Integrated Design and Process Technology, **IDPT-2002**
Printed in the United States of America, June, 2002

Formal Relationship between Petri Nets and Graph Grammars as Basis for Animation Views in GenGED

Roswitha Bardohl, Claudia Ermel, Julia Padberg

Institute for Software Engineering and Theoretical Computer Science
Technical University of Berlin, Germany
{**rosi,lieske,padberg**}**@cs.tu-berlin.de**

ABSTRACT: Specification techniques like Petri nets allow for the formal description and analysis of systems. Although tool support exists for many different Petri net classes and tasks, a domain-specific animation of net behavior, however, is not yet supported by many Petri net tools.

In this contribution, we present a formal approach for the generic specification of several Petri net classes including animation views. The approach follows the notions of GenGED, a tool for the visual specification of visual languages based on algebraic graph transformation. Moreover, we give a proof of the semantical compatibility of Algebraic High-Level Petri nets and their representation as graph grammars in GenGED. The proof is based on the formal semantics of Petri net behavior and the construction of graph derivations as pushouts in the category of graphs and graph morphisms.

Based on the behavior equivalence, we can define animation views for specific Petri nets in GenGED: the animation view of a system modeled as a Petri net consists of a domain-specific layout and animation rules according to the graph grammar representation of the Petri net.
Keywords: Petri nets, visual languages, graph grammars, stepwise animation

I. INTRODUCTION

Visual modeling techniques are of growing interest for an application-oriented presentation of models given by a formal specification. Especially, for non-experts in formal modeling, a layout of the model and its behavior simulation in the application domain would be desirable. Based on the GenGED approach for the generic specification of visual languages (VLs), we define a relationship between the formal model of Petri nets and a so-called animation view by giving a layout for a model as icons in the application domain.

The GenGED environment supports the visual specification of visual environments that are generated from the specification. Concerning editing, the specification of either syntax-directed [1] or free editing [4] is supported. VLs describing processes, like Automata, Statecharts or Petri nets, usually are suitable for simulating the behavior. The behavior is defined by a simulation specification which is executable in a generated VL environment [2]. All environment specifications in GenGED are based on the well-defined concepts of algebraic graph transformation and graphical constraint solving. These concepts have been used as a unifying representation for various Petri net classes in GenGED [3], [9].

Many different encodings of Petri nets into graph grammars have been proposed in the literature [16]. We focus on a very natural encoding of nets into grammars, namely we regard a net as a graph grammar acting on graphs representing the markings of a net. A Petri net transition is represented as a graph rule consuming the tokens in the pre-domain and generating the tokens in the post-domain of the transition. Places are represented as graph vertices which are preserved by the rule. This is indicated by dashed arrows in Fig. 1 showing an example for the translation of a transition from a Place/Transition net (short P/T net) to a graph rule in the Single-Pushout (SPO) approach to attributed graph transformation [11]. The left-hand side L of the graph rule holds the pre-domain tokens to be consumed and the right-hand side R holds the post-domain tokens to be generated. For a specification of P/T nets as graph grammars in the Double-Pushout (DPO) approach see [5].

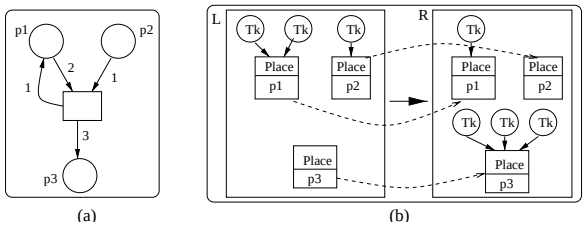


Fig. 1. (a) A transiton of a P/T net,(b) the corresponding graph rule

Obviously, such a representation satisfies the properties of the token game in Petri nets: a rule can be applied to a given marking iff the corresponding transition is enabled, and the firing of a transition produces the same marking as the SPO construction.

In this paper, we adapt the approach of Korff and Ribeiro in [10] and show that the semantics of an Algebraic High-

Level (AHL) net (based on sets of traces) is compatible (i.e. isomorphic) to the operational semantics of the graph grammar encoding it within the GenGED framework. The main difference in the encoding of AHL nets as graph grammars between [10] and our approach lies in the representation of markings. In [10], [5], markings are discrete graphs (token nodes labeled by place names). In GenGED, tokens and places are modeled as attributed nodes of different types, connected by edges. The fact of semantic compatibility between Petri nets and their representation as graph grammars is the formal basis for the definition of an animation view of a Petri net in an application-specific layout. The basic idea is to generate animation rules from the graph grammar representation of the net behavior by adapting the visual elements in the rules to an animation-specific layout (*animation view*).

In our running example of a simple elevator modeled as an AHL net, the animation view shows directly the movements of the elevator cabin between the different floors. In order to keep the example simple, the AHL net does not model the control mechanism to call the elevator but only the movements *up* and *down* (see Fig. 2). In general, the system states show in which floor the elevator is, and whether it is moving or not. The firing conditions of the transitions *up* and *down* control that the elevator does not move out of the range of the possible floors: in Fig. 2 a house with five floors is modeled because the SPEC-algebra A binds the constant $MaxFloor$ to the value 5 and $MinFloor$ to the value 0. The model requires the algebraic datatype specification Nat of natural numbers and uses the datatype operations "$+$" and "$-$" for computing the current floor number. The two constants $MaxFloor$ and $MinFloor$ fix the height of the house the elevator is working in. It is obvious that the constants in our data type specification together with the firing conditions of the transitions restrict the domain of data values for tokens to the range $\{MinFloor, .., MaxFloor\}$. The initial marking specifies that in the beginning the elevator must be in the ground floor, in the state not moving (token "0" on place *not moving*).

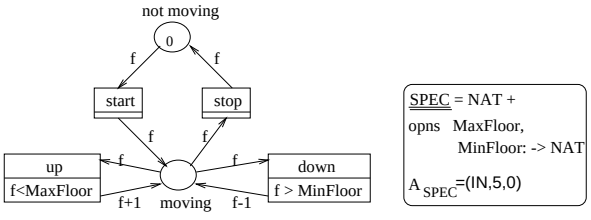


Fig. 2. A basic elevator as AHL net

A conceivable animation view of the AHL net is illustrated in Fig. 3. We show two snapshots of the animation view according to two possible markings of the net in Fig. 2. The elevator is illustrated as part of a building showing the actual number of floors. The elevator cabin which is visualized as box with doors is intended to move between the floors of the building. When the elevator is in the state *moving* as shown in the left part of Fig. 3, the cabin is positioned in the analogous floor and the doors are closed. This snapshot corresponds to a token "3" on place *moving*. When the elevator has stopped (state *not moving*), the cabin doors are open. This is shown in the right snapshot of Fig. 3, and corresponds to a token "2" on place *not moving*.

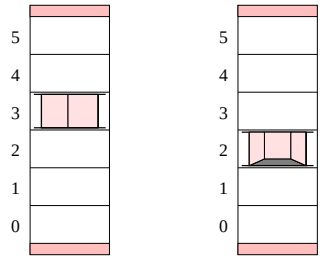


Fig. 3. Animation view snapshots

The actions that can be performed in the animation view of our model correspond to the transitions in the AHL net. Thus, for example, when the elevator has stopped (state *not moving*, it is not possible to perform the actions *up* or *down* because only the transition *start* is enabled in the corresponding net.

The paper is structured as follows: In the next section we briefly review the GenGED concepts which are illustrated by the specification of the AHL net language. In Section III we show the semantic equivalence of AHL nets and their graph grammar representation as formal foundation for the animation view specification. On this formal basis, in Section IV, a sample animation view specification for our elevator model is given.

II. Defining the Petri Net Language within GenGED

The GenGED approach and environment was developed in order to support the visual specification of visual languages (VLs) in an easy way, and to generate visual environments. In addition to this overall requirement it was desired to find a suitable formalism allowing for easy extensions of the application: the GenGED approach is based on algebraic graph transformation and graphical constraint solving techniques. A concrete AHL net like that in Fig. 2, for example, is represented by an attributed graph (an algebra) to a given type graph (a signature). In both the type graph and the instance graphs (also called visual sentences), we distinguish the abstract syntax (the symbols and links) and the concrete syntax (the layout). The abstract syntax is represented by an attributed graph. The vertices represent visual symbols such as a place or a transition of a Petri net, and the edges describe the logically meaningful spatial relations between symbols, e.g. that a token belongs to a place. Vertex attributes define data values, like names for places

or transitions. The layout of symbols and links is given by further vertex attributes denoting the shape, position, color, etc., and graphical constraints which have to be satisfied by each visual sentence. Although we concentrate on the abstract syntax in the following sections, we briefly sketch the concrete syntax also.

Similar to formal textual languages, a visual language is specified by an alphabet for symbol and link types (the type graph), and a grammar consisting of a start sentence and a set of syntax rules. A visual language VL is then the set of all visual sentences that can be derived by applying syntax rules [1]. These original concepts of GenGED for syntax-directed editing have been extended for free editing (parsing) [4] and simulation [2]. This means, visual grammars are interpreted differently, however, this topic is beyond the scope of this paper. Instead we concentrate on a brief introduction of the concepts, namely that of an alphabet and a grammar, illustrating the power of VL specifications using GenGED.

An alphabet is given by a set of symbol and link types of a specific VL as well as a graphical constraint satisfaction problem. The alphabet for the AHL net language is shown in Fig. 4.

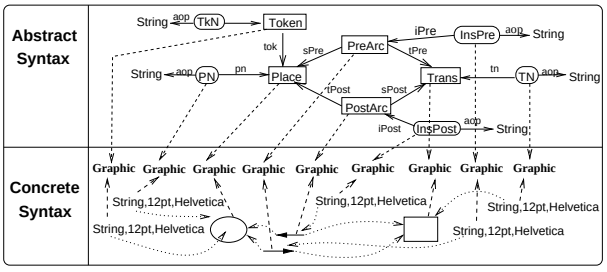


Fig. 4. Visual alphabet for AHL nets

Due to the abstract syntax level, there are symbols as Place, Trans (Transition), etc. in Fig. 4 which are partly attributed by datatypes like PN (Place Name) or TV (Token Value) of type String. These abstract syntax items are linked via directed edges. Note that we distinguish arcs running from places to transitions (type PreArc) and from transitions to places (type PostArc). The concrete syntax level is described by further vertex attributes of type Graphic and by graphical constraints specifying the intended layout. We use dotted arrows at the concrete syntax level in order to indicate the graphical constraints.

A VL grammar is based on the specific alphabet and the powerful means of graph transformation: grammar rules are not restricted to be context-free but context-sensitive. The left-hand side (LHS) of a rule does not comprise a single non-terminal symbol but a complete sentence, as well as the right-hand side (RHS). A rule expresses either the insertion or deletion (or both) of symbols. The rule insPre(ipt) depicted in Fig. 5 defines the operations for inserting an arc between a place and a transition. The newly created arc is inserted together with the arc inscription defined by the rule parameter ipt.

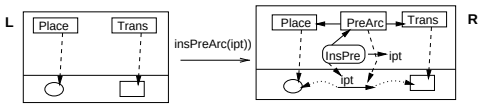


Fig. 5. Syntax rule from the AHL net grammar

A rule can be applied to a given sentence by mapping the abstract syntax items of the rule's LHS to items of the sentence. Then, the sentence is transformed according to the rule by inserting or deleting items to/from the sentence.

Please note that we omit the complete specification of the AHL net language for the sake of space; the reader is referred to [3] for more details. Instead we discuss the semantical compatibility of Petri nets and their representation in GenGED based on the alphabet shown in Fig. 4.

III. Semantical Compatibility of Petri Nets and their Representation in GenGED

Before we prove the semantical compatibility of the Petri net semantics and the graph grammar semantics of an AHL net representation within GenGED, we briefly review the formal definitions of AHL nets and attributed graph grammars.

A. AHL Nets

We review the basic definition of AHL nets and their behavior as given in [13]. Further information about similar high-level Petri net approaches can be found for instance in [17], [14], [8]. Note that the pre and post-domain of a transition is given by a multiset of pairs of terms and places, i. e. as elements of a free commutative monoid.

Definition III-A.1 (AHL Net) *An* Algebraic High-Level net *(AHL net)* $N = (SPEC, P, T, pre, post, cond, A, m)$ *consists of an algebraic specification* $SPEC = (S, OP, E; X)$ *with equations* E *and additional variables* X *over the signature* (S, OP) *[7], sets* P *and* T *of places and transitions respectively, pre- and post-domain functions* $pre, post : T \rightarrow (T_{OP}(X) \times P)^{\oplus}$ *assigning to each transition* $t \in T$ *the pre- and post-domains* $pre(t)$ *and* $post(t)$ *(see below) respectively, a firing condition function* $cond : T \rightarrow \mathcal{P}_{fin}(EQNS(S, OP, X))$ *assigning to each transition* $t \in T$ *a finite set* $cond(t)$ *of equations over the signature (S,OP) with variables* X *and an* (S, OP, E)*-algebra* A *[7].*

A marking m *is a multiset of pairs of data elements of* A *distributed on places:* $m \in (A \times P)^{\oplus}$. △

$T_{OP}(X)$ is the set of terms with variables X over the signature (S, OP) [7], $T_{OP}(X) \times P$ being the cartesian product and $S^{\oplus}$ the free commutative monoid over a set S. This means that $pre(t)$ (and similar $post(t)$) is of

the form $pre(t) = \sum_{i=1}^{n}(term_i, p_i) \quad (n \geq 0)$ with $p_i \in P, term_i \in T_{OP}(x)$. Thus, the multiset $\{p_1, ...p_n\}$ is the pre-domain of t with arc-inscription $term_i$ for the arc from p_i to t if the elements of $\{p_1, ..., p_n\}$ are distinct (unary case) and arc-inscription $term_{i1} \oplus ... \oplus term_{ik}$ for $p_{i1} = ... = p_{ik}$ (multi case).

Definition III-A.2 (Behavior of AHL Nets) *Enabling and firing of transitions is defined as follows: let $Var(t)$ be the set of local variables occurring in $pre(t), post(t)$ and $eqns(t)$. An assignment $asg_A : Var(t) \to A$ is called consistent for $t \in T$ if the equations $cond(t)$ are satisfied in A under asg_A. A transition $t \in T$ is enabled under a consistent assignment $asg_A : Var(t) \to A$ and a marking $m \in (A \times P)^{\oplus}$, if $pre_A(t, asg_A) \leq m$.*

The marking $pre_A(t, asg_A)$ – analogously $post_A(t, asg_A)$ – is defined for $pre(t) = \sum_{i=1}^{n}(term_i, p_i)$ by $pre_A(t, asg_A) = \sum_{i=1}^{n}(\overline{asg}_A(term_i), p_i)$, where $\overline{asg}_A = xeval_A(asg_A) : T_{OP}(Var(t)) \to A$ is the extended evaluation of terms under assignment asg_A [7].

The successor marking m' is defined in the case of t being enabled by $m' = m \ominus pre_A(t, asg_A) \oplus post_A(t, asg_A)$ and gives raise to a firing step *$m[t, asg_A\rangle m'$.* △

Definition III-A.3 (Semantics of AHL Nets) *The set of all steps $m[t, asg_A\rangle m'$ of an AHL net N is denoted by $AHLSteps_N$. A sequence $\sigma \in AHLSteps_N^{\infty}$ is called* firing sequence *iff $\sigma = \lambda$ or $\sigma \neq \lambda$ and $m_1 = m, \sigma_i = m_i[(t_i, asg_{A_i})\rangle m_{i+1}$ for all $i \in dom(\sigma)$.*

The semantics $Sem^{AHL}(N)$ of an AHL net N is defined by the set of all firing sequences of N. △

B. Attributed Graph Transformation

In the theory of algebraic graph transformation, a graph is given by disjoint sets indicating vertices and edges from a source vertex to a target vertex. Vertices may be additionally enhanced by attributes that are used to store data together with the vertices[1]. Such attributes are elements of an attribute algebra.

Definition III-B.1 (Attributed Graph) *An* attributed graph *$G = (G_V, G_E, G_A, s^G, t^G, attr^G)$ consists of a set of* vertices *G_V, a set of* edges *G_E, an algebra $G_A \in Alg(Sig)$, where $Alg(Sig)$ is the category of all Sig-algebras and Sig-homomorphisms, source and target functions $s^G, t^G : G_E \to G_V$, and an attribution function $attr^G : G_V \to G_A$, connecting vertices with attributes (data elements of the algebra).* △

Definition III-B.2 (Attributed Graph Morphism) *A* (partial) attributed graph morphism *$f : G \to H$ between the attributed graphs G and H is a tuple $f = (f_V, f_E, f_A)$ consisting of partial functions $f_V : G_V \to H_V$ and $f_E : G_E \to H_E$ such that $s^H(f_E(e)) = f_V(s^G(e))$ and $t^H(f_E(e)) = f_V(t^G(e))$ for all $e \in dom(f_E)$ and all $v \in dom(f_V)$, and a homomorphism $f_A \in Alg(Sig)$ such that $f_A(attr^G(v)) = attr^H(f_V(v))$ for all $v \in dom(f_V)$.* △

[1] In general, attributes for edges are possible, too. In the GenGED approach, however, we only need vertex attributes.

Attributed graphs and graph morphisms form the cocomplete category AGG in the Single-Pushout approach to algebraic graph transformation ([11], [12], [6]).

In general, graph transformation defines a rule-based manipulation of graphs. (For an overview of the main approaches see [15].) Graph rules can be used to capture the dynamical aspects of systems. The resulting notion of graph grammars (consisting of a start graph and a set of graph rules) generalizes Chomsky grammars from strings to graphs. The start graph represents the initial state of the system, whereas the set of rules describes the possible state changes that can occur in the system. A rule comprises two graphs: a left-hand side L and a right-hand side R, and a partial graph morphism r from L to R.

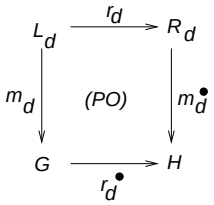
Definition III-B.3 (Attributed Graph Grammar) *An* attributed graph rule *$r : L \to R$ is an attributed graph morphism such that r_V and r_E are injective and r_A is the identity on a quotient term algebra $T_{OP}(X)$ with variables in X.*

An attributed graph grammar *$AGG = (S, P)$ consists of an attributed graph S (the start graph) and a set P of attributed rules (or productions) $r \in P$.* △

The application of a rule $r \in P$ to a graph G (also called *derivation*) requires a total morphism from L to G, called *match*. We here restrict the match to injective morphisms only. Thus we avoid the problem that two distinct nodes of the LHS are identified by the match, and the rule is applied to a graph modelling less tokens on the pre-domain places than the number expected for enabling the corresponding transition. Note that this problem does not occur in the DPO approach [5] because of the gluing condition which requires that the nodes of the LHS which shall be deleted are not merged by the match.

Formally, a derivation step from G to H with rule r and match m is defined by a pushout construction (see [11]) and is denoted by $G \overset{r,m}{\Longrightarrow} H$.

Definition III-B.4 (Derivation Step) *Let $AGG = (S, P)$ be an attributed graph grammar. A* derivation step *d of an attributed graph G where $G_A = S_A$ with rule $r_d \in P$ at match m_d is a pushout of m_d and r_d in AGG where $r^{\bullet}_{dA} = id_{S_A}$.*



△

A derivation step according to the pushout construction consists of three steps transforming the graph G to graph H. First, the vertices in L which do not have an image via r in R are deleted in G. Thereafter, the vertices in R without origin in L are created and appended to the graph. (Vertices in L which are mapped to R by r are preseved by the rule.) As a last step, all edges which are not defined any more are deleted from the graph.

Sometimes the application of a rule shall be constrained w.r.t. the attribution, as for example $x > 7$ (*conditional rule*). In this case we specify the attribution algebra of the rule equationally, i.e. the required equations are syntactically added to the rule implying that the terms annotating the rule shall be interpreted in the corresponding quotient term algebra.

Definition III-B.5 (AGG Semantics) *The set of all derivation steps in AGG is denoted by $GGSteps_{AGG}$. A sequence $\sigma \in GGSteps^{\infty}_{AGG}$ is called a* sequential derivation *of AGG, also denoted by $S[\sigma\rangle_{AGG}$ if either $\sigma = \lambda$ or $\sigma \neq \lambda$ and $G_{\sigma[1]} = S, H_{\sigma[i]} = G_{\sigma[i+1]}$ for all $i \in dom(\sigma)$.*

The semantics $Sem^{AGG}(AGG)$ is the set of all sequential derivations of AGG. △

C. Translating AHL Nets to Graph Grammars

The translation of AHL nets into graph grammars generalizes that of P/T nets into graph grammars as sketched in the introduction. The marking of a net is transformed to a graph consisting of bundles of vertices of type Token (the tokens). Each Token vertex is connected to a vertex of type Place representing its place and attributed by a data value from the net's algebra. Thus, each token of the net can be expressed in the graph as an edge from an attributed vertex of type Token to a vertex of type Place attributed by its name.

The initial marking of an AHL net N is translated to the start graph S of a graph grammar $AGG = (S, P)$ (Def. III-C.1), whereas the transitions together with their pre and post-domains are translated to rules $r \in P$ (Def. III-C.5).

Definition III-C.1 (Translation of Markings) *Let $m \in (A \times P)^{\oplus}$ be a marking of the AHL net $N = (P, T, pre, post, SPEC, cond, A, m)$.*
The translation of m is given by the attributed graph

$$Tr_m(m) = G = (G_V, G_E, G_A, s^G, t^G, attr^G)$$

with

- *$G_V = P \cup \widetilde{m}$. The multiset $m \in (A \times P)^{\oplus}$ here is given by the set $\widetilde{m} = \{(a, p, i) \in A \times P \times \mathbb{N} \mid 0 < i \leq m(a, p)\}$, where multiple occurrences of the same element in m are numbered by i in $\widetilde{m}$.*
- *$G_E = \{e_{tk} \mid tk = (a, p, i) \in \widetilde{m}\}$,*
- *$G_A = A$,*
- *$s^G : G_E \to G_V$ with $s^G(e_{(a,p,i)}) = (a, p, i)$*

and $t^G : G_E \to G_V$ with $t^G(e_{(a,p,i)}) = p$,

- *$attr^G : G_V \to G_A$ with*

$$attr^G(v) = \begin{cases} p & v = p \in P \\ a & v = (a, p, i) \in \widetilde{m} \end{cases}$$

This means that places are attributed by their names and tokens by their data values.

The backward translation of G to a marking of N is defined as follows:

$$M(G) = \sum_{e_{tk} \in G_E} (attr^G(s^G(e_{tk})), attr^G(t^G(e_{tk}))).$$

△

Fact III-C.2 (Compatibility of $Tr_m(m)$) *Let $m \in (A \times P)^{\oplus}$ be a marking of an AHL net N, and $Tr_m(m) = G$ be the translation of m to G and $M(G)$ the backward translation as defined in Def. III-C.1. Then, $M(Tr_m(m)) = m$.* △

Proof III-C.3 (Compatibility of $Tr_m(m)$)

$$\begin{aligned} & M\left(Tr_m(m)\right) = M(G) \\ & \quad \text{with } G \text{ defined as in Def. III-C.1} \\ = \;& \textstyle\sum_{e_{tk} \in G_E} (attr^G((a, p, i)), attr^G(p)) \\ = \;& \textstyle\sum_{e_{tk} \in G_E} (a, p) = m \end{aligned}$$

△

Example III-C.4 (Marking of Elevator Net) *Fig. 6 shows the AHL net modeling our basic elevator marked by the initial marking and the translation to an attributed graph (the start graph of the corresponding AGG).*

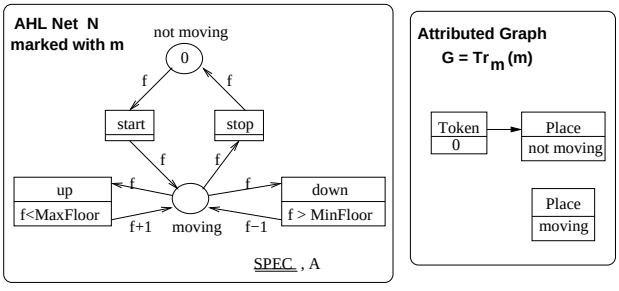


Fig. 6. Translation of the elevator net's initial marking to an attributed graph

The only edge in the translated graph formally is given by $e_{(0,not\_moving,1)}$. △

We now define the translation of the transitions in N to rules $r \in P$ encorporating the firing behavior: the left-hand side of each rule contains the pre-domain, the right-hand side the post-domain.

Definition III-C.5 (AHL Net Translation) *Let $N = (SPEC, P, T, pre, post, cond, A, m)$ be an AHL net. Then the translated attributed graph grammar is defined as $Tr^{AGG}(N) = (S_{Tr}, P_{Tr})$ where the start graph $S_{Tr} = Tr_m(m)$ is the translated initial marking of the net according to Def. III-C.1, and the set of behavior rules $P_{Tr} = \{r_t : L_t \to R_t \mid t \in T\}$ corresponds to the firing behavior of the transitions $t \in T$ with $Env_t = \{p \mid p \in pre(t) \cup post(t)\}$ being the environment of transition t. In each behavior rule $r_t : L_t \to R_t$, the left-hand side L_t corresponds to the pre-domain $pre(t)$, and the right-hand side R_t to the post-domain $post(t)$. Thus, in the following definition of the graph B_t, the set $TermSet$ is defined by $TermSet = PreSet$ for $B_t = L_t$ and by $TermSet = PostSet$ for $B_t = R_t$, where $PreSet = \{(term, p, i) \mid pre(t)(term, p) \geq i > 0\}$ corresponds to $pre(t)$ and $PostSet = \{(term, p, i) \mid post(t)(term, p) \geq i > 0\}$ corresponds to $post(t)$.*

Each behavior rule side $B_t = L_t, R_t$ is an attributed graph defined as follows:

$$B_t = (B_V, B_E, A', s^B, t^B, attr^B)$$

with

- $B_V = Env_t \cup TermSet$.
- $B_E = \{e_{ts} \mid ts = (term, p, i) \in TermSet\}$,
- $A' = T_{OP}(Var(t))|_{cond(t)}$,
- $s^B, t^B : B_E \to B_V$ *with* $s^B(e_{(term,p,i)}) = (term, p, i)$ *and* $t^B(e_{(term,p,i)}) = p$
- $attr^B : B_V \to T_{OP}(Var(t))|_{cond(t)}$ *with*

$$attr^B(v) = \begin{cases} p & v = p \in P \\ term & v \in TermSet \end{cases}$$

The rule morphism $r_t : L_t \to R_t = (r_{t_V}, r_{t_E}, r_{t_A}) = (id_P, \emptyset, id_{A'})$ is the identity on the place vertices (the vertices $v \in PreSet$ are not in the domain of r_t) and the identity on the terms in the quotient term-algebra. △

By the construction of $Tr^{AGG}(N)$ it is obvious that S_{Tr} is a well-defined attributed graph and that all rules in P_{Tr} are attributed rules. Thus, $Tr^{AGG}(N)$ is an attributed graph grammar.

Example III-C.6 (AGG for Elevator Net) *ex:TransNet Let N be our Elevator Net shown in Fig. 2. Let AGG be the translated graph grammar $Tr^{AGG}(N) = (G, P)$ according to Def. III-C.5. The start graph G is shown in Fig. 6, whereas Fig. 7 shows the rules in P corresponding to the transitions $t \in T$.*

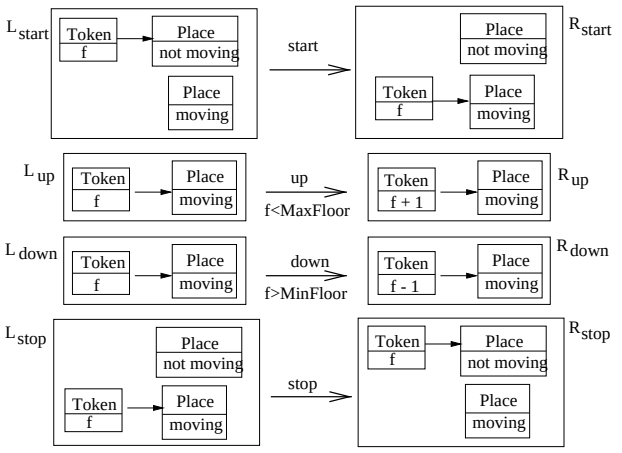


Fig. 7. Translation of the elevator net's transitions to rules

△

The following theorem states the compatibility between the semantics of an AHL net and the semantics of its translation into an AGG.

Theorem III-C.7 (Semantical Compatibility) *The semantics of an AHL net N and the semantics of the translation $Tr^{AGG}(N)$ are compatible, denoted by $Sem_{AGG}(Tr^{AGG}(N)) \cong Sem_{AHL}(N)$.*

The semantics of a net is given in terms of sets of firing sequences (firing steps), and the graph grammar semantics as sets of derivation sequences. △

Proof III-C.8 (Semantical Compatibility) *We We have to show that*

1. *the initial marking m of N is translated to the start graph G of the graph grammar $GG = (G, P) = Tr^{AGG}(N)$,*

2. *for each firing step $s = m[t, asg\rangle m' \in AHLSteps_N$ there is a derivation step $d \in GGSteps_{GG}$ with $M(G) = m$ s.t. the resulting marking m' corresponds to graph H of the pushout d, i.e. $M(H) = m'$,*

3. *each firing sequence $\sigma \in Sem^{AHL}(N)$ corresponds to a derivation sequence $\sigma' \in Sem^{AGG}(GG)$. This means, for all pairs $\sigma_i = (m_i[t_i, asg_i\rangle m_i') \in AHLSteps_N, i = 1..2$ s.t. $m_1' = m_2$, we have $H_1 = G_2$ in the corresponding pushout diagrams.*

ad 1. *Holds by the definition of $Tr^{AGG}(N)$.*

ad 2. *Let $s = m[t, asg\rangle m' \in AHLSteps_N$ be a firing step in N. Then d is constructed as follows:*

- ***Rule:*** *$r_t : L_t \to R_t$ as defined in Def. III-C.5*
- ***Match:*** *$m_{asg} : L_t \to G = (m_V, m_E, m_A)$ with the vertices being matched by $m_V((term, p, i)) = (\overline{asg}_A(term), p, i)$, $m_V(p) = p$. Edges in L_t are mapped by $m_E(e_{pr}) = e_{tk}$ with $pr = (term, p, i)$ and $tk = (\overline{asg}_A(term), p, i)$. $m_A = \overline{asg}_A$ maps each term in $T_{OP}(X)$ to its extended assignment in A.*

The match m_{asg} satisfies the graph morphism properties: $s^G(m_E(e_{pr})) = s^G(e_{tk}) = (a,p,i) = (\overline{asg}_A(term),p,i) = m_V((term,p,i)) = m_V(s^L(e_{pr}))$ *(analogously* $t^G(m_E(e_{pr})) = m_V(t^L(e_{pr}))$*).*

- ***Derivation step:*** *Using rule r_t and match m_{asg}, we obtain for $G = Tr_m(m)$ a derivation step in $GGSteps_{GG}$ as a pushout of r_t and m_{asg} in AGG (see Def. III-B.4). This means, we obtain a graph H and morphisms $r_t^\bullet : G \to H$, $m_t^\bullet : R_t \to H$ where $r^\bullet_{tA} = id_A$.*
By the construction of pushouts in AGG, the graph H is defined by $H = (H_V, H_E, H_A, s^H, t^H, attr^H)$ with $H_V = G_V - m_{asg}(PreSet) \uplus m^\bullet_{asg}(PostSet)$, $H_E = G_E - m_{asg}(L_E) \uplus m^\bullet_{asg}(R_E)$.
If $v \in m_V(G_V)$ then $attr^H(v) = attr^G(v)$, if $v \in m^\bullet_{asg}(PostSet)$ then $attr^H(v) = m^\bullet_{asg}(attr^{R_V}(v))$.
We show now that $M(H) = m'$:

$$
\begin{aligned}
M(H) &= \textstyle\sum_{e\in H_E}(attr^H(s^H(e)), attr^H(t^H(e))) \\
&= \textstyle\sum_{e\in G_E}(attr^G(s^G(e)), attr^G(t^G(e))) \\
&\quad \ominus \textstyle\sum_{e\in m_{asg}(L_E)}(attr^G(s^G(e)), attr^G(t^G(e))) \\
&\quad \oplus \textstyle\sum_{e\in m^\bullet_{asg}(R_E)}(attr^R(s^R(e)), attr^R(t^R(e))) \\
&= m \ominus \textstyle\sum_{e_{(a,p,i)}\in m_{asg}(L_E)}(a,p) \\
&\quad \oplus \textstyle\sum_{e_{(a,p,i)}\in m^\bullet_{asg}(R_E)}(a,p) \\
&= m \ominus \textstyle\sum_{(term,p,i)\in PreSet}(\overline{asg}_A(term),p,i) \\
&\quad \oplus \textstyle\sum_{(term,p,i)\in PostSet}(\overline{asg}_A(term),p,i) \\
&= m \ominus \textstyle\sum_{i=1}^{n}(\overline{asg}_A(term),p) \\
&\quad \text{with } (term,p,i) \in PreSet, n = |PreSet| \\
&\quad \oplus \textstyle\sum_{i=1}^{k}(\overline{asg}_A(term),p) \\
&\quad \text{with } (term,p,i) \in PostSet, k = |PostSet| \\
&= m \ominus pre_A(t, asg_A) \oplus post_A(t, asg_A) \\
&= m'
\end{aligned}
$$

ad 3. *Let $\sigma_i = (m_i[t_i, asg_i\rangle m_i') \in AHLSteps_N$, $i = 1..2, m_1' = m_2$. Let d_i be the derivation steps constructed for transitions t_i under marking m_i in N according to Def. III-C.5. We have to show that $H_1 \cong G_2$.*
Due to Theorem III-C.7 (item 2) we know that $M(H_1) = m_1' = m_2$. By the pushout construction of d_2, the marking m_2 is translated to $Tr_m(m_2) = G_2$, and by Fact III-C.2 we have $M(G_2) = M(Tr_m(m_2)) = m_2$. Thus, $M(H_1) = M(G_2)$, and by the construction of attributed graphs from markings according to Def. III-C.1, we conclude that $H_1 \cong G_2$. △

IV. Animation of Petri Nets

In order to bridge the gap between the underlying formal, descriptive specification as a Petri net and a natural dynamic visual representation of processes being simulated, we suggest the definition of animation views for Petri nets. Of course, the behavior shown in the animation view has to correspond to the behavior defined in the original Petri net. In Section IV-A we give guidelines for the specification of an animation view for a specific Petri net and present an application-specific animation grammar for our elevator net. To ensure a consistent mapping of the Petri net behavior, we propose in Section IV-B a graph grammar based view transformation from the Petri net to its animation view.

A. Animation View

In our approach, both the Petri net and its animation view consist of visual sentences based on the same abstract visual alphabet. They differ in the concrete syntax, i.e. we define different layouts for the same underlying process model.

We suggest the following guidelines for the definition of an animation view layout for all symbols and links from a Petri net alphabet (cf. Fig. 4): places in a Petri net give meanings to tokens by defining their properties. In a net-independent animation view, places are not needed any more because properties of tokens now are incorporated in the concrete layout/position of the tokens themselves. Therefore, we visualize places as symbols of the *fixed part* of the animation view, i.e. the part of the view which is not changed by animation.

The *animated part* consists of the symbols which are changed during animation and corresponds to the tokens of the Petri net. Transitions are replaced by rule names in the animation view. (Rule names are visible in the user interface of a generated environment in order to trigger a state transformation step corresponding to a firing step of the transition in the Petri net view.)

Arcs in the Petri net define the firing behavior in a static way. They are not shown in the animation view as the behavior now is defined by the animation rules and visualized by their application (the animation).

Example IV-A.1 (Animation View) *The elevator net as illustrated in Fig. 2, is a visual sentence of our AHL net language. The animation view of this sentence has been already motivated by Fig. 3 where an elevator cabin moving between the floors of a house is visualized. The abstract syntax of the alphabet of the animation view is equal to that of Fig. 4 but the concrete layout differs.*

The fixed part of the animation view consists of the symbol for the elevator shaft. The tokens – corresponding to the animated part of the animation view – model the different locations of the elevator cabin. A token on place not moving *corresponds to a cabin icon with open doors, whereas a token on place* moving *is visualized by an icon showing a cabin with closed doors.*

Fig. 8 shows the abstract syntax underlying both views (the Petri net and the animation view) for our elevator model and depicts some of the connections to the different layouts. △

Based on the abstract syntax, we define the generation of an animation view by grammar rules that transform all possible states of the Petri net into an appropriate state of

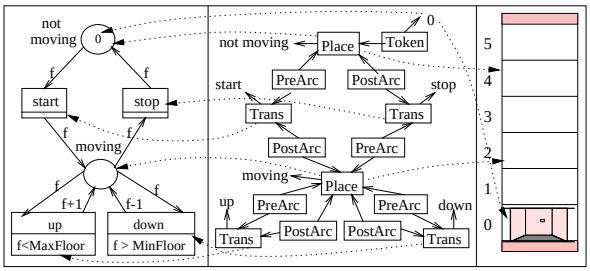


Fig. 8. Visual syntax of the elevator net and its animation view

the animation view (view transformation rules). The generation of the animation view in Example IV-A.1 is described in Section IV-B.

The behavior of the system in the animation view then is given by a set of animation rules on the VL sentences of the animation view. The abstract syntax of the animation rules equals the abstract syntax of the behavior rules for the Petri net (see Fig. 7 for the abstract behavior grammar of the elevator net). We call the grammar containing the animation rules, the *animation grammar*.

Example IV-A.2 (Animation Grammar) *Fig. 9 shows the animation grammar for our elevator model. Each animation rule corresponds to the behavior rule of the same name from the behavior grammar in Fig. 7 and thus to a transition in the elevator net.*

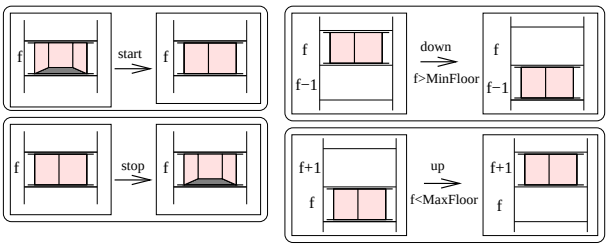


Fig. 9. Animation grammar for the elevator model

△

B. Generating Animation Views

The aim, of course, is to construct the animation grammar in a way that the animation is consistent to the behavior specification. Therefore, we now define the generation of an animation view by grammar rules that transform all possible states of the system model into an appropriate state of the animation view (view transformation rules). On the basis of these view transformation rules it is possible to enforce coherence between the behavior grammar of the original visual process model (the Petri net) and the animation grammar of the animation view. The view transformation rules allow to transform the VL sentences from the old layout to the new layout and the behavior rules into the animation rules.

In general, each of the view transformation rules transforms a subset of all possible Petri net markings to the animation view by combining elements of the abstract syntax with new concrete syntax elements (i.e. by giving them a different layout). After applying the view transformation rules, the VL sentence denoting the initially marked Petri net is transformed into a corresponding animation view.

Formally, view transformation rules operate on the union of both the visual syntax of the Petri net and the visual syntax of its animation view. Although the abstract syntax is the same, the concrete syntax (the layout) differs. In the following example, therefore, graphics from both concrete syntax definitions (net layout and animation view layout) are shown in the same rules.

Example IV-B.1 (View Transformation Rules) *Fig. 10 shows the view transformation rules needed for the generation of the animation view from the initially marked elevator net (both depicted in Fig. 8). We define one view transformation rule for the marking of each place. In the view transformation rules in Fig. 10, the abstract syntax remains, but the symbols are re-linked to the new animation view graphics as they are introduced in the right-hand sides of the rules.*

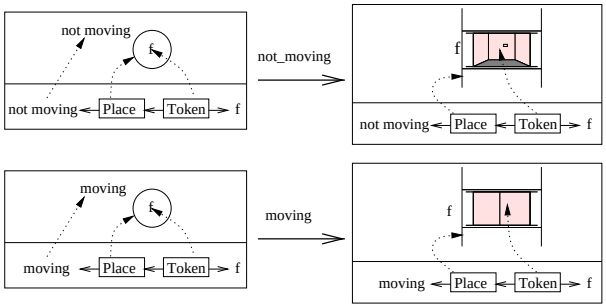


Fig. 10. View transformation rules for the elevator model

△

The behavior of a Petri net now can be transferred to the animation view by applying the view transformation rules to the LHSs and RHSs of the behavior rules.

Example IV-B.2 (Generating Animation Rules) *Fig. 11 illustrates the derivation of an animation rule by applying rules from the view transformation grammar in Fig. 10 to the behavior rule start from Fig. 7.*

△

A stepwise animation of the system can be performed by applying the animation rules in the animation view of the system model. An extension of the approach towards a more sophisticated animation would be the presentation of system behavior not as discrete steps but as a movie (”smooth” animation), i.e. showing a series of intermediate states for the firing of one transition. With this aim in mind, an animation framework as proposed in [19] could be combined with the GenGED environment.

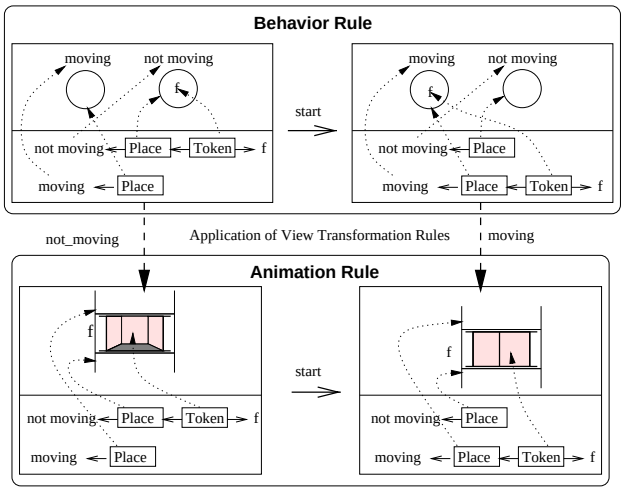


Fig. 11. Derivation of an animation rule with view transformation rules

V. Conclusion

In this paper, we have given a formal approach for the representation of Petri nets in GenGED in terms of attributed graph grammars. In this encoding, graphs represent the markings of a net, and for each transition, a graph rule models the firing behavior. We show that the semantics of an AHL net and the semantics of its representation in GenGED (the behavior specification) are compatible. On this formal basis it becomes possible to define animation views for Petri nets in an application-specific layout where the animation is formally derived from (and thus consistent to) the behavior specification.

In our running example we have specified and animated a simple elevator model. This is formally specified using AHL nets and animated in a freely chosen layout.

It remains to develop behavior and animation views for other visual modeling languages than Petri nets. Hence, a formal theory allowing to handle behavior and animation of various VL models in a generic way based on the GenGED approach is the overall aim of our work.

On the practical side, the GenGED tool environment has to be extended in order to be able to manage the combination of different VLs with the same abstract but different concrete syntax. Based on these extensions, the GenGED tool environment will be able to handle different views of a VL model, including support for the visualization of behavior in terms of the formal model itself (e.g. token game simulation) and animation in a domain-specific layout as discussed in this paper.

Acknowledgements This work has been partially supported by the DFG-Researcher Group Petri Net Technology and by the German-Brazilian project Graphit. Many thanks also to our colleague Milan Urbasek for carefully reading a previous version and to our anonymous referees for valuable hints.

References

[1] R. Bardohl. GenGED – *Visual Definition of Visual Languages based on Algebraic Graph Transformation*. Verlag Dr. Kovac, 2000. PhD thesis, Technical University of Berlin, Dept. of Computer Science, 1999.

[2] R. Bardohl, K. Ehrig, C. Ermel, A. Qemali, and I. Weinhold. GenGED – Specifying Visual Environments based on Visual Languages. In H.-J. Kreowski, editor, *Proc. of APPLIGRAPH Workshop on Applied Graph Transformation (AGT 2002)*, 2002. to appear.

[3] R. Bardohl, C. Ermel, and H. Ehrig. Generic Description of Syntax, Behavior and Animation of Visual Models. Technical Report 2001/19, Technische Universität Berlin, 2001. ISSN 1436-9915.

[4] R. Bardohl, T. Schultzke, and G. Taentzer. Visual Language Parsing in GenGED. *Electronic Notes of Theoretical Computer Science*, 50, June 12–13 2001.

[5] A. Corradini and U. Montanari. Specification of Concurrent Systems: From Petri Nets to Graph Grammars. In G. Hommel, editor, *Proc. Workshop on Quality of Communication-Based Systems, Berlin, Germany*. Kluwer Academic Publishers, 1995.

[6] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, and A. Corradini. Algebraic Approaches to Graph Transformation II: Single Pushout Approach and Comparison with Double Pushout Approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, chapter 4, pages 247–312. World Scientific, 1997.

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

[8] H. Ehrig, J. Padberg, and G. Rozenberg. Behaviour and Realization Construction for Petri Nets Based on Free Monoid and Power Set Graphs. Technical report, Technical University Berlin TR 94-15, 1994.

[9] C. Ermel, R. Bardohl, and H. Ehrig. Specification and Implementation of Animation Views for Petri Nets. In Weber et al. [18], pages 75–92.

[10] M. Korff and L. Ribeiro. Formal Relationship between Graph Grammars and Petri nets. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science, Williamsburg '94, LNCS 1073*, pages 288 – 303. Springer, 1995.

[11] M. Löwe. Algebraic Approach to Single-Pushout Graph Transformation. *TCS*, 109:181–224, 1993.

[12] M. Löwe, M. Korff, and A. Wagner. An Algebraic Framework for the Transformation of Attributed Graphs. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 14, pages 185–199. John Wiley & Sons Ltd, 1993.

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

[14] W. Reisig. Petri Nets and Algebraic Specifications. *Theoretical Computer Science*, 80:1–34, 1991.

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

[16] H. J. Schneider. Graph Grammars as a Tool to Define the Behaviour of Process Systems: From Petri Nets to Linda. In *Proc. Fifth International Workshop on Graph Grammars and their Application to Computer Science*, pages 7–12, Williamsburg, Va., USA, 1994.

[17] J. Vautherin. Parallel Specification with Coloured Petri Nets and Algebraic Data Types. In *Proc. of the 7th European Workshop on Application and Theory of Petri nets*, pages 5–23, Oxford, England, jul. 1986.

[18] H. Weber, H. Ehrig, and W. Reisig, editors. *2nd Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, Berlin, Germany, Sept. 2001. Researcher Group Petri Net Technology, Fraunhofer Gesellschaft ISST.

[19] C. Weidauer. Animations-Framework in Java. Systematische Animationsentwicklung mit Mehrschichtenarchitektur. *Informatik - Forschung und Entwicklung, Band 15, Heft 2*, pages 83 –91, June 2000.