Scenario Animation for Visual Behavior Models

– A Generic Approach Applied to Petri Nets –

Claudia Ermel and Rosi Bardohl
Technische Universität Berlin
{lieske, rosi}@cs.tu-berlin.de

Abstract

Visual modeling techniques including UML as well as graph and net based techniques are of growing interest for software system specification and development. The GenGED approach developed at the Technical University of Berlin allows already the generic description of visual modeling languages based on formal graph transformation and graphical constraint solving techniques and tools.

In this paper, the GenGED approach is reviewed and extended in order to allow the description of dynamic behavior and animation of systems. The basic idea is to define visual simulation and animation rules on top of the rules defining the corresponding visual modeling language and to allow a domain specific layout for an animation view of the system. The visual language of Petri nets and a specific Petri net modeling a network protocol serve as running example. The system view is given by means of Place/Transition nets. The animation view of the network model shows directly the sending of messages between hosts in a network graph.

1 Introduction

Visual modeling techniques provide an intuitive, yet precise way in order to express and reason about concepts at their natural level of abstraction. The success of visual techniques in computer science and engineering resulted in a variety of methods and notations addressing different application domains and different phases of the development process. Despite the wide-spread usage of visual modeling techniques there is a lack of well-understood (and integrated) methodologies for specifying their syntax and semantics. Until now there exists no equivalent to Backus-Naur-Form which would be the notation for defining the syntax of a visual language (VL). The same applies to type systems, deductive proof methods, operational or denotational semantics for visual modeling techniques.

In the literature one can find several formalisms for the definition of VLs. All currently used grammars for syntax definition of visual languages, such as picture layout grammars [14], or constraint multiset grammars [24], use textual instead of visual notations, i.e., multi-dimensional representations have to be coded into one-dimensional strings. Using graph grammars and graph transformation as underlying formal basis, we have a much more natural and itself visual formalism for the definition of VLs [26, 8].

Formal specification techniques, like Petri nets [25] and Statecharts [17] allow on the one hand the formal description and analysis of systems, and on the other hand they support the intuitive understanding of the system because of the visual nature of Petri net and Statechart models. A well-known example is the specification of a clock by the visual means of the Statechart language. In order to have an intuitive understanding of the clock it is even better to have an animation view which shows directly dynamic changes of the clock in addition to the dynamic behavior of the Statechart modeling the clock. Such kind of domain specific animation is supported by the STATEMATE tool [18] for Statecharts but not yet by high-level Petri net tools like DESIGN/CPN [19]. In both cases, however, a formal relationship between the system model based on Statecharts or Petri nets and a corresponding animation view is still missing.

In this paper, we give a general approach how to present behavior and animation of a system which is modeled using an arbitrary visual modeling language (short *VL*). For this purpose, we need a generic description of VLs including Statecharts, Petri nets and suitable diagram techniques from UML [27]. As a basis, we use our general approach with tool support for the generic description of common VLs, called GENGED [2], developed at the Technical University of Berlin. The GENGED approach is based on algebraic graph transformation [11] and graphical constraint solving techniques [16] and has been successfully applied to a variety of VLs, including Nassi-Shneiderman diagrams, Petri nets and simplified versions of UML class diagrams and Statecharts [3, 4, 2, 5, 10]. In the GENGED approach a VL is specified by a visual alphabet and a visual syntax grammar. The GENGED environment [2, 6] supports the generic description of VLs and the generation of graphical editors and simulators for VL diagrams of the specified VLs. The behavior of a VL model (e.g. a suitable class of visual diagrams in the language VL) is given in our proposed GENGED framework by a set of simulation rules which define the operational semantics of the underlying system. In the case of Petri nets, the VL model consists of all Petri nets with the same net structure but different markings, and the simulation rules are in one-to-one correspondence to the well-known firing rules of the transitions in the Petri net ([23, 22]).

An animation view for a VL model basically is defined by a new layout of the language elements, i.e. a combination of suitable icons according to the specific application domain of the model. More precisely, the animation view of our system is a VL model over a VL alphabet which has been extended by symbols with a different application domain specific layout of the system states together with the corresponding animation rules defining state transitions of the system. In fact, the simulation rules of the original VL model (e.g. a Petri net) are transformed coherently into animation rules for the animation view defining the state transitions of the system in the new animation layout. In contrast to related approaches and tools for the animation of Petri net processes (see e.g. the SimPEP-tool for the animation of low-level nets in PEP [20, 15]), the generic framework GENGED offers a basis for a more general formalization of model behavior which is applicable to various Petri net classes and other visual modeling languages.

The paper is organized as follows: In Section 2 we introduce our running example, namely an echo algorithm modeled as P/T net for testing the availability of connections in a network. For the specification of this algorithm, we define a VL for P/T nets. Furthermore, we show a possible animation view of the system as motivation for the further sections. The GENGED approach is reviewed in Section 3. Here we show how to specify VLs and VL models in GENGED, focusing on a VL for Petri nets and the

simulation of our network model. In Section 4 we define the animation view for a specific VL model. In order to have compatibility with the simulation rules according to Section 3, we define *view transformation rules* leading from the original VL model (start diagram and simulation rules) to the intended animation view of the VL model (diagram in the view layout and animation rules). In Section 5 we discuss related and future work.

2 Scenario Animation: An Example

The term "scenario" is used with different meanings in different contexts. We therefore state a definition from [13] where a scenario is informally defined to be a form of interaction sequence between a system and a set of actors.

Definition 2.1 (Scenario)
A scenario is an ordered set of events, usually interactions between a system and a set of actors external to the system. It may comprise a concrete sequence of interaction steps or a set of possible interaction steps. △

Scenarios can be defined by a specification defining one specific system state as the start state of the scenario and describing possible transitions from one state to another state. Interaction consists in the selection of one out of several possible state transitions leading from the current state to the next one. Technically, in our graph-grammar-based approach, a set of possible interaction steps is defined by a simulation grammar whose start diagram defines the start state of the scenario. The simulation rules represent simulation steps: each rule contains in the left-hand side the situation which exists before the event has occurred, and in the right-hand side the changed situation after the event. Concrete sequences of interaction steps then are all possible derivation sequences that can be obtained by consequently applying rules from the simulation grammar starting with the start diagram.

Example 2.2 (A Scenario of the Network Model)
As an example for a scenario, we model the echo algorithm for a network of hosts. One distinguished host (Boss) tests the connections between all hosts in the network. The algorithm works as follows: Boss sends a broadcast message to every other host in the network. Each host receiving a message sends out a message to every other host in the network (his target group), but not to the sender of the initial message, and waits for replies. After a host has received messages from all hosts in his target group, he sends a final message to Boss. Thus, after Boss got messages back from all hosts in the network, he knows that the connections between all host nodes are working fine.

As described in the next section, we define a P/T net VL in GenGED such that we can draw P/T nets, formally VL diagrams of our P/T net VL.

In a VL model the states of the system are given as VL diagrams. All VL diagrams in our VL model consist of the same net structure and differ by the token distribution. Fig.1 shows one state of our VL model, the VL diagram with the initial marking (*Init*). We model a network with a Boss host and two other hosts. Boss is ready to send his initial message and the two other nodes in the network (Host1 and Host2) are ready to receive the message and to react to it.

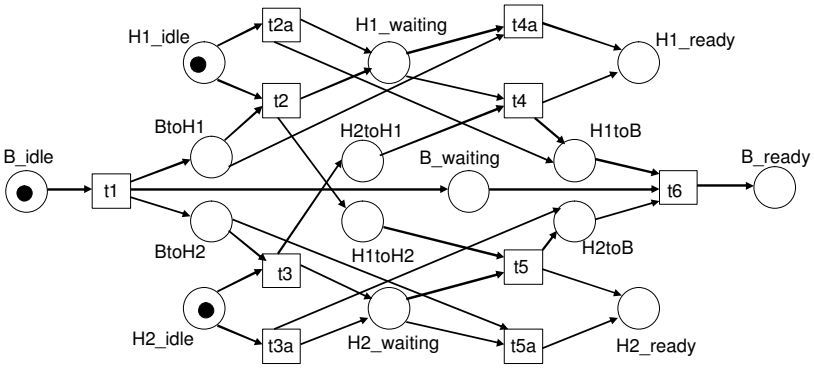

Figure 1: **VL diagram** $Init$ **modeling the initial state of the Network model**

A scenario in the sense of Def. 2.1 is given by an ordered set of simulation steps together with a start state. The user interaction consists of triggering the next simulation step in case that there is a choice of actions in the current system state. Note that in the case of the VL for Petri nets, a scenario is a trace in the marking graph. Fig. 2 shows a possible scenario. Boss sends his message to Host1 and Host2; Host1 forwards the message to Host2; Host2 himself forwards his own message from Boss to Host1; Host1 is ready now, as he has sent messages to every one else (to Host2) and he has received messages from everyone else (from Host2). As a last action, Host1 replies to Boss. The same situation applies to Host2. He is ready as well and as his last action he sends a message to Boss. The final event is that Boss gets in a "ready" state because he now has received replies from all hosts in the network.

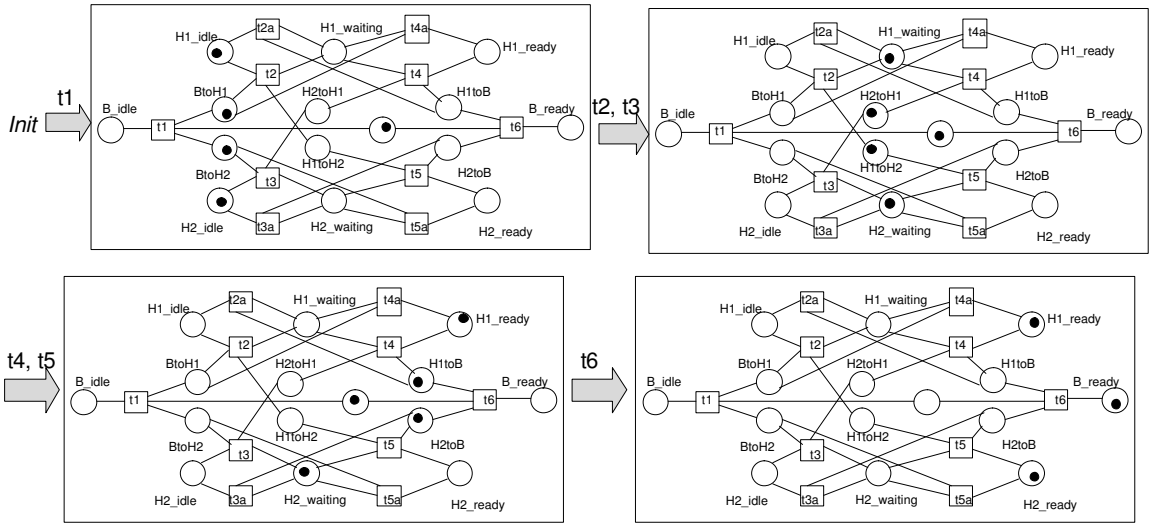

Figure 2: **Scenario of the Network Model, modeled as sequence of VL diagrams**

◊

To support an intuitive understanding of system behavior, especially for non-experts in the specific formal modeling language, it is desirable to have a visualization of (parts of) the VL model in the application domain and show the behavior directly in the layout of the application domain. This is also important for teaching purposes.

Considering our network example, we only want to know whether each host in the network is getting a message at some time and forwarding it to everyone else until he reaches the "ready" state. But the Petri net contains so many places and transitions modeling the control conditions that we cannot immediately see whether the behavior is modeled correctly. In a visualization we would like to see only a graph of network nodes where an arc appears between two of them if a message is sent from one node to another. This can be done by defining visualizations only for the few marked places of the net which concern the sending of a message. In this case, the visualization cuts out details that have to be present in the Petri net but make it rather difficult to validate the behavioral requirements.

Example 2.3 (The Host Graph animation view for the Network model) The animation view for the network model is the *host graph*: The hosts are represented as nodes, and a message from one host to another is shown by an arc between the two respective host nodes. A host who is in the state "ready" is highlighted in a different color. Thus, the scenario in Fig. 2 can be presented in the host graph animation view as depicted in Fig. 3. Here, it is much easier to see in which order messages are sent. Also, it is easier to simulate all relevant scenarios and thus to validate the correctness of the echo algorithm. ◊

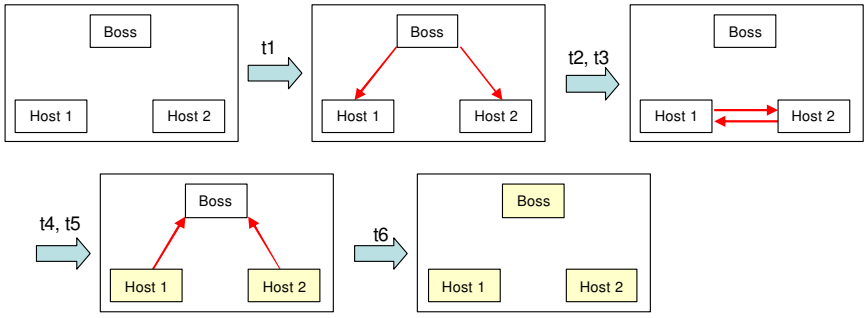

Figure 3: ***Host graph* animation view of the Network model**

For a representation of a VL model's behavior directly in an application-specific layout (e.g. the host-graph view), the graphical objects used in the VL model need to be mapped onto icons belonging to the layout of the view. It is very important that precision introduced by the VL model is carried over to the animation view in the sense that the representation of a particular requirement should not deviate from or even contradict the actual meaning of the requirements as given by the VL model.

Another issue concerned with capturing the process of visualizing dynamic behavior, is the nature of animation. Animation differs considerably from the notion of *simulation* as realized e.g. in GenGED. The network scenario depicted in Fig. 2 is an example for a simulation run. *Simulation* visualizes state changes within the means of the VL model itself. The user who validates a model sees a graph, a statechart or a Petri net, where simulation steps are carried out by switching to another graph to another marking (as in the Petri net in Fig. 2), or by highlighting another state (as in a statechart). Moreover, simulation relies on discrete steps and cannot depict changes continuously, e.g. the motion of an object. *Animation* visualizes the state changes in an animation view

which shows the model behavior using graphics from the application domain. Moreover, state changes can be depicted dynamically in the sense that the changes of graphical attributes (motion, change of size or color, ..) can be presented as continuous movies. In this paper, we advocate the integration of continuous animation in GenGED on top of the simulation features.

In the next two sections we describe how VL models including VL diagrams like in Fig. 1 and animation views like the host graph view can be defined and related such that the behavior is transferred from the model to the animation view in a coherent way. Moreover, we sketch how the simulation steps in a scenario can be enhanced in the animation view by more sophisticated animation features like e.g. continuous motion of objects.

3 The GenGED Approach

GenGED (short for Generation of Graphical Environments for Design) [3, 12] is based on the well-defined concepts of algebraic graph transformation [7].

The process for the validation of requirements using GenGED is carried out in two steps, the specification of a visual modelling language VL (described in Section 3), and the specification of a certain VL model (see Section 3). A VL model consists of a start VL diagram edited using the VL diagram editor generated from the VL specification, and a set of simulation rules. From the VL model specification, the VL simulation environment is generated allowing the user to interact visually with the VL model by generating scenarios in order to validate the VL model's properties. The two-step workflow for using the GenGED environment is illustrated in Fig. 4.

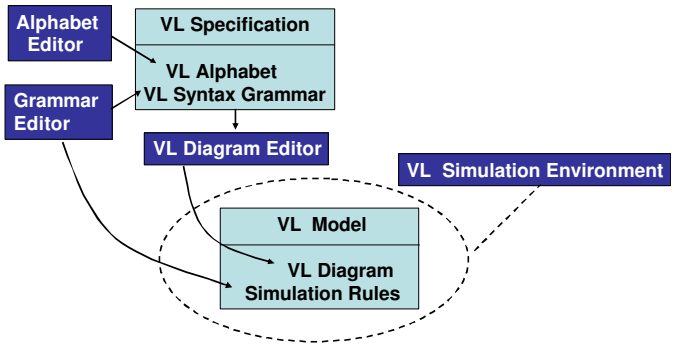

Figure 4: **The GenGED environment**

VL Specification

A VL is primarily defined by its VL alphabet (a symbol type graph and a layout description). The type graph defines the symbols and their links (abstract syntax) as well as the graphics for each symbol type (concrete syntax). Conceptually, the layout for each symbol is added to the abstract syntax by linking the symbol vertices to graphics. The layout constraints for the symbols is defined by a graphical constraint satisfaction problem Csp restricting the scope of constraint variables (position and size of graphics)

and has to be solved by an adequate variable binding in each diagram over the VL alphabet (see e.g.[3]). Moreover, a set of editing rules has to be given limiting the number of meaningful models (syntax grammar).

The VL alphabet together with the VL syntax grammar comprise the VL specification, the basis for the generation of a VL specific graphical editor for diagrams corresponding to the specified VL.

VL diagrams are instances over a VL alphabet, e.g. graphs typed over the type graph. The Csp of the VL alphabet then is used to compute a concrete configuration of valid variable bindings for the positions and sizes of graphics in a VL diagram.

Example 3.1 (VL specification of the P/T net VL)
The VL alphabet for P/T nets as shown by Figures 5 (a) in general comprise places, graphically given by circles, transitions which are visualized as rectangles, and arcs between places and transitions given by arrows. Additionally, places and transitions may be attributed by strings (their names). We omit arc inscriptions here and assume for our simple P/T net class a uniform arc weight of 1. In Figure 5, we use rectangles for the abstract syntax of lexical symbols and rounded rectangles for the abstract syntax of attribute symbols. The dashed arrows indicate the combination of the symbols' abstract syntax with their layout graphics. Some layout constraints are illustrated by dotted arrows at the concrete syntax level. For example, one constraint ensures that the place name is always written "above" the ellipse. Another constraint on token graphics ensures that the black tokens belonging to the marking of one place are always drawn "inside" that place.

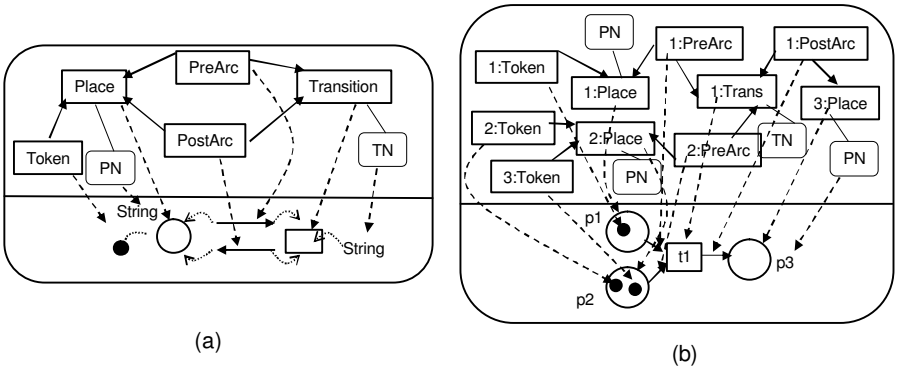

Figure 5: **(a) VL alphabet of the P/T net VL and (b) VL diagram typed over the VL alphabet**

One VL diagram according to this alphabet showing a P/T net with a marking, is depicted in Fig. 5 (b). Here, in the upper part of the figure, the nodes are instances of the respective symbol types in the type graph. ◊

The editing of a VL diagram (an instance of a VL) is realized by applying syntax rules (covering the insertion and deletion of symbols as well as the modification of symbol attributes) to the respective VL diagram in the generated visual editor. For rule applications, the graph transformation engine Agg [11, 1] is used.

Simulation Specification

The behavior of a VL model is specified by VL rules called *simulation rules* which represent transitions between system states. The VL model comprises all VL diagrams that can be generated by applying simulation rules beginning from the start diagram.

Example 3.2 (P/T net simulation)
In the case that the VL model is a P/T net over the P/T net alphabet, the behavior (the token game) can be described by simulation rules which correspond to firing the transitions of the net. More precisely, we have to ensure that

- a transition in the net is enabled if and only if the corresponding rule is applicable to the visual sentence corresponding 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 VL diagram modeling a marked Petri net. The left-hand side of each simulation rule defines the applicability condition, i.e. a subnet (a transition and the places connected to it) and a certain minimal marking (the transition's pre domain). By applying the rule, we replace an occurrence of its left-hand side by the right-hand side. The rule removes the tokens from the transition's pre domain and adds the required tokens to the places in its post domain. This approach to Petri net simulation can be applied to various types of low-level and high-level Petri nets. The simulation rule for the transition $t1$ depicted in Fig. 1 is shown in Fig. 6. The complete simulation grammar for our network

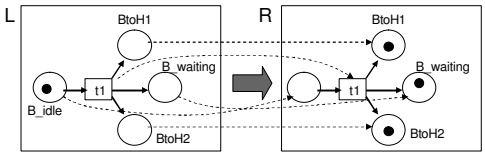

Figure 6: **Network model simulation rule for the transition** t1

model (see Fig. 1) consists of 10 simulation rules (one for each transition of the net). Simulation rules can be generated automatically for specific Petri nets. The algorithm transforms each transition in the net into an simulation rule as described above. To apply a simulation rule to a diagram of our VL model (the net depicted in Fig. 1), we have to find a mapping from the objects (nodes and edges) in L to the objects in the graph. Rule t1 is applicable to the initial state in Fig. 1 transforming the initial state into its successor state (depicted in the upper left of Fig. 2.

$\Diamond$

The application of a rule to a VL diagram resulting in a VL diagram modified according to the rule is called a derivation. In the case of a simulation grammar, the derivations are called simulation steps because the rules model the transitions from one system state to another. One derivation sequence of a VL model corresponds to a scenario. The choice of rules and rule matches and the resulting rule applications leading to such a scenario is called *simulation* of a VL model.

4 Generation of Scenario Views in GenGED

In this section, we explain how animation views are defined by an extension of the VL alphabet defined so far (called *Kernel Alphabet* from now on) to a VL alphabet for the animation view, called *View Alphabet*, and how the VL model is transformed to the animation view according to the view alphabet.

Extending the Kernel Alphabet to a View Alphabet

The extension embeds new symbol types that are visualized in a new layout but are connected to the old symbol types of the kernel alphabet to allow a coherent translation from all diagrams in the old layout to the animation view.

Example 4.1 (View alphabet for the Network model)
Fig. 7 shows the view alphabet for the host graph view of the network model, an extension of the general P/T net alphabet. Nodes for the new symbols (host, messages,

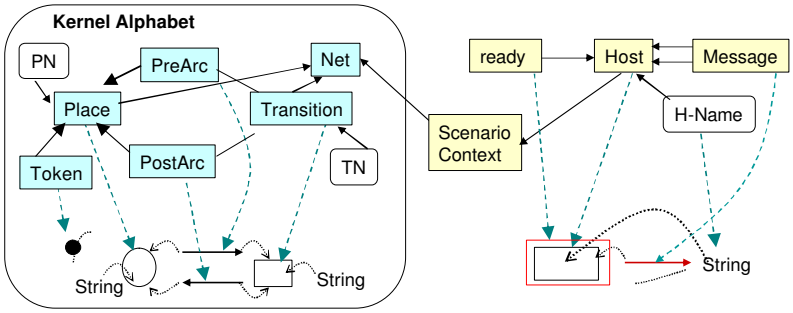

Figure 7: **View alphabet for the host graph view of the network model**

ready state) have been added and linked correspondingly. The new structures needed for the view alphabet are connected to the kernel alphabet by connecting the two root nodes *Net* and *Scenario Context*. Both are abstract nodes and are not visualized in our example. ◊

Note that the original kernel alphabet is not changed by the construction of a view alphabet; even the original layout of the old symbol types is still available.

Translating the VL Model according to a View Alphabet

As sketched in Section 2, our aim is the integration of animation operations (such as continuous motion of graphics) with the simulation features in GenGED in a way that the behavior of the VL model is carried over consistently to the animation view. Animation views should be developed systematically and be driven by the underlying behavior (formalized as simulation grammar in GenGED) for which the visualization is used. Hence, our approach is based on a formal view transformation graph grammar which is used transform the underlying VL model to its new layout in the animation view.

We define a *view transformation* based on the view alphabet. This view transformation is formalized as graph grammar whose rules are applied on the one hand to the VL diagram representing the start state of our VL model, and on the other hand to the VL model's simulation rules. The resulting transformed simulation rules are called *animation rules* and can be additionally enhanced by operations for continuous changes of objects such as motions or changes of size or color.

Example 4.2 (View Transformation Grammar for the Network Model) Fig. 8 shows the abstract syntax of the view transformation rules for the host graph view of our network model. The new layout of a VL diagram whose abstract syntax

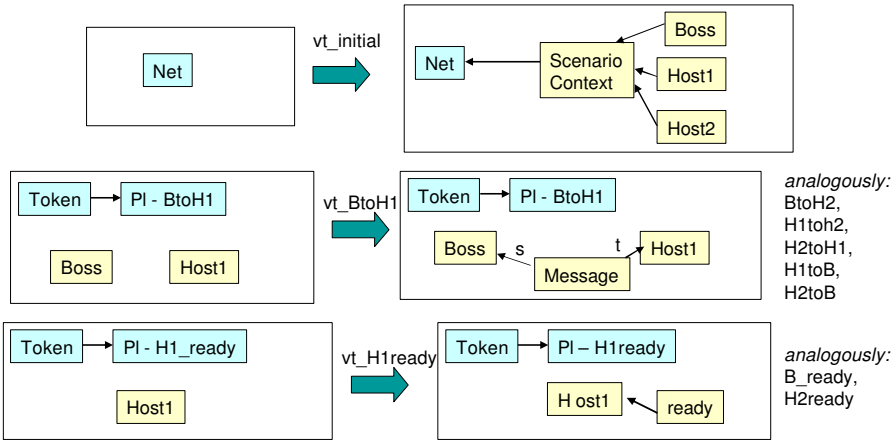

Figure 8: **View transformation grammar generating the host-graph view of the network model**

has been extended by view transformation is fixed already by the concrete syntax for the new symbols as defined in the view alphabet.

The view transformation grammar is used to transform a VL model to a animation view layout in two steps: Firstly, the start diagram of the VL model is transformed by applying the view transformation rules to it as long as any rules are applicable. Secondly, the simulation rules (see e.g. Fig. 6) are transformed into the animation view layout as well by applying the view transformation rules to the left-hand side and to the right-hand side of each simulation rule. The result of a view transformation after applying the view transformation rules to each side of the simulation rule `t1` is shown in Fig. 9.

◊

Animation of Scenarios in the Animation View

The implementation of the concepts presented so far in the GenGED environment is work in progress [9]. An animation editor allows to enrich the transformed simulation rules for the animation views by *animation operations* realizing continuous changes of graphics such as moving, appearing or disappearing, growing or shrinking or changing the color. The view designer defines these animation operations visually. More than one animation operation can be defined for one rule: a time-line diagram at the bottom

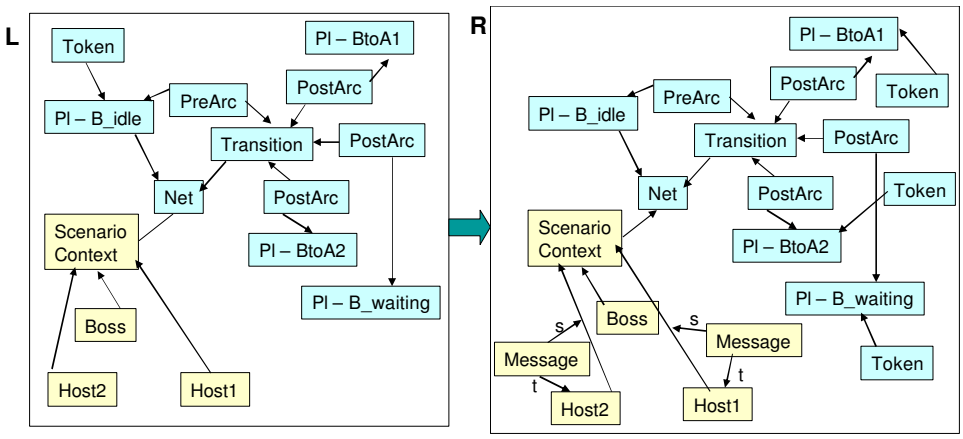

Figure 9: **A simulation rule transformed into the layout for the host-graph animation view**

of the screen shows the starting time and duration for each animation operation and allows the view designer to change them. Animation operations are executed during rule application, such that not only discrete steps from one system state to another are shown but rather a continuously animated change of the scene. Scenario animation then comprises the application of these enriched simulation rules (called animation rules) in the layout of the animation view to VL model states. Fig. 10 shows the animation editor where the upper part depicts the animation rule. Both rule sides are shown in one panel. where objects to be deleted and objects to be generated by the rule are high-lighted by different colors. In the lower part, the time line for the synchronization of starting and ending times for animation operations is shown. In our example, one linear-move operation starts half a second after the beginning of the animation step and lasts 3 seconds. The second linear-move operation starts after the first operation is finished and ends after 3 further seconds.

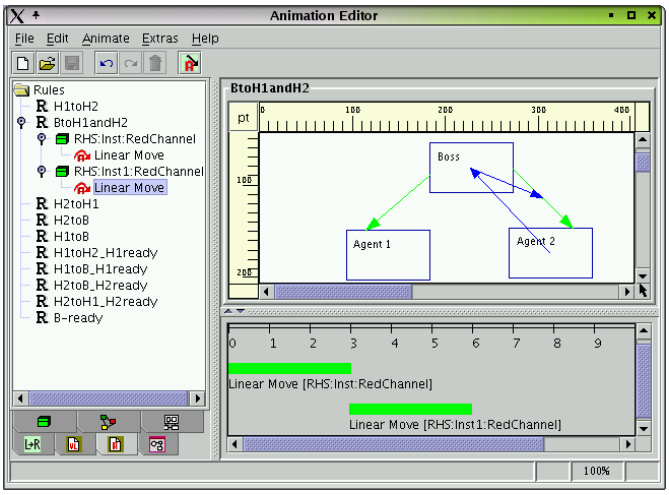

Figure 10: **The animation editor in the extended GenGED environment**

Single animation steps can be viewed in the animation environment by applying an animation rule to a VL diagram. Animation sequences can be recorded by performing

a sequence of animation rule applications. The complete animation then is stored in the XML-based SVG format (Scalable Vector Graphics [28]) and can be viewed by any external SVG viewer tool.

Summary: The Complete Methodology in GenGED

Summarizing, we presented a methodology enriching the GenGED environment by means to define animation views and their animation in a generic way. Our approach is based on graph grammars which allow flexible model transformations for various purposes. Fig. 11 presents the GenGED environment (whose basic features were depicted in Fig. 4) now extended by the animation view methodology proposed in this paper.

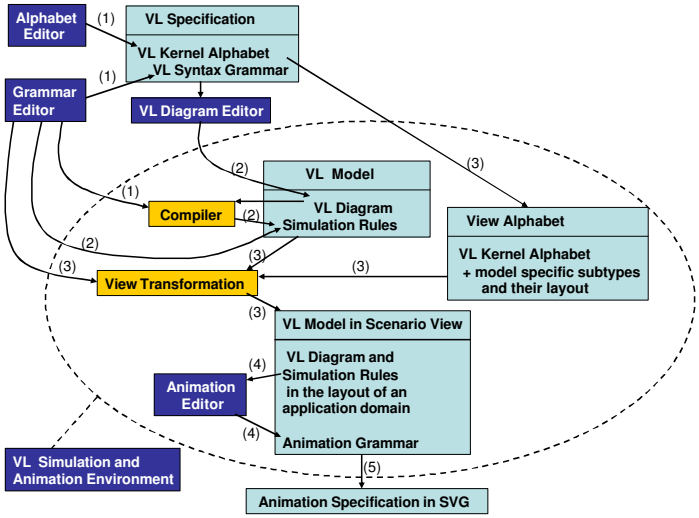

Figure 11: **The GenGED environment extended by features for animation view definition and animation**

We explain Fig. 11 by adding to the workflow different roles for users of the GenGED environment (different roles need not necessarily be taken by different persons) and describing who is doing what:

- The *language designer (1)* defines the VL Specification by using the Alphabet Editor to define the VL Kernel Alphabet and using the Grammar Editor to define the VL Syntax Grammar. Additionally (if the VL is a visual behavior modelling language), he defines the operational semantics (the Compiler in terms of a compiler grammar using again the Grammar Editor.
- The *model designer (2)* uses the VL Specification to edit a VL diagram and evokes the Compiler (an algorithm) to generate Simulation Rules from his VL Diagram. Alternatively, the Simulation Rules can be defined "by hand" using the Grammar Editor. The VL Diagram together with the Simulation Rules comprise the VL Model.
- The *view designer (3)* specifies a Scenario View by defining the View Alphabet (extending the VL Kernel Alphabet by subtypes according to the VL Model). Additionally, he defines the View Transformation Grammar over the view alphabet.

Applying the View Transformation Grammar to the VL Model, he generates a transformed VL Model in the Scenario View.

- The *animation designer (4)* uses the Animation Editor to enhance the transformed simulation rules for the animation view by animation operations and thus constructs Animation Rules for the animation view.
- the *model validator (5)* works in the VL Simulation and Animation environment by loading a VL Model (either in the original layout or in the layout of a animation view) and simulating (or animating) its behavior by applying the rules of the corresponding simulation (or animation) grammar. He can also generate animation sequences and export them to the SVG format, to be viewed by an external SVG viewer tool.

5 Conclusion

We have reasoned about the benefits of a visual environment for the employment of visual modelling techniques by discussing the GenGED approach. In general, existing tools supporting visual modelling are restricted to a fixed visual modelling language. The advantage of the GenGED approach is to support the generation of a small application specific visual modelling environment including the systematic derivation of animation views in the layout of an arbitrary application domain. This is done by means of a formal view transformation grammar, where the resulting animation rules are enhanced by attributes for e.g. continuous motion of icons.

Due to the generic and modular definition of syntax, behavior and animation for formal visual models, the presented framework reduces considerably the amount of work to realize a domain specific animation of a system's behavior. Yet, it would be even more desirable to have an interconnection between GenGED and other tools supporting the definition of VL models, e.g. the large world of Petri net or UML tools. The motives for such a tool interconnection are obvious: Petri net tools which are focussed on formal analysis of their models could profit from the animation view support offered by GenGED, whereas GenGED might export a Petri net to a Petri net tool for formal analysis. As a first step, a file exchange between GenGED and the Petri net tool infrastructure Petri Net Kernel (PNK) [21] has been realized by the implementation of an XML conversion between the XML file formats of the PNK and GenGED . Up to now, P/T nets edited with the PNK can be converted to the GenGED format and vice versa. Work is in progress to support the conversion of other Petri net classes as well. Thus, the generation of animation views in GenGED becomes possible for Petri nets which have been edited by the PNK or imported from other tools to the PNK.

Future work will be done to enhance the GenGED environment in order to model and check animation views.

As views play an important role not only for animation, we will consider the abstraction of our methodology to allow more general aspect-oriented views such as the combination of various diagram languages in UML. Adequate case studies using different visual modelling techniques will be investigated to validate the usefulness of our approach towards a rapid prototyping environment for visual modelling, simulation and

animation of animation views.

References

[1] AGG Homepage. http://tfs.cs.tu-berlin.de/agg.

[2] 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.

[3] R. Bardohl. A Visual Environment for Visual Languages. *Science of Computer Programming (SCP)*, 44(2):181–203, 2002.

[4] R. Bardohl and C. Ermel. Visual Specification and Parsing of a Statechart Variant using GenGEd. In *Statechart Modeling Contest at IEEE Symposium on Visual Languages and Formal Methods (VLFM'01)*, pages 3–5, Stresa, Italy, September 5–7 2001. http://www2.informatik.uni-erlangen.de/VLFM01/Statecharts/.

[5] 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.

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

[7] R. Bardohl, G. Taentzer, M. Minas, and A. Schürr. Application of Graph Transformation to Visual Languages. 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 105–181. World Scientific, 1999.

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

[9] K. Ehrig. Concepts and Implementation of a Generator for Animation Environments for Visual Modelling Languages (in German). Master's thesis, Technische Universität Berlin, 2003. In Preparation.

[10] C. Ermel. Generierung eines graphischen Editors für Algebraische High-Level-Netze mit GenGEd. Student's Project Status Report, 1998.

[11] 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.

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

[13] M. Glinz. Improving the Quality of Requirements with Scenarios. In *Proc. of the Second World Congress for Software Quality (2WCSQ), Yokohama*, pages 55 – 60, September 2000.

[14] E. Golin. Parsing Visual Languages with Picture Layout Grammars. *Journal of Visual Language Computing*, 2(4):216–231, 1991.

[15] Bernd Grahlmann. The State of PEP. In M. Haeberer A. editor, *Proceedings of AMAST'98 (Algebraic Methodology and Software Technology)*, volume 1548 of *Lecture Notes in Computer Science*. Springer-Verlag, January 1999.

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

[17] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, (8):231–274, 1987.

[18] D. Harel et al. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, 1990.

[19] K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN. A Reference Manual*. Meta Software Cooperation, 125 Cambridge Park Drive, Cambridge Ma 02140, USA, 1991.

[20] M. Kater. SimPEP: 3D-Visualisierung und Animation paralleler Prozesse. Diplomarbeit, Universität Hildesheim, 1998.

[21] E. Kindler and M. Weber. *The Petri Net Kernel - Documentation of the Application Interface, Revision 2.0, http://www.informatik.hu-berlin.de/top/pnk/index.html*. Forschergruppe Petrinetz-Technologie an der Humboldt-Universität zu Berlin, Januar 1999.

[22] 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.

[23] H.-J. Kreowski and A. Wilharm. Net Processes correspond to Derivation Processes in Graph Grammars. *TCS*, 44:275 – 305, 1987.

[24] K. Marriott. Constraint Multiset Grammars. In *Proc. IEEE Symposium of Visual Languages (VL'94), USA, 1994*, pages 118 – 125, 1994.

[25] W. Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.

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

[27] *Unified Modeling Language – version 1.3*, 2000. Available at http://www.omg.org/uml.

[28] WWW Consortium (W3C). *Scalable Vector Graphics (SVG) 1.0 Specification. http://www.w3.org/TR/svg*, 2000.