Generation of Animation Views for Petri Nets in GenGED*

Claudia Ermel, Roswitha Bardohl, and Hartmut Ehrig

Institut für Theoretische Informatik und Softwaretechnik,
Technische Universität Berlin,
Email: {lieske,rosi,ehrig}@cs.tu-berlin.de,
WWW home page: http://tfs.cs.tu-berlin.de

Abstract. Formal specification techniques like Petri nets allow for the formal description and analysis of systems. Tool support exists for many different Petri net classes for editing, simulating and analyzing formal models. A domain-specific animation of net behavior going beyond the well-known token game, however, is not yet supported in most cases. In this paper, we present a formal approach for the generic specification of animation views for different Petri net classes based on GenGED and graph transformation.
The GenGED approach, developed at the TU Berlin, allows for the generic description of visual modeling languages including different Petri net classes. In our framework, the animation view of a system modeled as a Petri net consists of a domain-specific visual layout and an animation view according to the firing behavior of the Petri net class. The basic idea is to generate visual animation rules based on visual syntax rules defining the corresponding Petri net language. We propose a view transformation from the classical Petri net layout to the animation layout. The well-known producer/consumer system modeled as an elementary Petri net serves as running example. We provide an animation view for the application domain *kitchen* where producing and consuming is visualized by icons for *baking* and *eating* cakes, respectively.

1 Introduction

1-Intro

The use of visual modeling techniques today is indispensable in software system specification and development. Especially, specification techniques for communication-based systems must provide means for modeling distributed systems and concurrent behavior. Petri nets allow already the formal specification and analysis of concurrent or distributed systems and the visual description of net models by the graphical notation of nets. This kind of graphical visualization of nets, however, is not always sufficient.

* This work is supported by the joint research project "DFG-Forschergruppe Petri Net Technology" (H. Weber, Coordinator) at TU Berlin and HU Berlin and by the project Graphit (DLR, Germany / CNPq, Brazil).

In order to support an intuitive understanding of Petri net behavior, especially for non-Petri net experts, it is desirable to have a layout of the model in the application domain. However, there are no specific tools to support an application-specific layout of system states and transformations modeled by Petri nets. Moreover, a formal relationship between the system model based on Petri nets and a corresponding layout of the model as icons from the application domain is missing. Such a support, called animation view for Petri nets in our framework, is presented in this paper based on our general framework for animation of visual languages in [2, 4] and in [13]. The animation view shows directly the states and dynamic changes of a system. In our sample application domain of a kitchen, the animation view of the well-known producer/consumer system visualizes producing as baking and consuming as eating cakes. Our paper is based on a generic approach how to visualize behavior and animation of a system given as a diagram of a specific visual language (VL) which defines e.g. a Petri net class. For this purpose, we use GenGED [1] as generic description of VLs, especially we consider different types of Petri nets (low-level and high-level Petri nets).

The GenGED approach is based on algebraic graph transformation and graphical constraint solving techniques and tools [1, 15, 16] and has been successfully applied to a variety of VLs, including different versions of UML class diagrams, Nassi-Shneiderman diagrams, statecharts as well as low-level and high-level Petri nets [8, 1, 3, 6, 2]. The corresponding visual environment [1, 7] supports the generic description of VLs and the generation of language-specific graphical editors.

Our approach to define the animation of Petri nets relies on the formal basis of GenGED for specifying VLs, including visual modeling techniques like Petri nets. In order to clarify the necessities for such descriptions let us take a closer look on diagrams like those shown in Figure 1. On the one hand, we have some graphical *symbols* like classes and associations in a UML class diagram or like places, transitions and arcs in a Petri net. On the other hand, there are spatial *relations* between symbols, e.g. an association arrow must start at the border of a class symbol, or an arc in a Petri net always connects a place and a transition.

In analogy to formal textual languages, for a specific VL an *alphabet* is defined over which sentences (diagrams), can be constructed. This visual alphabet captures all information about symbols, their relations and layout conditions. Yet, it is not sufficient to construct diagrams over an alphabet. In this case, there would be diagrams with illegal syntactical constellations (like multiple tokens on one place in a diagram over an alphabet for elementary Petri nets). Thus, like in formal textual languages, we have to give some *rules* defining the generation of syntactically legal diagrams over an alphabet. Together with a *start diagram*, these rules form a *syntax grammar*. The alphabet and the grammar of a VL establish a VL specification over which diagrams can be generated. Using graph grammars as underlying formal basis, we have a natural visual formalism for

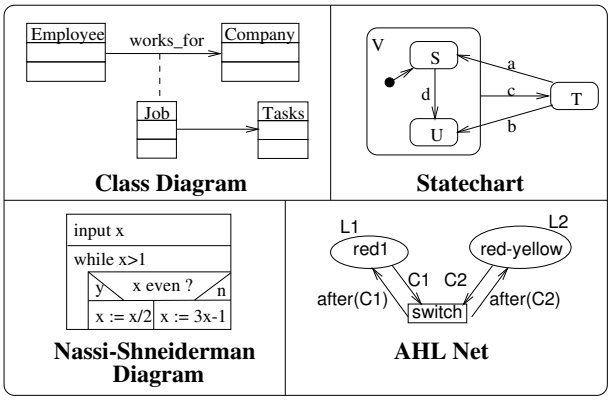

Fig. 1. Diagrams from Visual Languages

the definition of VLs. Moreover, the well-defined concepts of VL specifications in GenGED offer the basis for the specification of animation.

The behavior of a Petri net is given in our proposed GenGED framework by a set of behavior rules which correspond to the firing behavior of the transitions ([5, 9, 17]). An animation view for a Petri net basically is defined by a new layout of the Petri net tokens in each marking, i.e. a combination of suitable icons according to the specific application domain of the model. Additionally, the behavior rules modeling the firing behavior are transformed into animation rules for the animation view defining the state transitions of the system in the new animation layout. In order to have compatibility with the behavior rules, we define view transformation rules leading from the behavior rules to the animation rules of the animation view. In the case of our simple producer/consumer example, the animation rules directly define the production and consumption of cakes in the kitchen.

The paper is organized as follows: In Section 2 we introduce our running example, a producer/consumer system specified as elementary Petri net. We present an animation view of the system in the domain of a kitchen where two persons are baking (producing) and eating (consuming) cakes, respectively. Section 3 at first reviews the formal specification technique of algebraic graph grammars. Then we go into more detail and explain the GenGED approach applying its concepts to the specification of the visual syntax of a Petri net language. The GenGED framework is extended to incorporate the specification of behavior and animation of Petri nets in our main section 4. The current state of the implementation of the concepts in the GenGED environment is sketched in Section 5.

2 Example: A Producer/Consumer System

2-Example

As running example for the formal specification of a system model and its animation view, we use the well-known specification of a producer/consumer system as elementary Petri net. This example is (like the reader/writer protocol) one of the basic models for communication-based systems: two independent agents (the *producer* and the *consumer*) communicate via a channel (the *buffer*). The producer sends messages (writes) to the channel, and the consumer receives (reads) them from the channel. Rather than visualizing the flow (reading/writing) of messages, we suggest an animation view where real goods are produced and consumed. Thus, the underlying idea is visualized in a more concrete way.

The visual language (VL) we specify by using the GenGED approach defines the common graphical representation of elementary nets.

ProdCons

Fig. 2 shows the Petri net modeling the behavior of a producer who is producing and delivering goods, and a client (consumer) who is removing the goods from a buffer and consuming them. The places modeling the different buffer states on the one hand ensure that goods are delivered only if the consumer needs them (the buffer is empty). On the other hand, the consumer can consume a new good only if the buffer was filled by the producer in advance. The left subnet consisting of the places ready to deliver and ready to produce and the transitions produce and deliver is the specification part corresponding to the producer. Analogously, the right subnet (the places ready to consume and ready to remove and the transitions remove and consume) corresponds to the consumer. The places buffer filled and buffer empty and the arcs to the producer and consumer subnets model the buffer, i.e. the interface between producer and consumer.

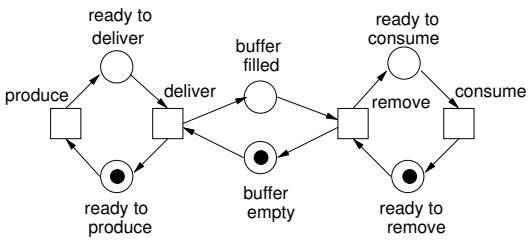

Fig. 2. ProdCons Net *Producer/Consumer*

PC-AnimView

A possible animation view of the net is illustrated in Fig. 3. The producer and the consumer are visualized as symbols representing a mother and her child in a kitchen. The mother is producing cakes and the child is consuming them.

The producer subnet corresponds to the mother standing near the stove baking (producing) cakes, and putting (delivering) them onto the table (the buffer), whereas the consumer subnet is visualized as a child taking (removing) the cakes from the table and eating (consuming) them. As the Petri net marking models a state of the system where both producer and consumer are ready

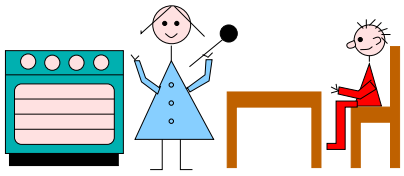

Fig. 3. Animation view for *Producer/Consumer*

(waiting) to produce respectively consume, and the buffer is empty, there is no cake to be seen in our animation view, but the mother is ready to produce one, and the child is waiting for it.

The transitions of the Petri net in Fig. 2 are not visualized in the animation view. In the user interface of the GenGED environment they correspond to animation rules visualized in an extra menu (see Section 6). The animation then is triggered by the user who selects an animation rule whereupon an action is performed (if the respective transition is enabled in the corresponding Petri net marking). Thus the behavior of the Petri net (the token game) can be simulated in the animation view.

Of course, different animations for the same system model are feasible. For example, communication between two partners in general might be visualized by special symbols for requests and answers, or by animating the contents of a message appropriately.

3 Defining Petri Net Languages within GenGED

In this section we review the basic concepts used for generic description of syntax, behavior and animation of Petri nets using GenGED. In general these basic concepts are given by algebraic graph transformation which is briefly introduced in Section 3.1. In the GenGED approach graph transformation is applied to the generic description of visual languages (VLs) consisting of a visual alphabet and a visual grammar. We review the GenGED concepts with the focus on the specification of a place/transition Petri net language: in Section 3.2 we give a visual alphabet of Petri nets, and in Section 3.3 we propose a visual syntax grammar which is based on the visual alphabet.

3.1 Review of Graph Transformation

In GenGED, diagrams as the Petri net in Fig. 2 are visual sentences of a VL, i.e. they consist of an abstract syntax level (the symbols and links) and a concrete syntax level (their layout). Diagrams are formalized as attributed graph structures, a generalization of attributed graphs. Attributed graph structures allow to define arbitrary graphical symbols as sorts and their connections as operations in a corresponding attributed graph structure signature. The formalization of rule applications as categorical pushout construction in the category of attributed

graph structures is slightly different to the construction for graphs and allows a cleaner separation of operations on graphs and data type attributes. In this paper, it is sufficient to keep in mind that all attributed graphs are attributed graph structures and that the transformation of visual sentences by rules of a visual grammar works in a way similar to attributed graph transformation. Hence, in this section we review the main concepts of attributed graph transformation [19] as they are used within GenGED. A detailed discourse on the formal backgrounds can be found in [1].

We illustrate the use of attributed graph grammars in GenGED by specifying simple place/transition nets (P/T nets) as graphs and sketching their manipulation as graph rules. Our P/T nets allow multiple black tokens for each place, but restrict the arc weight to one for all arcs, therefore we have no arc inscriptions in a net. The sample elementary net in Fig. 2 then can be expressed as a sentence of our specified P/T net language.

In the theory of algebraic graph transformation, a graph is given by two disjoint sets (graph objects), called *nodes* (vertices) and *edges* (directed arcs) from a source node to a target node. Every graph object is typed over a *type graph*. Fig. 4 (a) represents a graph with six nodes and five arcs (between them). The nodes are of type *Place* (white circles), *Transition* (rectangles) and *Token* (black dots). The arcs representing Petri net arcs are of type *ArcPT* and *ArcTP* (solid lines), whereas tokens belonging to a place are represented by *Token* nodes connected to a *Place* node by arcs of type *tk*. The corresponding type graph is shown in Fig. 4 (b). Here, the nodes and arcs represent the types themselves, whereas the graph objects in Fig. 4 (a) can be seen as instances of these types. Note that the type graph poses some restrictions on possible instances as, e.g. instances must not have arcs connecting two places or two transitions.

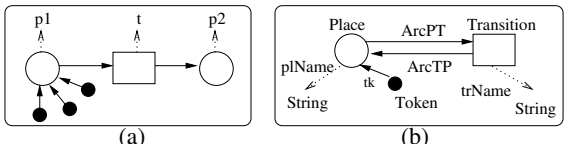

Fig. 4. A Graph (a) typed over the Type Graph (b)

Nodes and arcs may be additionally labeled by *attributes* which are used to store data together with the graph objects. In this paper we will only use attributes for nodes. In the type graph attributes are denoted by an edge carrying an attribute type name connecting a node to its attribute type (a set). In the instance graphs an attribute edge will connect a node with the current value of that attribute. In Fig. 4, the type graph (b) specifies that a *Place* node contains an attribute named *plName* of data type *String* denoting the name of a place. In the instance graph (a) the value of this attribute is a concrete name for each place. We allow abstract data types for attributes, i.e. we consider not only the sets of types, but also operations on these types. In particular, the use

of abstract data types allows us to use variables and terms as attributes (by choosing a term algebra as attribute algebra). As we will see later on, this is useful for a specification of behavior as graph grammar.

A relationship between two graphs can be expressed by a *graph morphism* g which maps the nodes and arcs of the first graph G to nodes and arcs of the second graph H, denoted by $g : G \to H$. The graph objects in G are called *origins* and in H *images*. The mappings have to be *type compatible* (nodes and arcs are mapped to nodes and arcs of the same type) and *compatible with structure* (the source/target node of an arc is mapped to the source/target node of the arc's image). Attribute values (if any) also have to coincide.

Graph transformation defines a rule-based manipulation of graphs[1]. Graph grammars (consisting of a start graph and a set of graph rules) generalize 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 (or LHS) and a right-hand side R (or RHS), and a graph morphism $r : L \to R$ between the graph objects of L and R. Graph objects in L which do not have an image via r in R are deleted; graph objects in R without original in L are created, and graph objects in L which are mapped to R by r are preserved by the rule.

The application of a rule to a graph G (derivation) requires a mapping from the rule's left-hand side L to this graph G. This mapping, called *match*, is a graph morphism $m : L \to G$. A match marks the graph objects in the working graph that participate in the rule application, namely the graph objects in the image of m. The rule application itself consists of three steps. First, the graph objects marked in the rule for deletion are deleted. Thereafter, the new graph objects are appended to the graph. As a last step, all dangling arcs are deleted from the graph. The graph transformation results in a transformed graph H. Fig. 5 shows the application of a rule inserting an arc between a place and a transition to a graph G representing a Petri net. The resulting graph H contains the Petri net after the application of the rule, i.e. the arc has been inserted.

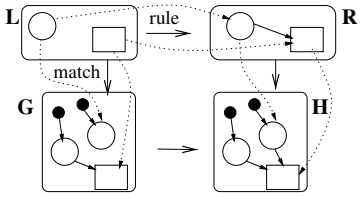

Fig. 5. Application of a Graph Rule

Rules may contain variables and terms as attributes. Using attributed graphs, the attribute values or variables of the rule's left-hand side have to match as

[1] We here follow the Algebraic Single-Pushout approach to graph grammars [19, 10].

well. An attribute variable is bound to an attribute value in the mapped graph object by the match. In the transformed graph, the attribute values are evaluated depending on the rule's right-hand side and result in a constant value.

The GenGED approach allows the generic description of visual languages based on VL specifications. A VL specification consists of a visual alphabet of pictorial objects and a visual syntax grammar. VL sentences (diagrams) can be derived from the start graph by applying the grammar rules.

3.2 The Visual Alphabet

alphabet

In general, a diagram consists of a set of *symbol graphics* that are spatially related. We offer graphical constraints for these spatial relationships, called *link constraints*. Symbol graphics and link constraints concern the layout of diagrams, called *concrete syntax*. The logical part of diagrams is called *abstract syntax*. The combination of both syntactical levels is called *visual syntax* level and can be represented by attributed graphs.

A *visual alphabet* establishes a type system for *symbols* and *links*, i.e. it defines the vocabulary of a VL. Note that in an alphabet, the symbol and link types have to be unique as well as the link arcs have to be acyclic.

Symbol graphics and graphical constraints specify layout conditions. In addition to logical (node) attributes as considered in Section 3.1, symbol graphics define a further kind of attributes for all abstract symbol nodes. Graphical constraints specify layout conditions. They are given by (in-)equations over constraint variables denoting the positions and sizes of graphical objects. For example, the constraint "point p1 lies always to the left of point p2" can be expressed as in-equation $p1.x \leq p2.x$ over the x coordinates of both points. The set of all constraint variables and constraints define a constraint satisfaction problem (Csp) that has to be solved by an adequate variable binding in a diagram over the alphabet.

3.1-Review

exa:VisPT

Definition 1 (Visual P/T Net Alphabet).
The visual alphabet of the P/T net language, called P/T net alphabet, is briefly illustrated as a graph in Fig. 6. We use rectangles for the abstract syntax of lexical symbols and rounded rectangles for the abstract syntax of attribute symbols. The dashed arrows mark the connections of the abstract syntax and the concrete syntax level.

PT-Alphabet

In the P/T net alphabet, the attribute symbol for the place name is called PlName and linked by pn to the lexical symbol Place. The attribute symbol for the transition name is called TrName and linked by tn to the lexical symbol Transition. Each name is given by a String data type that is to be written in a certain text font and text size. We distinguish arcs that run from places to transitions (ArcPT) and arcs that run from transitions to places (ArcTP). Both kinds of arcs have a certain source and target symbol where they are linked to (depicted by the edges spt, tpt, ttp, stp, short for source/target of place-transition arc resp. transition-place arc).

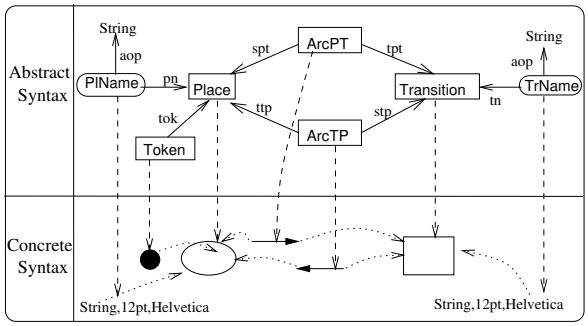

Fig. 6. PT-Alphabet: Concrete Syntax of the P/T Net Alphabet

Some link constraints are illustrated by dotted arrows between the symbol layouts. The constraints force a specific layout of diagrams typed over the visual alphabet. For example, one constraint ensures that the place name is always "near" the ellipse. Another constraint on the connection of a token symbol and its place states that a token is always visualized "inside" the corresponding place symbol. △

Note that the visual alphabet of the AHL net language [4] is similar to the visual alphabet of P/T nets presented above. The difference is given by arc inscriptions and the modeling of tokens. In the AHL net alphabet, tokens are modeled as string data types instead of pictorial objects as in the P/T net alphabet. Arc inscriptions are modeled as string data types, too. Obviously, the definition of alphabets for different Petri net classes in GenGED is straightforward. Especially for the design of visualizations of new Petri net features, GenGED offers a simple way to generate a prototype editor. For an example, see [14] where a visual language for model evolution is defined consisting of a combination of AHL nets and class diagrams.

Let us proceed with the P/T net language. An example for a visual sentence over our P/T net alphabet is our producer/consumer system in Fig. 2. Here, only symbols and links from our alphabet are used and connected according to the alphabet (the type graph). All graphical constraints are satisfied. For illustration we show the visual syntax of a subnet of this system in Fig. 7.

As already mentioned, the visual alphabet establishes a type system for all possible instances. Such instances also occur in visual syntax grammars we consider in the following section.

3.3 The Visual Syntax Grammar

The visual alphabet is the basis to define the syntax grammar for our P/T net language. The syntax grammar is represented by a graph grammar: it consists of a *start diagram* and a finite set of *graph rules*. The start diagram and both sides of a rule are diagrams typed over a specific alphabet, as well as the diagrams which

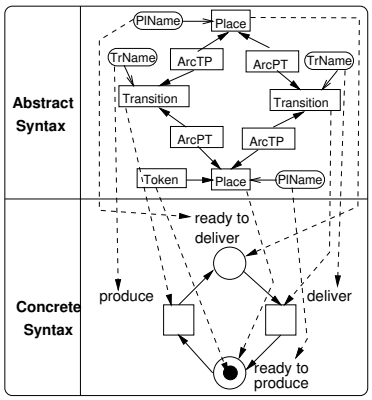

Fig. 7. PT-Sentence over the P/T Net Alphabet

can be derived by applying grammar rules. In our graph grammar rules, we use in addition to the left-hand rule sides so-called negative application conditions (NACs) which restrict the application of a rule. An NAC is a graph containing a forbidden graph pattern. The rule must not be applied to a graph if there is a match from the NAC to the graph, i.e. the forbidden pattern is found in the graph.

In Def. 2 we define the syntax grammar for the P/T net language. This definition has to be fixed once and allows the generation of a syntax-directed editor for arbitrary P/T nets (sentences corresponding to the defined syntax of the language). We define the insertion of the symbols Place, Transition, ArcPT, ArcTP and Token, as well as their graphical relation.

Definition 2 (P/T Net Syntax Grammar).
Fig. 8 illustrates a syntax grammar for our P/T net language which is based on the visual alphabet in Def. 1. In our P/T net grammar, the start sentence is empty. The first rule supports the insertion of a place together with a place name; the NAC requires that the place with the user-defined name is not already in the sentence where the rule is going to be applied to. The second rule analogously supports the insertion of a transition symbol. The next two rules allow for the insertion of arcs, either running from a place to a transition (insArcPT) or running from a transition to a place (insArcTP). The NACs forbid the application if there is already an arc. Last but not least, tokens can be inserted by applying the rule addToken. △

The application of a rule to a diagram G is obtained via a match morphism on the abstract syntax level as explained in Section 3.1. The derivation of the abstract syntax of diagram H has to be extended by its concrete syntax according to the alphabet. The diagram-specific Csp of H is derived from the Csp of the alphabet, and a Csp satisfaction (a solution) is computed. Fig. 9 illustrates the application of the rule *InsArcPT*. This rule allows the insertion of an arc

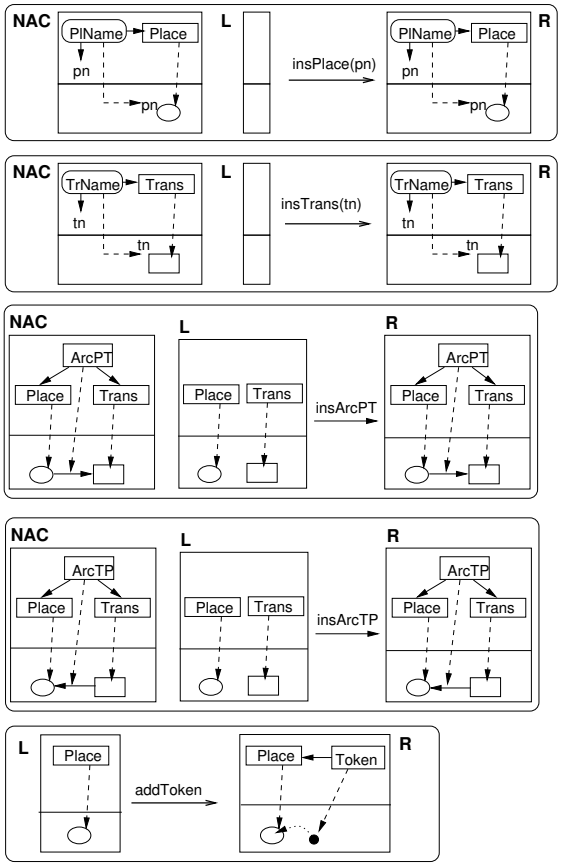

Fig. 8. Visual Syntax Grammar for the P/T Net Language

between a place and a transition in our visual P/T net language. The match and rule morphisms are indicated by the numbers of corresponding nodes.

A VL is generated by applying syntax grammar rules. The sentences occurring in the grammar as well as those which are derived by applying grammar rules are typed over the corresponding visual alphabet. Hence, a VL is generated by a VL specification.

exa:PN-VL

Definition 3 (Visual P/T Net Language).
The visual P/T net language is given by the VL specification

P/T Net Specification = (P/T Net Alphabet, P/T Net Syntax Grammar)

where the P/T Net Alphabet has been defined in Def. 1, and the P/T Net Syntax Grammar is given in Def. 2. △

exa:VisPT

exa:PN-VG

The visual P/T net language as regarded so far is the basis to define behavior rules and domain-specific animation views as explained in the following section.

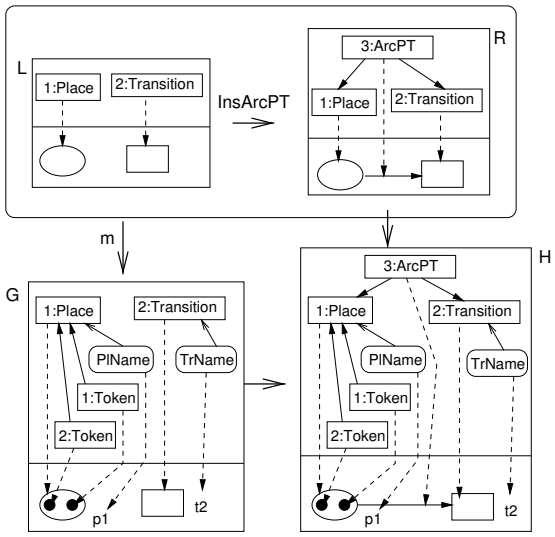

Fig. 9. Application of the Rule *InsArcPT*

4 Animation of Petri Nets within GenGED

4-Animation

In order to bridge the gap between the underlying formal, descriptive specification of a process model (i.e. a Petri net) and a natural dynamic visual representation of processes being simulated, we suggest the definition of an animation view for a visual process model. On the one hand, this animation view has to be readily comprehensible; people who are non-specialists in the modeling technique of Petri nets should be able to observe functional behavior of the model. On the other hand, the behavior shown in the animation view has to correspond to the behavior defined in the original process model. In Section 4.1, we consider the graph grammar based concepts for the behavior specification of Petri nets. These concepts are explained along our Producer/Consumer net. Thereafter, in Section 4.2 we give guidelines for the specification of an animation view for a specific Petri net and present an application-specific animation grammar for our Producer/Consumer net. To ensure a consistent mapping of the Petri net behavior, we propose in Section 4.3 a graph grammar based view transformation from the Petri net to its animation view.

4.1 Behavior of Petri Nets

4.1-Behavior

In Petri nets and similar visual process models, two aspects can be considered. The first aspect concerns the topological structure of the model, i.e., which visual elements exist and how are they linked to each other. The second aspect concerns the behavior of the modeled system, i.e., the flow of control within processes and the flow of communication between processes. In the previous section, we dealt

with the visual specification of the Petri net language, i.e. we specified what a Petri net *is*, but not what it *does*. Therefore, in this section we focus on simulating the dynamic behavior of P/T nets by a visual grammar approach based on our P/T net language.

In the literature one can find several graph grammar based approaches for the specification of Petri net behavior: Schneider [21] summarizes different approaches to define the behavior of process systems (e.g. Petri nets, event structures and actors) by graph grammars. He states that graph grammars are well suited to describe in a uniform way not only the syntactical structure of visual process models but also their semantic properties. In Petri nets, the basic form of a state transformation is the firing of a transition. The straightforward technique for behavior simulation therefore is playing the token game.

One of the first who discussed the relationship between graph grammars and Petri nets is Kreowski [18]. He associates a graph rule to each transition, with the rule being applicable if and only if the transition is allowed to fire. Tokens within a place are modeled as a bundle of new nodes connected to the node representing the place. This approach is able to handle both places with bounded capacity and places with unbounded capacity and can easily be extended to individual tokens.

Parisi-Presicce et al. [20] use a structured alphabet for labeling places with tokens. This alphabet has to allow changes of node labels in rule morphisms. In the case of high-level Petri nets, multiple and individual tokens can be represented using multisets as labels and the multiset inclusion as the structure of the alphabet. A further step is to allow arbitrary categories to label the places. In [2], we modeled tokens in AHL nets as arbitrary algebraic data types which are attributes of *Token* nodes.

We consider a Petri net as the set of all sentences over the Petri net language with the same net structure whose markings are given by an initial marking and all possible successor markings that can be reached by arbitrary transition firing steps. The behavior of a P/T net is defined by a visual grammar, called *behavior grammar*: the start sentence corresponds to the initial system state (the initial marking), and the behavior rules capture all possible transition firing steps in the net. When constructing a behavior grammar whose rules correspond to firing of the transitions of the net we have to ensure that

- a transition in the net is enabled if and only if the corresponding rule is applicable to the net,
- firing a transition in the net corresponds to a derivation step in the grammar and vice versa.

The token game then can be simulated by applying the rules of the grammar to a Petri net. The left-hand side defines the applicability condition, i.e. the marking corresponding to the transition's pre-domain. The right-hand side describes the effect of the transition, i.e. the marking of the pre domain places is removed, and the required tokens are added to the post domain places. This approach to Petri net simulation can be applied to various types of low-level and high-level Petri nets (see e.g. [4]). For each Petri net type, therefore, a specific net's behavior

grammar can be generated automatically according to the Petri net type's firing behavior.

exa:beh

Example 1 (Behavior Grammar for Producer/Consumer System). Fig. 10 illustrates the behavior grammar for the Producer/Consumer System. The start graph is the initially marked net as depicted in Fig. 2. Note that the P/T net behavior grammar is defined on top of the visual P/T net grammar of our P/T net language using the same visual alphabet.

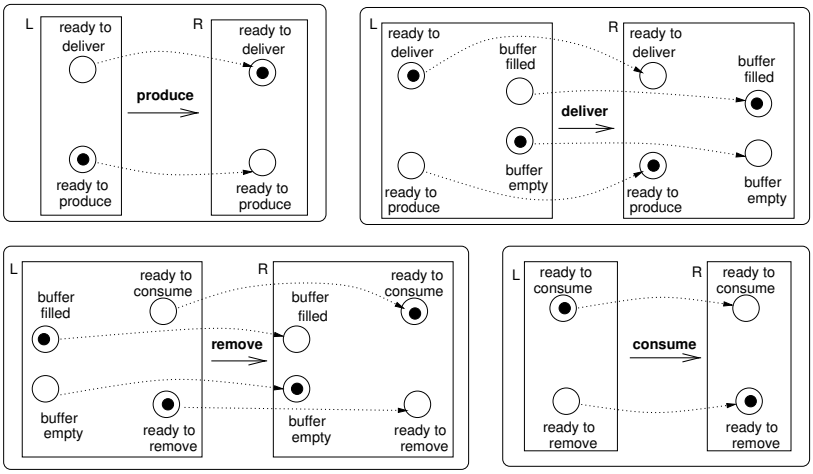

Fig. 10. PN-BehGrammar for the *Producer/Consumer* Net

◊

sec:AnimView

4.2 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 (like the alphabet from Fig. 6):

- 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 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 token game of the Petri net.

- Transitions are replaced by rule names in the animation view which are the user interface to trigger a state transformation step corresponding to a firing step of the transition in the Petri net view.
- Arcs in the Petri net have the function to define the firing behavior in a static way. They are not needed in the animation view as the behavior now is defined by the animation rules and visualized by their application (the animation). Thus, arcs are not visualized in the animation view.

exa:AV-PC

Example 2 (Animation View).
The *Producer/Consumer* net as illustrated in Fig. 2, is one sentence of our P/T net language we defined in Def. 3. The animation view of this sentence has been already motivated by Fig. 3 where two people in a kitchen are visualized. The abstract syntax of the alphabet of the animation view is equal to that of Fig. 6 but the concrete layout differs.

The fixed part of the animation view consists of the symbols for the mother standing besides the stove (corresponding to the producer places ready to produce and ready to deliver), the table (the buffer places buffer filled and buffer empty) and the child sitting on a chair (the consumer places ready to remove and ready to consume). The tokens – corresponding to the animated part of the animation view – model the different locations of a cake. A token on place ready to produce means that a cake may be taken out of the stove but is not yet to be seen. In the concrete syntax of the animation view, the token is marked as invisible. The same holds for tokens on the places buffer empty (no cake on the table) and ready to remove (the child has got no cake). Therefore, in the animation view of the initially marked net (Fig. 3), no cake is to be seen.

Fig. 11 shows our net with the only possible successor marking of the initial marking. Here, a token lies on place ready to deliver: the cake has been taken out of the stove and is visible in the animation view on top of the stove. Successor markings would be visualized by the cake put onto the table (token on place buffer filled) and the cake on the lap of the child (token on place ready to consume). The abstract syntax underlying both views (the Petri net and the animation view) also is depicted in Fig. 11, together with some of the respective 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 the animation view (view transformation rules). The generation of the animation view in Example 2 is described in Section 4.3.

The behavior of the system in the animation view 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 e.g. Example 1 for a behavior grammar of a specific Petri net). We call the grammar containing the animation rules, the *animation grammar*.

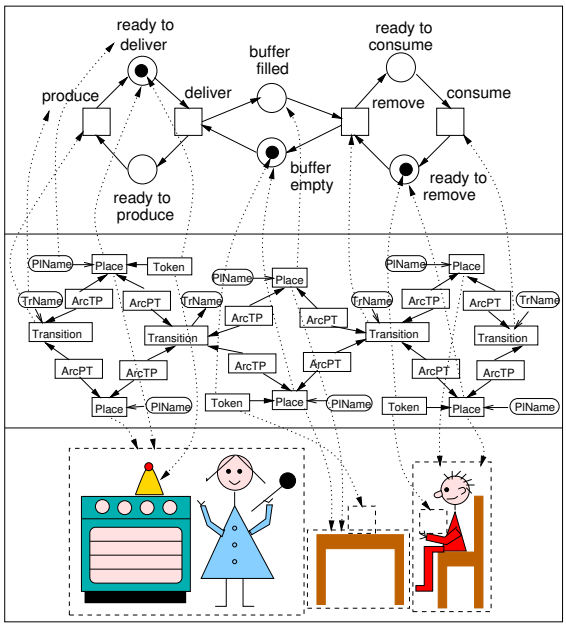

Fig. 11. PC-AnimView: Possible Animation View of the Petri Net *Producer/Consumer*

exa:AG-PC

Example 3 (Animation Grammar).
Fig. 12 shows the animation grammar for our producer/consumer system. Each animation rule corresponds to the behavior rule of the same name from the behavior grammar in Fig. 10.

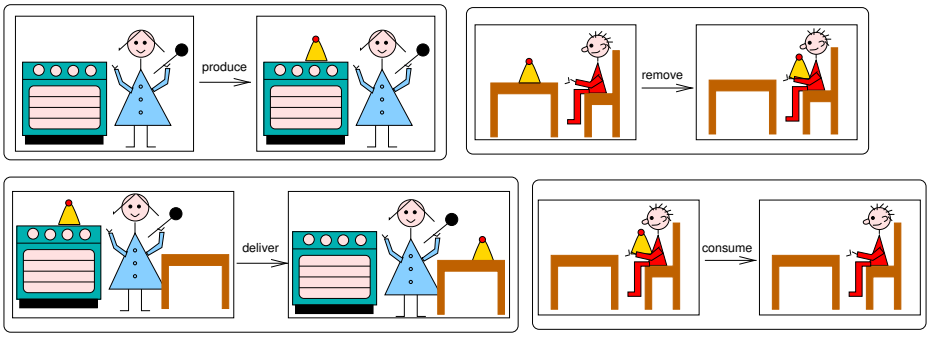

Fig. 12. PC-AnimGrammar: Animation Grammar for the *Producer/Consumer System*

◊

sec:GenAV

4.3 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). The view transformation rules must be defined by the animation view designer. On the basis of these 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 part of the Petri net to a part of the animation view by combining elements of the abstract syntax with new concrete syntax elements (i.e. by giving them a different layout). The concrete syntax of the Petri net transitions, arcs and the attributes is invisible in the animation view. After application of all view transformation rules in a suitable way, 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 the visual syntax of the Petri net and the visual syntax of its animation view because both of them contain different graphics. In the following example, therefore, graphics from both concrete syntax definitions (net layout and animation view layout) are shown in the same rules.

exa:PC-TrafoGrammar

Example 4 (View Transformation Rules).
PC-TrafoRules
Fig. 13 shows the view transformation rules needed for the generation of the
PC-AnimView ProdCons
animation view depicted in Fig. 3 from the Petri net depicted in Fig. 2. We define a view transformation rule for each part of the animation view that has a symbol for an underlying fixed part, i.e. for the producer, the buffer and the consumer. Note that the layout of a token in the animation view depends on the place it belongs to. For example, a token on place buffer filled is shown as a cake on the table, whereas a token on place buffer empty is invisible in the animation view. In the view transformation rules in Fig. 13, 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.
PC-TrafoRules

◊

By applying the view transformation rules from the view transformation
exa:PC-TrafoGrammar ProdCons
grammar described in Example 4 to the Petri net in Fig. 2, we generate the
PC-AnimView
animation view depicted in Fig. 3. The behavior of a Petri net now can be transferred consistently to the animation view by applying the view transformation rules to the LHSs and RHSs of the behavior rules.

exa:deriv.TL

Example 5 (Generating Animation Rules).
PC-DerivAnimRule
Fig. 14 illustrates the derivation of an animation rule by applying rules from the
PC-TrafoRules PN-BehGrammar
transformation grammar in Fig. 13 to a behavior rule in Fig. 10.

◊

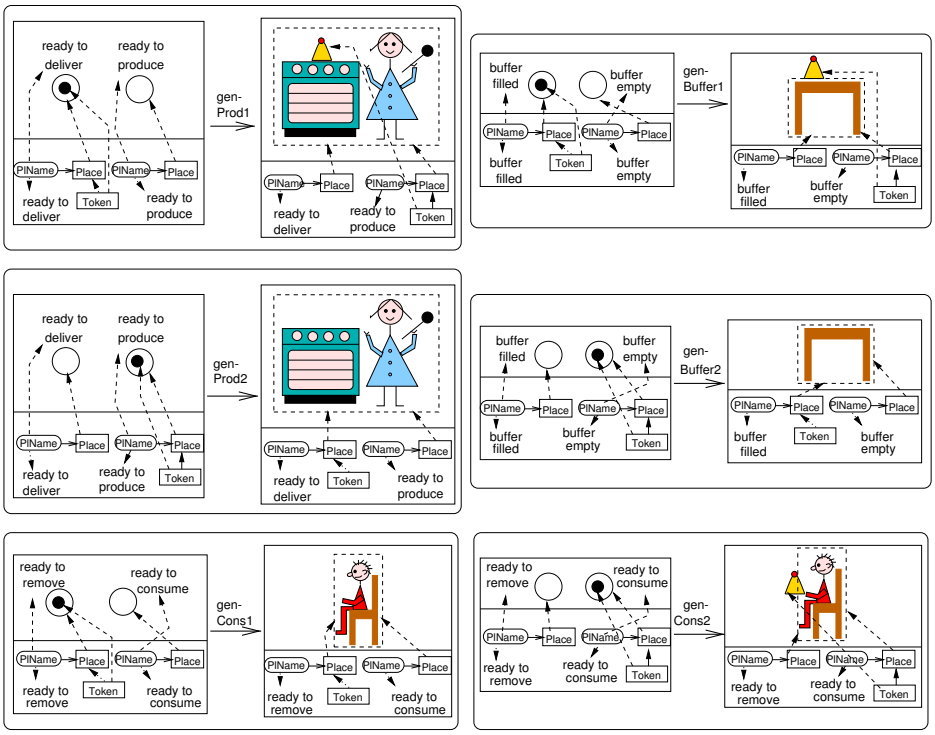

Fig. 13. [illegible] Transformation Rules for the Producer/Consumer System

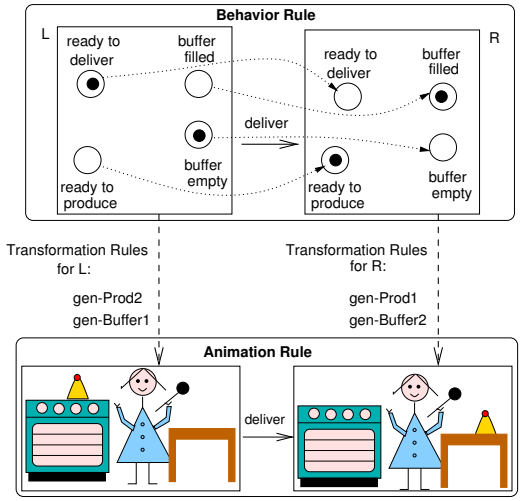

Fig. 14. [illegible] Animation Rule with View Transformation Rules

A first rough animation of the system can be performed by applying the animation rules in the animation view of the system model. A nice 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 [22] could be combined with the GenGED environment.

5 Implementation

5-Implementation

According to the constituents of a VL-specification, the GenGED environment as sketched in Fig. 15 comprises two major components: the *Alphabet Editor* and the *Grammar Editor* for the visual definition of VLs. From the VL definition using these editors, a VL specification is generated which is the input of the *Graphical Editor* for syntax-directed diagram drawing. This means that the language-specific editing commands of the Graphical Editor are given by the grammar rules of the visual grammar. Hence, not only a VL is specified but the VL-specific Graphical Editor also. Note that we distinguish two kinds of users, namely users defining a VL (*language-designer*), and those who use a Graphical Editor.

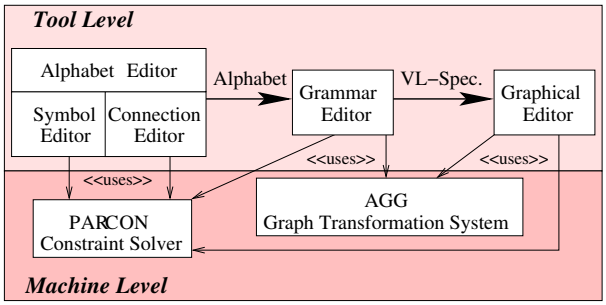

Fig. 15. GenGEdEnv about the GenGED environment

To assure the graphically correct drawing, all GenGED editors use the constraint solver ParCon [16]. The transformation of diagrams via rule application in the Grammar Editor and the Graphical Editor is done by the graph transformation system Agg [15]. The GenGED environment is implemented in Java, so is the Agg system. The ParCon constraint solver – implemented in C – is available for Linux and Solaris, thus GenGED runs on these two platforms.

5.1 The Visual Syntax

The specification of a Visual Alphabet is implemented as *Alphabet Editor* which is a bundle of two sub-editors – the *Symbol Editor* and the *Connection Editor*. A snapshot of the both the Symbol Editor and the Connection Editor is shown in Figure 16.

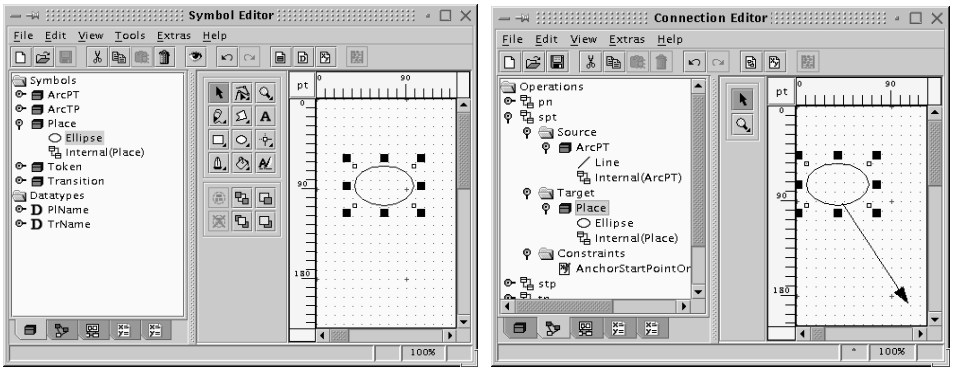

Fig. 16. The Symbol Editor and the Connection Editor of the Alphabet Editor

In the Symbol editor, the language elements are defined: For each symbol type in the abstract syntax the user gives a unique symbol name (e.g. Place) and a symbol graphic (e.g. an ellipse) and possibly some symbol constraints. For the definition of the symbol graphics, the Symbol Editor works similar to well-known vector editors except that the grouping of symbols is handled using graphical constraints to connect the primitives in a symbol graphic. Available primitives are e.g. lines, poly-lines, rectangles, ellipses, images (GIF/JPEG) and text. The primitives' properties like color, line width or text properties can be edited, too. Attribute symbols appear as independent graphical objects in the Symbol Editor.

The Connection Editor supports the definition of links between symbols. In order to define a link, the user can select any two symbols as source and target of the link in the abstract syntax (e.g. Place and ArcPT). A constraint dialog supports the definition of link constraints in the concrete syntax.

The definition of a visual syntax grammar is supported by the Grammar Editor available in the GenGED environment. The Grammar Editor gets an alphabet as input. From this input, so-called *alphabet rules* are generated defining the editing commands of the Grammar Editor. The set of alphabet rules comprises rules for the insertion and deletion of symbols. In the snapshot of the Grammar Editor shown in Fig. 17 one can see the alphabet rule for the insertion of a transition name in the upper part.

The lower part of the Grammar Editor denotes the *working areas*: here we build the start diagram, and the LHS and RHS (or LHS and NACs, respectively)

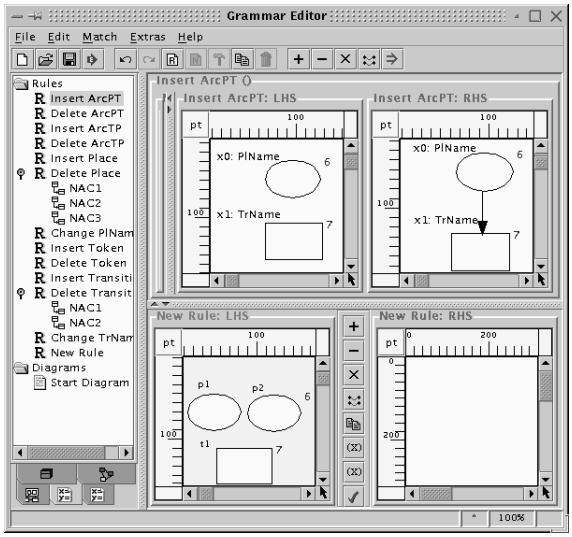

Fig. 17. Syntax Editor

of a VL-rule, add mappings between the two rule sides and edit the rule parameters. Applying a rule with rule parameter to a diagram in one of the working areas, the user is first asked to define the match morphism, i.e., to map the symbols of the rule's LHS to type-consistent symbols in the diagram. Then, the user has to give a value (or a variable) for the parameter such that the expressions in the RHS can be evaluated during transformation.

The next step is to export the set of VL-rules and the start diagram into a visual language grammar[2]. Then, the Graphic Editor uses the grammar rules to provide the language-specific editing commands. Fig. 18 shows the generated Graphic Editor where our Producer/Consumer Petri net is drawn in the edit panel using the visual grammar rules of the syntax grammar. The syntax grammar rules can be selected in the tree view at the left-hand side in order to edit symbols.

5.2 Behavior and Animation

It is possible to generate behavior rules for arbitrary Petri nets automatically according to the general definition of firing transitions in nets of the specific Petri net class. (For the class of AHL nets see [5]). The algorithm for generating behavior rules for Elementary nets is sketched in Def. 4. It will be used for the implementation of our simulation and animation concepts in the GenGED environment. Then, for the well-known Petri net types, it is not necessary for the user to specify the behavior for each specific net. Instead, the behavior rules for the transitions of a net can be generated by the tool.

[2] The alphabet is added automatically, so in fact we export a VL-specification.

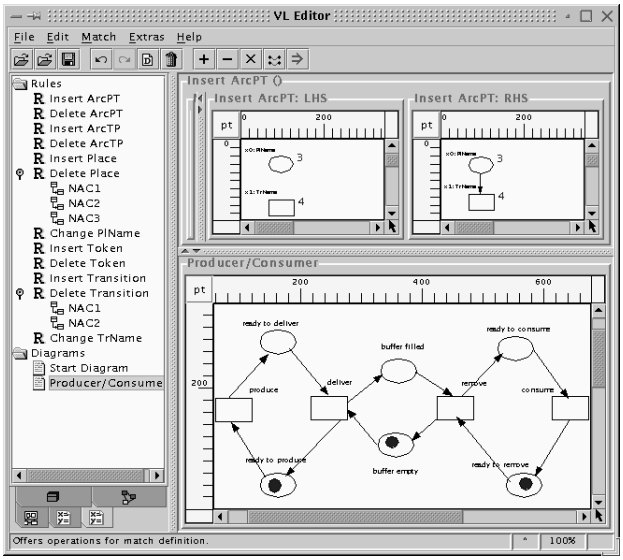

Fig. 18. [illegible] for Petri nets

def.en2agg **Definition 4 (Translation of Elementary Nets to Graph Grammars).** Each transition $t \in T$ is translated to an attributed graph grammar rule $r_t : L_t \to R_t$. The attributed graphs in the LHS L_t and the RHS R_t of such a rule both contain nodes for all places in the pre and post domain of t. In L_t, the places in the post domain are not marked. The marking of the pre domain places p_i is computed as follows:

for each arc $a : p_i \to t$
  generate a Token vertex;
  connect the Token vertex by an edge of type tok to place p_i;

Analogously, in R_t, only places in the post domain p_j become marked:

for each arc $a : t \to p_j$
  generate a Token vertex;
  connect the Token vertex by an edge of type tok to place p_j;

Moreover, an NAC is added to the rule $r_t : L_t \to R_t$ containing the places from the post domain $post(t)$ marked by one token each. This NAC ensures that the rule is applied only if the places in the post domain of the transition are unmarked. △

As the automatic generation of behavior rules is not yet implemented in the GenGED environment, we define them in the same way as the syntax rules: For the definition of Petri net behavior, again the Grammar Editor is started. After editing the behavior rules the Petri net simulator (similar to the Graphic Editor

but with behavior rules instead of syntax rules) is generated. Fig. 19 illustrates the simulator for our Producer/Consumer net. Applying a rule this time means to simulate the firing of a transition. In the screenshot, rule produce is selected and the match is indicated by equal numbers for corresponding objects. The application of the rule removes the token from the place ready to produce and adds a new token to the place ready to deliver.

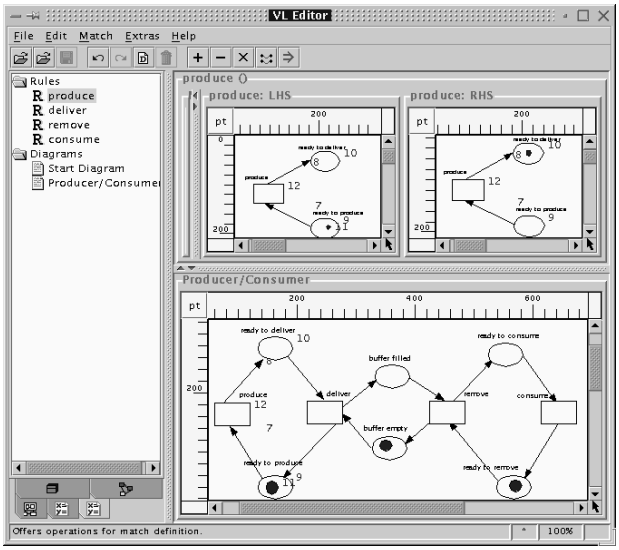

Fig. 19. PN-Simulator for the Producer/Consumer Net

In the corresponding animation grammar the animation rules have the layout of the animation view. The implementation of support for defining the view transformation grammar is still work in progress. Here, we need a user interface allowing the animation view designer to define the view transformation rules. These rules are to be applied suitably (in a user-controlled order) to the LHSs and RHSs of each behavior rule. Thus, the view transformation interface has to support the application of rules to rules in order to generate animation rules from behavior rules. Moreover, we need to define rule constraints to ensure specific layouts for symbol connections which are valid only if a specific rule is applied. For instance, in the Producer/Consumer animation view, the cake symbol is placed on top of the table symbol when applying rule deliver, but it is placed on top of the stove symbol when applying rule produce. Fig. 20 illustrates the desired animation view interface.

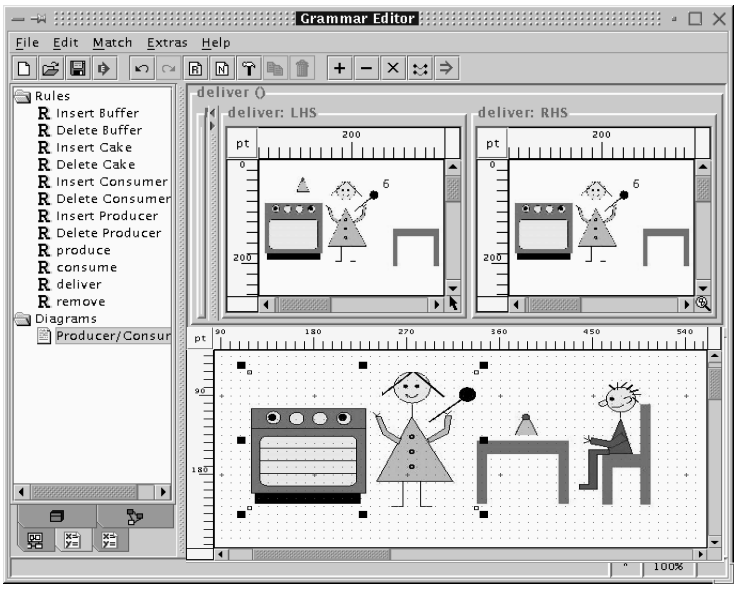

Fig. 20. Animation View for the Producer/Consumer Net

6-Conclusion

6 Conclusion

We have presented the GenGED approach supporting the generic description of visual languages and provided concepts for an extension of GenGED in order to allow the description and implementation of application-specific animation views for communication-based systems modeled as Petri nets. An animation of the Petri net behavior then shows directly the state transitions in the layout of the application domain. An animation view is realized by graphical icons corresponding to parts of the system. During animation, some icons are changed according to the selected state transition. GenGED is based on graph transformation. This provides a natural formal basis to express Petri nets and their animation views as graphs, whereas the behavior of the model (i.e. transition firing steps resp. state transitions in the animation view) can be formalized as graph grammars.

A specification of a specific Petri net type has to be provided once as visual language specification (graphical symbols and syntax rules) and allows the generation of a graphical editor for arbitrary nets of the specified Petri net type. The specifications of different Petri net types all have the same basis (places, transitions, arcs and tokens as symbols) and only differ in the representation of tokens (black dots or strings) and in their firing rule.

The behavior grammar for a specific Petri net can be generated automatically according to the firing rule of the corresponding net type: Each transition is converted to a graph grammar rule whose left/right-hand side corresponds to the transition's pre/post domain.

The designer of an animation view therefore only has to specify the correspondence of tokens to icons from the animation view representing the application domain. This relation of a Petri net and its animation view is defined as view transformation graph grammar and allows to map the behavior of the Petri net consistently to the animation view. Due to the generic and modular definition of syntax, behavior and animation for different Petri net types, the presented framework reduces considerably the amount of work to realize an application-domain animation of a system modeled as a specific Petri net.

It remains to develop a formal theory to handle behavior and animation of visual process models in general (i.e. covering other Petri net classes and modeling techniques like statecharts or message sequence charts). The theory should include formal definitions for an automatic generation of behavior rules for well-known Petri net classes and a formal transformation of the states and behavior rules of a general visual process model into is animation view. Implementational work is still in progress, i.e., concerning the user interface for defining view transformation rules, rule constraints and the mapping of rules to rules.

Within the Petri Net Baukasten [11] the proposed animation framework will be an extension of the functionality provided by the Petri net tool infrastructure PNK and by the external tools integrated over the PNK. In order to offer the features of the extended GenGED tool environment to PNK users, an XML conversion between the XML file interchange formats of the PNK and GenGED has been implemented [13, 12]. Thus, it becomes possible on the one hand to use the editing, simulation and analysis features provided by the PNK and on the other hand to have a visual environment for the definition of domain-specific animation views for Petri nets provided by GenGED.

References

gBar00 [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.

gBEE00 [2] R. Bardohl, H. Ehrig, and C. Ermel. Generic Description, Behaviour and Animation of Visual Modeling Languages. In *Proc. Integrated Design and Process Technology (IDPT 2000)*, Dallas (Texas), USA, June 2000.

gBE01a [3] R. Bardohl and C. Ermel. Visual Specification and Parsing of a Statechart Variant using GenGEd. In *Proc. Symposium on Visual Languages and Formal Methods (VLFM'01)*, Stresa, Italy, September 5–7 2001.

gBEE01 [4] R. Bardohl, C. Ermel, and H. Ehrig. Generic Description of Syntax, Behavior and Animation of Visual Models. TR 2001/19, TU Berlin, 2001. ISSN 1436-9915.

pBEP02 [5] R. Bardohl, C. Ermel, and J. Padberg. Formal Relationship between Petri Nets and Graph Grammars as Basis for Animation Views in GenGED. In *Proc. IDPT 2002: Sixth World Conference on Integrated Design and Process Technology*, 2002. To appear.

gBER00a [6] R. Bardohl, C. Ermel, and L. Ribeiro. Towards Visual Specification and Animation of Petri Net Based Models. In *Proc. GRATRA 2000 - Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems*, pages 22–31. Technische Universität Berlin, March 2000.

gBNS00 [7] R. Bardohl, M. Niemann, and M. Schwarze. GenGEd – A Development Environment for Visual Languages. In *Int. Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE'99), LNCS 1779*, pages 233–240. Springer, 2000.

gBST01 [8] R. Bardohl, T. Schultzke, and G. Taentzer. Visual Language Parsing in GenGEd. *Electronic Notes of Theoretical Computer Science*, Vol. 50, No. 3, June 12–13 2001.

gCM95 [9] 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, 1995.

gEHK+97 [10] 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.

pWER+02 [11] H. Ehrig, W. Reisig, and H. Weber et al. The Petri Net Baukasten of the DFG-Forschergruppe PETRI NET TECHNOLOGY. In this Volume.

sKEhr01 [12] K. Ehrig. Converting XML Files with XSLT and XPath, `http://tfs.cs.tu-berlin.de/lehre/SS01/gragra.html`, 2001. Student's Project Status Report.

pEBE01 [13] C. Ermel, R. Bardohl, and H. Ehrig. Specification and Implementation of Animation Views for Petri Nets. In Weber et al., editors. *2nd Int. Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, Berlin, Germany, Sept. 2001. Fraunhofer Gesellschaft ISST, pages 75–92.

gEBP01 [14] C. Ermel, R. Bardohl, and J. Padberg. Visual Design of Software Architecture and Evolution based on Graph Transformation. In *Int. Workshop on Uniform Approaches to Graphical Process Specification Techniques (UNIGRA'01), at ETAPS'01*, 2001. Electronic Notes in Theoretical Computer Science, Vol. 44, No. 4.

gERT98 [15] C. Ermel, M. Rudolf, and G. Taentzer. The AGG-Approach: Language and Tool Environment. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools*, pages 551–603. World Scientific, 1999.

sGri96 [16] P. Griebel. *Paralleles Lösen von grafischen Constraints*. PhD thesis, University of Paderborn, Germany, February 1996.

gKR95a [17] 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.

gKre81 [18] H.-J. Kreowski. A Comparison between Petri-nets and Graph Grammars. In *LNCS 100*, pages 1–19. Springer Verlag, 1981.

gLow93 [19] M. Löwe. Algebraic Approach to Single-Pushout Graph Transformation. *TCS*, 109:181–224, 1993.

gPEM87 [20] F. Parisi-Presicce, H. Ehrig, and U. Montanari. Graph Rewriting with Unification and Composition. In *3rd Int. Workshop on Graph Grammars and their Application to Computer Science, LNCS 291*, Berlin, 1987. Springer Verlag.

gSch94b [21] 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.

sWei00 [22] C. Weidauer. Animations-Framework in Java. Systematische Animationsentwicklung mit Mehrschichtenarchitektur. *Informatik - Forschung und Entwicklung, Band 15, Heft 2*, pages 83 –91, June 2000.