics, component operations for the composition and component transformations and rule-based refinement.

There are various instantiations with different specification techniques, and in most detail the instantiation with Petri nets has been developed (e.g [**Pad02a, PU03, PE06**]). So, we have as a main result a comprehensive Petri net technique for model-based software development in the sense of [ERRW03]. This techniques that is available for different net classes comprises module concepts, module operations, compositional module semantics, additional horizontal structuring,

transformation and rule-based refinemet both for Petri nets as well as Petri net modules.

## 1.3 Justification of the Categorical Approach

Category theory (see e.g. [CAT04]) can roughly be described as a general mathematical theory of structures and systems of structures and is used not only in mathematics, but also in theoretical computer science. The advantages of category theory that are important for this thesis are among others the following:

- The specification of components is proposed for various different formal techniques. But this leads to the unsatisfactory situation that for each modeling technique there is at least one component concept. Category theory 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.
- Category theory enables an effective way for proving results. The main proof can be done at an abstract level and then the instantiations yield a large amount of results for very little proving effort. Moreover, the well-known "Diagram Chasing" is one of the main advantages as it allows carrying out proofs in a visual way (for this thesis e.g [Pad99, Pad05b]).
- Category theory helps separating the levels of abstraction. In this thesis there are three levels: The lowest level of abstraction are specifications, namely the Petri nets and Petri net modules of the ongoing example. The next level is that of specification formalisms, e.g. the Petri 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.
- Most importantly for this thesis category theory has been the basis for the integration of the generic component concept and adhesive high-level replacement systems. Both theories are formulated categorically, hence they have the same mathematical language for describing the notions and results.

## 1.4 Outline of the Thesis

Following the introduction we first relate our work to the integration paradigm for data type and process specifications. We first present the basic ideas of this paradigm [EO98] and subsequently outline the component concepts of various specification techniques. Then we discuss the relation to the component technique of this thesis and discuss recent work to formalize software architectures using the formal component technique in subsection 2.4. Section 3 first introduces the categorical formulation of the the transformation-based framework for generic components. The component concept, its semantics and the component operations are discussed. In subsection 3.2 we sketch the instantiations with algebraic specifications, graph transformations, automata and various Petri net classes. The instantiations with place/transition nets are given in more detail for Petri net modules in subsection 3.3, for safety properties in Petri net modules in subsection 3.4 and for the model-based semantics in subsection 3.5. We present an ongoing example of a production cell modeled with safety preserving Petri net modules. Section 4 deals with the transformation and rule-based refinement of specifications, first at the abstract level and then at the instantiation level. We review the basic concepts of adhesive high-level replacement systems in subsection 4.1 and its extension to rule-based refinement in subsection 4.2. Again we examine the instantiations with algebraic specifications, graph transformations, automata and various Petri net classes in subsection 4.3 and continue with a survey on our results for Petri net transformations and rule-based refinement of Petri Nets in subsection 4.4. There we illustrate transformation and rule-based refinement again in the ongoing example. In section 5 we first investigate transformations of components (subsection 5.1) and subsequently the rule-based refinement of components (subsection 5.2). Again we discuss instantiations (subsection 5.3) and

elaborate the transformations and rule-based refinement of Petri net modules (subsection 5.4). There we again present the ongoing example. The conclusion in section 6 summarizes the results and gives an outlook to further research tasks. Then the references end the summary of the thesis.

In the appendix A we first present the abstracts of those papers that are submitted as part of this thesis. Further publications that are thematically related are listed in appendix B and at last there is the publication list in appendix C.

# 2 Formal Specifications in the Integration Paradigm

In accordance with the integration paradigm the formal component technique can be instantiated with various different formal specification techniques for data types, e.g. algebraic specifications, higher order specifications and others, as well as process specification techniques, e.g. Petri nets, automata, and others. It satisfies the requirement to provide a modular structure of the system in terms of components and allows the composition of components. First we recall the classification and the integration paradigm [EO98].

## 2.1 Basic Views and Classification of Specification Formalisms

In order to obtain a first rough classification of specification formalisms we distinguish between basic formalism, concerning mainly one view of a system, and integrated formalisms taking care of different views and aspects. In this section we consider as basic views the data type and process view and give a survey of the corresponding basic specification formalisms. Let us point out that even basic specification formalisms can be used in different styles for the specification of different views of a system. However, most of them are especially useful for one particular view, which will be considered in our following classification.

**Data Type View and Formalisms**

The following specification formalisms are especially useful to specify data types and hence the functional view of systems. We distinguish the following classes of data type specification formalisms:

**Algebraic/Axiomatic Approach** This class of data type specification formalisms includes especially all kinds of algebraic and logic specification formalisms, where the axioms are ranging from classical equations, via conditional equations and Universal Horn to first and higher order logical formulas. As typical representatives of this class we consider the (classical) algebraic approach based on algebraic specifications in thesense of [EM85], and extensions of algebraic specifications, where states are defined by algebras as proposed in [DG94a, AZ95], and [Gro95, Gro97, Gro98], or processes are specified algebraically [EPB$^+$90].

**State/Model-Oriented Approaches** This class of state/model-oriented specification formalisms includes techniques like VDM [Jon90], Z [Spi89], B [Abr96] and abstract state machines [BH98], formerly called evolving algebras [Gur91]. As typical representative we consider Z. Data types are given by type definitions in Z. Data states are given by data state schemas, while data state transformations are given by operationschemas. Although Z has no specific features for processes we can also use Z in a dynamic style to define processes in layer 3 by state and operation schemas for reactive states.

**Class-Oriented Approaches** Class-oriented specification notations and formalisms are closely related to object-oriented modeling techniques, where TROLL [JSHS91] and OBLOG [Sea91] are examples of formal specification techniques and an algebraic approach is given in [PP91]. In

addition to the well-known static structure diagrams, including class and object diagrams, the universal modeling language UML also includes use case, sequence, collaboration, statechart, activity and implementation diagrams supporting mainly the dynamic view of systems. For the purpose of data type specifications we consider as typical representative class-diagrams in the sense of UML.

**Process View and Formalisms**

The process view of a system is concerned with the dynamic behavior of the system, especially with the processes or activities of a system which realize the different scenarios. We distinguish the following classes of process specification formalisms: Petri net approaches, process algebraic approaches, automata / statechart-oriented approaches, and graph transformation approaches. In each of these classes there are low level variants, where data types are only supported in a weak way by fixed data domains or alphabets, and also high level variants which are defined by integration with some data type specification technique.

**Petri Net Approaches** Petri nets are broad and widely applicated theory (e.g. [DRR04, RR98a, RR98b]). In the classical Petri net approaches, like elementary and place/transition nets, there are no distinguished data elements, but only black tokens. These approaches are called low level in contrast to high level net approaches, where we have colored tokens representing different dataelements from a given data type or domain (see [JR91]). The dynamic behavior of a place/transition net is based on the firing of transitions, where some token on the input places of a transition are removed and other ones are added on the output places. The system behavior can be given by all net processes that can be realized by the net.

**Process Algebraic Approaches** The class of process algebraic approaches includes especially Hoare's Communicating Sequential Processes (CSP) [Hoa85], Milner's Calculus for Communicating Systems (CCS) [Mil80] and the Π-Calculus [Mil89] and all kinds of variants. As a typical representative we consider CCS. CCS and similarly all the other basic approaches are based on a set of actions, process variables, expressions and equations leading to the definition of processes via a recursive set of process equations. The system behavior and architecture is mainly given by parallel composition of processes.

**Automata / Statechart-Oriented Approaches** The class of automata / statechart-oriented specification techniques includes all kinds of automata e.g input/output automata [LT87, LT89], statecharts [Har87], transition systems and event structures [Win88] which are suitable to model the process behavior of systems. Similar to the process algebraic case the basic approaches are usually based on a set of actions or basic data types only.

**Graph Transformation Approaches** The main idea of graph transformation approaches for system specification (see the "handbooks" [Roz97, EEKR99, EKMR99] or the ICGT-proceeding [CEKR02, EEPPR04]) is to model system states by graphs and state transformations by graph transformations. There are several different approaches how to model a graph transformation step. The system behavior is defined by all possible graph transformation sequences or graph processes, which can be defined similarly to Petri net processes [CMR96].

## 2.2 The Integration Paradigm

Above we have considered basic specification formalisms for the data type and process aspects of systems respectively. Now we discuss how to combine or integrate basic specification formalisms in order to handle combination and integration of different aspects. First we review the integration paradigm for the data type and process view as proposed in [EO98].

#### Integration Paradigm for Data Type and Process View

The integration of different kinds of data type and process specification techniques has become an important issue in system specification. Based on several well- known examples of integrated specification techniques we propose an integration paradigm for system specification which provides a unified approach at a conceptual level. The key idea is to consider four different layers which correspond to different kinds of integrated views of system specification. As typical examples for the integration of data type and process specification formalisms we consider algebraic high-level nets [PER95], an integration of algebraic specification and Petri nets, ?SZ [BGK98] an integration of Z and statecharts, and attributed graph transformation [ELO94], an integration of algebraic specification and graph transformation. In all these and several other examples like LOTOS [Bri89] an integration of algebraic specification and CCS, there is a common pattern how data type and process view are combined with each other. This common pattern has been formulated as an integration paradigm in [EO98, EPO01] and consists of four layers which are organized in a hierarchical way.

#### Layers of Integration

The first layer corresponds exactly to the data type view of the system. The following layers are integrated views of data type, data state and system architecture aspects, where each layer is based on the previous one.

**Layer 1: Data Types**
Layer 1 provides the description of data values and operations on data values for the system. The corresponding data type can be considered as static within the system.

**Layer 2: Data States and Transformations**
Layer 2 provides the description of data states and data state transformations. This includes all states and transformations, which in principle can occur in the system, even if the data states are non-realizable or non-reachable by the processes of the system.

**Layer 3: Processes**
Layer 3 provides processes based on layers 1 and 2. Processes are the activities of the system according to its aims. Especially they realize the scenarios which are required from the application point of view. Moreover, there are communication mechanisms for processes with each other and with the environment in the sense of concurrent systems.

**Layer 4: System Architecture**
The system architecture should provide a modular structure of the system in terms of components, where in general each component is given by data types and a set of communicating processes as defined on layer 3. Vice versa this means that there should be horizontal structuring mechanisms in each layer 1-3 leading to a module concept in layer 4 which allows to compose components with suitable notions of compositionality.

## 2.3 Component Concepts in the Integration Paradigm

The above discussed integration paradigm gives a classification and integrated view on specification formalisms and this helps to establish the conceptual similarities. Layer 4 states the need for adequate compostion techniques. In this thesis we provide such a formal component technique in form of a component concept that is generic enough to comprise various specification techniques. There are component operations, namely hierarchical composition and union, that allow the horizontal structuring. Component transformations, based on adhesive high-level replacement system, are the formal foundation for vertical structuring of the development or maintenance process. There is little doubt that the use of components gains by having a precise, mathematical foundation. Next we summarize some of the component concepts that are available in the area of formal specifications.

**Graph Transformations** In [HEET99] an extensive survey over module concepts for graph transformation systems are given. In [EE93] the first ides have been formulated in the style of algebraic module specification [EM90], demandind import and export interfaces. GRACE [HHKK00, KK02] uses an abstract definition of graph transformations that the covers the basic notions of most graph transformation approaches. In [AEH+99] a module concept is given that uses transformation units for information hiding, control, and procedural abstraction. In [Sim99, GPS99] a module concept for typed graph transformation systems has been presented that is on the concepts known from algebraic specification modules [EM90] and uses refinement relations between graph transformation systems in order to model the implementation of exported features in the body. Other module concepts are available in DIEGO (Distributed Encapsulated Graph Objects) [TS95] has a structuring concept based on the algebraic double pushout approach, and PROGRES (PROgrammed Graph REwrite Systems) [Sch97] is a graph transformation based programming languageand provides a modules in the style of UML packages [SW98]. In the CommUnity-approach [FM97, AFLW00] the architecture is given by a graph and its nodes are components that are implemented in CommUnity-programs. Composition of components and connectors are constructed as categorically, namely as colimits.

**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. [Jen92, Buc94, He96, Feh93]) as well as a variety concepts for connector mechanisms as communication, coordination or cooperation (e.g. [CH94, SB94, DJL00, DG94b]). In other approaches places and transitions of modules are merged by well-defined operations (e.g. [Kin95, BCMR91, BCM91, BS92]). Most attempts to Petri net modules (among others [CP00, DJL01, JL02]) do not provide Petri nets as interfaces. For a not so recent survey see [BC92]. 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. Object Coordination Nets [GGW99, GW01, Gie01] are somewhat more elaborate as they allow specifying simple protocols in their interfaces.

**Algebraic Specifications** In [EM90] an algebraic module concept - with self contained syntactical and semantical units and several compositional interconnection mechanisms - has been introduced. At its syntactical level there is a clear separation between import, body, and export part - connected by specification morphisms - and on the semantical level a functorial semantics transforming import into export models. As algebraic specifications have been generalized using specification frames [EBO91, EBO92] this approach can be used for other algebraic specification-techniques as well e.g. [CBEO99, JO99]. The Common Algebraic Specification Language (CASL) [BM04, CoF04] is a specification language based on first-order logic with induction that allows the specification of single software modules as well as structured specifications for the modular specification of modules.

**Process Algebras** A number of architecture description languages (e.g WRIGHT [AG96, AG97], DARWIN [MK96, MDEK95], LEDA [CPT99], AEmilia [BBS03] or PILAR [CECB05]) are based on process algebras. These approaches concentrate on the dynamic aspects of components, their behavior, their connection and communication using connectors and so on. They are only rarely used for the specification of a component and its hierarchical compoisition. Closely related to the approach here, are the connector architectures in [EPB+04] that also are based on the generic approach to components [EOB+02b]. There an instantiation with CSP is given for a generic framework for composition for components and connectors.

**Automata** The use of inpu/output automata [LT87, LT89] for the description of components and/or their interfaces is well established. In [dAH01] 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 [BOR04]

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 [MMHR04, RS02] 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 (for details see [Reu01]). These service effect automata represent the component specification and describe the relation of the interfaces, that is used for e.g. performace evaluation.

The main disadvantage of the above concepts is that do not allow a uniform approach to component and component based modeling. In view of the integration paradigm a unified component concept is more desireable, as it continues the integration of the specification concepts. In other areas uniform approaches have already been developed, e.g. in the area of Petri nets by parameterized net classes [EP97, PE01] or other approaches in [EJPR01], in the area of graph transformations by high-level replacement system [EHKP91, EEPT06] or in the area of algebraic specifications by institutions [GB84, GB92] or specification frames [EBO91, EBO92]. The formal component technique we present here covers a wide range specification techniques, from formal to informal. To illustrate the wide range we state here not only the instantiation we discuss later on, but as well techniques that are likely to be instantiated or can be considered as instantiations of a related approach (see section 2.4) to connector architectures. Instantiations of the formal component techniques as presented here are algebraic specifications, automata, algebraic high-level nets, graph transformation systems (see subsection 3.2). Statecharts [Har87] are a likely candidate for the instantiation. Statecharts as treated in [EGKP97] are shown to satisfy the HLR-conditions, so there are already rules and transformations. In [EBK+05, EPB+04, OP05] connector architectures that are based on the same generic components, have been instantiated with CSP, class digrams, state charts and sequence diagrams.

| | Data description | Process description | Integrated |
|---|---|---|---|
| Axiomatic | Alg. Spec. | | |
| Class Oriented | class diagrams* | statecharts* | |
| PN Approaches | | PT nets | AHL Nets |
| Process Algebra | | CSP* | |
| Automata | | det. automata, statecharts* | |
| GraTra | | GraTra | attr. GraTra* |

Table 1: Captured Specification Formalisms

The table 1 summarizes those specification techniques that are an instantition of the proposed component technique and those that are likely to be an instantition (marked with a *) with respect to the classification through the integration paradigm.

## 2.4 From Components to Software Architectures

The importance of architecture descriptions has become most obvious over the last decade (see e.g. [SDK+95, SG96, Gri98, HNS99, GMW97]). Various formalisms have been proposed to deal with the complexity of large software systems. In order to build up large software systems from smaller parts, a flexible component concept for software systems and infrastructures are a useful and widely accepted abstraction mechanism (see e.g. [Szy97, MBE+00, GT00]). Most architecture description languages involve in some way some formal technique. In **[Pad06]** a survey over the use of formal techniques for the description of software architectures is given. Exemplarily stated there are: Process algebras are used for various architecture description languages, e.g. Darwin [MK96, MDEK95], Wright [AG96, AG97] or AEmilia [BBS03], but none of these specifies

the component itself. SARA [EFRV86] is an early architecture description language using Petri nets for the description of the operational behavior. In [DDK01, Daw01] dualistic Petri nets are proposed. These describe the architecture using abstract representations of parallel process objects. In ZCL [dPJC00] is based on Z [WD96] a set theoretic specification language. Z schemes are used to describe the architecture structure as well as the dynamic changes. CommUnity [FM97] and COOL [Gru04] are architecture description languages that are founded on graph transformations. But in COOL no explicit component specification is given. CommUnity is the approach closest to the generic approach in this thesis.

Although there are many approaches available, only few are general enough to be used for different specification techniques. In [Pad02b] a first idea has been formalized for the graph-based description of software architectures and their transformation independently of a specification technique. This approach has been realized using the GenGEd environment [Gen] for the exemplary employment of architecture transformations in [PEB00, EBP01, BEP02]. The basic ideas of [Pad02b] have been the starting point for connector architectures [EPB+04]. In **[Pad06]** we advocate the general framework for connector architectures as given in [EPB+04] as a first step for an abstract unification of the diverging component and architecture concepts. This connector architecture framework extends the generic component framework [EOB+02b] to a specific kind of connector architectures motivated by architectural connections in the sense of Allen and Garlan [AG97]. In [EPB+04] architectures using connectors with multiple imports and components with multiple exports are considered that allow connecting one connector to several different components. They are built up from components and connectors in a noncircular way. The semantics of an architecture is defined by reduction step sequences, where in each reduction step one connector is composed with all adjacent components. The main result shows existence and uniqueness of the semantics as a normal form of reduction step sequences.

# 3 Formal Specification of Components

In this section we present our work concerning the generic concept of components in a categorical frame. We concentrate on the instantiation with Petri nets, but discuss various other instantiations with other specification techniques as well.

## 3.1 The Formal Component Technique

In **[PE06]** we present a categorical formalization of the concepts of the transformation-based framework for generic components using specific kinds of pushouts.

*3.1.1 Generic Components.* A generic framework for components $\mathcal{T} = (\mathbf{Cat}, \mathcal{I}, \mathcal{E})$ consists of an arbitrary category and two classes of morphisms $\mathcal{I}$, called import morphisms and $\mathcal{E}$, called export morphisms such that the extension conditions hold These conditions require that

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

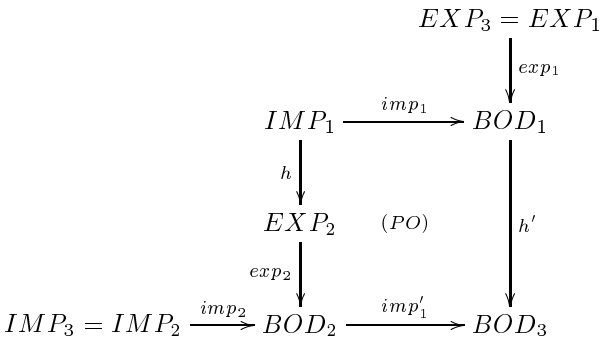
$$\begin{array}{ccc} A & \xrightarrow{e} & B \\ {\scriptstyle i}\downarrow & (\mathbf{1}) & \downarrow{\scriptstyle i'} \\ C & \xrightarrow[e']{} & D \end{array}$$

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.

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

*3.1.2 Component Operations.* We subsequently consider union and hierarchical composition. Union of components is achieved by a pushout construction, that – informally spoken – glues two components $COMP_1$ and $COMP_2$ together via some common subcomponent $COMP_0$, so the resulting component is given by $COMP_3 = COMP_1 +_{COMP_0} COMP_2$.

Hierarchical composition is a basic operation that allows composing components $COMP_1$ and $COMP_2$. It provides a connecting morphism $h : IMP_1 \to EXP_2$ from the import interface $IMP_1$ of $COMP_1$ to the export interface $EXP_2$ of $COMP_2$. We require the connector morphism to be a refinement morphism. Now we are able to define the composition $COMP_3 = COMP_1 \circ_h COMP_2$ as shown in the diagram below. Due to the requirements of a generic framework for components $\mathcal{T} = (\mathbf{Cat}, \mathcal{I}, \mathcal{E})$ the body $BOD_3$ of the component $COMP_3$ is constructed as a pushout of a plain and a refinement morphism.




The compatibility of the component operations with each other are achieved because of the colimit properties. This yields a kind of distributivity law for composition and union. Other kinds of such laws are stated informally in [**Pad02a**].

*3.1.3 Semantics.* According to the general requirements, components are self-contained units, with respect to syntax and semantics. Hence, it is necessary to have a semantics for each single component. Depending whether a semantics is available for the specification technique we have either a transformation-based component semantics or a model-based semantics

The main idea proposed in [EOB+02a] is a semantics that takes into account the environment of a component. The import interface of a component is described not by its class of models, but by its possible refinement transformations that we can find in the environment of the component. So, the semantical effect of a component is the combination of each possible import transformation, $trafo : IMP \Longrightarrow SPEC$ with the export transformation $exp : EXP \Longrightarrow BOD$ of the component. Since $IMP$ is included in $BOD$, we have to extend the import transformation from $IMP$ to $BOD$ in order to be able to compose both transformations. The transformation semantics of the component $COMP$ can be considered as a function $TrafoSem(COMP) : Trafo(IMP) \to Trafo(EXP)$, that maps transformations from the import to some specification to transformations from the export to some other specification.

In contrast to a transformation semantics we propose in [**PK05**] a model theoretic and functorial semantics as considered for algebraic module specifications in [EM90]. This semantics has the advantage to be constructive, so it is dependend on the model semantics of the underlying specification technique. Analogously to the semantics of algebraic specification we can use a contravariant functor for the mapping onto a class of (semantic) models. So we need a functor mapping the category of models over the import **Mod**(**IMP**) to the category of transition systems over the export net **Mod**(**EXP**). This semantic functor naturally depends on the morphisms that relate the interfaces to the body of the component. We define the functor $Sem : \mathbf{Mod}(\mathbf{IMP}) \to \mathbf{Mod}(\mathbf{EXP})$ by $Sem = V_r \circ F_m$. $V_r$ and $F_m$ are constructed using the morphisms $r$ and $m$. Now, we have the model semantics for components analogously to [EM90]. Each model of the import net is mapped

to the corresponding model of the export. This semantics takes some a model of the import specification and then constructs freely along the plain morphism $m : IMP \to BOD$, yielding a model of the body specification $BOD$. Then it forgets along the forgetful functor $V_r$ the internal details of the body and only represents that part of the model that is specified by the export $EXP$. Based on this notions we then obtain directly: internal and model correctness, compositional semantics with respect to component operations as union and hierarchical composition (as given for Petri net modules [Pad02a]).

## 3.2 Instantiations

*3.2.1 Algebraic Specifications.* Algebraic specification modules as defined by [EM90] have been the starting point for Petri net modules and the transformation-based component. So they fit nturally l into this approach. In [EM90] algebraic specification modules are given by parameter specification, an import specification, an export specification and body specification, that are connected by specification morphisms. There are various module operations as hierarchical composition, union, renaming actualization and others. The semantics is proven to be compositional and is a model semantics that is based on the model semantics of algebraic specifications. Due to the generalisation by specification frames [EBO91, EBO92] this theory is available to other variants e.g. [CBEO99, JO99] as well.

*3.2.2 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 [CMR96] 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 mapthe export to the body allows mapping one rule to combination of rules of the target graph transformation system. Without much doubt other classes of graph transformations in the double pushout approach are easy to be instantiated with.

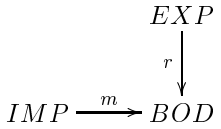
*3.2.3 Automata.* In **[Pad05a]** (and in detail in [Pad05b]) we use a version of deterministic input automatat, that is closely related to automata in [EP72]. But here we skip the output function. Basically, an automaton $A = (I, S, \delta : I \times S \rightharpoonup S)$ consists of the input alphabet $I$, 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 followers state $s'$. A plain morphisms $f = (f_I, f_S) : A_1 \to A_2$ maps the alphabet to the alphabet $f_I : I_1 \to I_2$ and the set of states to the set of states $f_S : S_1 \to S_2$. 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 subautomaton of the codomain. The category $\mathbf{Aut_r}$ is given by automata and refinement morphisms. 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.

*3.2.4 Abstract Petri Nets.* The component concept has been shown for place/transition nets in **[Pad05a]** and for algebraic high-level nets in [EOB+02b]. Abstract Petri Nets [Pad96] are an abstract frame that comprises various Petri net classes, e.g. elementary nets, place/transition nets, Colored Petri nets, algebraic high-level nets and others. Since it is an abstraction of algebraic high-level nets using specification frames it is straightforward to achieve the component technique at that level as well. This is the reason that in our papers we use the terminus Petri net modules (see below) because the notions can be easily transferred to various Petri net classes.

## 3.3 Petri Net Modules

In [**Pad02a**] 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 resents 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 transitions to subnets. 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 [Pad03], where we have developed a Petri net model of an automated telephone service center with a variety of telephone services.

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 $r : EXP \to BOD$ 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 mapped to a subnet.




*3.3.1 Module Operations.* In [**Pad02a**] we have introduced hierarchical composition and union. But note, there we have used specific conditions to ensure component-wise construction of pushouts with one plain morphism and one substitution morphism. In [PE06] we have used pushouts with one plain, injective morphism and one substitution morphism. And in [Pad05b] we have pushouts with one plain morphism and one substitution morphism, that can always be constructed, but are less intuitive for pushouts of one non-injective, plain morphism and one substitution morphism.

*3.3.2 Semantics.* In [**PE06**] we give a transformation-based semantics, but as there is meanwhile an model semantics available for Petri nets (see below in subsection 3.5) we directly have a model semantics due to the instantiation. So, we have a formal component technique for Petri nets that comprises component operations, namely hierarchical composition and union, as well as a model semantics.

*3.3.3 Example: Production Cell.* As an ongoing example we use a (most simplified) version of a production cell, consisting of a conveyor belt and an robot. The conveyor belt transports unformed metall blanks towards the robot until a photoelectric sensor indicates that a blank has its position for the robot to take it up. The belt stops and the robot takes the blank, moves it into the press. Concurrently the press opens and the right form is chosen. Then

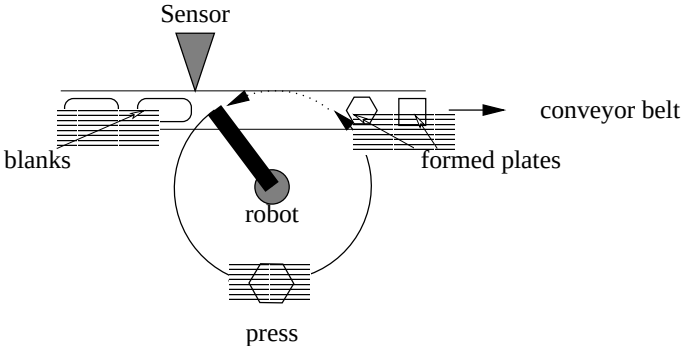
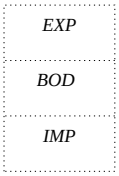



Figure 1: Production Cell

the blank gets forged in the press[2]. Next the robot puts the formed plate back to the belt and indicates that the belt can start moving again.

We present a simplified version of this production cell, that consists of two modules, that are composed hierarchically. Subsequently, both modules get modified in section 5.4. We use a notation, where the component is given as a box as depicted adjacently.




The module *MOD_Robot* is depicted in figure 2, where the export abstracts from the robots body net *BOD_R* with respect to the moving of the metal blank into the press. The import net *IMP_R* only models moving the blank to the robot and moving the formed plate away. The module *MOD_Belt* in figure 3 models the control of the conveyor belt, the export models moving the blank to the robot and moving the formed plate away. The body model that the sensor indicates the stopping of the belt and that putting the metal plate to the belt start the belt again. The import net models the sensor. In order to have different interfaces the import of module *MOD_Belt* models the stopping and starting of the belt explicitly, whereas the import of the module *MOD_Robot* does not. For the hierarchical composition the connecting morphism lets the export *EXP_B* of the imported module be an refinement of the import *IMP_R* of the importing module. So, the composition yields the module *MOD_Cell* in figure 4, where the body *BOD_RB* is obtained by gluing *BOD_R* and *BOD_B* together.

## 3.4 Safety Properties in Petri Net Modules

In [**Pad04**] this approach is extended to include safety properties. Safety properties in [**Pad04**] are formulas over markings and have translations along morphisms. An axiomatic expression is $\lambda p$, denoting that $\lambda \in \mathbb{N}$ tokens are on place $p$. We then can build logic formulae over such axioms, e.g. $4a \Longrightarrow 2b$ formalizes the statement that 4 tokens on place $a$ imply 2 tokens on place $b$. Safety properties describe invariants of the net behavior. So we use the henceforth operator $\Box$ to express that a formula shall hold for all reachable markings.

We ensure specific conditions that guarantee morphisms preserving safety properties. These are substitution morphisms that are place preserving: Any transition of the target net that has a place of the source net in its pre- or post-domain (i.e. there is an ingoing respectively an outgoing arc between the transition and the place) needs to have a source transition in the source net, so that the pre-domain and post-domain of the transition is preserved. In order to preserve safety properties the marking on the mapped places needs to stay the same. Places that are not in the image of the morphism may be marked arbitrarily. The notion of place-preserving morphisms is quite restrictive. But in order to preserve any safety property, we need to ensure that each transition of the target net has no effect or the same effect as one of the original transitions. This allows the definition of Petri net modules that preserve safety properties from the export net to the body net. As we desire a treatment of properties that is independent of the body net, we can use safety property preserving morphisms to relate the export net to the body net. If we require $r : EXP \to BOD$ to be safety property preserving, then any safety property holding in $EXP$ is preserved. A module $MOD = (IMP, EXP, BOD)$, where $r : EXP \to BOD$ is safety property preserving, is called a module with safety properties. The main intention [**Pad04**] is to simplify compositional reasoning by preserving safety properties throughout the construction of modules and its main result states that the composition of modules preserves safety properties as well.

*3.4.1 Example: Production Cell.* Both modules depicted in figure 2 and figure 3 are modules with safety properties, as the for both modules the substitution morphisms from the export to the body is place-preserving, that is, places that are present in the export do not get a new adjacent arc in the body. So, we have the safety property given as the temporal logic formula $\varphi = \Box R\_wait \vee R\_Press$ that holds in the export as well as in the body. Since the export morphism $EXP\_B \to BOD\_B$ is place-preserving as well, we can immediately follow that the safety condition $\varphi$ holds for the module *MOD_Cell* as well.

[2]For the sake of simplicity we treat robot and press as one single device.

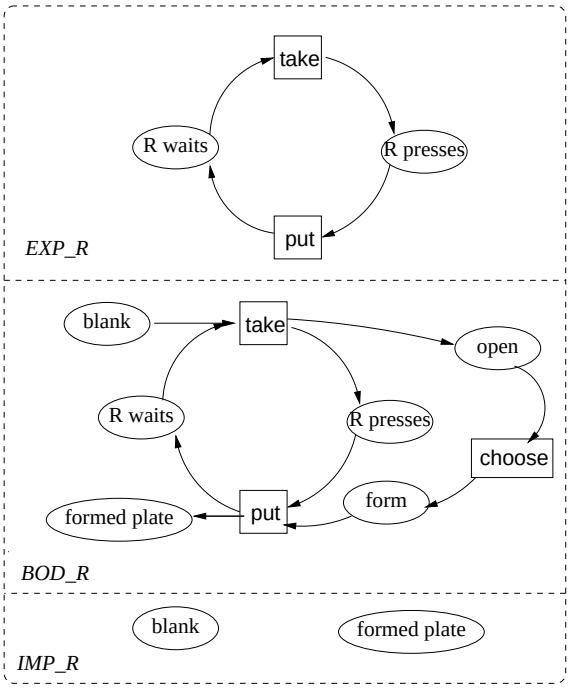



Figure 2: Module *MOD_Robot*

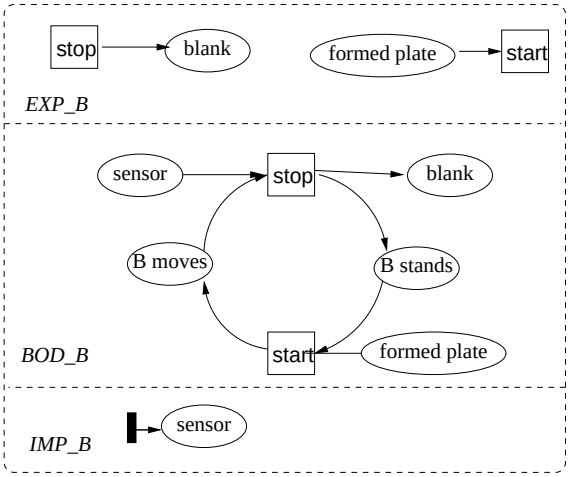



Figure 3: Module *MOD_Belt*

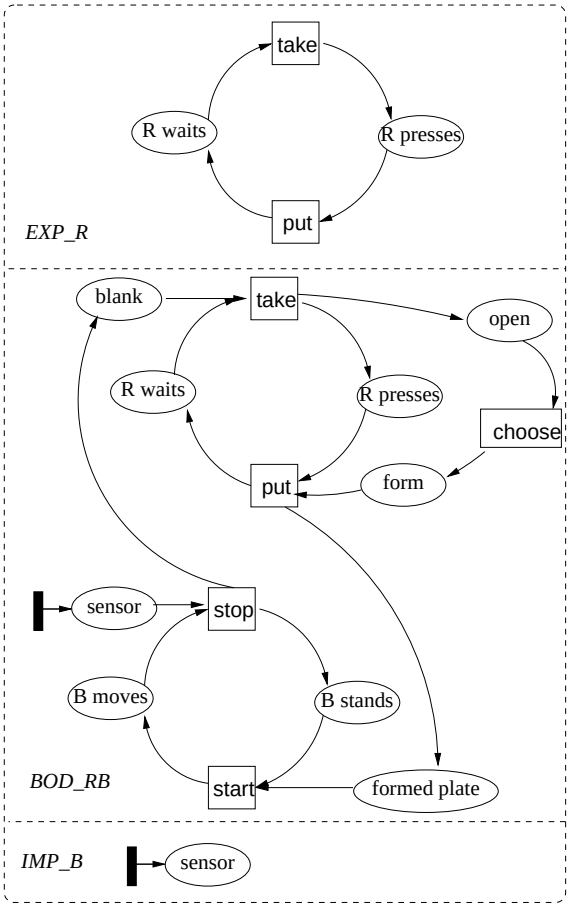


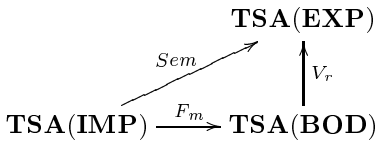

Figure 4: Module *MOD_Cell*

## 3.5 Model Semantics for Petri Nets and Petri net Modules

In [**PE06**] a transformation-based semantics for Petri net modules has been introduced based on the transformation-based approach to generic components [EOB+02b]. There the semantics is defined based on all transformations the import may undergo. The advantage of our approach in [**PK05**] is that it is constructive: The semantics of a module is based on the loose semantics presented in [**PK05**]. The reachability graph is a standard model of a place/transition net describing all possible sequences of firings of transitions starting from an initial marking. The loose semantics in [**PK05**] generalizes this net semantics in such a way that a firing of a transition can be refined by sequences of events. Moreover, we allow alternative possibilities for each such refinement. Typical examples of alternative sequences are the interleavings of independent events. Altogether, the loose semantics of a place/transition net consists of the class of transition systems

with alternative sequences of events including the reachability graph. The transitions systems can be viewed as models of a net, where a refinement of the enabled transitions and the representation of the states relate the net to the transition system. In particular, we allow refinements of transitions to be equivalence classes of alternative sequences. This class forms a category in a natural way. In [**PK05**] we show that each plain morphism between two place/transition nets induces a free construction between the corresponding semantic categories.

So we have the functor mapping the category of transition systems over the import net **TSA**(**IMP**) to the category of transition systems over the export net **TSA**(**EXP**).




Analogously to the model semantics for component this semantic functor (see [**PE06**]) depends on the morphisms that relate the interfaces to the body of the module. We define the functor $Sem$ : **TSA**(**IMP**) $\rightarrow$ **TSA**(**EXP**) by $Sem = V_r \circ F_m$. $V_r$ and $F_m$ are constructed using the morphisms $r$ and $m$. Now, we have the model semantics of Petri net modules as well.

# 4 Transformation and Rule-Based Refinement of Specifications

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. That means, instead of replacing one graph by another one, now one object is replaced by another one. Due to the categorical formulation of high-level replacement systems the focus is not on the structure of the objects but on the properties of the category.

## 4.1 Basic Concepts of Adhesive High-Level Replacement Systems

High-level replacement systems constitute a well-developed framework to use category theory for computer science (see also [EHKP91, EL93]. Based on formal grammars, more specifically on graph grammars [Ehr79], high-level replacement systems give a categorical description of an abstract rewriting technique. They have been motivated by the transfer of notions and results from graph grammars to other specification techniques, such as algebraic specifications in [EP91], High-level replacement systems can be considered as a general description of replacement systems, where a left-hand side of the rule is replaced by a right-hand side in the presence of an interface. These kinds of replacement systems have been introduced in [EHKP91] as a categorical generalization of graph transformations in the DPO-approach. High-level replacement systems are formulated for an arbitrary category **Cat** with a distinguished class $\mathcal{M}$ of morphisms, called $\mathcal{M}$-morphisms.

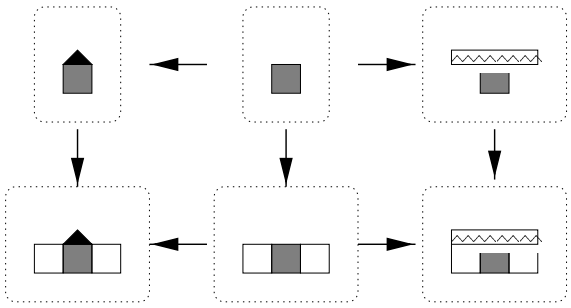
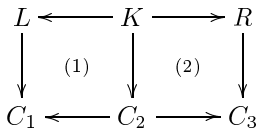


Figure 5: Abstract Example

Figure 5 illustrates the main idea for some arbitrary specification or structure. The rule given in the upper line describes that a black triangle is replaced by a long dotted rectangular, if there is a light grey square below the triangle. The transformation is given by the bottom line, where the replacement specified by the rule is carried out.

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,[3] and the morphisms $K \rightarrow L$ and $K \rightarrow R$ belong to a subclass of monomorphisms $\mathcal{M}$. 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 as pushout constructions (1) and (2) leading to a double pushout as depicted adjacently. An application of a rule is called direct transformation and describes the change of an object by the application of the rule. A sequence of these rule applications yields a transformation.




Adhesive high-level replacement (HLR) categories and systems are introduced as a new categorical framework for transformations in a broad sense in [EHPP04], which combines the well-known concept of HLR systems with the new concept of adhesive categories introduced by Lack and Sobocinski [LS04]. High-level replacement systems have been introduced in [EHKP91] to generalize the well-known double pushout approach from graphs [Ehr79] to various kinds of high-level structures, including also algebraic specifications and Petri nets. For a survey on graph and Petri net transformations see [EP04]. Basic results, like the local Church-Rosser, Parallelism and Concurrency Theorem can be achieved in weakly adhesive categories. Lack and Sobocinski [LS04] have introduced the notion of adhesive categories. They have presented the concept of *van Kampen squares*, short VK squares. Roughly spoken a VK square is a pushout square which is stable under pullbacks. The key idea of adhesive categories is the requirement that pushouts along monomorphisms are VK squares. in [EHPP04] we combine the advantages of HLR and of adhesive categories by introducing the new concept of (weak) adhesive HLR categories. Roughly spoken an adhesive HLR category is an adhesive category with a suitable subclass $\mathcal{M}$ of monomorphisms, which is closed under pushouts and pullbacks. Adhesive HLR categories are closed under product, slice, coslice and functor category constructions and that most of the important HLR properties of [EHKP91] are valid.

*4.1.1 Parallelism* 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 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. More details for horizontal structuring or parallelism are given in see [PER95] or [**Pad99**].

*4.1.2 Concurrency and pair factorization* 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, (see chapter 6 in [EEPT06])

*4.1.3 Embedding and local confluence* 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 (see chapter 6 in [EEPT06]).

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

## 4.2 Rule-Based Refinement

The abstract rewriting in HLR systems yields a stepwise modification of models. The need to develop a model in several steps arises as soon as large models are needed for the specification of a system. Our approach - as it is called - is based on rules and provides a visual way of expressing the development steps of a net. But modification alone is often not sufficient, so we extend in [**Pad99**] the existing approach. Since the model describes some desired system properties these need to be guaranteed after each development step. Verification of each intermediate model requires a lot of effort and hence is cost intensive. Obviously the idea of refinement concerns the modification of models so that system properties are preserved. Hence the verification of those properties needs only to be done once when introduced. Rule-based refinement modifies models using rules so that specific system properties are preserved. Preservation of system properties by a transformation is to be understood in the following way: If a model has a certain system property then the transformed model has the corresponding property as well. Preservation of system properties is of interest in many applications as it allows omitting the tedious verification of system properties at different stages of the development.

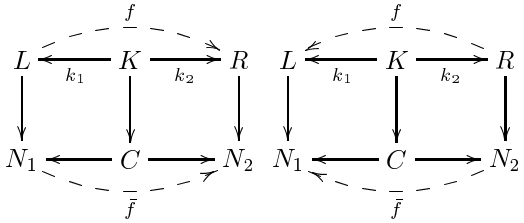
In [**Pad99**] we introduce into the theory of high-level replacement systems this notion of refinement that is motivated by refinement concepts from Petri nets. The concept of refinement is an important technique within software engineering for the stepwise development of systems. The refinement in high-level replacement is given by a morphism of a suitable category, relating the left with the right hand side of the rewriting rule. This category allows morphisms that are more complex than those that are given in the underlying HLR-category. We require here that the HLR-category is a subcategory of that category. This allows on the one hand the rule-based construction of transformations and on the other hand a description of the relation between the original and the refined parts. The compatibility between these, transformation and refinement is given by the theorems concerning sequential and parallel transformations. Thus the theory of high-level replacement systems is extended in a consistent way.

*4.2.1 Property Preserving Rules.* A pair $(r, f)$ is a property preserving rule, if $r = (L \xleftarrow{k_1} K \xrightarrow{k_2} R)$ is a rule with morphisms $k_1, k_2 \in \mathcal{M}$ and with

- either a property preserving morphism $f : L \dashrightarrow R$ s.t. $f \circ k_1 = k_2$.
- or a property respecting morphism $f : R \dashrightarrow L$ s.t. $f \circ k_2 = k_1$.

According to the notion of property preserving morphisms and rules, we can now define property preserving transformations. The general idea is that the application of a rule that preserves properties leads to a net transformation that also preserves these properties.

*4.2.2 Property Preserving Transformations.* Given a property preserving rule $(r = (L \leftarrow K \rightarrow R), f)$ with $f : L \rightarrow R$ being a property preserving morphism ($f : R \rightarrow L$ being property respecting, respectively). Then the direct transformation $C_1 \overset{(r,f)}{\Longrightarrow} C_2$ is a property preserving transformation[4] with a property preserving (respecting, respectively) morphism $\bar{f} : C_1 \dashrightarrow C_2$ ($\bar{f} : C_2 \dashrightarrow C_1$, respectively). The graphical representation of such transformations is depicted below.




Property preserving transformations represent a strong theoretical result with an obvious impact to the applications. The interpretation of the results is usually stated in form of so-called proof rules. The proof rule indicates the property (stated under the line) which can be derived for a system when certain assumptions (stated above the

[4] provided the morphisms satisfy certain assumptions

line) are fulfilled. For the property preserving transformations $N_1 \Longrightarrow N_2$ we have the following proof rule:

$$\frac{(r, f) \text{ is a property preserving rule; } N_1 \text{ satisfies the corresponding property } \mathfrak{P}}{N_2 \text{ satisfies the property } \mathfrak{P} \text{ too}}$$

Horizontal structuring techniques for high-level replacement systems, namely union and fusion have been introduced in [PER95] using pushouts and coequalizers. Compatibility between refinement and these structuring techniques is also discussed in [**Pad99**]. The compatibility results are crucial to meet one of the main demands in software development technology. This concerns the requirement for specification formalisms that allow horizontal (union and fusion) as well as vertical (refinement and transformation) structuring and provide sufficient criteria for the consistent use of both structuring dimensions.

## 4.3 Instantiations

*4.3.1 Algebraic Specifications.* The transformation of algenraic specifications in the sense of the DPO approach has been frst introduced in [EP91].Transformations of algebraic specification have been investiagted in [EGP99] in detail. there algebraic specifiaction modules are considered to be a transformation themselves. recently it has been shown that algebraic specifications with strict injective morphims are an adhesive category [EEPT06].

*4.3.2 Graph Transformation Systems.* Instantiation of adhesive HLR Systems (see e.g. [EEPT06]) comprises graph and typed graph transformation systems, hypergraph transformation systems and typed attributed graph transformation systems. In all these cases we have graphs and rules that yield the transformation system. Transformation of graph transformation systems s has been used [EHKZ05, ETB05] and they are based on transformations of HLR-systems in [PP96, PP01].

*4.3.3 Automata.* The automata given in section 3.2 are a comma category. Since weak adhesive HLR categories can be constructed as comma categories we immediately the instntiation with automata based on the category $\mathbf{Aut_p}$ as well.

## 4.4 Transformations and Rule-Based Refinement of Petri Nets

Petri nets transformations have been directly formulated as an instantiation of high-level replacement systems with place/transition nets in [EHKP91]. The framework of abstract Petri nets has been shown to be an HLR-category in [Pad96], so at that abtract level rules, transformations and the results mentioned above are provided. We directly obtain a theory of net transformations for elementary nets, place/transition nets, algebraic high-level nets and Colored Petri nets. Based on the net transformations in [**Pad99**] property preserving net transformations are introduced. Net transformations can be considered to preserve syntactical correctness, but usually they do not preserve semantical properties of the net. But this is not necessarily a disadvantage, as many steps of the development describe entirely new parts or change existing parts of the net radically. Then the preservation of the semantics is not appropriate. Rule-based refinement has been developd in several net classes based on net transformations. For a survey see [**PU03**].

*4.4.1 Example: Production Cell.* Here we illustrate a rule $r_{PN}$, that modifies the net for the conveyor belt in the following way: When the belt stops, the blanks shall be tested. Either they satisfy the quality standards or they need to be aborted and the belt has to start moving again. Being at the specification level in this section the rule is aplied to *BOD_B*. This rule is used furtheron to transform the entire module *MOD_Belt*. In Figure 6 we depict the transformation $BOD\_B \overset{r_{PN}}{\Longrightarrow} BOD\_B1$. It is a quite simple rule, as nothing is deleted only a few places and transitions are added.

Our work on rule-based refinement of Petri nets has been part of the research project "DFG-Forschergruppe PETRINETZ-TECHNOLOGIE". In **[PU03]** we present our results concerning Petri net transformations that preserve system properties. These properties comprise safety properties as well as liveness of Petri nets. The formal foundation is expressed in a categorical way in order to achieve an approach that is valid for different net classes. Rules and transformations of Petri nets are given by an instantiation of high-level replacement systems. Here we use the corresponding notions of replacement systems for various net classes, e.g. place/transition nets [PGH99], algebraic high-level nets **[PGE01]**, or Colored petri Nets [PG00, PU03].

Petri nets are an adequate specification technique for behavioral aspects of a system. So, the desired properties of the system to be specified usually concern the behavior of the model. These properties can be expressed in various ways, e.g in terms of Petri nets (as liveness, boundedness etc.), in terms of logic (e.g. temporal logic, logic of actions etc.) in terms of relation to other models (e.g. bisimulation, correctness etc.) and so on. Up to now we have focused on liveness of Petri nets and on safety properties in the sense of temporal logic. Liveness of nets means that no deadlock and even livelock of a net can occur, i.e. there always exists a firing sequence which enables any chosen transition from any reachable marking. A safety property is expressed by a logic formula stating facts about markings of a net. A formula is given in terms of numbers of tokens on places. For a place/transition net the static formula $2\mathbf{d} \wedge 3\mathbf{a}$ is true for a marking $m$ where at least 2 tokens are present on the place **d** and at least 3 tokens on the place **a**. The always operator $\Box$ in a safety property $\Box(2\mathbf{d} \wedge 3\mathbf{a})$ requires that the static formula $(2\mathbf{d} \wedge 3\mathbf{a})$ is true for all reachable markings from $m$.

*4.4.2 Example: Production Cell* The rule $r_{PN}$ is not place-preserving, as the place blank has no adjacent transition in the left-hand net, but in the right-hand net, the place blank is in the predomain of two transitions.

The next rule $r_{pp}$, depicted in figure 7, adds to the net *EXP_R* a counter that counts the number of formed plates that are put back to the conveyor belt. This rule is place-preserving and hence ist the transformation $EXP\_R \stackrel{r_{pp}}{\Longrightarrow} EXP\_R1$. So the temporal logic formula $\varphi = \Box R\_wait \vee R\_Press$ is preserved and holds in *EXP_R1* as well.

Summarising, we state that the extension of High-level replacement systems to rules and transformations preserving properties has the following impact on Petri nets:
Rule-based refinement comprises the transformation of Petri nets using rules while preserving certain net properties. For Petri nets the desired properties of the net model can be expressed, e.g in terms of Petri nets (as liveness, boundedness etc.), in terms of logic (e.g. temporal logic, logic of actions etc.) in terms of relation to other models (e.g. bisimulation, correctness etc.) and so on. We have investigated the possibilities to preserve liveness of Petri nets and safety properties in the sense of temporal logic.
We have for place/transition nets, algebraic-high level nets and Coloured Petri nets the following results for transformations and rule-based refinement presented in table 2. For more details see **[PU03]**.

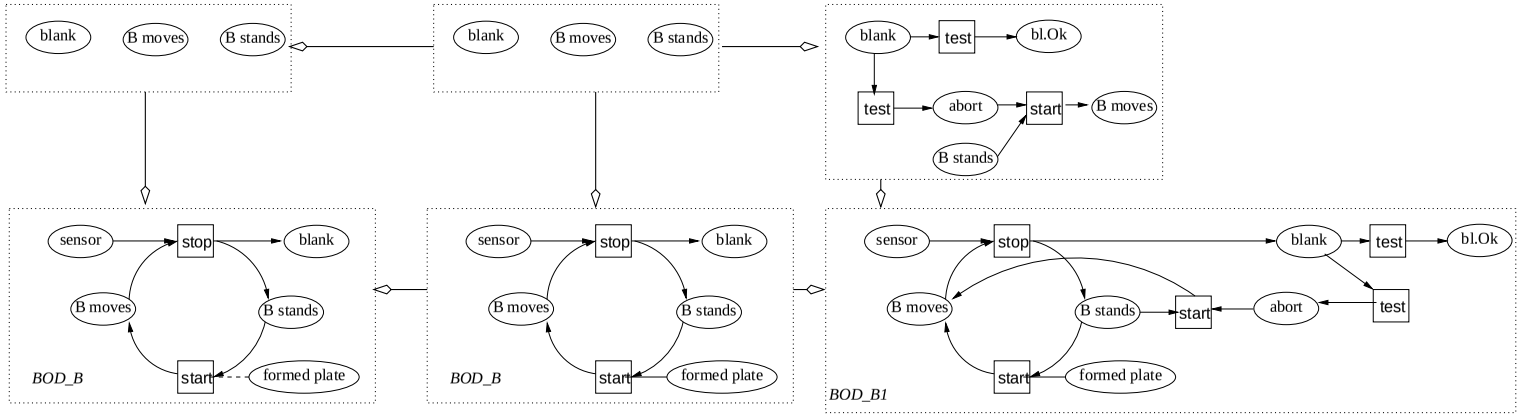



Figure 6: Transformation $BOD\_B \overset{r_{PN}}{\Longrightarrow} BOD\_B1$

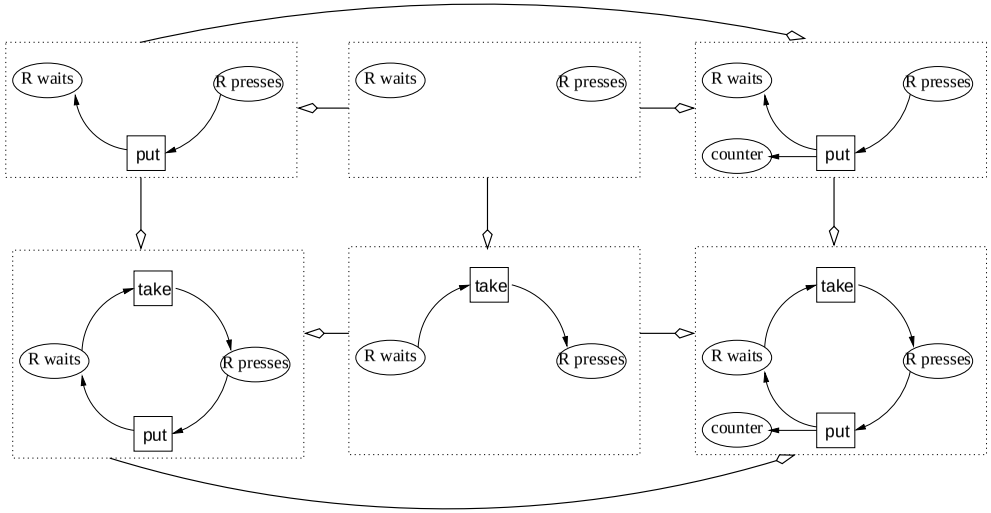



Figure 7: Transformation $EXP\_R \overset{r_{pp}}{\Longrightarrow} EXP\_R1$

| Notion/Results | PT-nets | AHL-nets | CPNs |
|---|---|---|---|
| Rules, Transformations | √ | √ | √ |
| Safety property preserving transformations with<br>transition-gluing morphisms<br>place-preserving morphisms | <br>√<br>√ | <br>√<br>√ | <br>√<br>√ |
| Safety property introducing transformations | √ | √ | √ |
| Liveness preserving transformations | √ | ? | ? |
| Liveness introducing transformations | √ | ? | ? |
| Church Rosser I + II Theorem | √ | √ | √ |
| Parallelism Theorem | √ | √ | √ |
| Union | √ | √ | √ |
| Fusion | √ | √ | √ |
| Union Theorems I+II | √ | √ | √ |
| Fusion Theorem | √ | √ | √ |

**Table 2.** Results for Petri net transformations and rule-based refinement [**PU03**]

# 5 Transformation and Rule-Based Refinement of Components

In this section we sketch how the notions of transformation and rule-based refinement can be carried over to components.

## 5.1 Transformations of Components

In [**Pad05a**] the integration of the transformation framework for generic componets with adhesive high-level replacement systems yields tranformations of generic components. Basically the idea is that the transformation of each part of the component, i.e. the export, the import, and the body, is a a transformation in the underlying specification category. Naturally, there is the precondition that the specification category is 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 involved morphism classes: as the morphism needed for the adhesive HLR system are usually simpler as the refinement morphisms we have defined the class of refinement morphisms forming a supercategory of the morphisms used for the construction.

Therefore we have to extend the generic approach in section 3.1 by relating the morphism classes used for the transformation and the components.

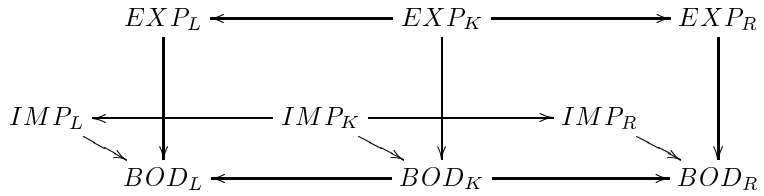
*5.1.1 Adhesive HLR framework for generic components.* This leads to the adhesive HLR framework for generic components. The adhesive HLR framework for generic components $\mathcal{A} = (\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M})$ consists of $\mathbf{Cat_p}$ the category of specifications with plain morphisms, $\mathbf{Cat_r}$ the category of specifications with refinement morphisms and $\mathcal{M}$ a subclass of plain monomorphisms. $\mathbf{Cat_r}$ has to be a supercategory of $\mathbf{Cat_p}$, there has to be the functor $\mathsf{Inc} : \mathbf{Cat_p} \to \mathbf{Cat_r}$ an inclusion in the sense that $Obj_{\mathbf{Cat_p}} = Obj_{\mathbf{Cat_r}}$. $(\mathbf{Cat_p}, \mathcal{M})$has to be a weak adhesive HLR category. and $\mathbf{Cat_r}$ has to have pushouts sa well as pullbacks where at least one morphism is

in $Inc(Mor_{\mathbf{Cat_p}})$. The adhesive HLR framework for generic components $(\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M})$ is related to the generic framework by choosing the import morphisms to be plain morphisms, i.e. $\mathcal{I} = Inc(Mor_{\mathbf{Cat_p}})$, and the export morphisms to be refinement morphisms, i.e. $\mathcal{E} = Mor_{\mathbf{Cat_r}}$ (see [**Pad05a**]).

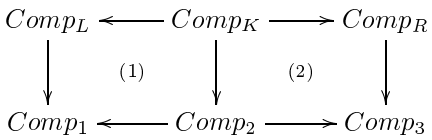
A component morphism is then given by three morphisms of the underlying specification category, ie. $comp = (comp_I, comp_B, comp_E)$ where we have $comp_I : IMP_1 \to IMP_2$, $comp_B : BOD_1 \to BOD_2$, and $comp_E : EXP_1 \to EXP_2$. The compability condition is straightforward, namely the morphisms between the parts of the component have to be compatible with the import and the export morphism.

In [**Pad05a**] was shown that given an adhesive HLR framework for generic components $(\mathbf{Cat_p}, \mathbf{Cat_r}, \mathcal{M})$ the category **Comp** of components is an weak adhesive HLR category. This yields directly the following notions.

*5.1.2 Rules.* A rule is given as usual on the level of components by two morphisms $Comp_L \leftarrow Comp_K \rightarrow Comp_R$. But at the level of the specifications we have the following situation:




*5.1.3 Transformations.* A transformation is again given by two pushouts (1) and (2) in the category **Comp** of components.




Transformations of components are then given by the double-pushout again, and we achieve directly the results holding for adhesive HLR-categories, namely Church-Rosser Theorems, Parallelism Theorem, Concurrency Theorem and so on (see section 4.1).

*5.1.4 Compatibilty.* As adhesive HLR system have been instantiated with generic components new questions of compatibility arise. Namely, are the operations at the level of components compatible with the transformation concept. 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. One of the main results in [**Pad05a**] 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 guaranties 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. Stated informally and provided a suitable indepence condition holds, we have:

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.

## 5.2 Rule-Based Refinement

As the assumptions for the property preserving transformations as given in section 4.2 hold in any adhesive HLR category (see [EEPT06]) 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. This is a direct result of the extension to property preserving transformation as sketched in section 4.2. So, again 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 $COMP_1 \Longrightarrow COMP_2$ and a property preserving rule $(r, f)$ we have the following proof rule:

$$\frac{(r,f) \text{ is a property preserving rule; } COMP_1 \text{ satisfies the corresponding property } \mathfrak{P}}{COMP_2 \text{ satisfies the property } \mathfrak{P} \text{ too}}$$

## 5.3 Instantiations

*5.3.1 Algebraic Specifications.* If we have the case that $\mathcal{E} = \mathcal{I} = Mor_{\mathbf{ASPEC}}$ as in the case of algebraic specifications then we have directly that the category of components is a weakly adhesive HLR category, where the class $\mathcal{M}$ is given componentwise by the corresponding class of the underlying specification category. This is due to the fact that the category of components is a functor category over any diagram functor having the adjacent diagram as the domain. So, we directly have rules and transformation for algebraic specifivcation modules. Moreover, the results as presented before hold as well.

*5.3.2 Graph Transformation Systems.* The graph transformation modules in the sense of [GPS99, Sim99, Sim02] are an instatiation of the generic component concept and typed graph transformations, on which the modules are based, are adhesive HLR systems [EEPT06]. As there are already transformations of HLR systems [PP96, PP01], it is obvious, that transformations of graph transformation modules can be achieved straightforwardly as well.

*5.3.3 Automata.* The instantiation with determinstic automata as given in section 3.2 yields that $(\mathbf{Aut_p}, \mathcal{M})$ with $\mathcal{M}$ the class of injective morphisms is an adhesive HLR-category, due to the comma category construction. Automata component categories satisfy the adhesive HLR framework for generic components (see paragraph 5.1.1) where $\mathcal{A}_{Aut}$ consists of $\mathbf{Aut_p}$ the category of automata with plain morphisms, $\mathbf{Aut_r}$ category of automata with refinement morphisms and $\mathcal{M}$ the class of plain, injective morphisms, see [**Pad05a**] and in more detail [Pad05b].

*5.3.4 Petri Nets.* Again, abstract Petri nets are HLR-systems (see paragraph [Pad96]) and there is a comprehensive theory on rule-based refinement of Petri nets. So, the instantiation with abstract Petri nets is straighforward and yields transformations and rule-based refinement for various classes of Petri net components.

## 5.4 Transformation and Rule-Based Refinement of Petri Net Modules

In [Pad05a] is shown that Petri net modules (see section 3.3) are an instantiation of the adhesive HLR framework for generic components (see 5.1.1) that consists of the category of place/transition nets with plain morphisms, the category of place transition nets with subsititution morphisms and the class of plain, injective morphisms.

*5.4.1 Example: Production Cell.* We first transform both modules applying to each one rule. Subsequently we illustrate that applying the composed rule to the composed module yields the same result, thus demonstrating the compatibilty of hierarchical composition with transformation. The module *MOD_Belt* as depicted in figure 3 is extended by introducing a separate quality test for the metal blanks before the robot takes them. Either it is good enough or it is aborted. Then the belt has to start moving again. In Figure 8 the rule $r_B$ for changing the module is depicted. It consists of three modules, each with export, body and import. In the export part, the new transitions testp and testp as well as the new places bl.OK and abort are added. The names of places and transitions indicate the mappings.The transformation of the body part corresponds to the Petri net transformation depicted in figure 6. The result of the $MOD\_Belt \overset{r_B}{\Longrightarrow} MOD\_Belt\_2$

is depicted in Figure 9.

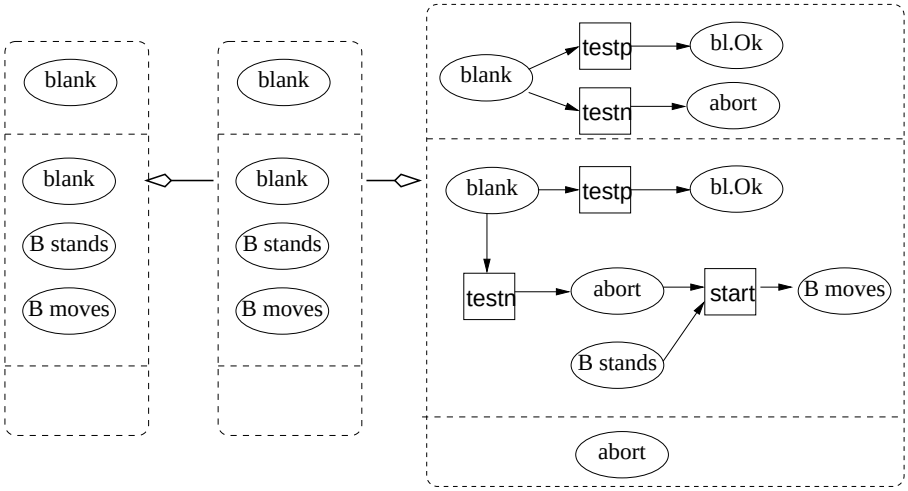



Figure 8: Rule $r_B$

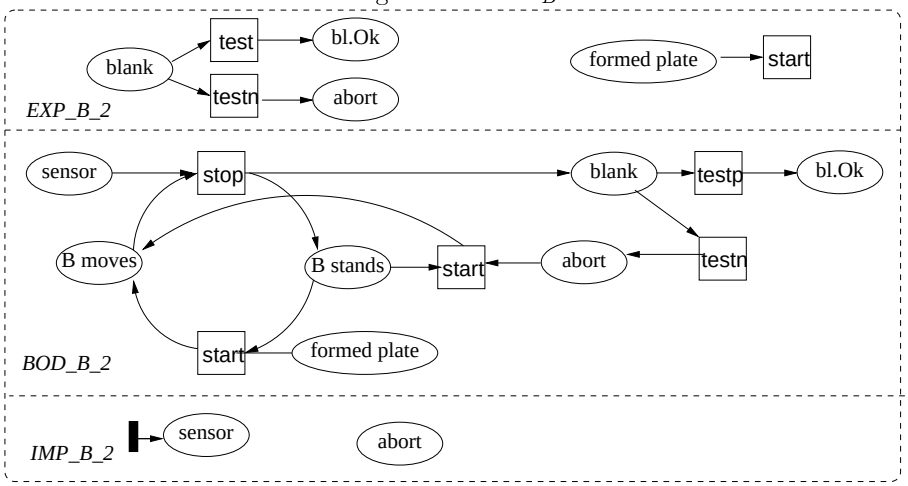



Figure 9: Module *MOD_Belt_2*

The next rule $r_R$ (in figure 10) is applied to the module *MOD_Robot* (see in figure 3). The resulting transformation first deletes the place blank, then the new transition test and the new place bl.OK are added. The tranformation $MOD\_Robot \overset{r_R}{\Longrightarrow} MOD\_Robot\_2$ yields the module depicted in figure 11. The composition of $MOD\_R\_2 \circ MOD\_B\_2$ yields the module in figure 12. Compatibility of hierarchical composition with transformation is a central result for changing componenent within their environment. The compatibilty conditions that the rules are composable and that nothing is deleted that the importing module requires, are satisfied in our example. The composed rule $r_R \circ r_B$ is given in figure 13. So we can directly transform $MOD\_Cell \overset{r}{\Longrightarrow} MOD\_Cell\_2$ using the rule $r = r_R \circ r_B$ as depicted in figure 13, where $r$ is constructed by hierarchical composition with $r_R \circ r_B$.

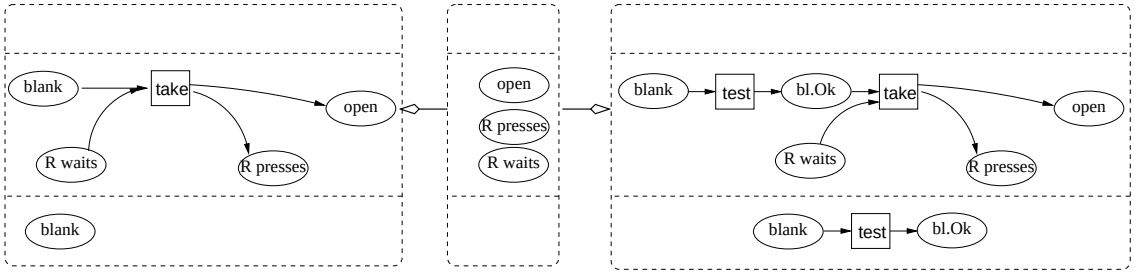



Figure 10: Rule $r_B$

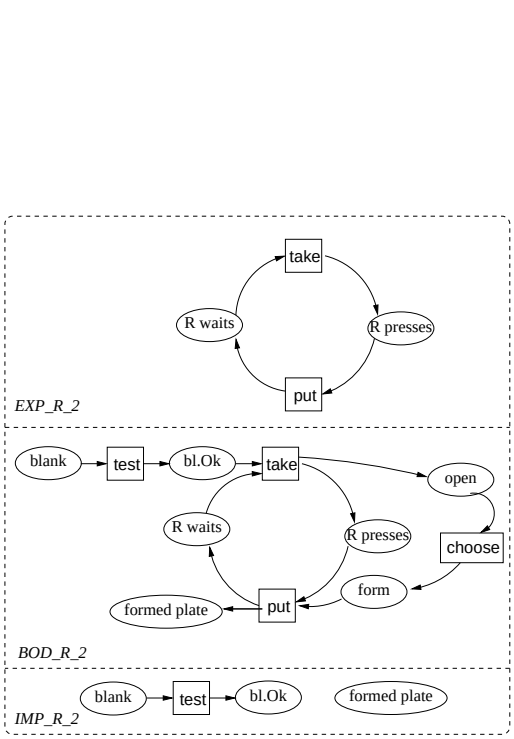



Figure 11: Module *Mod_Robot_2*

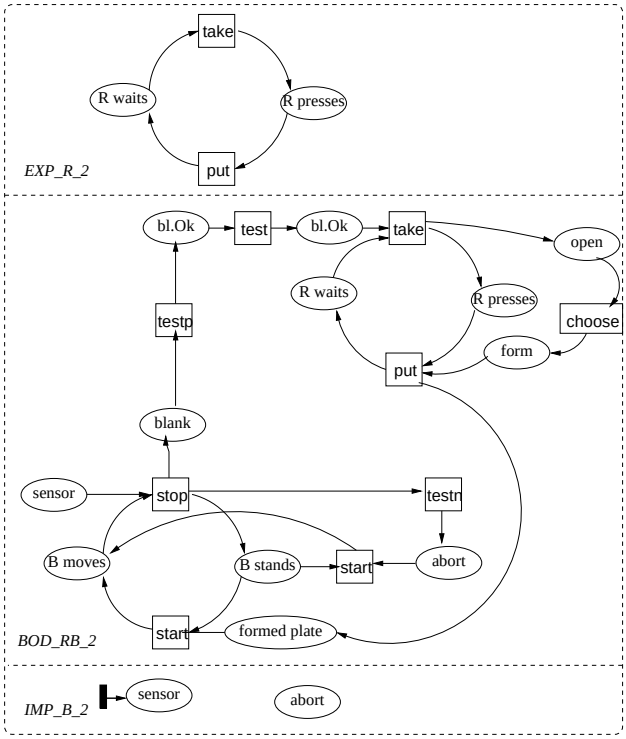



Figure 12: Module *MOD_Cell_2*

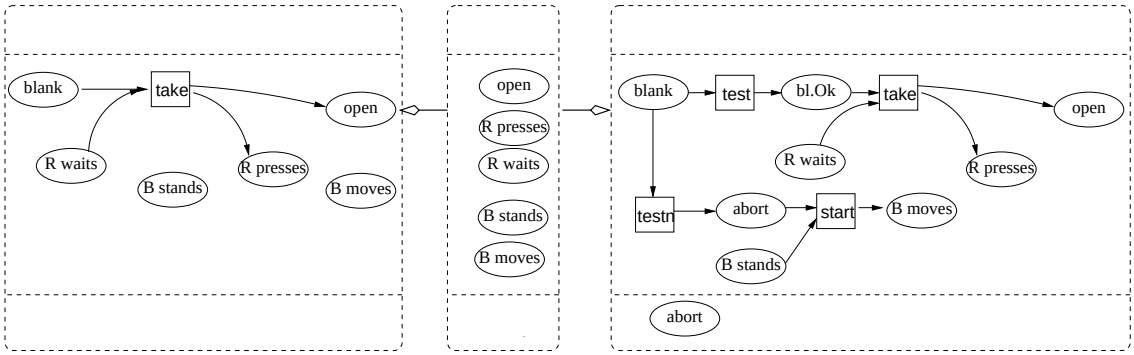



Figure 13: composed rule $r = r_R \circ r_B$

Summarizing we have the following situation:

$$\begin{array}{ccccc} MOD\_Robot & \circ & MOD\_Belt & = & MOD\_Cell \\ \Big\Downarrow{\scriptstyle r_R} & & \Big\Downarrow{\scriptstyle r_B} & & \Big\Downarrow{\scriptstyle r_R \circ r_B} \\ MOD\_Robot\_2 & \circ & MOD\_Belt\_2 & \cong & MOD\_Cell\_2 \end{array}$$

So, even within the composed module *MOD_Cell_2* in figure 12 the transformations yield the same result as applied independently to the corresponding component. This means that no undesired side-effects of the transformation can happen.

*5.4.2 Rule-based refinement of Petri net modules with safety.* Petri net modules with safety properties in section 3.4 can now be preserved during transformation by using safety preserving rules as sketched in section 4.4 for the import, export and body part of the module.

*5.4.3 Example: Production Cell.* Obviously, the resulting module *MOD_Cell_2* still satisfies the safety property $\varphi = \Box R\_wait \vee R\_Press$. Unfortunately, the rules $r_R$ as well as $r_R$ are not place-preserving rules, as they both add transitions to places that have already been in the body net. The definition of place preserving morphims concerns all placesof the domain. An interesting extension would be to have the property preserving morphimsms (and hence the property preserving rules and transformations) only for those places on which the safety property is directly dependent.

The rule $r_{count}$ in figure 14 that adds the counter of the formed plates to export and body of module *MOD_Cell_2* is safety property preserving for the same reasons as rule $r_{pp}$ in example 4.4.2.

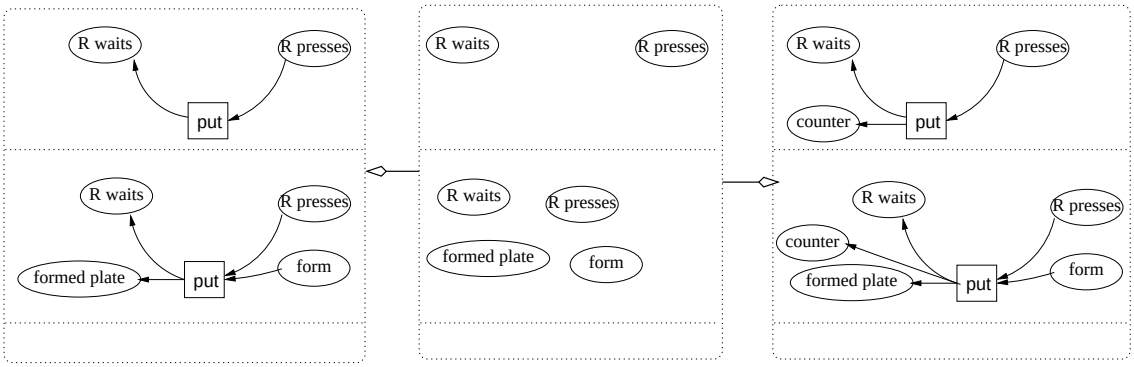



Figure 14: Safety property preserving rule $r_{count}$

# 6 Conclusion

## 6.1 Results and Instantiations

First we shortly summarize what results we have at the level of generic components, next we summarize the instantiations we have discussed in this thesis and at last we discuss the impact for Petri nets.

Based on a adhesive HLR framework for components (see paragraph 5.1.1) we have the following results:

- A categorical component concept (see paragraph 3.1.1),
- composition operations, hierarchical composition and union (see paragraph 3.1.2),
- compatibility results concerning component operations (see paragraph 3.1.2),
- a model based semantics as well as a transformation-based semantics (see paragraph 3.1.3),

– compatibility results concerning component operations and component semantics (see paragraph 3.1.3),
– rules for the transformation of components (see paragraph 5.1.2 and paragraph 5.1.3),
– results concering parallelism, concurrency and embedding of component transformations (see paragraphs 4.1.1 4.1.2 and 4.1.3),
– compatibility results concerning transformations with component operations (see paragraph 5.1.4),
– rule-based refinement of components (see subsection 5.2) and
– compatibility results concerning rule-based refinement with component operations (see subsection 5.2),

In table 3 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 names the specification technique, the second refres to the existence of a component concept in the sense of [EOB+02b], the third asks for the hierarchical composition, the next asks whether additional component operations are possible, e.g for algebraic module specifications there are union, renaming, actualization and others, and the last concerns the semantics. The entries either refer to some paragraph or subsection of this thesis or to the corresponding publication.

| Spec. Tech. | comp. | hier. comp. | other comp. ops | semantics |
|---|---|---|---|---|
| Det. input automata | par. 3.2.3 | yes | union | trafo-based |
| P/T nets | subsec. 3.3 | yes | union | both |
| AHL nets | [EOB+03] | yes | likely | trafo-based |
| Graph transf. sys. | [Sim02] | yes | union | |
| Local action systems | yes [Sim02] | yes | union | |
| Algebraic spec | yes, but with add. parameter [EM90] | yes | union, actualization, renaming, ... | model |
| CSP | as connector architectures in [EPB+04] | yes | | trafo-based |
| UML [5] | as connector architectures in [EBK+05, OP05] | yes | | trafo-based |

Table 3: Results concerning component operations

Table 4 lists whether specification techniques have a transformation concept in the sense of the DPO approach and whether there is rule-based refinenement. It lists whether component transformation or rule-based refinement of components is available. If the generic component concept can be used for a specific specification technique and the specification technique is an HLR category, then component transformation can be achieved in a straightforward way. Rule-based refinement is available in principle for theose specification techniques, that are HLR categories.

For Petri nets we have the following results, that present a comprehensive Petri net technique in the sense of [ERRW03] for place/transition nets, algebraic high-level nets and Colored Petri Nets. The detailed results for rule-based refinement have already been stated in subsection 4.4.

[5] class diagrams, state machines, sequence diagrams

| Spec. Tech. | Trafo of Spec | RBR of Spec | comp. trafo | RBR of comp |
|---|---|---|---|---|
| Det. input automata | par. 4.3.3 | in principle | par.4.3.3 | in principle |
| P/T nets | [EHKP91] | subsec. 4.4, (see table **??**) | subsec. 5.4 | subsec. 5.4 |
| AHL nets | [PER95] | subsec. 4.4, (see table **??**) | straight forward | straight forward |
| Graph transf. sys. | [EHKZ05, ETB05] or [PP96, PP01] | in principle | possibly | in principle |
| Alg. spec | par 4.3.1 | in principle | straightforward | in principle |

Table 4: Results concerning component transformations

| Notion/Results | PT-nets | AHL-nets | CPNs |
|---|---|---|---|
| Rules, Transformations | [EHKP91] | [PER95] | [PG00] |
| property preserving transformations | see table 2 | | |
| Compatibility results | see table 2 | | |
| Module concept | √ | √ | |
| Module semantics | model semantics | trafo-based sem. | |
| Module operations ( union, hierarch. com) | √ | √ | |
| Modules with safety | √ | √ | |
| Module transformations | √ | straightforward | |
| Rule-based Refinement of Modules | √ | straightforward | |

**Table 5.** Achieved results for Petri nets

## 6.2 Outlook

The task that first attracts attention is filling out the gaps in the tables above. Nevertheless, these are mostly mere technical exercises. More fascinating research areas

**Multiple interfaces** Most approach used in practice have component concept, where there are several interfaces. the extension of the component technique we have presented here can be extended so that there is a family of import and a family of exports. 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. Extending the component definition in this way requires the extension of the entire componet technique. In section 2.4 we have sketched the connection from a component technique to the formal description of software architecture. Based on components with multiple interfaces, the correspondence of the architecture graph and compostion operations can be given by graph transformation of the architecture graph.

**Components with properties** Petri net modules with safety are the starting point for the research on componets that have certain properties and preserve them under composition and transformation. The basis for this research is given by the componet technique presented here.

The treatment of specific properties is best done at the level of specification techniques. So, the instantiation and the extension of property preserving compnets in Petri nets, graph transformations or automata seems to be rewarding.

**Specifying Parameterized Contracts in Different Specification Techniques** Parameterized contracts are used for the performance evaluation of component-based software architectures. Up to now deterministic input automata have been used for the definition of parameterized contracts. The automata components presented in this thesis are a formalization of the component concept used for parameterized contracts. but the limitations of automata impede protocol-based interfaces. In [RHH05] a first step towards employing graph transformations has been made. To use Petri net modules for the specification of components and the reasoning about parameterized contracts is a promising work, especially as the adaption mechanism comes for free. Moreover, the comparison of various specification techniques for the definition and analysis of parameterized contracts is an interesting task and possible using the various instantiations of the component technique we have presented in this thesis.

# References


[Abr96] J.R. Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996.

[AEH+99] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34:1–54, 1999.

[AFLW00] L.F. Andrade, J. Fiadeiro, J.L Gouveia, A. Lopes, and M. Wermelinger. Patterns for coordination. In *Coordination Langauges and Models*, Lecture Notes in Computer Science 1906, pages 317–322. Springer-Verlag, 2000.

[AG96] R. Allen and D. Garlan. The Wright architectural specification language. Technical Report CMU-CS-96-TBD, School of Computer Science, Carnegie Mellon University, 1996.

[AG97] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 1997.

[AZ95] E. Astesiano and E. Zucca. D-oids: A model for dynamic data types. *Math. Struct. in Comp. Sci.*, 5(2):257–282, 1995.

[BBS03] S. Balsamo, M. Bernardo, and M. Simeoni. Performance evaluation at the software architecture level. Lecture Notes in Computer Science 2804, pages 207–258. Springer Verlag, 2003.

[BC92] L. Bernadinello and F. De Cindio. A survey of basic net models and modular net classes. ecture Notes in Computer Science 609; Advances in Petri Nets 1992, pages 304–351, 1992.

[BCM91] 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.

[BCMR91] E. Battiston, F. De Cindio, G. Mauri, and L. Rapanotti. Morphisms and Minimal Models for OBJSA Nets. In *12th International Conference on Application and Theory of Petri Nets*, pages 455–476, Gjern, Denmark, 1991.

[BEH+02] B. Braatz, H. Ehrig, K. Hoffmann, J. Padberg, and M. Urbášek. Application of Graph Transformation Techniques to the Area of Petri Nets. In H.-J. Kreowski, editor, *Proc. AGT 2002: APPLIGRAPH Workshop on Applied Graph Transformation*, pages 35–44, 2002.

[BEP02] R. Bardohl, C. Ermel, and J. Padberg. Transforming Specification Architectures by GenGED. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. First Int. Conference on Graph Transformation (ICGT'02)*, pages 30–44, 2002. Springer, Lecture Notes in Computer Science 2505.

[BGK98] R. Büssow, R. Geisler, and M. Klar. Specifying safety-critical embedded systems with statecharts and Z: a case study. In E. Astesiano, editor, *Proc. 1st Int. Conf. on Fundemantal Approaches to SoftwareEngineering (FASE'98)*, pages 71–87. Springer Lecture Notes in Computer Science 1382, 1998.

[BH98] E. Börger and J.K. Huggins. Abstract State Machines 1988 – 1998. *Bull. EATCS 64*, pages 71 – 87, 1998. Commented ASM Bibliography.

[BM04] Michel Bidoit and Peter D. Mosses. Casl *User Manual*. Lecture Notes in Computer Science 2900 (IFIP Series). Springer, 2004.

[BOR04] 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.

[Bri89] Brinksma, E. (ed.). Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour. ISO 8807, 1989. International Standard.

[BS92] M. Broy and T. Streicher. Modular functional modelling of Petri nets with individual tokens. *Advances in Petri Nets*, Lecture Notes in Computer Science 609, 1992.

[Buc94] P. Buchholz. Hierarchical high level Petri nets for complex system analysis. In *Application and Theory of Petri Nets*, Lecture Notes in Computer Science 815, pages 119–138. Springer, 1994.

[CAT04] Category theory, http://plato.stanford.edu/archives/spr2004/entries/category-theory/, Spring 2004.

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

[CECB05] M.l Barrio-Solórzano C. E. Cuesta, P. de la Fuente and M. Encarnación. An abstract process approach to algebraic dynamic architecture description. *The Journal of Logic and Algebraic Programming*, 63(2):177–214, 2005.

[CEKR02] A. Corradini, H. Ehrig, H-J Kreowski, and G. Rozenberg, editors. *Proc. 1st Int. Conference on Graph Transformation (ICGT'02)*. Springer Lecture Notes in Computer Science 2505, 2002.

[CH94] S. Christinsen and N.D. Hansen. Coloured Petri nets extended with channels for synchronous communication. In *Application and Theory of Petri Nets*, volume Lecture Notes in Computer Science 815, pages 159–178. Springer, 1994.

[CMR96] A. Corradini, U. Montanari, and F. Rossi. Graph Processes. *Special Issue of Fundamenta Informaticae*, 26(3,4):241–266, 1996.

[CoF04] CoFI (The Common Framework Initiative). CASL *Reference Manual*. Lecture Notes in Computer Science 2960 (IFIP Series). Springer, 2004.

[CP00] S. Christensen and L. Petrucci. Modular analysis of Petri nets. *Computer Journal*, 43(3):224–242, 2000.

[CPT99] C. Canal, E. Pimentel, and J. M. Troya. Specification and refinement of dynamic software architectures. In *Software Architecture*, pages 107–126. 1999.

[dAH01] 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.

[Daw01] Eugenio P. Dawis. Architecture of an SS7 protocol stack on a broadband switch platform using dualistic petri nets. In *2001 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 323–326, 2001.

[DDK01] Eugenio P. Dawis, Joaquin F. Dawis, and Wei Pin Koo. Architecture of computer-based systems using dualistic petri nets. In *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference*, October 2001.

[DG94a] P. Dauchy and M.C. Gaudel. Algebraic specifications with implicit states. *Tech. Report, Univ. Paris Sud*, 1994.

[DG94b] 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.

[DJL00] 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*, Lecture Notes in Computer Science 1825, pages 146–165. Springer, 2000.

[DJL01] 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*, Lecture Notes in Computer Science 2128. Springer, 2001.

[dPJC00] V.C. de Paula, G.R.B. Justo, and P.R.F. Cunha. Specifying and verifying reconfigurable software architectures. In *International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 21 – 31, 2000.

[DRR04] Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors. *4th Advanced Course on Petri Nets, ACPN 2003*, Lecture Notes in Computer Science 3098. Springer-Verlag, 2004.

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

[EBO91] H. Ehrig, M. Baldamus, and F. Orejas. Amalgamation and Extension in the Framework of Specification Logics and Generalized Morphisms. *Bulletin of the EATCS, No. 44*, pages 129 – 143, 1991.

[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 Lecture Notes in Computer Science 655, 1992.

[EBP01] C. Ermel, R. Bardohl, and J. Padberg. Visual Design of Software Architecture and Evolution based on Graph Transformation. In C. Ermel, H. Ehrig and J. Padberg, editors, *Electronic Notes in Theoretical Computer Science: Proc. Workshop on Uniform Approaches to Graphical Process Specification Techniques (UNIGRA'2001), Satellite Event of ETAPS'01*, volume 44, 2001. Elsevier Science Publishers.

[EE93] H. Ehrig and G. Engels. Towards a module concept for graph transformation systems. Technical Report 93-34, Leiden University (The Netherlands), 1993.

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*. World Scientific, 1999.

[EEPPR04] H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors. *Proc. 2nd International Conference on Graph Transformations, 2004*. Springer, Lecture Notes in Computer Science 3256, 2004.

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

[EFRV86] G. Estrin, R. S. Fenchel, R. R. Razouk, and M. K. Vernon. Sara (system architects apprentice): modeling, analysis, and simulation support for design of concurrent systems. *IEEE Trans. Softw. Eng.*, 12(2):293–311, 1986.

[EGKP97] H. Ehrig, R. Geisler, M. Klar, and J. Padberg. Horizontal and Vertical Structuring Techniques for Statecharts. In A. Mazurkiewicz and J. Winkowski, editors, CONCUR'97, Lecture Notes in Computer Science 1243, pages 181–195. Springer Verlag, 1997.

[EGP98] C. Ermel, M. Gajewsky, and J. Padberg. Rule-Based Refinement of High-Level Nets Preserving Safety Properties. In *Proceedings of ETAPS-FASE (Fundamental Approaches of Software Engineering)*, Lecture Notes in Computer Science 1382, pages 221–238. Springer Verlag, 1998.

[EGP99] 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.

[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.

[EHKZ05] C. Ermel, K. Hölscher, S. Kuske, and P. Ziemann. Animated Simulation of Integrated UML Behavior Models based on Graph Transformation. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, 2005. to appear.

[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)*, Lecture Notes in Computer Science 3256, pages 144–160, 2004. Springer.

[Ehr79] H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In *Graph Grammars and their Application to Computer Science and Biology*, pages 1–69. Springer Lecture Notes in Computer Science 73, 1979.

[EJPR01] H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors. *Advances in Petri Nets: Unifying Petri Nets*, Lecture Notes in Computer Science 2128. Springer, 2001.

[EKMR99] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution.* World Scientific, 1999.

[EL93] H. Ehrig and M. Löwe. Categorical principles, techniques and results for high-level replacement systems in computer science. *Applied Categorical Structures*, 1(1):21–50, 1993.

[ELO94] H. Ehrig, M. Löwe, and F. Orejas. Dynamic abstract data types based on algebraic graph transformations. Technical Report 94–37, TU Berlin, 1994.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, Berlin, 1985.

[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.

[EO98] H. Ehrig and F. Orejas. Integration Paradigm for Data Type and Process Specification Techniques. *Bull. EATCS 65, Formal Specification Column, Part 5*, 1998. also in [PRS01], pages 192 - 201.

[EOB+02a] H. Ehrig, F. Orejas, B. Braatz, M. Klein, and M. Piirainen. A Component Framework based on High-Level Replacement Systems. In *Proc. Int. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'02), Satellite Event of ICGT'02*, pages 124–138, 2002.

[EOB+02b] 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*, Lecture Notes in Computer Science 2306, pages 32–48. Springer, 2002.

[EOB+03] 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, 2003.

[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.

[EOP00] H. Ehrig, F. Orejas, and J. Padberg. Relevance, Integration and Classification of Specification Formalisms and Formal Specification Techniques. In *Proc. FORMS'99, Braunschweig, Germany*, pages 31–54. Forschrittberichte VDI, Reihe 12, Nr. 436, VDI-Verlag, 2000.

[EP72] H. Ehrig and Pfender, M. et al. *Kategorien und Automaten.* de Gruyter Lehrbuch, 1972.

[EP91] H. Ehrig and F. Parisi-Presicce. Algebraic specification grammars: a junction between module specifications and graph grammars. Lecture Notes in Computer Science, 532:292–310, 1991.

[EP97] H. Ehrig and J. Padberg. A Uniform Approach to Petri Nets. In Ch. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential - Theory - Cognition*, pages 219–231. Springer, Lecture Notes in Computer Science 1337, 1997.

[EP04] H. Ehrig and J. Padberg. Graph grammars and Petri net transformations. In *Lectures on Concurrency and Petri Nets Special Issue Advanced Course PNT*, Springer Lecture Notes in Computer Science 3098, pages 496–536, 2004.

[EPB+90] H. Ehrig, F. Parisi-Presicce, P. Boehm, C. Rieckhoff, C. Dimitrovici, and M. Große-Rhode. Combining data type and recursive process specifications using projection algebras. *Theoretical Computer Science*, 71:347–380, 1990.

[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* Eletronic Notes in Theoretical Computer Science, volume 108, pages 53–67, 2004.

[EPO01] H. Ehrig, J. Padberg, and F. Orejas. From Basic Views and Aspects to Integration of Specification Formalisms . In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Entering the 21st Century*, pages 202 – 214. World Scientific, Singapore, 2001.

[ERRW03] H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, editors. *Advances in Petri Nets: Petri Net Technology for Communication Based Systems.* Lecture Notes in Computer Science 2472. Springer, 2003.

[ETB05] C. Ermel, G. Taentzer, and R. Bardohl. Simulating Algebraic High-Level Nets by Parallel Attributed Graph Transformation. In Kreowski et al. [KMO+05].

[Feh93] R. Fehling. A concept of hierarchical Petri nets with building blocks. In *Advances in Petri Nets'93*, pages 148–168. Springer, 1993. Lecture Notes in Computer Science 674.

[FM97] J. L. Fiadeiro and T. Maibaum. Categorical semantics of parallel programm design. *Sciencs of Computer Programming*, 28:111–138, 1997.

[Gar03] D. Garlan. Formal modelin and analysis of software architecture: Components, connectors, and events. In *Formal Methods for Software Architecture*, Lecture Notes in Computer Science 2804, pages 1–24. Springer, 2003.

[GB84] J. A. Goguen and R. M. Burstall. Introducing institutions. *Proc. Logis of Programming Workshop Carnegie-Mellon*, 64:221–256, 1984.

[GB92] J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journals of the ACM*, 39(1):95–146, January 1992.

[Gen] GenGED Homepage. http://tfs.cs.tu-berlin.de/genged.

[GGW99] H. Giese, J. Graf, and G. Wirtz. Closing the Gap Between Object-Oriented Modeling of Structure and Behaviour. In R. France and B. Rumpe, editors, *UML'99 - The Second Int. Conference on The Unified Modeling Language*, Lecture Notes in Computer Science 1723, pages 534 – 549. Springer, 1999.

[Gie01] H. Giese. Object Coordination Nets 3.0: Reference Guide. Technical Report 1/01-I, University Münster, Computer Science, Distributed Systems Group, 2001.

[GMW97] D. Garlan, R. Monroe, and D. Wile. Acme: An Architecture Description Interchange Language. In *Proc. of CASCON'97*, pages 169–183, 1997. http://www.cas.ibm.ca/cascon/cfp.html.

[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)*, pages 137–151. Springer Lecture Notes in Computer Science 1589, 1999.

[Gri98] F. Griffel. *Componentware – Konzepte und Techniken eines Softwareparadigmas*. dpunkt Verlag, 1998.

[Gro95] M. Große-Rhode. *Specification of Transition Categories — An Approach to Dynamic Abstract Data Types*. PhD thesis, TU Berlin, 1995.

[Gro97] M. Große–Rhode. Transition specifications for dynamic abstract data types. *Applied Categorical Structures*, 5:265–308, 1997.

[Gro98] M. Große–Rhode. Algebra transformation systems and their composition. In E. Astesiano, editor, *Fundamental Approaches to Software Engineering (FASE'98)*, pages 107–122. Springer Lecture Notes in Computer Science 1382, 1998.

[Gru04] L. Grunske. *Strukturorientierte Optimierung der Qualitätseigenschaften von softwareintensiven technischen Systemen im Architekturentwurf*. PhD thesis, Universität Potsdam, 2004.

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

[Gur91] Y. Gurevich. Evolving algebras, a tutorial introduction. In *Bull. EATCS 43*, pages 264 – 284. Springer, 1991.

[GW01] H. Giese and G. Wirtz. The OCoN Approach for Object-Oriented Distributed Software Systems Modeling. *Computer Systems Science & Engineering*, 16(3):157–172, 2001.

[Har87] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[He96] X. He. A Formal Definition of Hierarchical Predicate Transition Nets. In *Application and Theory of Petri Nets*, Lecture Notes in Computer Science 1091, pages 212–229. Springer, 1996.

[HEET99] R. Heckel, H. Ehrig, G. Engels, and G. Taentzer. Classification and Comparison of Modularity Concepts for Graph Transformation Systems. In H. Ehrig, G. Engels, J.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools*, pages 669 – 690. World Scientific, 1999.

[HHKK00] R. Heckel, B. Hoffmann, P. Knirsch, and S. Kuske. Simple modules for grace. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Theory and Application of Graph Transformation: 6th Int. Workshop*, Lecture Notes in Computer Science 1764. Springer, 2000.

[HNS99] C. Hofmeister, R. Nord, and D. Soni. *Describing Software Architecture in UML*, pages 145–159. Kluwer Academic Publishers, 1999.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[Jen92] 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.

[JL02] 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.

[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*, Lecture Notes in Computer Science 1709, pages 1778–1797. Springer, 1999.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM.* Prentice Hall Internatinal, New Jersey, 1990.

[JR91] K. Jensen and G. Rozenberg, editors. *High-Level Petri-Nets: Theory and Application.* Springer Verlag, 1991.

[JSHS91] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-oriented specification of information systems: The troll language. Technical Report 91-04, TU Braunschweig, 1991.

[Kin95] E. Kindler. *Modularer Entwurf verteilter Systeme mit Petrinetzen.* PhD thesis, Technische Universität München, Institut für Informatik, 1995.

[KK02] P. Knirsch and S. Kuske. Distributed graph transformation units. In A. Corradini, H. Ehrig, H.J. Kreowski, and G. Rozenberg, editors, *Proc. 1. Int. Conference on Graph Transformation (ICGT)*, Lecture Notes in Computer Science 2505, pages 207–222, 2002.

[KMO+05] H.-J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, editors. *Formal Methods in Software and Systems Modeling: Essays Dedicated to Hartmut Ehrig on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science 3393. Springer, 2005.

[LS04] S. Lack and P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, Lecture Notes in Computer Science 2987, pages 273–288. Springer, 2004.

[LT87] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987.

[LT89] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3), 1989.

[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.

[MDEK95] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In W. Schafer and P. Botella, editors, *Proc. 5th European Software Engineering Conf. (ESEC 95)*, Lecture Notes in Computer Science 989, pages 137–153. Springer, 1995.

[Mil80] R. Milner, editor. *A calculus for communicationg systems.* Springer Lecture Notes in Computer Science 92, 1980.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

[MK96] J. Magee and J. Kramer. Dynamic Structures in Software Architecture. In *Proc. 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 3–14, 1996.

[MMHR04] J. Matevska-Meyer, W. Hasselbring, and R. Reussner. Software architecture description supporting component deployment and system runtime reconfiguration. In *Proceedings of Workshop on Component-Oriented Programming (WCOP 2004)*, 2004.

[Mon98] S. Montenegro. Formale Methoden in der Softwareentwicklung heute und morgen. *Elektronik Zeitschrift*, 21, 1998.

[OP05] F. Orejas and S. 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.

[Pad96] J. Padberg. *Abstract Petri Nets: A Uniform Approach and Rule-Based Refinement.* PhD thesis, Technical University Berlin, 1996. Shaker Verlag.

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

[Pad02a] J. Padberg. Petri net modules. *Journal on Integrated Design and Process Technology*, 6(4):121–137, 2002.

[Pad02b] 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.

[Pad03] J. Padberg. Case study: Modelling telecom services with petri net modules. In Roswitha Bardohl and Hartmut Ehrig, editors, *Electronic Notes in Theoretical Computer Science*, volume 82. Elsevier, 2003.

[Pad04] J. Padberg. Safety properties in Petri net modules. *Journal on Integrated Design and Process Technology*, 2004. Accepted.

[Pad05a] Julia Padberg. Integration of the generic component concepts for system modeling with adhesive HLR systems. *EATCS Bulletin 87*, pages 138 - 155, 2005.

[Pad05b] Julia Padberg. High-level replacement systems for software components. Technical Report 2005-07, Technische Universität Berlin, 2005.

[Pad06] J. Padberg. Formale Techniken für die Beschreibung von Software-Architekturen. In R. Reussner and W. Hasselbring, editors, *Handbuch der Software-Architektur*, pages 465-476, d-punkt Verlag, 2006.

[PE01] J. Padberg and H. Ehrig. Parametrized Net Classes: A Uniform Approach to Petri Net Classes. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, Lecture Notes in Computer Science 2128, pages 173–229. Springer, 2001.

[PE06] J. Padberg and H. Ehrig. Petri net modules in the transformation-based component framework. *Journal of Logic and Algebraic Programming*, Volume 67, Issues 1-2, pages 198-225, 2006.

[PEB00] J. Padberg, C. Ermel, and R. Bardohl. Rule-Based and Visual Model Evolution using GenGEd. In *Proc. of Satellite Workshops of 27th Int. Colloqium on Automata, Languages, and Programming (ICALP'2000)*, pages 467–475, 2000. Carleton Scientific.

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

[PG00] J. Padberg and M. Gajewsky. Rule-Based Refinement of Petri Nets For Modeling Train Control Systems . In Š. Kozák and M. Huba, editors, *Petri Nets in Design, Modelling and Simulation of Control Systems, Special Session at the IFAC Conference on Control Systems Design*, pages 299–304, 2000.

[PGE98] J. Padberg, M. Gajewsky, and C. Ermel. Rule-Based Refinement of High-Level Nets Preserving Safety Properties. In E. Astesiano, editor, *Fundamental Approaches to Software Engineering*, pages 221–238. Springer Verlag, Lecture Notes in Computer Science 1382, 1998.

[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.

[PGH99] J. Padberg, M. Gajewsky, and K. Hoffmann. Incremental Development of Safety Properties in Petri Net Transformations. In G. Engels and G. Rozenberg, editors, *Theory and Application of Graph Transformations (TAGT'98)*, Lecture Notes in Computer Science 1764, pages 410–425. Springer Verlag, 1999.

[PHG00] J. Padberg, K. Hoffmann, and M. Gajewsky. Stepwise Introduction and Preservation of Safety Properties in Algebraic High-Level Net Systems. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, pages 249–265. Springer Verlag, Lecture Notes in Computer Science 1783, 2000.

[PK05] J. Padberg and H.-J. Kreowski. Loose semantics of Petri nets. In Kreowski et al. [KMO+05], pages 370–384.

[PP91] F. Parisi-Presicce and A. Pierantonio. An Algebraic Approach to Inheritance and Subtyping. In *Proc. ESEC 1991*, Lecture Notes in Computer Science 550, pages 364 –379. Springer, 1991.

[PP96] F. Parisi-Presicce. Transformation of Graph Grammars. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science*, Lecture Notes in Computer Science 1073, 1996.

[PP01] 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.

[PRS01] G. Paun, G. Rozenberg, and A. Salomaa, editors. *Current Trends in Theoretical Computer Science: Entering the 21st Century*. World Scientific, Singapore etc., 2001.

[PU03] J. Padberg and M. Urbášek. Rule-Based Refinement of Petri Nets: A Survey. In Ehrig et al. [ERRW03], pages 161–196.

[Reu01] R. Reussner. *Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten*. PhD thesis, Universität Karlsruhe (Technische Hochschule), 2001.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[RHH05] R. Reussner,J. Happe and A. Habel. Modelling Parametric Contracts and the State Space of Composite Components by Graph Grammars. In M. Cerioli, editor, *Fundamental Approaches to Software Engineering (FASE'05 )*, pages 80-95. Springer Lecture Notes in Computer Science 3442, 2005..

[RR98a] Reisig, W. and Rozenberg, G., editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[RR98b] Reisig, W. and Rozenberg, G., editors. *Lectures on Petri Nets II: Applications* , volume 1492 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[RS02] R. 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.

[SB94] C. Sibertin-Blanc. Cooperative Nets. In *Application and Theory of Petri Nets'94*, pages 471–490. Springer Lecture Notes in Computer Science 815, 1994.

[Sch97] A. Schürr. Programmed graph replacement systems. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1998.

[SDK+95] M. Shaw, R. Deline, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for Software Architecture and Tools to Support Them. *IEEE Transactions on Software Engineering*, 21(4):314–315, 1995.

[Sea91] A. Sernadas and et. al. OBLOG – Object oriented Logic: An Informal Introduction. Technical report, INESC, Lisbon, 1991.

[SG96] M. Shaw and D. Garlan. *Software Architecture - Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[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 .

[Spi89] J. M. Spivey. *The Z Notation — A Reference Manual*. Prentice Hall, 1989.

[SW98] A. Schuerr and A.J. Winter. Uml Packages for PROgrammed Graph REwrite Systems. In *Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98)*, Paderborn, Germany, 1998.

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

[TS95] G. Taentzer and A. Schürr. DIEGO, another step towards a module concept for graph transformation systems. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. http://www.elsevier.nl/locate/entcs/volume2.html.

[UP02] 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.

[WD96] J. Woodcock and J. Davies. *Using Z.* Prentice-Hall, 1996.

[Web99] H. Weber. Continuous Engineering of Communication and Software Infrastructures. Lecture Notes in Computer Science 1577, pages 22–29. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[Win88] G. Winskel. Introduction to Event Structures. In G. Rozenberg J.W. de Bakker, W.-P. de Roever, editor, *Linear Time, Branching Time, and Partial Order in Logics and Models of Concurrency*, pages 364 – 397. Lecture Notes in Computer Science 354, 1988.

# A Publications Submitted as Part of this Thesis

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

Based on the well-known theory of high-level replacement systems – a categorical formulation of graph grammars – we present here the results concerning refinement of high-level replacement systems. Motivated by Petri nets, where refinement is often given by morphisms, we give a categorical notion of refinement. This concept is called $\mathcal{Q}$-transformations and is established within the framework of high-level replacement systems. The main idea is to supply rules with an additional morphism, which belongs to a specific class $\mathcal{Q}$ of morphisms. This leads to the new notions of $\mathcal{Q}$-rules and $\mathcal{Q}$-transformations. Moreover, several concepts and results of high-level replacement systems are extended to $\mathcal{Q}$-transformations. These are sequential and parallel transformations, union, and fusion, based on different colimit constructions. The main results concern the compatibility of these constructions with $\mathcal{Q}$-transformations that is the corresponding theorems for usual transformations are extended to $\mathcal{Q}$-transformations. Finally, we demonstrate the application of these techniques for the special case of Petri nets to a case study concerning the requirements engineering of a medical information system.

[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.

The concept of rule-based modification developed in the area of algebraic graph transformations and high-level replacement systems is a powerful concept for vertical stucturing of Petri nets. This includes low-level and high-level Petri nets, especially algebraic high-level nets which can be considered as an integration of algebraic specifications and Petri nets. In a large case study rule-based modification of algebraic high-level nets has been applied successfully for the requirements analysis of a medical information system. The main result in this paper extends rule-based modification of algebraic high-level nets such that it preserves safety properties formulated in terms of temporal logic. For software development based on rule-based modification of algebraic high-level nets as a vertical development strategy this extension is an important new technique. It is called rule-based refinement. As a running example an important safety property of a medical information system is considered and is shown to be preserved under rule-based refinement.

The coauthors Dipl.Inf. M. Gajewsky (email: gajewski@wortfeilerei.de) and Dipl.Inf C. Ermel email: lieske@cs.tu-berlin.de) belong to the research group that has been headed by J. Padberg. The main results have been the the work of J. Padberg.

[Pad02a] J. Padberg. Petri net modules. *Special Issue on Component Based System Development, Journal on Integrated Design and Process Technology*, 2002.

Here we present a new module concept for Petri nets that is based on the component concepts of *Continuous Software Engineering (CSE)*. According to that concept two distinguished interfaces are required. These are import and export interfaces. The import describes the assumptions on the environment, e.g. in terms of used components. The export gives an abstraction of the functionality and presents e.g. the offered services. The modules for Petri we introduce here consist of three nets for the import, the body of the module and the export. The import net $IMP$ states the prerequisites the

modules assumes. The body net $BOD$ represents the internal functionality. The export net $EXP$ gives an abstraction of the body that can be used by the environment. We provide module operations to compose larger modules from basic ones. Operations to compose Petri net modules are crucial as the main purpose is composition. In most approaches module concepts come along with just one operation. A great advantage is achieved having different possibilities to compose modules, as it increases the convenience of modeling large systems. We propose three different operations, composition, disjoint union, and union. Our main result in this contribution is that these module operations are compatible with each other.

[PU03] J. Padberg and M. Urbášek. Rule-Based Refinement of Petri Nets: A Survey. In: Advances in Petri Nets: Petri Net Technology for Communication Based Systems. *Lecture Notes in Computer Science 2472*, pages 161–196. Springer, 2003.

This paper provides a thorough survey of our work on rule-based refinement. Rule-based refinement comprises the transformation of Petri nets using rules while preserving certain system properties. Petri net rules and transformations are expressed by morphisms and pushouts. This allows an abstract formulation of our notions independent of a specific Petri net class, as place/transition nets, elementary nets, predicate/transition nets etc. Hence, it is adequate to consider our approach as rule-based refinement of Petri nets in general. We have presented various results in recent years at different conferences. So this contribution gives an overview of our work in a compact form leaving out the technical details.

The coauthor Dr. M. Urbášek (email: M.Urbasek@Aquasoft.cz) was part of the research group that has been headed by J. Padberg. The survey covers part the work of DFG-Research-Group Petri Net Technology from 1997 -2002. Those publications that have been coauthored by Dr. Urbášek are part of his Phd. Thesis *Categorical Net Transformations for Petri Net Technology*, Technical University Berlin, May, 2003.

[Pad04] J. Padberg. Safety properties in Petri net modules. *Journal on Integrated Design and Process Technology*, 2004. Accepted.

This paper introduces safety properties in the temporal logic sense to Petri net modules. Petri net modules have been achieved by a transfer of algebraic specification modules to Petri nets. They consist of three nets; the interface nets import and export, and the body of the module. The import net states the prerequisites the modules assumes. The body net represents the internal functionality. The export net gives an abstraction of the body that can be used by the environment. The interfaces $IMP$ and $EXP$ are related to the body $BOD$ via morphisms. We make precise what it means that a Petri net module has specific safety properties. We differentiate between explicit and implicit properties. Explicit safety properties are stated additionally to the export net. Implicit are those properties that hold in the export net without being stated explicitly.The main advantage of our approach are module operations to compose larger modules from basic ones. We can show that the composition of modules preserves safety properties:
Given two modules with implicit or explicit safety properties then the composition of these modules is again a module with implicit or explicit safety properties.

[PK05] J. Padberg and H.-J. Kreowski. Loose semantics of Petri nets. In Formal Methods in Software and Systems Modeling, *Lecture Notes in Computer Science 3393*, pages 370–384. Springer, 2005.

Here we propose a new loose semantics for place/transition nets based on transition systems and generalizing the reachability graph semantics. The loose semantics of a place/transition net reflects all its possible refinements and is given as a category of transition systems with alternative sequences of events over the net. The main result

states that each plain morphism between two place/transitions nets induces a free construction between the corresponding semantic categories.

The coauthor Prof. Dr. Kreowski (email: kreo@tzi.de) has provided many helpful suggestions and a lot of discussion, the main concepts and results have been worked out by J. Padberg.

[Pad05a] Julia Padberg. Integration of the generic component concepts for system modeling with adhesive HLR systems. *EATCS Bulletin 87*, pages 138 - 155, 2005.

The integration of two well established theories is presented in this paper, namely the generic framework for components in system modeling and the adhesive HLR systems. The first theory describes components and composition at an abstract level that is independent of the specification technique. The second theory formalizes rules and transformations for an abstract replacement at the same abstract level.

The main results are the definition of a weak adhesive HLR category for components, its immediate results as Church-Rosser Theorem, Parallelism Theorem and Concurrency Theorem, and the new compatibility result for the transformation and hierarchical composition. We discuss the instantiation with deterministic automata, Petri nets and others. For these instantiations we immediately have the results mentioned above.

[PE06] J. Padberg and H. Ehrig. Petri net modules in the transformation-based component framework. *Journal of Logic and Algebraic Programming*, Volume 67, Issues 1-2, pages 198-225, 2006.

Component-based software engineering needs to be backed by thorough formal concepts and modeling techniques. This paper combines two concepts introduced independently by the two authors in previous papers. On one hand, the concept of Petri net modules , and on the other hand a generic component framework for system modeling. In this paper we develop a categorical formalization of the transformation based approach to components that is based on pushouts. This is the frame in which we show that Petri net modules can be considered as an instantiation of the generic component framework. This allows applying the transformation based semantics and compositionality result of the generic framework to Petri net modules. In addition to general Petri net modules we introduce Petri net modules preserving safety properties which can be considered as another instantiation of pushout based formalization of the generic framework.

The coauthor Prof. Dr. Ehrig (email: ehrig@cs.tu-berlin.de) has contributed mainly the parts concerning the generic framework.

[Pad06] J. Padberg. Formale Techniken für die Beschreibung von Software-Architekturen. In R. Reussner and W. Hasselbring, editors, *Handbuch der Software-Architektur*, pages 465-476, d-punkt Verlag, 2006.

The importance of architecture descriptions has become most obvious over the last decade and various formalisms have been proposed to deal with the complexity of large software systems. Most architecture description languages involve in some way some formal technique. In this paper a survey over the use of formal techniques for the description of software architectures is given. Exemplarily stated there are process algebras, Petri nets, logic calculi, graph transformations, set-based approaches, algebraic specifications. Although there are many approaches available, only few are general enough to be used for different specification techniques. So we additionally present a generic approach to componets and software architectures that has been introduced in [EPB+04].

# B Further Publications Related to this Thesis


[BEH+02] B. Braatz, H. Ehrig, K. Hoffmann, J. Padberg, and M. Urbášek. Application of Graph Transformation Techniques to the Area of Petri Nets. In H.-J. Kreowski, editor, *Proc. AGT 2002: APPLIGRAPH Workshop on Applied Graph Transformation*, pages 35–44, 2002.

[BEP02] R. Bardohl, C. Ermel, and J. Padberg. Transforming Specification Architectures by GenGED. In A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. First Int. Conference on Graph Transformation (ICGT'02)*, pages 30–44, 2002. Springer, Lecture Notes in Computer Science 2505.

[EBP01] C. Ermel, R. Bardohl, and J. Padberg. Visual Design of Software Architecture and Evolution based on Graph Transformation. In C. Ermel, H. Ehrig and J. Padberg, editors, *Electronic Notes in Theoretical Computer Science: Proc. Workshop on Uniform Approaches to Graphical Process Specification Techniques (UNIGRA'2001), Satellite Event of ETAPS'01*, volume 44, 2001. Elsevier Science Publishers. .

[EGKP97] H. Ehrig, R. Geisler, M. Klar, and J. Padberg. Horizontal and Vertical Structuring Techniques for Statecharts. In A. Mazurkiewicz and J. Winkowski, editors, CONCUR'97, Lecture Notes in Computer Science 1243, pages 181–195. Springer Verlag, 1997.

[EGP98] C. Ermel, M. Gajewsky, and J. Padberg. Rule-Based Refinement of High-Level Nets Preserving Safety Properties. In *Proceedings of ETAPS-FASE (Fundamental Approaches of Software Engineering)*, Lecture Notes in Computer Science 1382, pages 221–238. Springer Verlag, 1998.

[EGP99] 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.

[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)*, Lecture Notes in Computer Science 3256, pages 144–160, 2004. Springer.

[EJPR01] H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors. *Advances in Petri Nets: Unifying Petri Nets*, Lecture Notes in Computer Science 2128. Springer, 2001.

[EOP00] H. Ehrig, F. Orejas, and J. Padberg. Relevance, Integration and Classification of Specification Formalisms and Formal Specification Techniques. In *Proc. FORMS'99, Braunschweig, Germany*, pages 31–54. Forschrittberichte VDI, Reihe 12, Nr. 436, VDI-Verlag, 2000.

[EP97] H. Ehrig and J. Padberg. A Uniform Approach to Petri Nets. In Ch. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential - Theory - Cognition*, pages 219–231. Springer, Lecture Notes in Computer Science 1337, 1997.

[EP04] H. Ehrig and J. Padberg. Graph grammars and Petri net transformations. In *Lectures on Concurrency and Petri Nets Special Issue Advanced Course PNT*, Springer Lecture Notes in Computer Science 3098, pages 496–536, 2004.

[EPO01] H. Ehrig, J. Padberg, and F. Orejas. From Basic Views and Aspects to Integration of Specification Formalisms . In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Entering the 21st Century*, pages 202 – 214. World Scientific, Singapore, 2001.

[Pad02b] 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.

[Pad03] J. Padberg. Case study: Modelling telecom services with petri net modules. In Roswitha Bardohl and Hartmut Ehrig, editors, *Electronic Notes in Theoretical Computer Science*, volume 82. Elsevier, 2003.

[Pad05b] Julia Padberg. High-level replacement systems for software components. Technical Report 2005-07, Technische Universität Berlin, 2005.

[PE01] J. Padberg and H. Ehrig. Parametrized Net Classes: A Uniform Approach to Petri Net Classes. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, Lecture Notes in Computer Science 2128, pages 173–229. Springer, 2001.

[PEB00] J. Padberg, C. Ermel, and R. Bardohl. Rule-Based and Visual Model Evolution using GenGEd. In *Proc. of Satellite Workshops of 27th Int. Colloqium on Automata, Languages, and Programming (ICALP'2000)*, pages 467–475, 2000. Carleton Scientific.

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

[PG00] J. Padberg and M. Gajewsky. Rule-Based Refinement of Petri Nets For Modeling Train Control Systems . In Š. Kozák and M. Huba, editors, *Petri Nets in Design, Modelling and Simulation of Control Systems, Special Session at the IFAC Conference on Control Systems Design*, pages 299–304, 2000.

[PGE98] J. Padberg, M. Gajewsky, and C. Ermel. Rule-Based Refinement of High-Level Nets Preserving Safety Properties. In E. Astesiano, editor, *Fundamental Approaches to Software Engineering*, pages 221–238. Springer Verlag, Lecture Notes in Computer Science 1382, 1998.

[PGH99] J. Padberg, M. Gajewsky, and K. Hoffmann. Incremental Development of Safety Properties in Petri Net Transformations. In G. Engels and G. Rozenberg, editors, *Theory and Application of Graph Transformations (TAGT'98)*, Lecture Notes in Computer Science 1764, pages 410–425. Springer Verlag, 1999.

[PHG00] J. Padberg, K. Hoffmann, and M. Gajewsky. Stepwise Introduction and Preservation of Safety Properties in Algebraic High-Level Net Systems. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, pages 249–265. Springer Verlag, Lecture Notes in Computer Science 1783, 2000.

[UP02] 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.

# C Publication List of Julia Padberg

## Articles and Chapters in Books

1. Formale Techniken für Software-Architekturen. Book chapter in *Handbuch der Software-Architektur*, pp. 465-476, dPunkt Verlag, 2006.
2. Petri net modules in the transformation-based component framework. Journal of Logic and Algebraic Programming 67(1-2), pp. 198-225, 2006, with H. Ehrig.
3. Integration of the generic framework for components for system modeling with adhesive HLR systems, Bulletin of the EATCS (87) pp. 138 - 155, 2005.
4. Adhesive high-level replacement categories and systems. Fundamenta Informatica, 2005, accepted, with H. Ehrig, A. Habel, U. Prange.
5. Safety properties in Petri net modules. Journal on Integrated Design and Process Technology, 2004, 8(4), 2004.
6. Petri net modules. Journal on Integrated Design and Process Technology, 6(4):105–120, 2002.
7. Evolutionary development of business process centred architectures using component technologies. Journal on Integrated Design and Process Technology, 5(3), 2001, with A. Sünbül, H. Weber.
8. Rule-based refinement of high-level nets preserving safety properties. Science of Computer Programming, 40, 2001, with M. Gajewsky, and C. Ermel.
9. Cooperability in train control systems specification of scenarios using open nets. Journal of Integrated Design and Process Technology 5(1), 2001, with L. Jansen, H. Ehrig, E. Schnieder, R. Heckel.
10. Categorical approach to horizontal structuring and refinement of high-level replacement systems. Applied Categorical Structures, 7(4), 1999.
11. From basic views and aspects to integration of specification formalisms Bulletin of the EATCS 69, 1999, with H. Ehrig, F. Orejas.
12. Classification of Petri nets using adjoint functors. Bulletin of the EACTS 66, 1998.
13. Algebraic High-Level Net Transformation Systems. Mathematical Structures in Computer Science Vol 2, pp. 217-256, 1995, with H. Ehrig and L. Ribeiro.
14. How to Transfer Concepts of Abstract Data Types to Petri Nets? Bulletin of the EATCS 62, 1997, with H. Ehrig, A. Merten.
15. Linking Algebraic High Level Nets and Dynamic Abstract Data Types. Bulletin of the EATCS 54, 1994, with H. Ehrig.

## Edited collections and Special Issues

16. Special Issue on Component Based System Development H. Ehrig, J. Padberg. Journal on Integrated Design and Process Technology 6(4), 2002.
17. Advances in Petri Nets: Unifying Petri Nets H. Ehrig, G. Juhás, J. Padberg, G. Rozenberg. Lecture Notes in Computer Science 2128. Springer, 2001.
18. Proc. of the 1st. Int. Workshop on Uniform Approaches to Graphical Process Specification Techniques H. Ehrig, J. Padberg, and C. Ermel. Electronic Notes in Theoretical Computer Science, 44(4), 2001.
19. Special Issue on Integration and Collaboration Based on Graphical Techniques. H. Ehrig, M. Goedicke, J. Padberg Journal on Integrated Design and Process Technology 5(1), 2001.
20. Newsletter of the European Association of Software Science and Technology Editor: J. Padberg, http://www.easst.org/newsletter/index.html, Volumes 1-10; 2000-2005.

## Conference Papers and Contributions to Books

21. Loose Semantics of Petri Nets. Lecture Notes in Computer Science 3393, 2005, with H. J. Kreowski.
22. Graph Grammars and Petri Net Transformations. Lecture Notes in Computer Science 3098, 2004, with H. Ehrig.
23. Rule-based refinement of Petri nets: A survey. Lecture Notes in Computer Science 2472, Springer, 2002, with M. Urbàek.
24. Rule invariants in graph transformation systems for analyzing safety-critical systems. Lecture Notes in Computer Science 2505, Springer, 2002, with B. Enders.
25. Transforming specification architectures by GenGED. Lecture Notes in Computer Science 2505, Springer, 2002, with R. Bardohl, C. Ermel.
26. High-level net processes. Lecture Notes in Computer Science 2300, Springer, 2002, with H. Ehrig, K. Hoffmann, P. Baldan, R. Heckel.
27. From basic views and aspects to integration of specification formalisms. Current Trends in Theoretical Computer Science: Entering the 21st Century World Scientific, 2001, with H. Ehrig, F. Orejas.
28. Classification of Petri nets using adjoint functors. Current Trends in Theoretical Computer Science: Entering the 21st Century World Scientific, 2001.
29. Introduction to parameterized net classes. Lecture Notes in Computer Science 2128. Springer, 2001, with H. Ehrig.
30. Behaviour and realization construction for Petri nets based on free monoid and power set graphs. Lecture Notes in Computer Science 2128. Springer, 2001, with H. Ehrig, G. Rozenberg.
31. Stepwise introduction and preservation of safety properties in algebraic high-level net systems. Lecture Notes in Computer Science 1783. Springer, 2000, with M. Gajewsky, K. Hoffmann.
32. New concepts for high-level Petri nets in the application domain of train control systems. Proc. Vol.2, 9th Symposium Control in Transportation Systems, 2000, with P. Schiller, H. Ehrig.
33. Double-pullback graph transitions: a rule-based framework with incomplete information. Lecture Notes in Computer Science 1764, Springer, 2000, with H. Ehrig, R. Heckel, M. Llabres, F. Orejas, G. Rozenberg.
34. Relevance, integration and classification of specification formalism and formal specification techniques. Fortschritt-Berichte VDI 12(436), VDI Verlag, 2000, with H. Ehrig, F. Orejas.
35. Modeling train control systems: from message sequence charts to Petri nets. Fortschritt-Berichte VDI 12(436), VDI Verlag, 2000, with O.Kluge, H. Ehrig.
36. Rule-based refinement of high-level nets preserving safety properties. Lecture Notes in Computer Science 1382, Springer, 1999, with M. Gajewsky, C. Ermel.
37. Construction and Characterization of Double Pullback Transitions. Lecture Notes in Computer Science 1764, 1999, with H. Ehrig, R. Heckel, M. Llabres, F. Orejas, G .Rozenberg.
38. Incremental development of safety properties in Petri net transformations. Lecture Notes in Computer Science 1764, Springer, 1999, with M. Gajewsky, K. Hoffmann.
39. Horizontal and vertical structuring techniques for statecharts. Lecture Notes in Computer Science 1243, Springer, 1998., with H. Ehrig, R. Geisler, M. Klar.
40. Uniform approach to Petri nets. Lecture Notes in Computer Science 1337, Springer, 1997, with H. Ehrig.
41. The category of typed graph grammars and its adjunctions with categories of derivations. Lecture Notes in Computer Science 1037, Springer, 1996, with A. Corradini, H. Ehrig, M. Löwe, U. Montanari.

## Refereed Workshop Papers

42. Case Study: Modelling Telecom Services with Petri Net Modules Electronic Notes in Theoretical Computer Science, 82(7), 2003.
43. Application of Graph Transformation Techniques to the Area of Petri Nets: An Overview Proc. of APPLIGRAPH Workshop on Applied Graph Transformation, 2002, with B. Braatz, H. Ehrig, K. Hoffmann, M. Urbásek.
44. Basic ideas for transformations of specification architectures. Electronic Notes in Theoretical Computer Science, 2002.
45. Visual Design of Software Architecture and Evolution based on Graph Transformation Electronic Notes on Theoretical Computer Science 44(4), 2001, with C. Ermel, R. Bardohl.
46. Preserving liveness with rule-based refinement of place/transition systems. Proc. 6th World Conference on Integrated Design and Process Technology, CD ROM, 2002, with M. Urbáek.
47. Formal relationship between Petri nets and graph grammars as basis for animation views in GenGED. Proc. 6th World Conference on Integrated Design and Process Technology, CD ROM, 2002, with R. Bardohl, C. Ermel
48. Rule-Based and Visual Model Evolution using GenGEd Proc. Workshop on Graph Transformation and Visual Modeling Techniques, 2000. with R. Bardohl, C. Ermel
49. Abstract Petri nets as a uniform approach to high-level Petri nets. WADT 98, Lecture Notes in Computer Science 1589, Springer, 1999.
50. Graph Transformations and Other Rule-Based Formalisms with Incomplete Information Proc. 6th Int. Workshop on Theory and Application of Graph Transformation, 1998, with H. Ehrig, G. Rozenberg.
51. Reverse Petri Net Technology Transfer: On the Boundary of Theory and Application Proc. Formal Methods Pacific, 1997, with H. Ehrig, M. Gajewsky, S. Lembke.
52. Requirements engineering of a medical information system using rule-based refinement of Petri nets. Proc. Conference on Integrated Design and Process Technology, 1996, with C. Ermel, H. Ehrig.
53. Abstract datatype semantics for algebraic high-level nets using dynamic abstract datatypes. Proc. of the Int. Workshop on Quality of Communication-Based Systems, Kluwer Academic Publishers,1995.
54. Functorial Semantics for Safe Graph Grammars Using Prime Algebraic Domains and Event Structures. Proc. Int. Workshop on Graph Grammars and Their Application to Computer Science, 1994, with A. Corradini, H. Ehrig, M. Löwe, U. Montanari.
55. Algebraic High-Level Nets - Petri Nets Revisited. Proc. Joint ADT-COMPASS Workshop, Lecture Notes in Computer Science 785,1994, with H. Eh rig, L. Ribeiro.

## PhD Thesis

56. Abstract Petri nets: Uniform approach and rule-based refinement. PhD Thesis TU Berlin, Shaker-Verlag. 1996.