Electronic Notes in Theoretical Computer Science 44 No. 4 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume44.html 23 pages

Representation and Execution of Petri Nets Using Rewriting Logic as a Unifying Framework

Mark-Oliver Stehr, José Meseguer, and Peter Csaba Ölveczky

Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA

Abstract

We give a high-level overview of our recent results which extend and unify the lines of research initiated by Meseguer and Montanari under the motto "Petri nets are monoids", and by Marti-Oliet, Meseguer and other authors in their linear logic axiomization of Petri nets. In particular, we investigate the use of rewriting logic, which was partially inspired by the two aforementioned approaches, as a unifying semantic framework for different Petri net models. To this end, we equip place/transition nets with a rewriting semantics which is sound and complete in the strong categorical sense of a natural isomorphism between the Best-Devillers process semantics and the semantics obtained via rewriting logic. In addition to place/transition nets we consider algebraic net specifications which subsume colored Petri nets and we show how a corresponding sound and complete rewriting semantics can be established at this level. Furthermore, we discuss how rewriting logic can be used to represent other important extensions of the basic Petri net model such as place/transition nets with test arcs and timed Petri nets. Apart from the conceptual unification of different models in a field which has to cope with increasing diversity, this work has interesting practical applications ranging from the execution and analysis of Petri net models, using a rewriting engine such as Maude, to formal verification taking advantage of the logical side rather than the operational side of rewriting logic.

1 Introduction

In this paper we give a high-level overview of our recent results [51] which attempt to contribute to the general goal of unifying Petri net models by studying in detail the unification of a wide range of such models within rewriting logic [42]. Using rewriting logic as a logical and semantic framework, we show how place/transition nets, nets with test arcs, algebraic net specifications, colored Petri nets, and timed Petri nets can all be naturally represented. Our work extends in substantial ways previous work on the rewriting logic representation of place/transition nets [42], nets with test arcs [43], algebraic net specifications [62], and timed Petri nets [56].

The representations in question associate a rewrite specification to each net in a given class of Petri net models in such a way that concurrent computations in the original net naturally coincide with concurrent computations in the associated rewrite specification. That is, we exhibit appropriate bijections between Petri net computations and rewriting logic computations, viewed as equivalence classes of proofs, that is, as elements of the free model associated to the corresponding rewrite specification [42].

Furthermore, for certain classes of nets, namely place/transition nets and a general form of algebraic net specifications, which subsume the well-known class of colored Petri nets, we show that the representation maps into rewriting logic are *functorial*; that is, that they map in a functorial way net morphisms to rewrite specification morphisms. In addition, such functorial representations can be further extended to the level of *semantic models*, yielding *semantic equivalence theorems* (in the form of natural isomorphisms of functors) between well-known semantic models for the given class of Petri nets and the free models of the corresponding rewrite theories or, more precisely, models obtained from such free models by forgetting some structure.

As we further explain in the body of the paper, this work, including the above-mentioned functorial semantics and the semantic equivalences, generalizes in some ways, and complements in others, a substantial body of work initiated by the second author in joint work with Ugo Montanari under the motto "Petri nets are monoids" [46,39,40,49,18,47,48,50,10,11], in which categorical models are naturally associated as semantic models to Petri nets, and are shown to be equivalent to well-known "true concurrency" models. Our work is also related to linear logic representations of Petri nets [39,40,3,9,8,22]. All this is not surprising, since, as explained in [42], both the categorical place/transition net models of [46] and the linear logic representations of place/transition nets inspired rewriting logic as a generalization of both formalisms. But, as shown in this paper, the extra algebraic expressiveness of rewriting logic is very useful to model in a simple and natural way not only place/transition nets, but also *high-level nets*, such as algebraic net specifications, colored Petri nets, and timed Petri nets.

Our proposed unification of Petri net models is not only of conceptual interest. Given that, under reasonable assumptions, rewrite theories can be executed, the representation maps that we propose provide a uniform operational semantics in terms of efficient logical deduction. Furthermore, using a rewriting logic language implementation such as Maude [17,16], or the Real-Time Maude tool in the timed case [55,54], it is possible to use the results of this paper to create execution environments for different classes of Petri nets. In addition, because of Maude's reflective capabilities [15], the Petri nets thus represented cannot only be executed, but they can also be formally analyzed and model checked by means of *rewriting strategies* that explore and analyze at the metalevel the different rewriting computations of a given rewrite specification.

The purpose of this paper is to introduce the main ideas, concepts and results of our work in a somewhat informal style and with no proofs. For a much more detailed presentation we refer the reader to [51].

2 Preliminaries

2.1 Membership Equational Logic

Membership equational logic (MEL) [7,44] is a many-sorted logic with subsorts and overloading of function symbols. It can express partiality very directly by defining membership in a sort by means of membership equational conditions. In accordance with the terminology introduced in the references above we refer to the types in the logic as *kinds*, and we view the *sorts* for each kind as unary predicates. The atomic sentences are *equalities* $M = N$ for terms M,N of the same kind, and *memberships* $M : s$ for M a term and s a sort, both of the same kind. Sentences of MEL are universally quantified Horn clauses on the atoms. In contrast to an entirely loose or entirely initial semantics of membership equational theories, in practice a mixed specification style is used, where certain subtheories are intended to be equipped with initial interpretations, or they are interpreted freely over their parameter specifications. To make such restrictions on the models explicit in the specification we enrich a membership equational theory by initiality and freeness constraints [20,30], and refer to this enriched theory as a *membership equational logic specification (MES)*. MESs together with their morphisms form a category **MES**. Given a MES $\mathcal{S}$ we also have the category of $\mathcal{S}$-algebras $\mathbf{Mod}(\mathcal{S})$. Each MES morphism $H : \mathcal{S} \to \mathcal{S}'$ induces an obvious forgetful functor $\mathbf{Mod}(H) : \mathbf{Mod}(\mathcal{S}') \to \mathbf{Mod}(\mathcal{S})$ that we also write as $\mathbf{U}_H$. In fact, we have a contravariant functor $\mathbf{Mod} : \mathbf{MES} \to \mathbf{Cat}^{\mathrm{op}}$.

Given a MES $\mathcal{S}$, the *operational semantics* [7], that can be used to efficiently execute a specification under certain assumptions, is explained using a refinement of $\mathcal{S}$, namely by viewing $E_{\mathcal{S}}$ as being composed of a set $E^S_{\mathcal{S}}$ of *structural axioms* and a set $E^C_{\mathcal{S}}$ of *computational axioms*, i.e., $E = E^S_{\mathcal{S}} \uplus E^C_{\mathcal{S}}$. The equations in $E^C_{\mathcal{S}}$ can be seen as reduction rules that operate modulo the equational theory induced by $E^S_{\mathcal{T}}$. In order for a MES to be efficiently *executable* we impose certain restrictions concerning variables as well as confluence, termination, sort-deceasingness and regularity [7].

2.2 Rewriting Logic

In the simplified setting of [42] a rewrite specification $\mathcal{R}$ consists of a single-sorted signature $\Omega_{\mathcal{R}}$, a set $E_{\mathcal{R}}$ of equations over $\Omega_{\mathcal{R}}$, and a set $R_{\mathcal{R}}$ of labelled rewrite rules of the form $\forall\ X\ .\ l : M \to N$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$, where l is a label, M and N are $\Omega_{\mathcal{R}}$-terms, and $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ is a $\Omega_{\mathcal{R}}$-condition[1] over the variable set X. The rewrite rules in $R_{\mathcal{R}}$ are applied *modulo* the equations $E_{\mathcal{R}}$. *Rewriting logic (RWL)* has rules of deduction to infer all rewrites, i.e., those sentences of the form $P : M \rightarrow N$ that are valid in a given rewrite specification [42]. A *rewrite* $P : M \rightarrow N$ means that the term M *rewrites* to the term N modulo $E_{\mathcal{R}}$, and this rewrite is witnessed by the *proof term* P. Apart from general (concurrent) rewrites $P : M \rightarrow N$ that are generated from identity and atomic rewrites by parallel and sequential composition, rewriting logic classifies its most basic rewrites as follows: a *one-step (concurrent) rewrite* is generated by parallel composition from identity and atomic rewrites and contains at least one atomic rewrite, and a *one-step sequential rewrite* is a one-step rewrite containing exactly one atomic rewrite.

From a more general point of view, rewriting logic is parameterized by the choice of its underlying equational logic, which can be single-sorted, many-sorted, order-sorted and so on. In the design of the Maude language [17,16], *membership equational logic* has been chosen as the underlying equational logic. To introduce rewriting logic over membership equational logic, abbreviated as $\mathrm{RWL}_{\mathrm{MEL}}$ or just RWL, we assume an underlying MES $\mathcal{S}_{\mathcal{R}}$ with a distinguished *data subspecification* $\mathcal{S}^D_{\mathcal{R}}$. The data subspecification specifies the static data part of the system, whereas the remaining part of $\mathcal{S}_{\mathcal{R}}$ specifies the *state space* by introducing the *rewrite kinds*, i.e., kinds not already contained in $\mathcal{S}^D_{\mathcal{R}}$ whose terms correspond to states and therefore can be rewritten, together with their algebraic structure, which characterizes the possibilities of parallel composition. In the context of this paper the state space is always specified in a purely equational way. More precisely, a *rewrite specification (RWS)* $\mathcal{R}$ consists of a MES $\mathcal{S}_{\mathcal{R}}$ with a distinguished *data subspecification* $\mathcal{S}^D_{\mathcal{R}}$, a set of labels $L_{\mathcal{R}}$, and a set of rules $R_{\mathcal{R}}$ of the form $\forall\ X\ .\ l : M \rightarrow N$ `if` $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ where $l \in L_{\mathcal{R}}$, $\bar{\phi}_1 \wedge \ldots \wedge \bar{\phi}_n$ is a $\mathcal{S}_{\mathcal{R}}$-condition over X and $M, N \in \mathbf{T}_{\mathcal{R}}(X)_k$ in $\mathcal{S}_{\mathcal{R}}$ for a rewrite kind k. To simplify the exposition we identify either $E_{\mathcal{R}}$-equivalent or $E^S_{\mathcal{R}}$-equivalent terms in the context of a RWS $\mathcal{R}$ whenever we are concerned with the algebraic semantics or the operational semantics, respecively. A RWS morphism $H : \mathcal{R} \rightarrow \mathcal{R}'$ consists of a MES morphism $H_{\mathcal{S}} : \mathcal{S}_{\mathcal{R}} \rightarrow \mathcal{S}_{\mathcal{R}'}$ and a function $H_L : L_{\mathcal{R}} \rightarrow L_{\mathcal{R}'}$ such that $H_{\mathcal{S}}$ has a restriction $H_D : \mathcal{S}^D_{\mathcal{R}} \rightarrow \mathcal{S}^D_{\mathcal{R}'}$ to the data subspecification and for each rule $r \in R_{\mathcal{R}}$ there is a rule in $R_{\mathcal{R}'}$ that is $E_{\mathcal{R}'}$-equivalent to $H(r)$ up to a renaming of the variables, where H is lifted to rules in the obvious homomorphic way. RWSs together with their morphisms form a category that is denoted by **RWS**.

The *algebraic semantics* of rewriting logic is defined as follows. A model of a rewrite specification (RWS) $\mathcal{R}$ is a model A of the underlying MES $\mathcal{S}_{\mathcal{R}}$ together with an enriched categorical structure for each set $[\![k]\!]_A$, where k is a rewrite kind. The interpretation of $\_ : \_ \rightarrow \_$, which can be regarded

[1] Rewriting logic as presented in [42] admits rewrites in conditions of rules, but we do not exploit this possibility in the present paper.

as a ternary predicate, is given by the arrows of the category. Sequential composition of rewrite proofs is interpreted by arrow composition, and parallel composition operators are interpreted by enriching the category with an algebraic structure as it has been specified for the rewrite kinds in $\mathcal{S}_\mathcal{R}$. In order to be a model, the category has to satisfy a number of natural requirements, namely, functoriality w.r.t. the algebraic structure that is relevant for the rewrite kinds, the equations in $\mathcal{S}_\mathcal{R}$ that are relevant for the rewrite kinds lifted to arrows, and for each rule of $\mathcal{R}$ the so-called exchange and decomposition laws. For a detailed description of these requirements we refer to [42]. The models of a RWS we consider in this paper are freely generated over models of the data subspecification $\mathcal{S}^D_\mathcal{R}$. In the important case where $\mathcal{S}^D_\mathcal{R}$ is interpreted initially, we obtain precisely the initial model described in [42].

The *operational semantics* of RWSs extends the operational semantics of MESs by applying both the equations $E^C_\mathcal{R}$ *and* rewrite rules $R_\mathcal{R}$ modulo the structural equations $E^S_\mathcal{R}$. In order for a rewrite specification to be efficiently *executable* we require that the underlying MES is executable and we impose a coherence requirement between equations and rules as explained in [51,68]. In fact, rewrite specifications satisfying this requirement can be executed using Maude.

3 Place/Transition Nets

We now give formal definitions of basic nets and define a PTN as a particular form of an inscribed net. Instead of just finite nets we admit infinite nets, but we restrict our attention to nets with transitions that can affect only a finite part of the marking (locality principle) so that each transition can be represented in a finitary way.

A *net* N consists of a set of *places* P_N, a set of *transitions* T_N disjoint from P_N, and a *flow relation* $F_N \subseteq (P_N \times T_N) \cup (T_N \times P_N)$ such that ${}^\bullet t = \{p \mid p \; F_N \; t\}$ and $t^\bullet = \{p \mid p \; F_N \; t\}$ are finite for each $t \in T_N$ (local finiteness). A net is *finite* iff the sets P_N and T_N are finite. Given nets N and N', a *net morphism* $H : N \to N'$ consists of functions $H_P : P_N \to P_{N'}$ and $H_T : T_N \to T_{N'}$ such that $H_P({}^\bullet t) = {}^\bullet H_T(t)$ and $H_P(t^\bullet) = H_T(t)^\bullet$. Nets together with their morphisms form a category **Net**. A *place/transition net (PTN)* $\mathcal{N}$ consists of: (1) a net $N_\mathcal{N}$ and (2) an *arc inscription* $W_\mathcal{N} : F_\mathcal{N} \to \mathbb{N}$. $W_\mathcal{N}$ is extended to $W_\mathcal{N} : (P_\mathcal{N} \times T_\mathcal{N}) \cup (T_\mathcal{N} \times P_\mathcal{N}) \to \mathbb{N}$ in such a way that $(x, y) \notin F_\mathcal{N}$ implies $W_\mathcal{N}(x, y) = 0$. Given PTNs $\mathcal{N}$ and $\mathcal{N}'$, a *PTN morphism* $H : \mathcal{N} \to \mathcal{N}'$ is a net morphism $H : N_\mathcal{N} \to N_{\mathcal{N}'}$ such that $W_{\mathcal{N}'}(p', t') = W_\mathcal{N}(p_1, t) + \ldots + W_\mathcal{N}(p_n, t)$ holds for all $p' \in P_{\mathcal{N}'}$, $t' \in T_{\mathcal{N}'}$, $t \in H^{-1}(t')$, and $\{p_1, \ldots, p_n\} = H^{-1}(p') \cap {}^\bullet t$ with distinct p_i, and the obvious similar condition also holds for $W_{\mathcal{N}'}(t', p')$. PTNs together with their morphisms form a category **PTN**.

The notion of net morphism we use here is more restrictive than the (topological) net morphisms used in [57] and close to, but slightly stronger than, the (algebraic) net morphisms used in [46]. The justification for our definition is that net morphisms should be morphisms in the sense of [57], and should preserve the behaviour in the strongest reasonable sense possible.

Let $\mathcal{N}$ be a PTN. A *marking* is a multiset of places. A *(concurrent) step* is a nonempty finite multiset of transitions. The *set of markings* and the *set of steps* of $\mathcal{N}$ are denoted by $\mathcal{M}_\mathcal{N}$ and $\mathcal{ST}_\mathcal{N}$, respectively. We define *preset* and *postset functions* $\partial_0, \partial_1 : T_\mathcal{N} \to \mathcal{M}_\mathcal{N}$ by $\partial_0(t)(p) = W(p,t)$ and $\partial_1(t)(p) = W(t,p)$, respectively. The *(concurrent) step semantics* of a place/transition net $\mathcal{N}$ is given by the labelled transition system which has $\mathcal{M}_\mathcal{N}$ as its set of states, $\mathcal{ST}_\mathcal{N}$ as its set of labels and a transition relation $\to\ \subseteq \mathcal{M}_\mathcal{N} \times \mathcal{ST}_\mathcal{N} \times \mathcal{M}_\mathcal{N}$ defined by $m_1 \xrightarrow{e} m_2$ iff there is a marking m such that, for all $p \in P_\mathcal{N}$, $m_1(p) = m(p) + \partial_0(e)(p)$ and $m_2(p) = m(p) + \partial_1(e)(p)$. Writing the occurrence rule in the way given above makes it evident that the occurrence of an action replaces its preset by its postset, whereas the remainder of the marking, here denoted by m, is not involved in this process. This is an important fact that is made formally explicit in the classical process semantics and in the more abstract Best-Devillers process semantics [6]. The latter is the semantics that we use in this paper since it reflects the *collective token philosophy* [10] according to which all our equivalence results are stated.

A *(strict) monoidal category (MC)* $\mathbf{C}$ is a category equipped with a monoidal operation $\_ \otimes_\mathbf{C} \_$ and an identity object $id_\mathbf{C}$ such that $\_ \otimes_\mathbf{C} \_$ is an associative bifunctor with left and right identity $id_\mathbf{C}$. A monidal category morphism $h : \mathbf{C} \to \mathbf{C}'$ is a functor that preserves $\_ \otimes \_$ and id, i.e., $h(u \otimes_\mathbf{C} v) = h(u) \otimes_{\mathbf{C}'} h(v)$ and $h(id_\mathbf{C}) = id_{\mathbf{C}'}$. If in addition $\_ \otimes \_$ is commutative, then we have a *(strictly) symmetric (strict) monoidal category (SMC).* The category of SMCs is denoted by **SMC**. The *Best-Devillers process semantics* $\mathbf{BDP}(\mathcal{N})$ of a PTN $\mathcal{N}$ is given by an SMC that has markings as objects and Best-Devillers processes as arrows. Arrow composition is given by sequential composition, the monoidal operation is given by parallel composition, and the identity for an object m is given by the Best-Devillers process without transitions with origin m and destination m. **BDP** is extended to a functor $\mathbf{BDP} : \mathbf{PTN} \to \mathbf{SMC}$ [12].

3.1 Rewriting Semantics

Rewriting logic can provide a direct semantics of PTNs following the motto “Petri nets are monoids” advocated in [46]. In fact, the categorical semantics presented in that work and also the relation between PTNs and linear logic explained in [40] inspired the development of rewriting logic.

As an example consider the PTN modeling an instance of the well-known banker’s problem depicted in Fig. 1, which models the situation of a bank loaning money to (in this case two) clients. This PTN can be represented by the following RWS given in Maude syntax [17,16], which consists of a MES specification and a set of rewrite rules. As usual in Maude, the rewrite kind `[Marking]` is implicitly introduced by introducing a sort `Marking` of this kind.

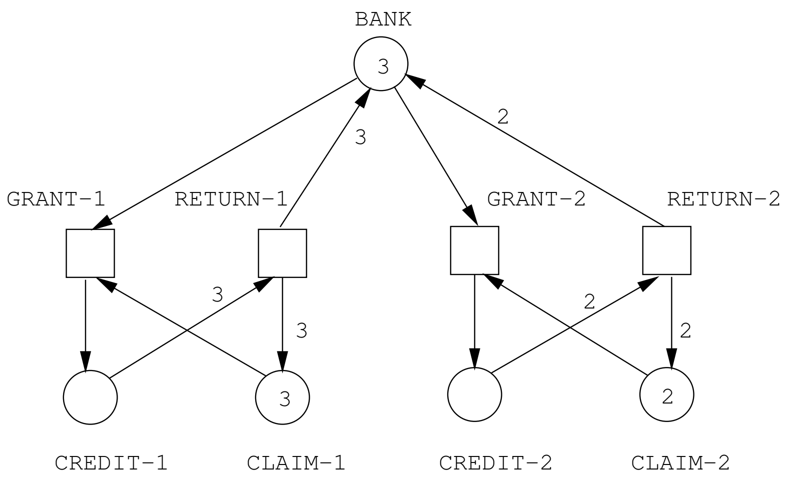

Fig. 1. Banker's problem with two clients

```
sort Marking .

op empty : -> Marking .
op __ : Marking Marking -> Marking [assoc comm id: empty] .

ops BANK CREDIT-1 CREDIT-2 CLAIM-1 CLAIM-2 : -> Marking .

rl [GRANT-1]  : BANK CLAIM-1 => CREDIT-1 .

rl [RETURN-1] : CREDIT-1 CREDIT-1 CREDIT-1 =>
                BANK BANK BANK CLAIM-1 CLAIM-1 CLAIM-1 .

rl [GRANT-2]  : BANK CLAIM-2 => CREDIT-2 .

rl [RETURN-2] : CREDIT-2 CREDIT-2 =>
                BANK BANK CLAIM-2 CLAIM-2 .
```

Here we have applied the translation of PTNs into rewriting logic suggested in [42], which is closely related to the translation of PTNs into linear logic [40]. A marking is represented as an element of the finite multiset sort `Marking`. The constant `empty` represents the empty marking and `__` is the corresponding multiset union operator. Associativity, commutativity and identity laws are specified as structural equations by the operator attributes in square brackets. For each place p there is a constant p, called *token constructor*, representing a single token residing in that place. In fact, under the initial semantics `Marking` is a multiset sort over tokens generated by these token constructors. For each transition t there is a rule, called *transition rule*, labelled by t and stating that its preset marking may be replaced by its postset marking.

In order to control the execution of a RWS the user can specify a strategy which successively selects rewrite rules and initiates rewriting steps. For instance, in the case of the banker's example it is possible to define an execution strategy that avoids states which are necessarily leading to a deadlock such that the banker stays always in the "safe" part of the state space. In applications such as net execution and analysis the choice of a strategy will be guided by the need to explore the behaviour of the system under certain conditions. Strategies are well-supported by the Maude engine via reflection [17,16], i.e. the capability to represent rewrite specifications as objects and control their execution at the meta-level, which makes Maude a suitable tool not only for executing Petri nets but also for analyzing such nets using strategies for (partial) state-space exploration and model checking.

In general, the rewriting semantics for place/transition nets can be conceived as a functor from the category **PTN** to the category **SMRWS** of symmetric monoidal RWSs (SMRWSs) that will be introduced next. The characteristic feature of SMRWSs is that their underlying specification has a single rewrite kind `[Marking]` that is specified to be a free commutative monoid over a set of constants. A RWS $\mathcal{R}$ is a *symmetric monoidal RWS (SMRWS)* iff the following conditions are satisfied: (1) $\mathcal{S}^D_{\mathcal{R}}$ is empty. (2) $\mathcal{S}_{\mathcal{R}}$ contains precisely the following: (a) a kind `[Marking]` together with operator symbols `empty` $: \to$ `[Marking]`, `__` $:$ `[Marking] [Marking]` $\to$ `[Marking]`; (b) any number of operator symbols of the general form $p : \to$ `[Marking]`; (c) structural axioms stating that `__` is associative, commutative and has `empty` as the identity. (3) Rules in $R_{\mathcal{R}}$ do not have conditions and do not contain any variables. A SMRWS morphism is a RWS morphism that preserves `[Marking]`, `empty` and `__`. SMRWSs together with their morphisms form a subcategory of **RWS** denoted **SMRWS**.

The rewriting semantics of PTNs is then defined as follows: Given a PTN $\mathcal{N}$ the *rewriting semantics* of $\mathcal{N}$ is the smallest SMRWS $\mathbf{R}(\mathcal{N})$ such that: (1) $\mathcal{S}_{\mathbf{R}(\mathcal{N})}$ contains a *token constructor* $p : \to$ `[Marking]` for each place $p \in P_{\mathcal{N}}$; (2) $\mathbf{R}(\mathcal{N})$ has a label t and a rule called a *transition rule*, namely,

$$t : \underbrace{p_1 \dots p_1}_{W(p_1,t)} \;\cdots\; \underbrace{p_m \dots p_m}_{W(p_m,t)} \to \underbrace{p_1 \dots p_1}_{W(t,p_1)} \;\cdots\; \underbrace{p_m \dots p_m}_{W(t,p_m)}$$

for each transition $t \in T_{\mathcal{N}}$ assuming $P_{\mathcal{N}} = \{p_1, \dots, p_m\}$ with distinct p_i. $\mathbf{R}$ can be extended to a functor $\mathbf{R} : \mathbf{PTN} \to \mathbf{SMRWS}$ that maps each PTN morphism $H : \mathcal{N} \to \mathcal{N}'$ to the unique SMRWS morphism $G : \mathbf{R}(\mathcal{N}) \to \mathbf{R}(\mathcal{N}')$ with $G_L(t) = H(t)$ for each $t \in T_{\mathcal{N}}$ and $G_{\mathcal{S}}(p) = H(p)$ for each $p \in P_{\mathcal{N}}$.

The main result in this section states that for a PTN $\mathcal{N}$ the Best-Devillers process semantics $\mathbf{BDP}(\mathcal{N})$ coincides with the initial semantics of $\mathbf{R}(\mathcal{N})$ in the strongest possible categorical sense of a natural isomorphism. This theorem is closely related to Corollary 33 in [18], which states that the presentation of an SMC denoted by $\mathcal{T}(\mathcal{N})$ provides a complete and sound axiomatization of Best-Devillers processes. $\mathcal{T}(\mathcal{N})$ is given by a direct inductive equational definition, whereas here we use a rewriting logic as a formal specification language to express the same category. Below we use a functor $\mathbf{V} \circ \Sigma\mathbf{I} : \mathbf{SMRWS} \to$

SMC, where $\Sigma\mathbf{I}$ sends an SMRWS to its initial model, and $\mathbf{V}$ forgets the algebraic structure that is beyond a SMC.

Theorem 3.1 *There is a natural isomorphism* $\widehat{\boldsymbol{\tau}} : \mathbf{BDP} \to \mathbf{V} \circ \Sigma\mathbf{I} \circ \mathbf{R}$ *between the functors* $\mathbf{BDP} : \mathbf{PTN} \to \mathbf{SMC}$ *and* $\mathbf{V} \circ \Sigma\mathbf{I} \circ \mathbf{R} : \mathbf{PTN} \to \mathbf{SMC}$.

Corollary 3.2 *The rewrite specification* $\mathbf{R}(\mathcal{N})$ *provides a sound and complete axiomatization of the Best-Devillers processes of the PTN* $\mathcal{N}$.

3.2 Petri Nets with Test Arcs

In this section we illustrate how the techniques for giving a rewriting logic semantics to place/transition nets can be extended to deal with the important class of place/transition nets with *test arcs* [14,52,69,13]. Petri nets have been equipped with test arcs (also called *read arcs*, or *positive contexts* in contextual nets [52]) to naturally model cases where a certain resource may be *read without being consumed* by a transition.

Formally, a place/transition net with test arcs $\mathcal{N}$ is a place/transition net together with a set of *test arcs* $TA_{\mathcal{N}} \subseteq P_{\mathcal{N}} \times T_{\mathcal{N}}$. We define the *context function* $\partial_{TA} : T_{\mathcal{N}}^{\oplus} \to \mathcal{M}$ on finite multisets e of transitions by $\partial_{TA}(e)(p) = 1$ if there is a transition $t \in e$ with $(p, t) \in TA_{\mathcal{N}}$, and by $\partial_{TA}(e)(p) = 0$ otherwise. The *step semantics* of a place/transition net with test arcs is defined as for place/transition nets (see Section 3) with the modification that for $m_1 \xrightarrow{e} m_2$ to hold we require additionally that, for each place $p \in P_{\mathcal{N}}$, $\partial_{TA}(e)(p) \leq m(p)$.

We propose a rewriting semantics for a place/transition net with test arcs, defined in terms of a rewrite specification $\mathbf{R}(\mathcal{N})$ similar to the one in Section 3.1, but specifying tokens by means of a kind `[Place]` and two operators $\texttt{[\_]}, \langle\_\rangle$: `[Place]` $\to$ `[Marking]`, so that a token residing at place p is represented by the term `[`p`]`. An occurrence of `[`p`]` may not be shared by more than one rewrite at the same time; to allow simultaneous rewrites with *read-only* access to a token at place p, we consider a token `[`p`]` to be equivalent to an arbitrary number of read-only tokens of the form $\langle p\rangle$. This can be accomplished, using a technique described in [43], by adding to our specification $\mathbf{R}(\mathcal{N})$ an operator $\{\_|\_\}$: `[Marking] [Nat]` $\to$ `[Marking]` and two "copying" axioms[2] `[`p`]` $= \{p\,|\,0\}$ and $\{p\,|\,n\} = \{p\,|\,n{+}1\}\langle p\rangle$, where p and n are variables ranging, respectively, over `[Place]` and `[Nat]`.

A transition t which consumes the tokens $a_1, \ldots, a_n$, produces the tokens $b_1, \ldots, b_m$, and "reads" the tokens $c_1, \ldots, c_k$, is modeled by a rewrite rule

$$t : \texttt{[}a_1\texttt{]} \ldots \texttt{[}a_n\texttt{]}\ \langle c_1\rangle \ldots \langle c_k\rangle \longrightarrow \texttt{[}b_1\texttt{]} \ldots \texttt{[}b_m\texttt{]}\ \langle c_1\rangle \ldots \langle c_k\rangle.$$

[2] The counting of the read-only copies and their read-only use guarantee that all the copies must have been "folded back together" in order for the original token to be engaged in a transition that consumes the token.

The resulting categorical semantics of such rewrite specifications is closely related to the one proposed by Bruni and Sassone in [13].

4 High-Level Petri Nets

We use the term *high-level Petri nets* to refer to a range of extensions of PTNs by individual tokens, a line of research that has been initiated by the introduction of *predicate/transition nets* in [28,29,27]. High-level Petri nets make use of an underlying formalism, such as first-order logic in the case of predicate/transition nets, to describe the information that is associated with each token and its transformation. *Colored nets*[3] introduced in [33] are another quite general model of this kind with a more set-theoretic flavour. They generalize PTNs in such a way that tokens can be arbitrary set-theoretic objects. Quite different from, but closely related to, colored nets are high-level Petri nets that use an algebraic specification language as an underlying formalism [66,5,67,60,58,59,19,4]. In this paper we subsume such approaches under the general notion of *algebraic net specifications*, parameterized over an underlying equational specification language. The main feature that algebraic net specifications have in common with predicate/transition nets is that an algebraic net specification does not necessarily specify a single colored net, but instead denotes a *class* of colored nets that satisfy the specification. In the following we first define colored nets, and then we introduce algebraic net specifications over MEL, a straightforward generalization of algebraic net specifications over many-sorted equational-logic (MSA). Both, algebraic net specifications and rewriting logic are specification formalisms that admit a variety of models. From an even more general point of view that is only briefly sketched in this paper, one can define colored net specifications parameterized over an underlying logic. In fact, predicate/transition nets can essentially be regarded as colored net specifications over first-order logic. From this more general point of view we restrict our attention in this paper to the particular class of colored net specifications over MEL, that we also call algebraic net specifications (over MEL), to establish a systematic connection to rewriting logic (over MEL).

4.1 Colored Nets and Colored Net Specifications

Algebraic net specifications will be introduced later as a formal specification language for colored nets. In the following we define the most general set-theoretic version of colored nets [33].

Colored nets are nets with places, transitions, and arcs inscribed with additional information given by functions C and W. The color set $C(p)$ of

[3] In fact, the nets introduced in [33] are called *colored Petri nets (CPNs)*, but this name has later been used for the more syntactic version introduced in [34], which is also the sense for which we would like to reserve this term (see below).

a place p is the set of possible objects p can carry. The color set $C(t)$ of a transition t can be seen as a set of modes in which t may occur. The arc inscription W defines a multiset of objects ("colored" tokens) that are transported by an arc when the associated transition occurs. In fact, this multiset may depend on the mode in which the transition occurs, which is why $W(p,t)$ and $W(t,p)$ take the form of functions in the definition below.

A *colored net (CN)* $\mathcal{N}$ consists of: (1) a finite net $N_{\mathcal{N}}$; (2) a set of *color sets* $CS_{\mathcal{N}}$; (3) a *color function* $C_{\mathcal{N}} : P_{\mathcal{N}} \cup T_{\mathcal{N}} \to CS_{\mathcal{N}}$; and (4) an *arc inscription* $W_{\mathcal{N}}$ on $F_{\mathcal{N}}$ such that $W_{\mathcal{N}}(p,t) : C_{\mathcal{N}}(t) \to C_{\mathcal{N}}(p)^{\oplus}$ and $W_{\mathcal{N}}(t,p) : C_{\mathcal{N}}(t) \to C_{\mathcal{N}}(p)^{\oplus}$. $W_{\mathcal{N}}$ is extended to a function on $(P_{\mathcal{N}} \times T_{\mathcal{N}}) \cup (T_{\mathcal{N}} \times P_{\mathcal{N}})$ in such a way that $(p,t) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(p,t)(b) = \emptyset$, and $(t,p) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(t,p)(b) = \emptyset$ for each $b \in C_{\mathcal{N}}(t)$. CNs together with suitable morphisms form a category denoted by **CN**.

Although CNs can be seen as a generalization of PTNs, there is a more fundamental justification for introducing CNs, namely, that a CN is just a convenient abbreviation for a typically rather complex PTN [34,27]. Indeed, this connection can be exploited to lift low-level concepts such as markings, safe processes, and Best-Devillers processes to the higher level. This is achieved by the following flattening functor $(\_)^{\flat} : \mathbf{CN} \to \mathbf{PTN}$ which associates to each CN the PTN obtained by "spatial unfolding." Given a CN $\mathcal{N}$, we define the *flattening* $\mathcal{N}^{\flat}$ of $\mathcal{N}$ as the unique PTN that satisfies: (1) $P_{\mathcal{N}^{\flat}} = \{(p,c) \mid p \in P_{\mathcal{N}}, c \in C_{\mathcal{N}}(p)\}$; (2) $T_{\mathcal{N}^{\flat}} = \{(t,b) \mid t \in T_{\mathcal{N}}, b \in C_{\mathcal{N}}(t)\}$; (3) $W_{\mathcal{N}^{\flat}}((p,c),(t,b)) = W_{\mathcal{N}}(p,t)(b)(c)$; and (4) $W_{\mathcal{N}^{\flat}}((t,b),(p,c)) = W_{\mathcal{N}}(t,p)(b)(c)$ for $p \in P_{\mathcal{N}}, c \in C_{\mathcal{N}}(p), t \in T_{\mathcal{N}}, b \in C_{\mathcal{N}}(t)$.

Colored Petri nets, a more syntactic, finitary version of colored nets based on an underlying programming language, are proposed in [34]. A point worth remarking is that this definition leaves open the particular choice of the underlying programming language. We use $\mathbf{CPN}_{\mathcal{L}}$ to abbreviate the class of colored Petri nets over a programming language $\mathcal{L}$.

As a useful concept, we informally introduce *colored net specifications (CNS)* which capture the essential idea shared by predicate/transition nets and algebraic net specifications, namely, that they denote an entire class of colored nets instead of just a single one. In fact, there is a general concept of CNSs that is parameterized by an underlying logic. A logic has a deductive system and a model-theoretic semantics, a concept that can be formalized by general logics [41] which contain institutions [30] as the model-theoretic component. We denote by $\mathbf{CNS}_{\mathcal{L}}$ the class of colored net specifications over the underlying logic $\mathcal{L}$. Possible candidates for $\mathcal{L}$ include equational logics such as many-sorted equational logic (MSA), order-sorted equational logic (OSA), or membership equational logic (MEL). We refer to CNSs over such equational logics also as *algebraic net specifications (ANS)*, and we denote by $\mathbf{ANS}_{\mathcal{L}}$ the class of algebraic net specifications over $\mathcal{L}$. Obviously, there are other possible choices for the underlying logic, such as full first-order logic (as in predicate/transition nets), a version of higher-order logic, or a higher-order algebraic specification language (as in [32]).

4.2 Algebraic Net Specifications

In the following we use the term *algebraic net specification (ANS)* to specifically refer to ANSs over MEL, since MEL is sufficiently expressive to cover other commonly used algebraic specification languages such as MSA and OSA [44]. The use of MEL is particularly attractive, because it is weak enough to admit initial models. Indeed, under the initial semantics (which can be internally specified using constraints in the data subspecification) an ANS denotes a unique CN. Another benefit of the use of membership equational logic is that, under the restrictions mentioned in Section 2.1, it comes with a natural operational semantics (which is actually implemented in the Maude engine) so that it can be used directly as a programming language or, more generally, as a metalanguage to specify the logical and operational semantics of other specification or programming languages. As a consequence, colored Petri nets in $\mathbf{CPN}_{\mathcal{L}}$ which use $\mathcal{L}$ as a programming language can be seen as a special case of algebraic net specifications in $\mathbf{ANS}_{\mathrm{MEL}}$ if the semantics of $\mathcal{L}$ can be specified in MEL.

Due to the fact that MEL generalizes MSA in an obvious way, ANSs over MEL are a straightforward generalization of ANSs over MSA, i.e., many-sorted algebraic net specifications. Disregarding the issue of the underlying specification language, the definition we give below is equivalent to the one in [36,37], generalizing [58] by so-called *flexible arcs*, which transport variable multisets of tokens in the sense that the number of tokens transported by an arc is not fixed but can depend on the mode in which the associated transition occurs.

An ANS presupposes an underlying specification that has a multiset kind for each place domain. A *MES of finite multisets* over a kind k consists of: (1) kinds k and $\texttt{[FMS}_k\texttt{]}$ with operator symbols $\texttt{empty}_k : \texttt{[FMS}_k\texttt{]}$, $\texttt{single} : k \to \texttt{[FMS}_k\texttt{]}$, $\_\,\_ : \texttt{[FMS}_k\texttt{]}\ \texttt{[FMS}_k\texttt{]} \to \texttt{[FMS}_k\texttt{]}$; (2) equational axioms stating that $\_\,\_$ is associative, commutative and has $\texttt{empty}_k$ as identity; and (3) a constraint stating that this theory is free over k.

The subsequent definition of algebraic net specifications should be regarded as an instance of CNSs over a logic $\mathcal{L}$ choosing MEL for $\mathcal{L}$. In fact, the only requirements that $\mathcal{L}$ has to meet is that it has a notion of type and that it is expressive enough to axiomatize multisets.

An *algebraic net specification (ANS)* $\mathcal{N}$ consists of: (1) a MES $\mathcal{S}_{\mathcal{N}}$; (2) a finite net $N_{\mathcal{N}}$; (3) a *place declaration*, i.e., a function $D_{\mathcal{N}} : P_{\mathcal{N}} \to K_{\mathcal{S}_{\mathcal{N}}}$ assigning a kind $D_{\mathcal{N}}(p)$ to each place $p \in P_{\mathcal{N}}$ such that $\mathcal{S}_{\mathcal{N}}$ includes a MES of finite multisets over $D_{\mathcal{N}}(p)$; (4) a *variable declaration*, i.e., a function $V_{\mathcal{N}}$ on $T_{\mathcal{N}}$ associating to each transition $t \in T_{\mathcal{N}}$ a kinded variable set $V_{\mathcal{N}}(t)$; (5) an *arc inscription*, i.e., a function $W_{\mathcal{N}}$ on $F_{\mathcal{N}}$ such that for $p \in P_{\mathcal{N}}$, $t \in T_{\mathcal{N}}$, (a) $(p, t) \in F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(p, t) \in \mathbf{T}_{\mathcal{S}_{\mathcal{N}}}(V_{\mathcal{N}}(t))_{\texttt{[FMS}_{D_{\mathcal{N}}(p)}\texttt{]}}$ and (b) $(t, p) \in F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(t, p) \in \mathbf{T}_{\mathcal{S}_{\mathcal{N}}}(V_{\mathcal{N}}(t))_{\texttt{[FMS}_{D_{\mathcal{N}}(p)}\texttt{]}}$; and (6) a *guard definition*, i.e., a function $G_{\mathcal{N}}$ on $T_{\mathcal{N}}$ with $G_{\mathcal{N}}(t)$ being an $\mathcal{S}_{\mathcal{N}}$-condition over $V_{\mathcal{N}}(t)$. $W_{\mathcal{N}}$ is extended to a function on $(P_{\mathcal{N}} \times T_{\mathcal{N}}) \cup (T_{\mathcal{N}} \times P_{\mathcal{N}})$ such that $(p,t) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(p,t) = \mathtt{empty}_{D_{\mathcal{N}}(p)}$ and $(t,p) \notin F_{\mathcal{N}}$ implies $W_{\mathcal{N}}(t,p) = \mathtt{empty}_{D_{\mathcal{N}}(p)}$. ANSs together with suitable morphisms form a category **ANS**. An *interpreted ANS* $\mathcal{IN} = (\mathcal{N}, A)$ consists of a ANS $\mathcal{N}$ and a $\mathcal{S}_{\mathcal{N}}$-algebra A. Interpreted ANSs together with suitable morphisms form a category **IANS**.

Interpreted ANSs are considerably richer than CNs, since they contain their specification together with a model equipped with a corresponding algebraic structure. In this sense they are similar to concrete predicate/transition nets [28,29,27] and algebraic high-level nets [21]. In fact, interpreted ANS, concrete predicate/transition nets [26] and algebraic-high-level nets [21] can be regarded as instances of a general notion of *interpreted CNSs.*[4] The transition from interpreted ANSs to CNs can be described by a forgetful functor as follows.

Given an interpreted ANS $(\mathcal{N}, A)$, the *CN semantics* of $(\mathcal{N}, A)$ is given by the CN $\mathbf{CN}(\mathcal{N}, A)$ defined as follows: (1) the underlying net $N_{\mathbf{CN}(\mathcal{N},A)}$ is precisely $N_{\mathcal{N}}$; (2) the color function $C_{\mathbf{CN}(\mathcal{N},A)}$ is defined by $C_{\mathbf{CN}(\mathcal{N},A)}(p) = [\![D_{\mathcal{N}}(p)]\!]_A$ for $p \in P_{\mathcal{N}}$ and $C_{\mathbf{CN}(\mathcal{N},A)}(t) = B_{\mathcal{N},A}(t)$ for $t \in T_{\mathcal{N}}$, where $B_{\mathcal{N},A}(t)$ is the set of *valid bindings*, i.e., the set of assignments $\beta : V_{\mathcal{N}}(t) \to A$ satisfying $G_{\mathcal{N}}(t)$; (3) the set of color sets $CS_{\mathbf{CN}(\mathcal{N},A)}$ is the smallest set that contains all $C_{\mathbf{CN}(\mathcal{N},A)}(x)$ for $x \in P_{\mathcal{N}} \cup T_{\mathcal{N}}$; and (4) the arc inscription $W_{\mathbf{CN}(\mathcal{N},A)}$ is defined by $W_{\mathbf{CN}(\mathcal{N},A)}(p,t)(\beta) = [\![W_{\mathcal{N}}(p,t)]\!]_{A,\beta}$ and $W_{\mathbf{CN}(\mathcal{N},A)}(t,p)(\beta) = [\![W_{\mathcal{N}}(t,p)]\!]_{A,\beta}$ for $p \in P_{\mathcal{N}}$ and $t \in T_{\mathcal{N}}$. **CN** is extended to a functor $\mathbf{CN} : \mathbf{IANS} \to \mathbf{CN}$.

We can lift the flattening functor $(\_)^\flat : \mathbf{CN} \to \mathbf{PTN}$ to interpreted ANSs, denoting also by $(\_)^\flat : \mathbf{IANS} \to \mathbf{PTN}$ the composition $(\_)^\flat \circ \mathbf{CN}$. Using this flattening functor we furthermore can lift $\mathbf{BDP} : \mathbf{PTN} \to \mathbf{SMC}$ by defining $\mathbf{BDP} : \mathbf{IANS} \to \mathbf{SMC}$ as $\mathbf{BDP} \circ (\_)^\flat$.

4.3 Rewriting Semantics

We now define for an arbitrary ANS its associated rewriting semantics and explain in which sense the rewriting semantics is equivalent to the Best-Devillers process semantics of ANSs, which we have defined by lifting the Best-Devillers process semantics of PTNs to ANSs via the flattening construction. First we generalize symmetric monoidal RWSs (SMRWSs) to extended symmetric monoidal RWSs (ESMRWSs) which will serve as a suitable domain for the rewriting semantics. Notice that in ESMRWSs the data subspecification is not required to be empty. A second difference w.r.t. SMRWSs is that token constructors are extended to multisets and place linearity equations are added to reflect the distributed nature of places.

A RWS $\mathcal{R}$ is an *extended symmetric monoidal RWS (ESMRWS)* iff the

[4] To be precise, arc inscriptions have to be restricted, since flexible arcs are not available in predicate/transition nets and algebraic high-level nets.

following conditions are satisfied: (1) $\mathcal{S}_{\mathcal{R}}$ extends $\mathcal{S}^D_{\mathcal{R}}$ by: (a) a new kind `[Marking]` and new operator symbols `empty` : `[Marking]`, `__` : `[Marking]` `[Marking]` $\rightarrow$ `[Marking]`; (b) any number of new operator symbols of the general form p : $[\mathtt{FMS}_k] \rightarrow$ `[Marking]`, where k is a kind in $\mathcal{S}^D_{\mathcal{R}}$ such that $\mathcal{S}^D_{\mathcal{R}}$ includes a MES of finite multisets over k; (c) equational axioms stating that `__` is associative, commutative and has identity `empty`; and (d) the *place linearity equations* $p(\mathtt{empty}_k) = \mathtt{empty}$, $\forall\ a, b : [\mathtt{FMS}_k]\ .\ p(a\ b) = p(a)\ p(b)$ for each operator $p : [\mathtt{FMS}_k] \rightarrow$ `[Marking]` introduced above. (2) Rules in $R_{\mathcal{R}}$ contain only variables with kinds in $\mathcal{S}^D_{\mathcal{R}}$ and have $\mathcal{S}^D_{\mathcal{R}}$-conditions. ESMRWSs together with suitable morphisms, which preserve `[Marking]`, `empty` and `__`, form a subcategory of **RWS** that is denoted by **ESMRWS**.

An *interpreted ESMRWS* $(\mathcal{R}, A)$ consists of an ESMRWS $\mathcal{R}$ and a $\mathcal{S}^D_{\mathcal{R}}$-model A. Interpreted ESMRWSs together with suitable morphisms form a category **IESMRWS**.

Given an ANS $\mathcal{N}$, the *rewriting semantics* of $\mathcal{N}$ is the smallest ESMRWS $\mathbf{R}(\mathcal{N})$ with an underlying data specification $\mathcal{S}^D_{\mathbf{R}(\mathcal{N})} = \mathcal{S}_{\mathcal{N}}$ such that: (1) $\mathcal{S}_{\mathbf{R}(\mathcal{N})}$ contains a *token constructor* p : $[\mathtt{FMS}_{D_{\mathcal{N}}(p)}] \rightarrow$ `[Marking]` for each place $p \in P_{\mathcal{N}}$; and (2) $\mathbf{R}(\mathcal{N})$ has a rule called *transition rule*, namely,

$$\begin{aligned}\forall\ V_{\mathcal{N}}(t)\ .\ t : (&p_1(W_{\mathcal{N}}(p_1, t))\ \ldots\ p_m(W_{\mathcal{N}}(p_m, t))) \rightarrow\\ &(p_1(W_{\mathcal{N}}(t, p_1))\ \ldots\ p_m(W_{\mathcal{N}}(t, p_m)))\ \mathtt{if}\ G_{\mathcal{N}}(t)\end{aligned}$$

for each transition $t \in T_{\mathcal{N}}$, assuming $P_{\mathcal{N}} = \{p_1, \ldots, p_m\}$ with distinct p_i. **R** can be extended to a functor **R** : **ANS** $\rightarrow$ **ESMRWS**, which in turn is naturally extended to a functor **R** : **IANS** $\rightarrow$ **IESMRWS** sending each interpreted ANS $(\mathcal{N}, A)$ to the interpreted ESMRWS $(\mathbf{R}(\mathcal{N}), A)$.

The theorem and the corollary below are stated in complete analogy to the corresponding results for PTNs. We use a functor $\mathbf{V} \circ \Sigma\mathbf{F}$: **IESMRWS** $\rightarrow$ **SMC**, where $\Sigma\mathbf{F}$ sends an interpreted ESMRWS $(\mathcal{R}, A)$ to its free model over A, and **V** forgets the algebraic structure that is beyond a SMC.

Theorem 4.1 *There is a natural isomorphism* $\widetilde{\boldsymbol{\tau}}$: **BDP** $\rightarrow$ $\mathbf{V} \circ \Sigma\mathbf{F} \circ \mathbf{R}$ *between the functors* **BDP** : **IANS** $\rightarrow$ **SMC** *and* $\mathbf{V} \circ \Sigma\mathbf{F} \circ \mathbf{R}$: **IANS** $\rightarrow$ **SMC** *(with* **R** : **IANS** $\rightarrow$ **IESMRWS** *and* $\mathbf{V} \circ \Sigma\mathbf{F}$: **IESMRWS** $\rightarrow$ **SMC***).*

Corollary 4.2 *The interpreted RWS* $\mathbf{R}(\mathcal{N}, A)$ *provides a sound and complete axiomatization of the Best-Devillers processes of the interpreted ANS* $(\mathcal{N}, A)$*.*

5 Timed Petri Nets

This section illustrates how an important class of *timed* Petri nets (see e.g. [1,53,31]), namely *interval timed Petri nets* (ITPNs), can be given a rewriting logic semantics. We define ITPNs similarly to the *interval timed colored Petri nets* (ITCPNs) proposed by van der Aalst [1]. ITCPNs appear in the context of colored nets, but to simplify the exposition and focus on real-time features, we abstract from the colors of the tokens and instead use atomic tokens (with timestamps). In addition, ITPNs have a notion of concurrent firing of transitions.

An ITPN is a PTN where the outgoing arcs are inscribed by time intervals denoting the range of possible firing delays of the produced tokens. The set TI of all time intervals, in a time domain $Time$, is the set $TI = \{ [r_1, r_2] \mid r_1, r_2 \in Time \wedge r_1 \leq r_2\}$. An ITPN $\mathcal{N}$ is a tuple $(P_{\mathcal{N}}, T_{\mathcal{N}}, F_{\mathcal{N}}, W_{\mathcal{N}}, D_{\mathcal{N}})$, where $(P_{\mathcal{N}}, T_{\mathcal{N}}, F_{\mathcal{N}}, W_{\mathcal{N}})$ is a PTN, and the function $D_{\mathcal{N}} : F_{\mathcal{N}} \cap (T_{\mathcal{N}} \times P_{\mathcal{N}}) \to TI^{\oplus}$, with $|D_{\mathcal{N}}(t,p)| = W_{\mathcal{N}}(t,p)$, is a delay inscription.

In the ITPN model, as in the ITCPN model, we attach to each token a *timestamp*, which indicates the time when a token becomes available. The set $\mathcal{M}_{\mathcal{N}}$ of *markings* of an ITPN $\mathcal{N}$ is, therefore, the set $(P_{\mathcal{N}} \times Time)^{\oplus}$ of all finite multisets of pairs (p, r) representing the presence of a token at place p with timestamp r. The *enabling time* of a transition is the maximum timestamp of the tokens to be consumed. Transitions are *eager* to fire (i.e., they fire as soon as possible). Therefore the transition with the smallest enabling time will fire first. Firing is an atomic action, producing tokens with a timestamp equal to the firing time plus some *firing delay* specified by the delay inscription. A finite multiset of transitions, firing at the same time constitutes a *(concurrent) step*. The formal definition of the step semantics of an ITPN is given in [56,51].

We have proposed in [56] a framework for modeling real-time and hybrid systems in rewriting logic by means of *real-time rewrite theories*, and have shown that a number of well-known models of real-time and hybrid systems can naturally be specified as such theories. A real-time rewrite theory is a rewrite theory including a sort *Time*, and an operator $\{\_\}$ which encloses the global state of the system and is used to ensure that time advances uniformly in all parts of the system. In addition to ordinary rewrite rules modeling instantaneous change in a system, a real-time rewrite theory may contain *tick rules* of the form $l : \{t\} \xrightarrow{\tau_l} \{t'\}$ **if** C, which model times elapse in a system, and where the term τ_l of sort *Time* denotes the duration of the rule. The total time elapse $\tau(\alpha)$ of a rewrite proof $\alpha : \{u\} \longrightarrow \{u'\}$ is defined as the sum of the time elapsed in each tick rule application in α. Even though it is useful to highlight the real-time aspects of a system using real-time rewrite theories, we have shown in [56] that, by adding an explicit clock, such theories are reducible to ordinary rewrite theories in a way that preserves all their expected properties.

In real-time systems, some actions are *eager*, that is, their application should take precedence over the application of time-advancing tick rules. We divide the rules of a real-time rewrite theory into *eager* and *lazy* rules, and define the *admissible rewrites* [56] to be the subset of all rewrites satisfying the additional requirement that a lazy rule may only be applied when no eager rule is applicable. The Real-Time Maude language and tool [54,55] supports the specification and analysis of real-time rewrite theories, including the possibility to define eager and lazy rules.

The rewrite semantics of interval timed Petri nets generalizes the rewrite semantics of (untimed) place/transition nets given in Section 3.1. For the sake of simplicity of the rewriting logic representation of ITPNs, we choose not to carry the timestamps in the tokens. Instead, a term $\texttt{[}p\texttt{]}$ of sort `VisibleMarking` represents a occurrence of a token at place p that is “visible”, i.e., available for consumption. A token that *will be* visible at place p in time r is represented by the term $\texttt{dly(}p, r\texttt{)}$, which has the sort `DelayedMarking` whenever $r \neq 0$.[5] The sort `Marking` is a supersort of the sorts `VisibleMarking` and `DelayedMarking`, and denotes multisets of these two forms of tokens, where multiset union is represented by juxtaposition. The function `mte` takes a term of sort `DelayedMarking` and returns the time elapse until the next delayed token becomes visible. The function `delta` models the effect of the passage of time on delayed tokens by decreasing their delays according to the time elapsed.

The rewrite semantics of an ITPN $\mathcal{N} = (P_\mathcal{N}, T_\mathcal{N}, F_\mathcal{N}, W_\mathcal{N}, D_\mathcal{N})$ is a real-time rewrite theory $\mathbf{R}(\mathcal{N})$ whose signature Ω and axioms E define the sort `Marking` and the functions `delta` and `mte`. The set of rules of $\mathbf{R}(\mathcal{N})$ consists of a *lazy* tick rule modeling time elapse and, for each transition t in $T_\mathcal{N}$, an *eager* rule

$$t : \overbrace{\texttt{[}p_1\texttt{]} \ \ldots \ \texttt{[}p_1\texttt{]}}^{W(p_1,t)} \ \ldots \ \overbrace{\texttt{[}p_n\texttt{]} \ \ldots \ \texttt{[}p_n\texttt{]}}^{W(p_n,t)} \longrightarrow$$

$$\underbrace{\texttt{dly(}p_1, x_{1,1}\texttt{)} \ \ldots \texttt{dly(}p_1, x_{1,W(t,p_1)}\texttt{)}}_{W(t,p_1)} \ \ldots \ \underbrace{\texttt{dly(}p_n, x_{n,1}\texttt{)} \ \ldots \texttt{dly(}p_n, x_{n,W(t,p_n)}\texttt{)}}_{W(t,p_n)}$$

$$\textbf{if } (l_{1,1} \leq x_{1,1} \leq u_{1,1}) \wedge \ldots \wedge (l_{1,W(t,p_1)} \leq x_{1,W(t,p_1)} \leq u_{1,W(t,p_1)}) \wedge \ldots$$

$$\wedge (l_{n,1} \leq x_{n,1} \leq u_{n,1}) \wedge \ldots \wedge (l_{n,W(t,p_n)} \leq x_{n,W(t,p_n)} \leq u_{n,W(t,p_n)})$$

where $P_\mathcal{N} = \{p_1, \ldots, p_n\}$, with p_i distinct, $D(t, p_i)$ is the multiset $\{[l_{i,1}, u_{i,1}], \ldots, [l_{i,W(t,p_i)}, u_{i,W(t,p_i)}]\}$ for each $p_i \in P_\mathcal{N}$, and $x_{i,j}$ are distinct variables of sort *Time*. For example, a transition `t` which consumes two tokens from place `a`, and one token from place `b`, and produces one token at each of the places `c` and `d`, with respective delay intervals $[l_c, u_c]$ and $[l_d, u_d]$, is represented by the following eager rewrite rule:

$$\texttt{t} : \ \texttt{[a] [a] [b]} \longrightarrow \texttt{dly(c, X) dly(d, Y)} \textbf{ if } l_c \leq \texttt{X} \leq u_c \ \wedge \ l_d \leq \texttt{Y} \leq u_d.$$

The following lazy tick rule, where `VM` is a variable of sort `VisibleMarking` and `DM` is a variable of sort `DelayedMarking`, advances time until the first delayed token becomes visible:

$$\texttt{tick} : \ \{\texttt{VM DM}\} \overset{\texttt{mte(DM)}}{\longrightarrow} \{\texttt{VM delta(DM, mte(DM)}\} \textbf{ if } \texttt{mte(DM)} \neq \infty.$$

[5] No interesting information about time is lost by this simplification, since the time when a firing of a transition occurs can always be extracted from the proof term which represents the entire computation.

The tick rule computes the time until the next delayed token becomes visible, and advances time by that amount. After such a tick, the tick rule is again enabled but, due to its being lazy, it will not be applied if the new visible token(s) enable some transition(s) (whose firing in turn could immediately trigger further instantaneous transitions).

We have shown in [56,51] a correspondence between step sequences in $\mathcal{N}$ and admissible rewrites in $\mathbf{R}(\mathcal{N})$, namely that each (concurrent) step in $\mathcal{N}$ can be represented by a one-step concurrent rewrite in $\mathbf{R}(\mathcal{N})$, taking place at the same time and performing the same multiset of transitions, and, conversely, that each one-step (concurrent) rewrite in $\mathbf{R}(\mathcal{N})$ is either an application of the `tick` rule advancing time, or corresponds to a step in $\mathcal{N}$, taking place at the same time.

6 Conclusions

In this paper we have given a high-level overview of how rewriting logic can be used as a semantic framework in which a wide range of Petri net models can be naturally unified. Specifically, we have explored how place/transition nets, nets with test arcs, algebraic net specifications, colored Petri nets, and timed Petri nets can all be naturally expressed in rewriting logic, and how well-known semantic models often coincide with (in the sense of being naturally isomorphic to) the natural semantic models associated to the rewriting logic representations of the given nets. The general way of representing Petri nets within rewriting logic that we propose is by no means limited to the net classes explicitly discussed in this paper. Although a careful study still has to be carried out, we briefly discuss in [51] some ideas about how similar representations could be defined for other Petri net classes, such as colored Petri nets based on higher-order programming languages [34], colored net specifications over higher-order logics, nets with macroplaces [2], nets with FIFO places [35,25,23], object-oriented variants of Petri nets [61,38], and object nets [63,64,24,65], where nets are viewed as token objects.

In our view, the unification of Petri net models within the rewriting logic framework is useful not only for conceptual reasons, but also for purposes of execution, formal analysis, and formal reasoning about Petri net specifications. Using the reflective and metalanguage capabilities of Maude, it is possible to build execution environments for Petri net specifications where the language description provided by the user and the user interaction could all take place at the Petri net level with which the user is familiar. Similarly, the Real-Time Maude tool [55] could offer corresponding capabilities for executing and analyzing timed Petri net models. Indeed, there is already a substantial body of experience and case studies on the Maude executable specification and formal analysis of consurrent and distributed systems in general (see [45]) and of Petri nets in particular [62,56], suggesting that the Maude-based development of execution and formal analysis environments for Petri nets is a clear and promising practical application of the conceptual unification of Petri net models suggested in this paper.

Acknowledgement

Support by DARPA through Rome Laboratories Contract F30602-C-0312 and NASA through Contract NAS2-98073, by Office of Naval Research Contract N00014-99-C-0198, and by National Science Foundation Grant CCR-9900334 is gratefully acknowledged. Part of this work is based on earlier work conducted by the first author at University of Hamburg, Germany and in the scope of the European Community project MATCH (CHRX-CT94-0452). Furthermore, we would like to thank Roberto Bruni, Narciso Martí-Oliet, Rüdiger Valk and the referees for their constructive criticism and many helpful suggestions.

References

[1] W. M. P. van der Aalst. Interval timed coloured Petri nets and their analysis. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472. Springer-Verlag, 1993.

[2] N. A. Anisimov, K. Kishinski, A. Miloslavski, and P. A. Postupalski. Macroplaces in high level Petri nets: Application for design inbound call center. In *Proceedings of the Int. Conference on Information System Analysis and Synthesis (ISAS'96), Orlando, FL, USA*, pages 153–160, July 1996.

[3] A. Asperti. A logic for concurrency. unpublished manuscript, November 1987.

[4] E. Battiston, F. De Cindio, and G. Mauri. OBJSA nets: a class of high-level nets having objects as domains. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 340 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[5] B. Berthomieu, N. Choquet, C. Colin, B. Loyer, J. M. Martin, and A. Mauboussin. Abstract data nets: Combining Petri nets and abstract data types for high level specifications of distributed systems. In *Proc. of the Seventh Workshop on Applications and Theory of Petri Nets, Oxford, UK*, pages 25–48, 1986.

[6] E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55:87–136, 1987.

[7] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236:35–132, 2000.

[8] C. Brown. Relating Petri nets to formulae of linear logic. Technical Report ECS-LFCS-89-87, Laboratory of Foundations of Computer Science, University of Edinburgh, June 1989.

[9] C. Brown and D. Gurr. A categorical linear framework for Petri nets. In *Proc. Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 208–218, June 1990.

[10] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. A comparison of Petri net semantics under the collective token philosophy. In J. Hsiang and A. Ohori, editors, *Proceedings of ASIAN'98, 4th Asian Computing Science Conference*, volume 1538 of *Lecture Notes in Computer Science*, pages 225–244. Springer-Verlag, 1998.

[11] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for Petri nets under the individual token philosophy. In *Proc. Category Theory and Computer Science, Edinburgh, Scottland, September 1999*, volume 29 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1999. `http://www.elsevier.nl/locate/entcs/volume29.html`.

[12] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. Functorial semantics for petri nets. *Information and Computation*, 2001. To appear.

[13] R. Bruni and V. Sassone. Algebraic models for contextual nets. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming. Proceedings 2000.*, volume 1853. Springer-Verlag, 2000.

[14] S. Christensen and N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, 1993.

[15] M. Clavel. Reflection in general logics and in rewriting logic, with applications to the Maude language. Ph.D. Thesis, University of Navarre, 1998.

[16] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. A tutorial on Maude. SRI International, March 2000, `http://maude.csl.sri.com`.

[17] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic.* Computer Science Laboratory, SRI International, Menlo Park, 1999. `http://maude.csl.sri.com`.

[18] P. Degano, J. Meseguer, and U. Montanari. Axiomizing the algebra of net computations and processes. *Acta Informatica*, 33:641–667, 1996.

[19] C. Dimitrovici, U. Hummert, and L. Petrucci. Semantics, composition and net properties of algebraic high-level nets. In G. Rozenberg, editor, *Advances in Petri Nets 1991*, volume 524 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[20] F. Durán and J. Meseguer. Structured theories and institutions. In M. Hofmann, G. Rosolini, and D. Pavlović, editors, *Proceedings of CTCS'99, 8th Conference on Category Theory and Computer Science, Edinburgh, Scotland, U.K., September 10-12, 1999*, volume 29, pages 71–90. Elsevier, 1999. `http://www.elsevier.nl/locate/entcs/volume29.html`.

[21] H. Ehrig, J. Padberg, and L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In *Recent Trends in Data Type Specification*, volume 785 of *Springer-Verlag*, pages 188–206, 1994.

[22] U. Engberg and G. Winskel. Petri nets as models of linear logic. In A. Arnold, editor, *CAAP'90*, volume 431 of *Lecture Notes in Computer Science*, pages 147–161. Springer-Verlag, 1990.

[23] J. Fanchon. FIFO-net models for processes with asynchronous communication. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 152–178. Springer-Verlag, 1992.

[24] B. Farwer. A linear logic view of object Petri nets. *Fundamenta Informaticae*, 37(3):225–246, 1999.

[25] A. Finkel and A. Choquet. FIFO nets without order deadlock. *Acta Informatica*, 25(1):15–36, 1988.

[26] H. J. Genrich. Equivalence transformation of PrT-nets. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424, pages 179–208. Springer-Verlag, 1990.

[27] H. J. Genrich. Predicate/transition nets. In *High-Level Petri Nets: Theory and Practice*, pages 3–43. Springer-Verlag, 1991.

[28] H. J. Genrich and K. Lautenbach. The analysis of distributed systems by means of predicate/transition-nets. In G. Kahn, editor, *Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 123–146, Berlin, 1979. Springer-Verlag.

[29] H. J. Genrich and K. Lautenbach. System modelling with high-level Petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[30] J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[31] H. M. Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 282–299. Springer-Verlag, 1993.

[32] K. Hoffmann. Run time modification of algebraic high level nets and algebraic higher order nets using folding and unfolding construction. In G. Hommel, editor, *Communication-Based Systems, Proceedings of the 3rd International Workshop held at the TU Berlin, Germany, 31 March – 1 April 2000*, pages 55–72. Kluwer Academic Publishers, 2000.

[33] K. Jensen. Coloured Petri nets and the invariant-method. *Theoretical Computer Science*, pages 317–336, 1981.

[34] K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use.*, volume 1 of *EATCS monographs on theoretical computer science.* Springer-Verlag, 1992.

[35] E. Kettunen, E. Montonen, and T. Tuuliniemi. Comparison of Pr-net based channel models. In *Proc. of the 12th IMACS World Conf.*, volume 3, pages 479–482, 1988.

[36] E. Kindler and W. Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, (51):16–31, December 1996.

[37] E. Kindler and H. Völzer. Flexibility in algebraic nets. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998, 19th International Conference, ICATPN'98, Lisbon, Portugal, June 1998, Proceedings*, volume 1420 of *Lecture Notes in Computer Science*, pages 345–384. Springer-Verlag, 1998.

[38] C. A. Lakos. From coloured Petri nets to object Petri nets. In M. Diaz G. De Michelis, editor, *Application and Theory of Petri Nets*, volume 935 of *Lecture Notes in Computer Science*, pages 278–297, Berlin, 1995. Springer-Verlag.

[39] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic. *Mathematical Structures in Computer Science*, 1:69–101, 1991.

[40] N. Martí-Oliet and J. Meseguer. From Petri nets to linear logic through categories: A survey. *International Journal of Foundations of Computer Science*, 2(4):297–399, 1991.

[41] J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987*, pages 275–329. North-Holland, 1989.

[42] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[43] J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In U. Montanari and V. Sassone, editors, *Proc. Concur'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 331–372. Springer-Verlag, 1996.

[44] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT '97, Tarquinia, Italy, June 3-7, 1997, Selected Papers*, volume 1376 of *Lecture Notes in Computer Science*, pages 18 – 61. Springer-Verlag, 1998.

[45] J. Meseguer. Rewriting logic and Maude: a wide-spectrum semantic framework for object-based distributed systems. In S. Smith and C.L. Talcott, editors, *Formal Methods for Open Object-based Distributed Systems, FMOODS 2000*, pages 89–117. Kluwer, 2000.

[46] J. Meseguer and U. Montanari. Petri nets are monoids. *Information and Computation*, 88(2):105–155, October 1990.

[47] J. Meseguer, U. Montanari, and V. Sassone. On the semantics of Petri nets. In W.R. Cleaveland, editor, *Proceedings of the Concur'92 Conference, Stony*

Brook, New York, August 1992, volume 630 of *Lecture Notes in Computer Science*, pages 286–301. Springer-Verlag, 1992.

[48] J. Meseguer, U. Montanari, and V. Sassone. On the model of computation of place/transition Petri nets. In *Proceedings 15th International Conference on Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 16–38. Springer-Verlag, 1994.

[49] J. Meseguer, U. Montanari, and V. Sassone. Process versus unfolding semantics for place/transition Petri nets. *Theoretical Computer Science*, 153(1–2):171–210, 1996.

[50] J. Meseguer, U. Montanari, and V. Sassone. Representation theorems for Petri nets. In C. Freska, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential, Theory, Cognition*, volume 1337 of *Lecture Notes in Computer Science*, pages 239–249. Springer-Verlag, 1997.

[51] J. Meseguer, P. C. Ölveczky, and M.-O. Stehr. Rewriting logic as a unifying framework for Petri nets. In H. Ehrig, G. Juhas, J. Padberg, and G. Rozenberg, editors, *Unifying Petri Nets*, Lecture Notes in Computer Science (Advances in Petri Nets). Springer-Verlag, December 2001. To appear.

[52] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32:545–596, 1995.

[53] S. Morasca, M. Pezzè, and M. Trubian. Timed high-level nets. *The Journal of Real-Time Systems*, 3:165–189, 1991.

[54] P. C. Ölveczky. *Specification and Analysis of Real-Time and Hybrid Systems in Rewriting Logic.* PhD thesis, University of Bergen, 2000. Available at `http://maude.csl.sri.com/papers`.

[55] P. C. Ölveczky and J. Meseguer. Real-Time Maude: A tool for simulating and analyzing real-time and hybrid systems. In *Third International Workshop on Rewriting Logic and its Applications*, 2000. To appear in *Electronic Notes in Theoretical Computer Science.*

[56] P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. To appear in *Theoretical Computer Science.* Available at `http://maude.csl.sri.com/papers`, September 2000.

[57] C. A. Petri. Nets, time and space. *Theoretical Computer Science*, 153(1–2):3–48, 1996.

[58] W. Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991.

[59] W. Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets.* Springer-Verlag, 1998.

[60] W. Reisig and J. Vautherin. An algebraic approach to high level Petri nets. In *Proceedings of the Eighth European Workshop on Application and Theory of Petri Nets*, pages 51–72. Universidad de Zaragoza (Spain), 1987.

[61] C. Sibertin-Blanc. Cooperative nets. In R. Valette, editor, *Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 471–490, Berlin, 1994. Springer-Verlag.

[62] M.-O. Stehr. A rewriting semantics for algebraic nets. In C. Girault and R. Valk, editors, *Petri Nets for Systems Engineering – A Guide to Modelling, Verification, and Applications.* Springer-Verlag, 2001. To appear.

[63] R. Valk. Petri nets as dynamical objects. In *Workshop Proc. 16th International Conf. on Application and Theory of Petri Nets, Torino, Italy*, June 1995.

[64] R. Valk. Petri nets as token objects: An introduction to elementary object nets. In J. Desel and M. Silva, editors, *Proceedings of the 19th International Conference on Application and Theory of Petri Nets, Lissabon, June 22-26, 1998*, volume 1420 of *Lecture Notes in Computer Science*, pages 1 – 25. Springer-Verlag, 1998.

[65] R. Valk. Relating Different Semantics for Object Petri Nets. Technical report, FBI-HH-B-266/00, Fachbereich Informatik, Universität Hamburg, 2000.

[66] J. Vautherin. *Un Modele Algebrique, Base sur les Reseaux de Petri, pour l'Etude des Systemes Paralleles.* These de Docteur Ingenieur, Univ. de Paris-Sud, Centre d'Orsay, June 1985.

[67] J. Vautherin. Parallel systems specifications with coloured Petri nets and algebraic specifications. *Lecture Notes in Computer Science: Advances in Petri Nets 1987*, 266:293–308, 1987.

[68] P. Viry. Rewriting: An effective model of concurrency. In C. Halatsis, D. Maritsas, G. Philokyprou, and S. Theodoridis, editors, *PARLE'94 – Parallel Architectures and Languages Europe, 6th Int. PARLE Conf. Athes, Greece, July 1994, Proceedings.*, volume 817 of *Lecture Notes in Computer Science*, pages 648–660. Springer-Verlag, 1994.

[69] W. Vogler. Partial order semantics and test arcs. In *Proc. MFCS'97*, volume 1295 of *Lecture Notes in Computer Science.* Springer-Verlag, 1997.