Graph-Rewriting Petri Nets

Géza Kulcsár(✉), Malte Lochau, and Andy Schürr

Real-Time Systems Lab, TU Darmstadt, Darmstadt, Germany
{geza.kulcsar,malte.lochau,andy.schuerr}@es.tu-darmstadt.de

Abstract. Controlled graph rewriting enhances expressiveness of plain graph-rewriting systems (i.e., sets of graph-rewriting rules) by introducing additional constructs for explicitly controlling graph-rewriting rule applications. In this regard, a formal semantic foundation for controlled graph rewriting is inevitable as a reliable basis for tool-based specification and automated analysis of graph-based algorithms. Although several promising attempts have been proposed in the literature, a comprehensive theory of controlled graph rewriting capturing semantic subtleties of advanced control constructs provided by practical tools is still an open challenge. In this paper, we propose graph-rewriting Petri nets (GPN) as a novel foundation for unifying control-flow and rule-application semantics of controlled graph rewriting. GPN instantiate coloured Petri nets with categorical DPO-based graph-rewriting theory where token colours denote typed graphs and graph morphisms and transitions define templates for guarded graph-rewriting rule applications. Hence, GPN enjoy the rich body of specification and analysis techniques of Petri nets including inherent notions of concurrency. To demonstrate expressiveness of GPN, we present a case study by means of a topology-control algorithm for wireless sensor networks.

1 Introduction

Graph-rewriting systems provide an expressive and theoretically founded, yet practically applicable and tool-supported framework for the specification and automated analysis of complex software systems [5]. Due to their declarative nature, graph-rewriting systems are inherently non-deterministic as (1) multiple graph-rewriting rules might be simultaneously applicable to an input graph and (2) the input graph might contain multiple matches for a rule application. The lack of expressiveness in *controlling* rule applications may obstruct a precise specification of more complicated graph-based algorithms involving sequential, conditional, iterative (and intentionally non-terminating), or even concurrent composition of rule applications. To tackle this problem, various tools (e.g., PROGRES [18], Fujaba [7] and eMoflon [14]) incorporate *controlled* (or, programmed) graph rewriting enriching declarative graph rewriting with additional constructs for explicitly restricting the order and matches of rule applications.

This work has been partially funded by the German Research Foundation (DFG) as part of project A1 within CRC1053–MAKI.

L. Lambers and J. Weber (Eds.): ICGT 2018, LNCS 10887, pp. 79–96, 2018.
https://doi.org/10.1007/978-3-319-92991-0_6

Bunke was one of the first to lay a formal foundation for controlled graph rewriting by proposing a generalization of programmed string grammars to graph grammars [2]. Since then, several promising attempts have been proposed to investigate essential semantic aspects of graph-rewriting processes. Schürr proposes a semantic domain for controlled graph-rewriting systems with sequencing, choice and recursion using a *denotational* fix-point characterization of valid input-output graph pairs [17]. The *transformation units* of Kreowski et al. [13] provide a high-level, flexible formalism to describe controlled graph-rewriting processes, however, as in the previous case, operational semantics and formal analysis aspects are out-of-scope. In contrast, Plump and Steinert propose an *operational* semantics for the programmed graph-rewriting language *GP*, mostly focusing on algorithmic aspects and data structures for executing *GP* programs on input graphs [16]. Further existing approaches, like those of Guerra and de Lara [9] as well as Wimmer et al. [20], integrate graph-based model transformations and control structures of coloured Petri nets for purposes similar to those pursued in this paper. Nevertheless, those approaches rely on out-of-the-box model-transformation engines and, thus, do not address issues arising from graph-rewriting theory. While *algebraic higher-order nets* by Hoffmann and Mossakowski [10] provide a means to handle graphs attached to tokens in Petri nets models, they do not extend PN transition semantics for algebraic graph rewriting as in our work. Corradini et al. propose a *trace* notion and a corresponding trace-based equivalence relation for (non-controlled) graph-rewriting processes [3]. Thereupon, Corradini et al. propose *graph processes* as a compact representation of sequences of rule applications incorporating a static notion of independence inspired by Petri nets [1,4].

To summarize, existing formalisms mostly focus on particular formal aspects concerning (mostly operational) semantics of (controlled) graph rewriting, whereas a comprehensive theory of controlled graph rewriting is still an open issue. From a theoretical point of view, recent approaches often lack a proper trade-off between *separation* and *integration* of the two competing, yet partially overlapping, theories involved in controlled graph rewriting, namely categorical graph-rewriting theory (e.g., DPO/SPO style) and process theory (e.g., operational/denotational semantics). As a result, essential theoretical concepts established in both worlds in potentially different ways (e.g., notions of nondeterminism, conflict, independence and concurrency) may be obscured when being combined into one formal framework, owing to the different interpretations of those concepts in their respective originating theory. As a crucial example, *concurrency* in graph rewriting usually amounts to independently composed, yet operationally interleaved rule application steps, thus leaving open the *truly* concurrent aspects of operational step semantics. From a practical point of view, recent approaches are often *incomplete*, missing semantic subtleties of advanced constructs provided by recent tools (e.g., negative application conditions, propagation of sub-graph bindings to synchronize matches of rule applications etc. [7,14,19]). (Note that another important aspect of graph rewriting in practice, namely attributed graphs, is out-of-scope for this paper.)

In this paper, we propose *graph-rewriting Petri nets* (*GPN*) as a novel foundation of controlled graph rewriting unifying control-flow and rule-application semantics of those systems in a comprehensive and intuitive way. To this end, we aim at integrating constructs for expressing control-flow specifications, sub-graph bindings and concurrency in controlled graph rewriting into a unified formal framework which allows for automated reasoning using state-of-the-art analysis techniques. GPN are based on coloured Petri nets (CPN) [11], a backward-compatible extension of Petri nets introducing notions of typed data attached to tokens (so-called colours) being processed by guarded transitions. In particular, GPN instantiate CPN with categorical DPO-based graph-rewriting theory where token colours denote type graphs and graph morphisms and transitions define templates for guarded graph-rewriting rule applications. In this way, rule applications are composeable in arbitrary ways in GPN, including inherent notions of concurrency. In addition, our GPN theory incorporates advanced constructs including complex transition guards and (concurrent) propagation of multiple graphs and/or sub-graph matches among rule applications. The resulting GPN theory preserves major theoretical properties and accompanying reasoning techniques of the underlying (coloured) Petri net framework such as reachability analysis as well as natural notions of conflict and independence of controlled graph-rewriting rule applications. To demonstrate expressiveness of GPN, we present a case study by means of a topology-control algorithm for wireless sensor networks and we describe how to reason about essential correctness properties of those systems by utilizing the underlying Petri net theory.

2 An Illustrative Example: WSN Topology Control

We first illustrate controlled graph rewriting by an example: a simplified *wireless sensor network* (*WSN*) in which autonomous, mobile sensors communicate through wireless channels. WSN communication topologies evolve over time (e.g., due to unpredictable status changes of nodes or links) thus potentially deteriorating topology quality. The quality of topologies is usually measured by performance metrics and other properties like energy consumption of nodes. *Topology control* (*TC*) is a widely used technique to maintain or improve quality of topologies in a proactive manner, by continuously adapting the status of communication links [15]. In a realistic WSN scenario, TC is necessarily employed in a decentralized setting, where an external *environment* sporadically impacts the current topology in an unpredictable manner. When analyzing WSN evolution, those changes are interleaved with proactively performed TC actions which might lead to inconsistent states or violations of essential WSN properties (e.g., network connectedness). Nevertheless, existing approaches assume that TC operates on a consistent snapshot of the entire topology thus abstracting from realistic environment behaviors and concurrency aspects [12].

Figure 1 shows four stages in the evolution of a sample WSN topology, starting from topology T_0 (Fig. 1a). We denote the current physical distance between sensor nodes by edges either being labeled *short* or *long* (`S`,`L`). From stage T_0

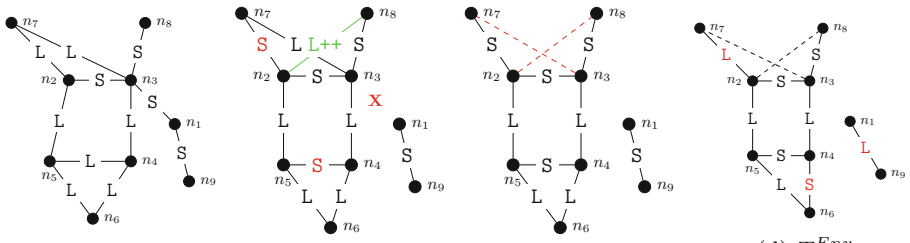
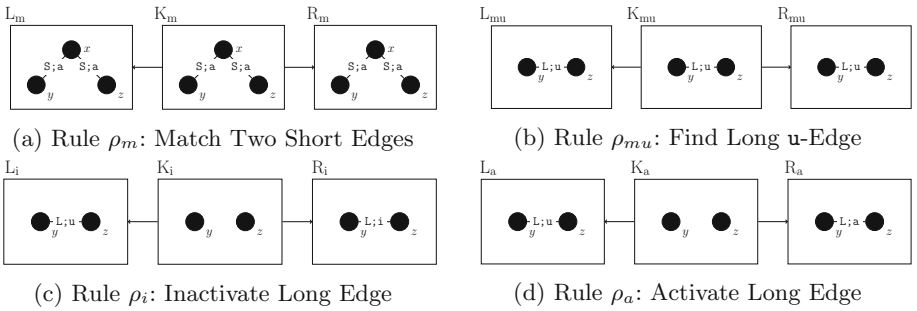


Fig. 1. WSN example: topology control and environmental changes

to T_1^{Env}, the environment causes the following changes: edges n_2n_7 and n_4n_5 become short (indicated by the new labels in red), a new long edge n_2n_8 appears (in green with label `L++`), and edge n_1n_3 falls out due to communication failure or insufficient range (indicated by a red cross), causing the topology to lose *connectedness*. In the next stage T_2^{TC}, a TC action *inactivates* the presumably redundant long edges n_3n_7 and n_2n_8 (dashed edges) as both can be replaced by an alternative path via two short edges with better quality measures [12]. Besides, no further long edges exist in the current topology matching this pattern. In the next stage T_3^{Env}, the environment changes n_1n_9 and n_2n_7 into long edges while n_4n_6 becomes short. The change of n_1n_9 does not lead to any TC action, whereas the change of n_2n_7 poses a problem now as the path which previously caused the inactivation of n_3n_7 is not short anymore. In contrast, edges n_4n_5 and n_4n_6 now provide a short path enabling inactivation of edge n_5n_6. In a realistic setting, the change of the distance of n_2n_7 might even cause a *deadlock*: if the TC inactivation action is performed concurrently to the environmental changes, the two short edges required for inactivation are not present anymore. To summarize, TC processes should fulfill the following basic *correctness properties*:

- **(P1a)** Redundancy reduction: TC should eventually inactivate long edges for which there are alternative paths via 2 short edges.
- **(P1b)** Connectedness preservation: TC must preserve connectedness of input-topology graphs.
- **(P2)** Liveness: TC must not deadlock due to concurrent interactions with environmental behaviors.

We next present a specification of TC actions as *graph-rewriting rules* to investigate those properties in more detail. As shown in Fig. 1, we use graphs for representing WSN topologies, with nodes denoting sensors (names given as single letters) and edges denoting bidirectional communication links via (volatile) channels. In addition to length attributes `S` (short) and `L` (long), a further edge attribute represents link status (*active*, *inactive* or *unclassified*) thus leading to six possible edge types `S;a`, `S;i`, `S;u`, `L;a`, `L;i` and `L;u`, respectively. Status *active* indicates that the link is currently used for communication, whereas



(a) Rule ρ_m: Match Two Short Edges

(b) Rule ρ_{mu}: Find Long u-Edge

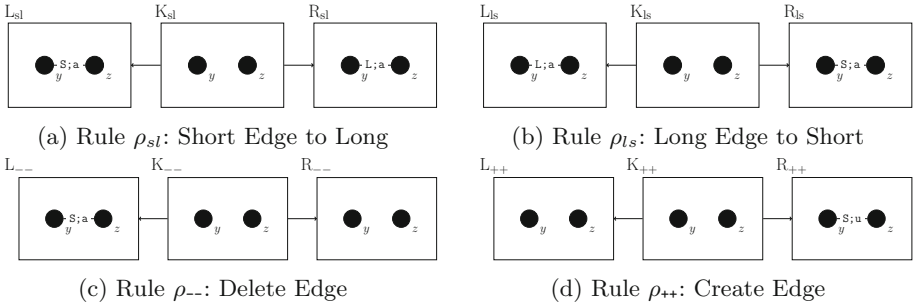
(c) Rule ρ_i: Inactivate Long Edge

(d) Rule ρ_a: Activate Long Edge

Fig. 2. Graph-rewriting rules for topology-control actions

inactive links are currently not in use. Edges with status *unclassified* are in use but require status revision, usually due to environmental changes. Based on this model, the DPO graph-rewriting rules in Fig. 2a–d specify (simplified) TC actions. A DPO-rule ρ consists of three graphs, where L is called *left-hand side*, R *right-hand side* and K *interface*. Graphs L and K specify *deletion* of those parts of L not appearing in K while R and K specify *creation* of those parts of R not appearing in K. An *application* of rule ρ on input graph G consists in (1) finding an occurrence (*match)* of L in G, (2) deleting elements of G according to L and K, and (3) creating new elements in G according to R and K, thus, yielding an *output graph* H. As a common restriction in the WSN domain, nodes in a WSN topology graph are only visible to their direct neighbor nodes thus limiting granularity of graph patterns within graph-rewriting rules, accordingly. For instance, rule ρ_m in Fig. 2a matches two neighboring active short edges (indicated by the label `S;a`) in topology graph G without performing any changes to G. Similarly, rule ρ_{mu} (Fig. 2b) matches a long unclassified (`L;u`) edge. The actual adaptations (rewritings) of the topology graph following those rule applications are performed by rule ρ_i and rule ρ_a (Fig. 2c–d), by inactivating or activating an (`L;u`) edge, respectively. A TC algorithm based on these rules defines a non-terminating TC process repeatably triggered by environmental changes of links as modeled by the rules in Fig. 3. In Fig. 3a–b, we only present the rules for status `a` and in Fig. 3c–d for edge type `S;a` (`S;u`); whereas the complete specification includes 14 such rules (six for length change as well as six for deletions and two for creations of links).

TC continuously searches for active short edge pairs (ρ_m) as well as long unclassified edges (ρ_{mu}), and then *either* inactivates a long unclassified edge (ρ_i) *if it forms a triangle exactly containing the previously matched short edges and the long edge or* it activates such an edge (ρ_a), otherwise. However, a TC algorithm specified in a purely declarative manner by means *uncontrolled* graph-rewriting rule applications is not sufficient for faithfully reasoning about the aforementioned properties. Hence, we conclude four major challenges to be addressed by a comprehensive *controlled* graph-rewriting formalism as will be presented in the following.



(a) Rule ρ_{sl}: Short Edge to Long

(b) Rule ρ_{ls}: Long Edge to Short

(c) Rule ρ_{--}: Delete Edge

(d) Rule ρ_{++}: Create Edge

Fig. 3. Graph-rewriting rules for the environment

(C1: Control-Flow Specification) Control-flow constructs for graph-rewriting processes, namely sequential-, negated-, conditional-, and iterative rule composition to restrict *orderings* of graph-rewriting rule applications.

(C2: Subgraph Binding) Data-flow constructs for propagating subgraph bindings among rules to restrict *matches* of graph-rewriting rule applications (cf. dependencies between rule ρ_m and the subsequent rules in Fig. 2).

(C3: Concurrency) Control-flow and data-flow constructs for concurrent rule composition and synchronization of graph-rewriting processes being compatible with solutions for **C1** and **C2**.

(C4: Automated Analysis) Techniques for automated analysis of correctness properties for algorithms specified as controlled graph-rewriting processes including **C1**, **C2**, **C3**.

3 Foundations

We first recall basic notions of graph rewriting based on category theory [5].

Definition 1 (Graphs, Typed Graphs). *A* (directed) graph *is a tuple* $G = \langle N, E, s, t \rangle$*, where* N *and* E *are sets of* nodes *and* edges*, and* $s, t : E \to N$ *are the* source *and* target functions*. The components of a graph* G *are denoted by* N_G*,* E_G*,* s_G*,* t_G*. A* graph morphism $f : G \to H$ *is a pair of functions* $f = \langle f_N : N_G \to N_H, f_E : E_G \to E_H \rangle$ *such that* $f_N \circ s = s' \circ f_E$ *and* $f_N \circ t = t' \circ f_E$*; it is an* isomorphism *if both* f_N *and* f_E *are bijections. We denote by* **Graph** *the category of graphs and graph morphisms.*

A *typed graph* is a graph whose elements are "labelled" over a fixed *type graph* TG. The category of *graphs typed over* TG is the slice category (**Graph** $\downarrow TG$), also denoted $\mathbf{Graph}_{TG}$ [4].

Definition 2 (Typed Graph-Rewriting System). *A* (TG-typed graph) DPO rule *is a span* $(L \xleftarrow{l} K \xrightarrow{r} R)$ *in* $\mathbf{Graph}_{TG}$*. The typed graphs* L*,* K*, and* R *are called* left-hand side, interface*, and* right-hand side*. A* (TG-typed) graph-rewriting system *is a tuple* $\mathcal{G} = \langle TG, \mathcal{R}, \pi \rangle$*, where* $\mathcal{R}$ *is a set of* rule names *and* π *maps rule names in* $\mathcal{R}$ *to rules.*

We write $\rho : (L \xleftarrow{l} K \xrightarrow{r} R)$ for rule $\pi(\rho)$ with $\rho \in \mathcal{R}$. A *rule application* of rule ρ on input graph G amounts to a construction involving two pushouts by (1) finding a *match* of L in G, represented by morphism m, (2) choosing a pushout complement $K \xrightarrow{k} D \xrightarrow{f} G$ for $K \xrightarrow{l} L \xrightarrow{m} G$, representing the application interface, and (3) creating a pushout $R \xrightarrow{n} H \xleftarrow{g} D$ over $K \xrightarrow{k} D \xrightarrow{f} G$, yielding *output graph* H by deleting and creating elements of G according to the rule span and match m. By $G \overset{\rho@m}{\Longrightarrow} H$ we denote application of rule ρ on input graph G at match m, producing output graph H.

We next revisit *coloured Petri nets* (*CPN*), constituting a conservative extension of plain *Petri nets* (*PN*). To this end, we employ the notations of *multisets*.

Definition 3 (Multiset). *A* multiset *ranged over a non-empty set S is a function $M : S \to \mathbb{N}$ for which the following holds.*

- *$M(s)$ is the* coefficient (number of appearances) *of $s \in S$ in M,*
- *$s \in M \Leftrightarrow M(s) > 0$* (membership),
- *$\mid M \mid = \sum_{s \in S} M(s)$* (size),
- *$(M_1 +\!+ M_2)(s) = M_1(s) + M_2(s)$* (summation),
- *$M_1 \ll\!= M_2 \Leftrightarrow \forall s \in S : M_1(s) \leq M_2(s)$* (comparison),
- *$(M_1 -\!- M_2)(s) = M_1(s) - M_2(s)$ iff $M_2 \ll\!= M_1$* (subtraction), *and*
- *$(n \ast\!\ast M)(s) = n * M(s)$* (scalar multiplication)

The symbol $\sum^{++}$ is defined on multisets according to summation $++$. By $\mathbb{N}^S$ we refer to the *set of all multisets* over set S and by $M_\emptyset$ we denote the *empty multiset* (i.e., $M_\emptyset(s) = 0$ for each $s \in S$). By *i's* we refer to the ith appearance of element s in multiset M, where $1 \leq i \leq M(s)$. As depicted, for instance, in Fig. 6, a PN consists of a set of *places* (circles), a set of *transitions* (rectangles) and a set of (directed) *arcs* (arrows) each leading from a place to a transition or vice versa. *Markings* of places are depicted as filled circles within places.

Definition 4 (Petri Net). *A* Petri net *is a tuple (P, T, A, M_0), where P is a finite set* (places)*, T is a finite set* (transitions) *such that $P \cap T = \emptyset$, $A \subseteq (P \times T) \cup (T \times P)$ is a set* (directed arcs)*, and $M_0 \in \mathbb{N}^P$ is a multiset* (initial marking)*. We use the notations ${}^\bullet t = \{p \in P \mid pAt\}$* (preplaces) *and $t^\bullet = \{p \in P \mid tAp\}$* (postplaces) *for $t \in T$ as well as ${}^\bullet p = \{t \in T \mid tAp\}$ and $p^\bullet = \{t \in T \mid pAt\}$ for $p \in P$, respectively. These notations naturally extend to multisets $X : P \cup T \to \mathbb{N}$ by ${}^\bullet X := \sum_{x \in S \cup T} X(x) \ast\!\ast {}^\bullet x$ and $X^\bullet := \sum_{x \in S \cup T} X(x) \ast\!\ast x^\bullet$.*

Transition $t \in T$ is *enabled* by marking $M \in \mathbb{N}^P$ if all preplaces of t are marked by at least one token. For each *occurrence* of enabled transition t in a *step*, one token is consumed from all preplaces and one token is added to each postplace of t.

Definition 5 (PN Step). *Let N be a PN and $M \in \mathbb{N}^P$ be a* marking*, then $M \xrightarrow{X} M'$ with non-empty $X \in \mathbb{N}^T$ and $M \in \mathbb{N}^P$ is a* step *of N iff (1) ${}^\bullet X \ll\!= M$* (enabledness) *and (2) $M' = (M -\!- {}^\bullet X) +\!+ X^\bullet$* (occurrence).

We may omit X as well as M' in denoting steps $M \xrightarrow{X} M'$ if not relevant.

Definition 6. *Let N be a Petri net.*

- *Marking M' is* reachable *from marking M in N iff there exists a sequence $M \to M'' \to \cdots \to M'$. By $\mathcal{R}_N(M) \subseteq \mathbb{N}^P$ we denote the set of markings reachable from M in N.*
- *Two transitions $t, u \in T$ are* concurrent*, denoted $t \smile u$, iff $\exists M \in \mathcal{R}_N(M_0) : M \xrightarrow{\{t,u\}} M'$.*
- *N is* k-bounded *iff $\forall M \in \mathcal{R}_N(M_0) : \forall p \in P : M(p) \leq k$. A* 1*-bounded Petri net is also called* 1-safe*.*
- *N is* live *iff $\forall M \in \mathcal{R}_N(M_0) \setminus \{M_h\} : M \to$*

Coloured Petri nets (CPN) extend PN by assigning typed data (*colours*) to tokens and by augmenting net elements with *inscriptions* denoting conditions and operations involving typed variables over token data. Let Σ be a finite non-empty set *(types or colours)*, V a finite set *(variables)* and $Type : V \to \Sigma$ a function *(type declaration)*. By $INSCR_V$ we denote the set of *inscriptions* over V and by $Type[v]$ we denote the set of values of variable v. Subscript MS as in $C(p)_{MS}$ denotes the object (e.g., $C(p)$) to be a multiset.

Definition 7 (Coloured Petri Net). *A* Coloured Petri Net (CPN) *is a tuple $(P, T, A, \Sigma, V, C, \Gamma, E, \Xi)$, where*

- *P is a finite set* (places)*,*
- *T is a finite set* (transitions) *such that $P \cap T = \emptyset$,*
- *$A \subseteq (P \times T) \cup (T \times P)$ is a set* (directed arcs)*,*
- *Σ is a finite, non-empty set* (colours)*,*
- *V is a finite set* (typed variables) *such that $\forall v \in V : Type[v] \in \Sigma$,*
- *$C : P \to \Sigma$ is a function* (place colours)*,*
- *$\Gamma : T \to INSCR_V$ is a function* (transition guards) *such that $\forall t \in T : Type[\Gamma(t)] = Bool$,*
- *$E : A \to INSCR_V$ is a function* (arc expressions) *such that $\forall a \in A : Type[E(a)] = C(p)_{MS}$ with $p \in P$ being the place related to a,*
- *$\Xi : P \to INSCR_\emptyset$ is a function* (initialization expressions) *such that $\forall p \in P : Type[\Xi(p)] = C(p)_{MS}$.*

CPN steps extend PN steps by *binding elements* (t, b) of transitions $t \in T$, where b is a function *(binding)* such that $\forall v \in Var(t) : b(v) \in Type[v]$ (i.e., assignments of values to variables occurring in inscriptions of the transition according to the type of the variable). By $B(t)$, we denote the set of all bindings of $t \in T$ and by BE, we denote the set of all binding elements of CPN $\mathcal{N}$. By $\langle var_1 = value_1, var_2 = value_2, \ldots \rangle$, or $\langle b \rangle$ for short, we denote value assignments to variables in binding b. Furthermore, by $Var(t)$, we denote the set of *free variables* of $t \in T$ (i.e., $v \in Var(t)$ either if v occurs in guard $\Gamma(t)$ of t or in inscription $E(a)$ of some arc $a \in A$ having t as source or target).

Definition 8 (CPN Step). *Let* $\mathcal{N}$ *be a CPN and* $\mathcal{M} : P \to \mathbb{N}^{\Sigma}$ *a* coloured marking*, then* $\mathcal{M} \overset{Y}{\Rightarrow} \mathcal{M}'$ *with* $Y \in \mathbb{N}^{BE}$ *and* $\mathcal{M} : P \to \mathbb{N}^{\Sigma}$ *is a* step *of* $\mathcal{N}$ *iff (1)* $\forall (t,b) \in Y : \Gamma(t)\langle b\rangle$ *and* $\forall p \in P : \sum^{++}_{(t,b)\in Y} E(p,t)\langle b\rangle \ll= \mathcal{M}(p)$ (enabledness) *and (2)* $\forall p \in P : \mathcal{M}'(p) = (\mathcal{M}(p) -- \sum^{++}_{(t,b)\in Y} E(p,t)\langle b\rangle) ++ \sum^{++}_{(t,b)\in Y} E(t,p)\langle b\rangle$ (occurrence).

As an example, Fig. 4 shows a basic CPN with one transition, whose inscription will be descried in the next section. The passing of input/output values in a CPN step is handled by arc and guard variables: variable G is bound to the value of input token 1‘**G**, and the produced output token is being bound to the value of variable H.

Coloured marking $\mathcal{M}'$ is *reachable* from $\mathcal{M}$ in $\mathcal{N}$ iff there exists a sequence of CPN steps from $\mathcal{M}$ to $\mathcal{M}'$, denoted as $\mathcal{M} \Rightarrow \mathcal{M}'' \Rightarrow \cdots \Rightarrow \mathcal{M}'$. By $\mathcal{R}^C_{\mathcal{N}}(\mathcal{M})$ we denote the set of coloured markings reachable from $\mathcal{M}$ in $\mathcal{N}$. The set of *reachable coloured markings* of $\mathcal{N}$ is denoted as $\mathcal{R}^C_{\mathcal{N}}(\mathcal{M}_0)$ where the *initial coloured marking* $\mathcal{M}_0$ of $\mathcal{N}$ is defined as $\forall p \in P : \mathcal{M}_0(p) = I(p)\langle\rangle$. Thereupon, all further PN notions of Definition 6 are adaptable to CPN, accordingly.

4 Graph-Rewriting Petri Nets

We now introduce *graph-rewriting Petri nets* (*GPN*) as an instantiation of CPN. To this end, we define *graph-rewriting inscriptions* to specify controlled graph-rewriting processes as GPN.

4.1 GPN Syntax

In GPN, token colours represent objects (i.e., graphs) and morphisms, respectively, from category $\mathbf{Graph}_{TG}$ and transitions correspond to DPO rule applications. Although the distinction between graphs and morphisms is not really necessary from a categorical point of view (as objects may be represented as identity morphisms), it is useful to handle *subgraph bindings* as first-class entity in GPN being distinguished from complete graphs when specifying inputs and outputs of GPN transitions. Notationally, colour set

$$\Sigma_{TG} = \{Obj(\mathbf{Graph}_{TG}), Mor(\mathbf{Graph}_{TG})\}$$

consists of two colours: objects (denoted in capital letters) and morphisms (denoted in non-capital letters) in $\mathbf{Graph}_{TG}$. In the following, we use the same typing for elements of $\mathbf{Graph}_{TG}$) in an obvious way for convenience.

By $INSCR_{TG,V}$, we denote the set of *graph-rewriting inscriptions* over a set of variables V, typed over Σ_{TG}. Transition guards in GPN are given as graph-rewriting inscriptions representing templates for diagrams in $\mathbf{Graph}_{TG}$ having a *bound* part as well as *free* variables thus expressing DPO rule applications potentially involving subgraph bindings. In an inscription I, the diagram template containing variable names is given by a finite category $\mathbf{FV_I}$, where the bound part (i.e., elements already fixed in $\mathbf{Graph}_{TG}$) is given by a binding

functor $B_I : \mathbf{BV_I} \to \mathbf{Graph}_{TG}$ with $\mathbf{BV_I}$ being a finite category. The structure of the diagram is given by the type of the variables as well as further structural properties of the inscription (cf. Φ_I in the next paragraph). The distinction between bound and free names directly corresponds to the respective notion of variables in CPN such that firing a GPN transition t consists in binding free variables of t by extending the binding functor to a complete diagram. Bound elements (from $\mathbf{BV_I}$) are denoted by bold letters (e.g., $\mathbf{L}, \mathbf{r}$) and free variables by sans-serif letters (e.g., G, b).

In addition, a proper graph-rewriting inscription has to involve further categorical properties to hold for the diagram for yielding the intended DPO semantics, including, for instance, commutativity of arrows, pushout squares, as well as non-existence or partiality of morphisms. We refer to this set of properties of inscriptions I by Φ_I. For instance, when utilizing the DPO diagram in Fig. 4b as transition guard I_ρ in which span ($\mathbf{L} \xleftarrow{\mathbf{l}} \mathbf{K} \xrightarrow{\mathbf{r}} \mathbf{R}$) is mapped by B_{I_ρ} to concrete graphs and morphisms according to rule ρ, Φ_{I_ρ} requires both rectangles to be pushouts (and thus to commute), indicated by (PO). In a concurrent setting, GPN inscriptions will be further generalized to compound operations with potentially multiple graphs and subgraph bindings involved. Adapting step semantics of CPN to GPN means that (inscription) *bindings* are functors mapping variable set $\mathbf{FV_I}$ into $\mathbf{Graph}_{TG}$ in a way that conforms to B_I thus completing the bound parts of the inscription.

Definition 9 (Graph-Rewriting Inscription). *A* graph-rewriting inscription $I \in INSCR_{TG,V}$ *is based on the following components:*

(i) a finite category of bound *variables* $\mathbf{BV_I}$ *with* $v \in \mathbf{BV_I} \Rightarrow v \in V$,
(ii) a finite category of free *variables* $\mathbf{FV_I}$ *with* $v \in \mathbf{FV_I} \Rightarrow v \in V$, $\mathbf{BV_I} \subseteq \mathbf{FV_I}$,
(iii) a binding functor $B_I : \mathbf{BV_I} \to \mathbf{Graph}_{TG}$ *such that* $\forall v \in \mathbf{BV_I} : Type(v) = Type(B_I(v))$ *and*
(iv) a set of categorical properties Φ_I *for (the images of)* $\mathbf{FV_I}$.

A binding B *for* I *is a functor* $B : \mathbf{FV_I} \to \mathbf{Graph}_{TG}$ *such that* $B\,|_{\mathbf{BV_I}} = B_I$ *and* $\forall v \in \mathbf{FV_I} : Type(v) = Type(B(v))$. $I\langle B \rangle$ *evaluates to true if* $B(\mathbf{FV_I})$ *satisfies* Φ_I.

A GPN defined over type graph TG is a CPN over Σ_{TG} with inscriptions $I \in INSCR_{TG,V}$.

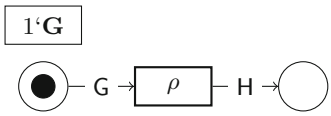
Definition 10 (Graph-Rewriting Petri Net). *A* graph-rewriting Petri net *(*GPN*) over type graph* TG *is a CPN with set of colours* Σ_{TG}*, transition guard function* Γ *and arc expression function* E *ranged over* $INSCR_{TG,V}$*, initial marking function* Ξ *ranged over* $INSCR_{TG,\emptyset}$.

We further require each transition of a GPN to consume and produces at least one graph token (i.e., having non-empty sets of pre- and postplaces).

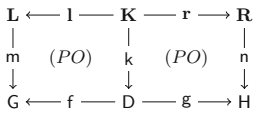
4.2 GPN Semantics

We first illustrate GPN semantics intuitively by describing the GPN pattern corresponding to a basic DPO rule application. Then, we provide a generic pattern for GPN transitions. To avoid bloat of notation, we simplify transition guards using a compact graphical notation.

DPO Rule Application. The most basic GPN pattern in Fig. 4 consists of a transition applying rule $\rho : (L \xleftarrow{l} K \xrightarrow{r} R)$ to input graph G, producing output graph H (both typed over TG). We denote the GPN transition guard simply



(a) Net Structure: DPO Rule Application with Input Graph G



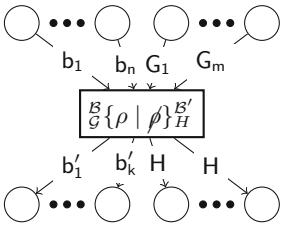
(b) Guard ρ: DPO Application of Rule ρ

Fig. 4. Basic GPN pattern: DPO rule application

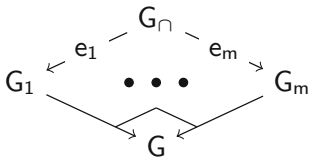
by ρ which is satisfied if $G \overset{\rho@m}{\Longrightarrow} H$ for some match m (cf. Definition 2ff). In GPN terms, guard ρ is satisfied if there is a respective inscription binding such that the complete diagram becomes a DPO rule application (indicated by (PO) as imposed in Φ_ρ).

Hence, input variable G is bound to the graph corresponding to the input token ($1'\mathbf{G}$ denoting a multiset consisting of a single graph $\mathbf{G}$) and the output token corresponds to the output graph bound to variable H. Sequential compositions of this basic GPN pattern specifies controlled sequences of DPO rule applications (incl. loops) restricted to the given ordering, as well as control-flow branches (choices) and joins (i.e., places with multiple outgoing/incoming arcs).

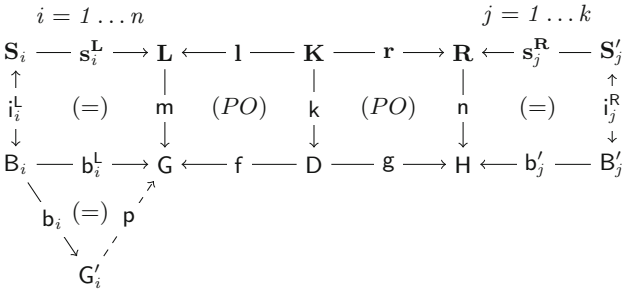
Generic GPN Transition Pattern. Figure 5 shows the most general form (net structure and transition guard) of GPN transitions including *concurrent* behaviors. As represented by the transition label in Fig. 5a, each GPN transition either denotes an application of a rule ρ or a *non-applicability condition* of ρ (i.e., requiring the left-hand side of ρ to *not* have any match in G, where output graph H is omitted in the transition label as G remains unchanged). The set of preplaces of transitions might include a (non-empty) set $\mathcal{G}$ of (simultaneously consumed) input graphs $G_1, G_2, \ldots, G_l$ to each of which either ρ is applied or a non-applicability condition is imposed. Application of rule ρ to a set of input graphs is performed by means of a non-deterministic *merge* as follows: (1) selecting (non-deterministically) a family of injective arrows $G_\cap \xrightarrow{e_l} G_l$, $l = 1 \ldots m$ from an arbitrary graph $G_\cap$ representing a common overlapping of each input graph, and (2) creating a co-limit object over that family in the category of typed



(a) Generic Transition



(b) Generic Guard: Graph Merge



(c) Generic Guard: Rule Application with Subgraphs

Fig. 5. Generic pattern for GPN transitions

graphs, as indicated in Fig. 5b. This diagram is part of the transition guard and the merged graph G is used as input for applying ρ to produce output H. Correspondingly, the set of postplaces might include *forks* into multiple (at least one) concurrent instances of H.

Subgraph Binding and Matching. In addition to control-flow dependencies restricting the *orderings* of rule applications, GPN further allow to express data-flow dependencies among rule applications for restricting the *matches* to which particular rules are to be applied in a given input graph (cf. e.g., Fig. 2, rules ρ_m and ρ_i (ρ_a)). GPN therefore allow to propagate parts of output graphs from preceding rule applications (*subgraph binding*) to be matched within input graphs of subsequent rule applications (*subgraph matching*) via additional tokens coloured by respective matching morphisms. To this end, GPN transitions are equipped with *synchronization interfaces* $R_1 \xleftarrow{s_R} S \xrightarrow{s_L} L_2$ between sequentially applied rules ρ_1 and ρ_2 to be a span relating the right-hand side of ρ_1 with the left-hand side of ρ_2. In Sect. 2, the rules ρ_i and ρ_a require left-hand side synchronization interfaces to ρ_m (cf. Fig. 2) in order to ensure that exactly those edges are matched for a node pair which have been previously matched by ρ_m. This concept is, however, not limited to GPN net structures with sequentially applied rules, but rather generalize to any kind of rule composition. In Fig. 5a, $\mathcal{B}'$ denotes the (possibly empty) set of outgoing *subgraph binding* morphisms, while

$\mathcal{B}$ denotes the (possibly empty) set of incoming *subgraph matching* morphisms of a GPN transition. The rightmost rectangle in Fig. 5c represents *subgraph binding* where the rule-application structure is extended by right-hand side synchronization interfaces of ρ (for some subsequent rule(s)). For each right-hand interface S'_j (indexed by $j = 1 \dots k$), the application of ρ produces an additional token b'_j coloured as morphism for that subgraph of H which is (1) within the co-match of the rule application and (2) corresponds to that part of the co-match designated by $S'_j \xrightarrow{s_j^R} R$. This property is captured by the commutation of the arrows $n \circ s_j^R$ and $b'_j \circ i_j^R$, denoted by (=). The bidirectionality of i_j^R denotes that i_j^R is an isomorphism. Conversely, the leftmost part (rectangle and triangle) of Fig. 5c represents *subgraph matching* for left-hand side synchronization interfaces (indexed by $i = 1 \dots n$, where an application of rule ρ additionally consumes a token coloured as morphism to be checked for compatibility with the input graph. The additional commutative diagram expresses that (1) the bound part B_i of the previous graph G'_i should be also present in the current input graph G (*partial* morphism p commuting with b in the lower triangle) and (2) the occurrence b_i^L of B_i in G should be isomorphic to the left-hand side synchronization interface (the leftmost commutative rectangle). Here, a *partial* graph morphism p between G' and G is a graph morphism $p : G_0 \to G$ where $\exists g : G_0 \to G'$ injective.

Properties of GPN Processes. We utilize the notion of reachable coloured markings of CPN (cf. Sect. 3) to characterize semantic correctness properties on GPN models as described in Sect. 2. In particular, we are interested in *completed* markings only consisting of tokens being coloured over $Obj(\mathbf{Graph}_{TG})$, whereas markings additionally containing tokens coloured over $Mor(\mathbf{Graph}_{TG})$ correspond to intermediate steps of GPN processes.

Definition 11 (GPN Language). *A marking $\mathcal{M}$ of GPN $\mathcal{N}$ is* completed *if for all $p \in P$ with $\mathcal{M}(p) \neq \emptyset$ it holds that $C(p) = Obj(\mathbf{Graph}_{TG})$. The* language *of $\mathcal{N}$ for initial completed marking $\mathcal{M}_0$ is defined as $\mathcal{L}(\mathcal{N}, \mathcal{M}_0) = \mathcal{R}_{\mathcal{N}}^{C\checkmark}(\mathcal{M}_0)$, where $\mathcal{R}_{\mathcal{N}}^{C\checkmark}(\mathcal{M})$ denotes the subset of completed markings reachable from marking $\mathcal{M}$.*

This definition characterizes GPN semantics in terms of input/output correspondences according to the definition of the graph language as the set of graphs generated by a set of rules applied to a given start graph [5]. Note that GPN is able to simulate the standard semantics of plain graph-rewriting systems, i.e., sets of rules: For a given rule set, such a GPN canonically consists of a single place (initially holding a single start graph token) and one transition for every rule, each having this place as its pre- and postplace.

Thereupon, a more elaborated process-centric characterization of GPN semantics might employ a labeled transition system (LTS) representation.

Definition 12 (GPN Processes). *The* LTS *of GPN $\mathcal{N}$ for initial completed marking $\mathcal{M}_0$ is a tuple $(S, s_0, \to, \mathcal{P})$, where $S = \mathbb{N}^{\Sigma_{TG}}$* (processes)*, $s_0 = \mathcal{M}_0$*

(initial process), $\mathcal{M} \xrightarrow{Y} \mathcal{M}'$ *iff* $\mathcal{M} \overset{Y}{\Rightarrow} \mathcal{M}'$ (labeled transitions) *and* $\mathcal{P} = \{pred \mid pred : Obj(\mathbf{Graph}_{TG}) \to Bool\}$ (predicates).

Predicates $pred \in \mathcal{P}$ might be used to define (computable) correctness properties $pred : Obj(\mathbf{Graph}_{TG}) \to Bool$ to be checked on graphs in reachable completed markings (e.g., connectedness as described in Sect. 2). In the next section, we provide an elaborated case study illustrating the intuition of the above definitions, by means of a GPN model for our WSN running example (cf. Sect. 2).

5 Case Study

We now demonstrate how GPN address the challenges in controlled graph-rewriting by revisiting our WSN and TC case study from Sect. 2.

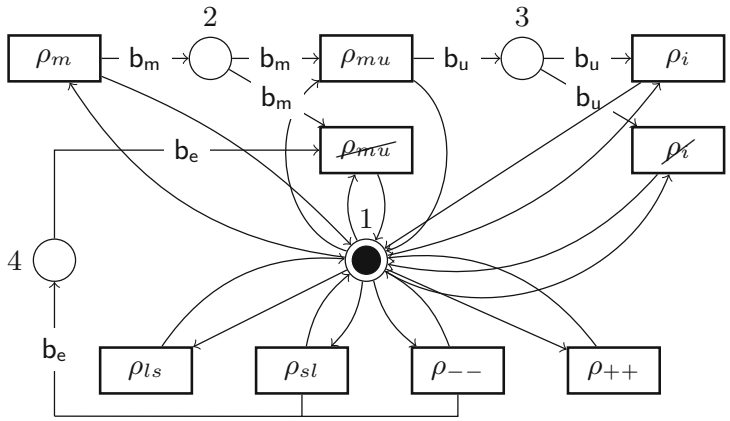


Fig. 6. GPN model for the WSN and TC scenario

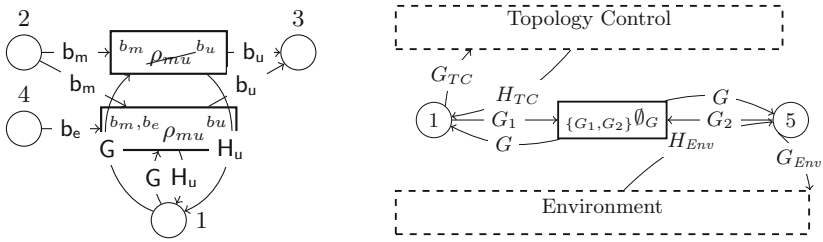
Control-Flow Specification (C1). A possible GPN specification of the WSN and TC scenario is shown in Fig. 6, where we omit arc labels denoting graph variables (but still include binding variables like $\mathsf{b_m}$) and we annotate places by numbers 1–4 for the sake of clarity. The environment behavior affecting the WSN topology graph comprises the following actions (lower part of the GPN): ρ_{ls} turns long edges into short, ρ_{sl} short edges into long, ρ_{--} deletes an edge while ρ_{++} creates a fresh (unclassified) edge. Moreover, the rules ρ_{sl} and ρ_{--} propagate their matches as subgraph bindings to the TC process (upper part of the GPN) as those changes influence TC decisions. This specification represents an idealistic (i.e., *synchronous*) model of TC, as all actions (TC as well as environmental) are assumed to be atomic (i.e., non-interfered and instantaneously globally visible). Thus, each transition/rule application consumes the

global token (current topology) at place 1 and immediately produces a token (evolved/adapted topology) to the same place.

Subgraph Matching/Binding (C2). Regarding the TC process, transition ρ_m first matches a pair of neighboring short edges in the current graph and passes this match as subgraph binding to place 2 for the next step (using the binding variable $\mathsf{b_m}$). Place 3 plays a similar role for passing the long edge to be inactivated from ρ_{mu} to ρ_i. For those rules, the parallel presence of their positive and negative forms amounts to a deterministic conditional *if-else* construct based on rule applicability. We omitted rule ρ_a in Fig. 2d from the model due to space restrictions. Transition $\not{\rho_{mu}}$ handles those case where the subgraph binding previously matched by ρ_m becomes outdated in the current graph due to concurrent changes by the environment (i.e., either one of the matched edges of the edge pair becomes long by action ρ_{sl} or it disappears by action ρ_{--}). However, the model might *deadlock* if ρ_{mu} is not applicable but place 4 has no token. This can be solved by introducing two new transitions as in Fig. 7a (where place numbers correspond to Fig. 6). With that extension, subgraph binding guarantees *liveness* as the new transitions capture the case above.

Concurrency (C3). The idealized (synchronous) GPN specification considered so far above does not yet sufficiently capture the concurrent (or even distributed) semantics aspects of TC for WSN, which shall be reflected (still in a simplified manner) in the GPN in Fig. 7b. Here, the dashed boxes *Topology Control* and *Environment* refer to the GPN components as depicted in upper and lower part of Fig. 6, respectively, such that place 1 retains its role (i.e., all its arcs are combined into $G_{TC}, H_{TC}, G_{Env}, H_{Env}$). As a consequence, both TC and the environment fork the current graph (place 1 and place 5, respectively) while performing their (concurrent and thus potentially conflicting) changes to the topology. Afterwards, the *merge transition* (cf. Fig. 5b) labeled ${}_{\{G_1,G_2\}}\emptyset_G$ (where $\emptyset$ is the empty rule leaving the input graph unmodified) integrates both local graphs into the new global topology graph. In this GPN model, the (non-deterministic) merge transition potentially yields many different subsequent GPN steps as both subprocesses do not synchronize their matches as done in the first GPN model. In a future work, we plan to study different possible trade-offs between both extrema presented here, by considering different possible degrees of granularity of synchronization of subgraph-matches and -merges using GPN. In this regard, the detection of *concurrent transitions* (cf. Definition 6) can be employed to refine the (inherently imprecise) notion of *critical pair analysis* techniques [5] in the context of controlled graph-rewriting semantics.

Automated Analysis (C4). Correctness properties like **P1a-b** and **P2** in Sect. 2 can be specified on the LTS process semantics of GPN using appropriate *predicates* according to Definition 12. For instance, a predicate for property **P1a** holds for topology graph G reached in a marking iff for each *inactive* edge xy, there is a node z with *short* edges xz and yz. Similarly, **P1b** (i.e., connectedness) holds iff for each pair of nodes x and y, there is a path between x and y not containing any inactive edge. Predicates **P1a** and **P1b** constitutes



(a) New Transitions for **P2** (b) Concurrent WSN Model

Fig. 7. Extensions of the WSN and TC model

correctness properties in terms of *invariants* to be checked for any reachable completed marking, whereas **P2** requires further capabilities to express *liveness* properties on the LTS semantics of the GPN model (e.g., using temporal logics). For automated reasoning, we may choose as initial marking any possible initial topology graph G_0 on place 1 to explore the set of reachable markings. In this regard, *1-safety* of the underlying PN of a GPN is of particular interest as most essential PN analysis problems become decidable or computationally cheaper [6]. Here, the GPN model in Fig. 6 is 1-safe due to the centralized structure with place 1, whereas the GPN model in Fig. 7b is unbounded as the environment may continuously produce an arbitrary number of tokens to place 5. Thus, applying existing PN analysis tools to GPN at least allows for exploring *uncoloured* reachable markings (e.g., for over-approximating pairs of potentially concurrent transitions). Concerning reachability analysis of *coloured* markings, CPN Tools constitutes a state-of-the-art simulation and analysis tool for CPN which we currently integrate in our graph-rewriting framework eMoflon in an ongoing work.

To summarize, while certain basic aspects of the case study are directly expressible in other recent modeling approaches (e.g., control flow in Henshin [19] and eMoflon [14], predicate analysis in Groove [8]), neither of those approaches addresses all the challenges **C1-4** (cf. Sect. 2). Crucially, explicit concurrency is considered by none of the existing approaches, whereas Petri nets and therefore also GPN are equipped with a natural notion of true concurrency, thus being particularly well-suited for our purposes.

6 Conclusion and Future Work

We proposed graph-rewriting Petri nets (GPN) to build a comprehensive theoretical foundation for modeling and automated analysis of controlled graph-rewriting processes. In particular, GPN provides a control-flow specification language for graph rewriting processes with a natural notion of concurrency. Additionally, GPN provide an explicit means to handle *sub-graph bindings*, being prevalent in practical applications of controlled graph rewriting (cf. **C1-3**

in Sect. 2). Furthermore, GPN constitutes an appropriate basis for automated semantic analysis of controlled graph-rewriting systems (cf. **C4** in Sect. 2).

Our plans for future work are twofold. First, we want to explore further theoretical properties of GPN based on existing (C)PN results (e.g., revised notion of parallel independence in the context of concurrent graph-rewriting processes as well as equivalence notions beyond input/output equivalence). Second, we plan to use GPN as back-end for existing controlled graph-rewriting modeling tools and languages (e.g., SDM in eMoflon [14]) as well as for developing novel modeling approaches directly based on GPN. This would allow us to apply existing analysis tools like CPN Tools [11] for automatically analyzing crucial properties of graph-based algorithms like our WSN case study.

References

1. Baldan, P., Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Löwe, M.: Concurrent semantics of algebraic graph transformation. In: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 3, pp. 107–187. World Scientific (1999). https://doi.org/10.1142/9789812814951_0003
2. Bunke, H.: Programmed graph grammars. In: Claus, V., Ehrig, H., Rozenberg, G. (eds.) Graph Grammars 1978. LNCS, vol. 73, pp. 155–166. Springer, Heidelberg (1979). https://doi.org/10.1007/BFb0025718
3. Corradini, A., Ehrig, H., Löwe, M., Montanari, U., Rossi, F.: Abstract graph derivations in the double pushout approach. In: Schneider, H.J., Ehrig, H. (eds.) Graph Transformations in Computer Science. LNCS, vol. 776, pp. 86–103. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-57787-4_6
4. Corradini, A., Montanari, U., Rossi, F.: Graph processes. Fundamenta Informaticae **26**(3–4), 241–265 (1996). https://doi.org/10.3233/FI-1996-263402
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
6. Esparza, J.: Decidability and complexity of Petri net problems — an introduction. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 374–428. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_20
7. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: a new graph rewrite language based on the unified modeling language and Java. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) TAGT 1998. LNCS, vol. 1764, pp. 296–309. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46464-8_21
8. Ghamarian, A.H., de Mol, M.J., Rensink, A., Zambon, E., Zimakova, M.V.: Modelling and analysis using GROOVE. Int. J. Softw. Tools Technol. Transf. **14**, 15–40 (2012). https://doi.org/10.1007/s10009-011-0186-x
9. Guerra, E., de Lara, J.: Colouring: execution, debug and analysis of QVT-relations transformations through coloured petri nets. Softw. Syst. Model. **13**(4), 1447–1472 (2014). https://doi.org/10.1007/s10270-012-0292-6
10. Hoffmann, K., Mossakowski, T.: Algebraic higher-order nets: graphs and petri nets as tokens. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2002. LNCS, vol. 2755, pp. 253–267. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40020-2_14

11. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Science & Business Media, Heidelberg (2009). https://doi.org/10.1007/b95112
12. Kluge, R., Stein, M., Varró, G., Schürr, A., Hollick, M., Mühlhäuser, M.: A systematic approach to constructing families of incremental topology control algorithms using graph transformation. Softw. Syst. Model. 1–41 (2017). https://doi.org/10.1007/s10270-017-0587-8
13. Kreowski, H.-J., Kuske, S., Rozenberg, G.: Graph transformation units – an overview. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Concurrency, Graphs and Models. LNCS, vol. 5065, pp. 57–75. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68679-8_5
14. Leblebici, E., Anjorin, A., Schürr, A.: Developing eMoflon with eMoflon. In: Di Ruscio, D., Varró, D. (eds.) ICMT 2014. LNCS, vol. 8568, pp. 138–145. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08789-4_10
15. Li, M., Li, Z., Vasilakos, A.V.: A survey on topology control in wireless sensor networks: taxonomy, comparative study, and open issues. Proc. IEEE **101**(12), 2538–2557 (2013). https://doi.org/10.1109/JPROC.2013.2257631
16. Plump, D., Steinert, S.: The semantics of graph programs. In: RULE 2009 (2009). https://doi.org/10.4204/EPTCS.21.3
17. Schürr, A.: Logic-based programmed structure rewriting systems. Fundam. Inf. **26**(3,4), 363–385 (1996). https://doi.org/10.3233/FI-1996-263407
18. Schürr, A., Winter, A.J., Zündorf, A.: The PROGRES-approach: language and environment. In: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2, pp. 487–550. World Scientific (1999). https://doi.org/10.1142/9789812815149_0013
19. Strüber, D., Born, K., Gill, K.D., Groner, R., Kehrer, T., Ohrndorf, M., Tichy, M.: Henshin: a usability-focused framework for EMF model transformation development. In: de Lara, J., Plump, D. (eds.) ICGT 2017. LNCS, vol. 10373, pp. 196–208. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61470-0_12
20. Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W.: Right or wrong?–verification of model transformations using colored petri nets. In: DSM 2009 (2009)