Electronic Communications of the EASST

ELECTRONIC COMMUNICATIONS OF THE EASST

Volume 4

# Proc. of 2nd International Workshop on Graph and Model Transformation 2006

ISSN 1863-2122

**Semantical Correctness of Simulation-to-Animation Model and Rule Transformation**

**Claudia Ermel, Hartmut Ehrig, and Karsten Ehrig**

**14 pages, 2006**

# Semantical Correctness of Simulation-to-Animation Model and Rule Transformation

**Claudia Ermel*, Hartmut Ehrig*, and Karsten Ehrig****
*Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Germany
{lieske,ehrig}@cs.tu-berlin.de
**Department of Computer Science, University of Leicester, UK
karsten@mcs.le.ac.uk

***Abstract.*** *In the framework of graph transformation, simulation rules are well-known to define the operational behavior of visual models. Moreover, it has been shown already how to construct animation rules in a domain specific layout from simulation rules. An important requirement of this construction is the semantical correctness which has not yet been considered. In this paper we give a precise definition for* simulation-to-animation *(S2A) model and rule transformations. Our main results show under which conditions semantical correctness can be obtained. The results are applied to analyze the S2A transformation of a Radio Clock model.*

**Keywords:** graph transformation, model and rule transformation, semantical correctness, simulation, animation

## 1 Introduction

In recent years, visual models represented by graphs have become very popular in model-based software development, as the wide-spread use of UML and Petri nets proves. For the definition of an operational semantics for visual models, the transformation of graphs plays a similar central role as term rewriting in the traditional case of textual models. The area of graph transformation provides a rule-based setting to express the semantics of visual models (see e.g. [Roz97]). The objective of *simulation rules* is their application to the states of a visual model, deriving subsequent model states, thus characterizing system evolution. A *simulation scenario*, i.e. a sequence of such simulation steps can be visualized by showing the states before and after each simulation rule application as graphs.

For validation purposes, simulation may be extended to a domain specific view, called *animation view* [EB04, EE05b, EHKZ05], which allows to define scenario visualizations in the layout of the application domain. The animation view is defined by extending the alphabet of the original visual modeling language by symbols representing entities from the application domain. The simulation rules for a specific visual model are translated to the animation view by performing a *simulation-to-animation model and rule transformation* ($S2A$ transformation), realizing a consistent mapping from simulation steps to animation steps which can be visualized in the animation view layout. $S2A$ transformation is defined by a set of $S2A$ graph transformation rules, and an additional formal construction allowing to apply $S2A$ rules to simulation rules, resulting in a new set of graph transformation rules, called *animation rules*.

Comparable theoretical research in the area of applying graph transformation rules to rules has been done by Parisi-Presicce [PP96]. His approach has provided the basis of our definition of $S2A$ transformations which additionally allows to transform not only the rule interfaces, and which also treats negative application conditions (NACs), both for the transforming rules and for the transformed rules.

An important requirement is the *semantical correctness* of the $S2A$ transformation in the sense that the behavior of the original model is preserved in the animation view. In this paper, we give a formal definition for $S2A$ transformations and show under which conditions semantical correctness can be obtained. In our approach, an $S2A$ transformation generates one animation rule for each simulation rule. Hence, our notion of semantical correctness implies that each animation step (obtained by applying an animation rule) corresponds to a simulation step of the original model. Please note that there are more general definitions for the semantical correctness of model transformations which establish a correspondence between one simulation step in the source model and a sequence of simulation steps in the target model. For $S2A$ transformation it is sufficient to relate single simulation and animation steps. Intermediate animation states providing smooth state transitions are possible nonetheless: They are defined by enriching an animation rule by animation operations to specify continuous changes of object properties. Since animation operations leave the states before and after a rule application unchanged, they do not influence the semantical correctness of $S2A$ transformation. Our approach has been implemented in the generic visual modeling environment GENGED [Gen]. The implementation includes an animation editor to define animation operations visually, and to export animation scenarios to the SVG format [WWW03].

There exist related tool-oriented approaches, where different visual representations are used to visualize a model's behavior. One example is the *reactive animation* approach by Harel [HEC03], where behavior is specified by UML diagrams. The animated representation of the system behavior is implemented by linking UML tools to pure animation tools like Macromedia FLASH or DIRECTOR [Mac04]. Hence, the mapping from simulation to animation views happens at the implementation level and is not specified formally. Analogously, different Petri net tools also offer support for customized Petri net animations (e.g. the SimPEP tool [Gra99] to animate transition firings of low-level Petri nets). In general, approaches to enhance the front end of CASE tools for simulating/animating the behavior of models are restricted to one specific modeling language. In our approach we integrate animation views at model level with graph transformation representations for different visual modeling languages based on a formal specification. This provides the model designer with more flexibility, as the modeling language to be enhanced by animation features, can be freely chosen.

The paper is organized as follows: Section 2 presents the basic concepts of simulation and animation, illustrated by our case study in Section 3. In Section 4, the main result on semantical correctness of $S2A$ transformation is given in the case without NACs. Extensions to cope with NACs are discussed. Explicit proofs for the case with NACs, and the semantical correctness of the complete case study is presented in the technical report [EEE06]. Section 5 discusses related work, and Section 6 concludes the paper.

## 2 Basic Concepts of Simulation and Animation

We use typed algebraic graph transformation systems (TGTS) in the double-pushout-approach (DPO) [EEPT06] which have proven to be an adequate formalism for visual language (VL) modeling. A VL is modeled by a type graph capturing the definition of the underlying visual alphabet, i.e. the symbols and relations which are available. Sentences or diagrams of the VL are given by graphs typed over the type graph. We distinguish abstract and concrete syntax in alphabets and models, where the concrete syntax includes the abstract symbols and relations, and additionally defines their layout. Formally, a VL can be considered as a subclass of graphs typed over a type graph $TG$ in the category $\mathbf{Graphs_{TG}}$.

For behavioral diagrams like Statecharts, an operational semantics can be given by a set of simulation rules $P_S$, using the abstract syntax of the modeling VL. A simulation rule $p = (L \leftarrow I \rightarrow R) \in P_S$ is a graph transformation rule, consisting of a left-hand side $L$, an interface $I$, a right-hand side $R$, and two injective morphisms. Applying rule $p$ to a graph $G$ means to find a match of $L \xrightarrow{m} G$ and to replace the occurrence $m(L)$ of $L$ in $G$ by $R$ leading to the target graph $G'$. In the DPO approach, the deletion of $m(L)$ and the addition of $R$ are described by two pushouts (a DPO) in the category $\mathbf{Graphs_{TG}}$ of typed graphs. A rule $p$ may be extended by a set of *negative application conditions (NACs)* [EEPT06], describing situations in which the rule should not be applied to $G$. Formally, match $L \xrightarrow{m} G$ satisfies NAC $L \xrightarrow{n} N$ if there does not exist an injective graph morphism $N \xrightarrow{x} G$ with $x \circ n = m$. A sequence $G_0 \Rightarrow G_1 \Rightarrow ... \Rightarrow G_n$ of graph transformation steps is called *transformation* and denoted as $G_0 \overset{*}{\Rightarrow} G_n$. A transformation $G_0 \overset{*}{\Rightarrow} G_n$, where rules from $P$ are applied as long as possible (i.e. as long as matches can be found satisfying the NACs), is denoted by $G_0 \overset{P\ !}{\Longrightarrow} G_n$.

We define a model's *simulation language* $VL_S$, typed over the simulation alphabet $TG_S$, as a sublanguage of the modeling language $VL$, such that all diagrams $G_S \in VL_S$ represent different states during simulation. Based on $VL_S$, the operational semantics of a model is given by a *simulation specification*.

**Definition 2.1** (*Simulation Specification*)
Given a visual language $VL_S$ typed over $TG_S$, i.e. $VL_S \subseteq \mathbf{Graphs_{TG_S}}$, a *simulation specification* $SimSpec_{VL_S} = (VL_S, P_S)$ over $VL_S$ is given by a TGTS $(TG_S, P_S)$ s.t. $VL_S$ is closed under simulation steps, i.e. $G_S \in VL_S$ and $G_S \Rightarrow H_S$ via $p_S \in P_S$ implies $H_S \in VL_S$. The rules $p_S \in P_S$ are called simulation rules. △

In order to transform a simulation specification to an animation view, we define an $S2A$ transformation $S2A = (S2AM, S2AR)$ consisting of a simulation-to-animation model transformation $S2AM$, and a corresponding rule transformation $S2AR$. The $S2AM$ transformation applies $S2A$ transformation rules from a rule set $Q$ to each $G_S \in VL_S$ as long as possible, adding symbols from the application domain to the model state graphs. The resulting set of graphs comprises the animation language $VL_A$.

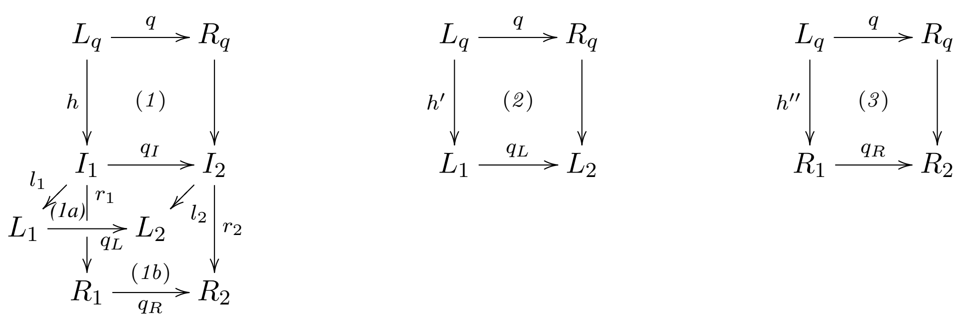
**Definition 2.2** (*$S2AM$-Transformation*)
Given a simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$ with $VL_S$ typed over $TG_S$ and a type graph $TG_A$, called animation type graph, with $TG_S \subseteq TG_A$, a *simulation-to-animation model transformation*, short $S2AM$-transformation,

$$S2AM : VL_S \rightarrow VL_A$$

is given by $S2AM = (VL_S, TG_A, Q)$ where $(TG_A, Q)$ is a TGTS with non-deleting rules $q \in Q$, and $S2AM$-transformations $G_S \overset{Q\ !}{\Longrightarrow} G_A$ with $G_S \in VL_S$. The *animation language* $VL_A$ is defined by $VL_A = \{G_A | \exists\, G_S \in VL_S \;\&\; G_S \overset{Q\ !}{\Longrightarrow} G_A\}$. This means $G_S \overset{Q\ !}{\Longrightarrow} G_A$ implies $G_S \in VL_S$ and $G_A \in VL_A$, where each intermediate step $G_i \overset{q_i}{\Longrightarrow} G_{i+1}$ is called *S2AM-step*. △

Our aim is not only to transform model states but to obtain a complete animation specification, including animation rules, from the simulation specification. Hence, we define a construction allowing us to apply the *S2A* transformation rules from $Q$ also to the simulation rules. The following definition extends the construction for rewriting rules by rules given by Parisi-Presicce in [PP96], where a rule $q$ is only applicable to a rule $p$ if it is applicable to the interface graph of $p$. In this paper, we want to add animation symbols to simulation rules even if the *S2A* transformation rule is *not* applicable to the interface of the simulation rule. Hence, we distinguish three cases in Def. 2.3. Case (1) corresponds to the notion of rule rewriting in [PP96], adapted to non-deleting *S2A* transformation rules. In Case (2), the *S2A* transformation rule $q$ is not applicable to the interface, but only to the left-hand side of a rule $p$, and in Case (3), $q$ is only applicable to the right-hand side of $p$.

**Definition 2.3** (*Transformation of Rules by Non-Deleting Rules*)
Given a non-deleting rule $q = (L_q \rightarrow R_q)$ and a rule $p_1 = (L_1 \overset{l_1}{\leftarrow} I_1 \overset{r_1}{\rightarrow} R_1)$, then $q$ is applicable to $p_1$ leading to a *rule transformation step* $p_1 \overset{q}{\Longrightarrow} p_2$ , if the precondition of one of the following three cases is satisfied, and $p_2 = (L_2 \overset{l_2}{\leftarrow} I_2 \overset{r_2}{\rightarrow} R_2)$ is defined according to the corresponding construction.




***Case (1)***
*Precondition (1)*: There is a match $L_q \overset{h}{\longrightarrow} I_1$.
*Construction (1)*: $I_2$, $L_2$, and $R_2$ are defined by pushouts $(1)$, $(1a)$ and $(1b)$, leading to injective morphisms $l_2$ and $r_2$.

***Case (2)***
*Precondition (2)*: There is no match $L_q \overset{h}{\longrightarrow} I_1$, but a match $L_q \overset{h'}{\longrightarrow} L_1$.
*Construction (2)*: $L_2$ is defined by pushout $(2)$, and $I_2 = I_1$, $R_2 = R_1$, $r_2 = r_1$, and $l_2 = q_L \circ l_1$.

***Case (3)***
*Precondition (3)*: There are no matches $L_q \overset{h}{\longrightarrow} I_1$ and $L_q \overset{h'}{\longrightarrow} L_1$, but there is a match $L_q \overset{h''}{\longrightarrow} R_1$.
*Construction (3)*: $R_2$ is defined by pushout $(3)$, and $L_2 = L_1$, $I_2 = I_1$, $l_2 = l_1$, and $r_2 = q_L \circ r_1$. △

The transformation of rules defined above allows now to define an $S2AR$ transformation of rules, leading to an $S2A$ transformation $S2A = (S2AM, S2AR)$ from the simulation specification $SimSpec_{VL_S}$ to the animation specification $AnimSpec_{VL_A}$.

**Definition 2.4** ($S2AR$-*Transformation*)
Given a simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$ and an $S2AM$-transformation $S2AM = (VL_S, TG_A, Q)$ then a *simulation-to-animation rule transformation*, short *$S2AR$-trafo*,

$$S2AR : P_S \to P_A,$$

is given by $S2AR = (P_S, TG_A, Q)$ and *$S2AR$ transformation sequence* $p_S \overset{Q\ !}{\Longrightarrow} p_A$ with $p_S \in P_S$, where rule transformation steps $p_1 \overset{q}{\Longrightarrow} p_2$ with $q \in Q$ (see Def. 2.3) are applied as long as possible. The *animation rules* $P_A$ are defined by $P_A = \{p_A | \exists\, p_S \in P_S \wedge p_S \overset{Q\ !}{\Longrightarrow} p_A \}$.

This means $p_S \overset{Q\ !}{\Longrightarrow} p_A$ implies $p_S \in P_S$ and $p_A \in P_A$, where each intermediate step $p_i \overset{q_i}{\Longrightarrow} p_{i+1}$ is called *$S2AR$-step*.

△

**Definition 2.5** (*Animation Specification and $S2A$ Transformation*)
Given a simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$, an $S2AM$ transformation $S2AM : VL_S \to VL_A$ and an $S2AR$ transformation $S2AR : P_S \to P_A$, then

1. $AnimSpec_{VL_A} = (VL_A, P_A)$ is called *animation specification*, and each transformation step $G_A \overset{p_A}{\Longrightarrow} H_A$ with $G_A, H_A \in VL_A$ and $p_A \in P_A$ is called *animation step*.
2. $S2A : SimSpec_{VL_S} \to AnimSpec_{VL_A}$, defined by $S2A = (S2AM, S2AR)$ is called *simulation-to-animation model and rule transformation*, short $S2A$ transformation.

△

## 3 Case Study: Radio Clock

In this section, we illustrate the main concepts of Section 2 by the well-known Radio Clock case study from Harel [Har87]. The behavior of a radio clock is modeled by the nested Statechart shown in Fig. 1 (a). The radio clock display can show alternatively the time, the date or allows to set the alarm time. The changes between the modes are modeled by transitions labeled with the event Mode. The nested state Alarm allows to change to modes for setting the hours and the minutes (transition Select) of the alarm time. A Set event increments the number of hours or minutes which are currently displayed.

A domain-specific animation view of the Radio Clock is illustrated in Fig. 1 (b). The two snapshots from a possible simulation run of the Statechart in Fig. 1 (a) correspond to the active state Set:Hours before and after the set event has been processed. The animation view shows directly the current display of the clock and indicates by a red light that in the current state the hours may be set. Furthermore,

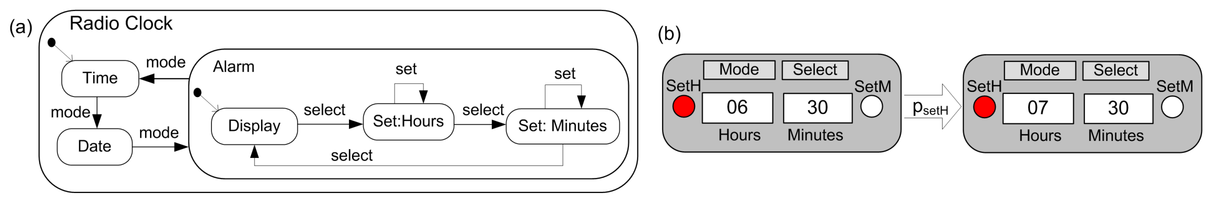



Figure 1: Radio Clock Statechart (a), and Animation View Snapshots (b)

buttons are shown either to proceed to the state where the minutes may be set (button Select), or to switch back to the Time display (button Mode).

The abstract syntax graph of the Radio Clock Statechart is the given by the graph $G_I$ in Fig. 2.

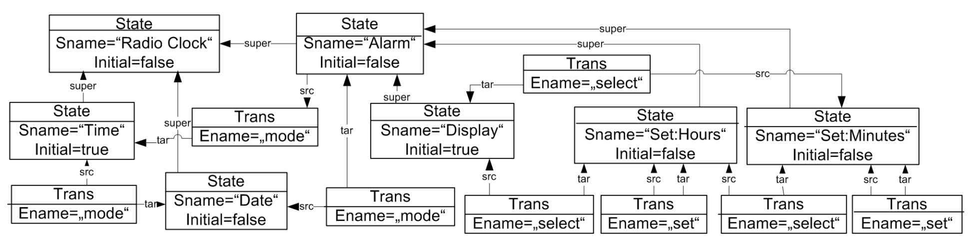



Figure 2: Abstract Syntax Graph $G_I$ of the Radio Clock Statechart

The set of model-specific simulation rules $P_S = \{p_{addObject}, p_{addEvent}, p_{downTime}, p_{downDisp}, p_{upAlarm}, p_{upClock}, p_{mode_{TD}}, p_{mode_{DA}}, p_{mode_{AD}}, p_{selectH}, p_{selectM}, p_{selectD}, p_{setH}, p_{setM}\}$ to be applied to $G_I$ contains initialization rules which generate the object node with attribute values for the initial alarm time, set the current pointer to the top level state Radio Clock, and fill the event queue. Additional simulation rules are defined which realize the actual simulation, processing the events in the queue. For each superstate there is a rule moving the current pointer from the superstate down to its initial substate. Analogously, there are rules moving the pointer from a substate to its superstate. For each transition there is a rule which moves the pointer from the source state of the transition to its target state and removes the triggering event from the queue. The full set $P_S$ of simulation rules is given in [EEE06]. Fig. 3 shows the sample simulation rule $p_{setH}$ for the transition set whose source and target is the state Set:Hours. In addition to processing the event set, this rule increments the hour value of the current alarm time.

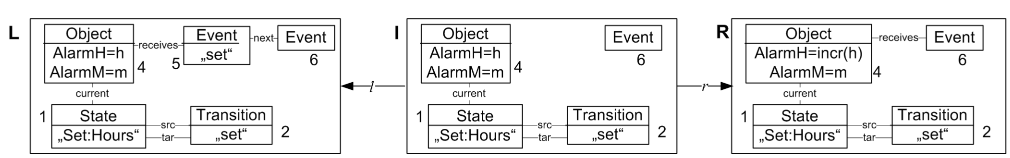



Figure 3: A Simulation Rule $p_{setH}$

The simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$ for the Radio Clock consists of the simulation language $VL_S$ typed over $TG_S$, where $TG_S$ is the simulation alphabet depicted in the left-hand side of

Fig. 4, $P_S$ is the set of simulation rules, and $VL_S$ consists of all graphs that can occur in any Radio Clock simulation scenario: $VL_S = \{G_S | \exists G_I \overset{P_S*}{\Longrightarrow} G_S\}$, where $G_I$ is the initial graph shown in Fig. 2.

Fig. 4 shows the animation view type graph $TG_A$, which is a disjoint union of the simulation alphabet $TG_S$, and the new visualization alphabet $TG_V$ shown in the right part of Fig. 4, which models the visualization symbols for a domain-specific view of the radio clock behavior. The three modes of the

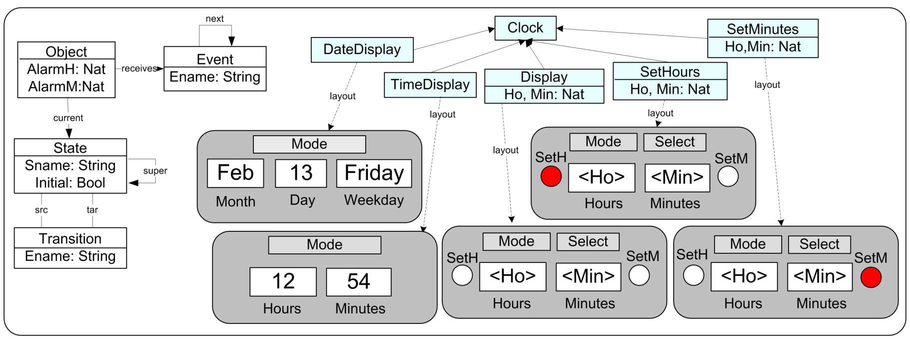



Figure 4: Simulation and Animation Alphabet

clock are visualized by five different displays: a date display, a time display, and three alarm displays showing the alarm time but differing in the states of two red lights which indicate the states Display (both lights off), Set:Hours (light SetH on), and Set:Minutes (light SetM on).

The $S2A$ transformation rules $Q = \{q_{Clock}, q_{Date}, q_{Time}, q_{Disp}, q_{SetH}, q_{SetM}\}$ add visualization symbols to the simulation rule graphs and to the initial radio clock graph. The initial $S2A$ rule $q_{Clock}$ adds the root symbol Clock to all graphs it is applied to. The remaining $S2A$ rules add visualization symbols depending on the state of the current pointer. We visualize only basic states which do not have any substates. Superstates are not shown in the animation view, as they are considered as transient, abstract states which are active on the way of the current pointer up and down the state hierarchy between two basic states, but have no concrete layout themselves.

The full set $Q$ of $S2A$ rules is given in [EEE06]. The top row of Fig. 5 shows the sample $S2A$ transformation rule $q_{setH}$ which adds a SetHours symbol and links it to the clock symbol in the case that the current pointer points to the state named "Set:Hours". The attributes are set accordingly. Note that each $S2A$ rule $q$ has to be applied at most once at the same match, which is formalized by a NAC $L_q \rightarrow N_q$, such that $N_q$ and $R_q$ are isomorphic. The Radio Clock $S2AM$ transformation $S2AM : VL_S \rightarrow VL_A$ is given by $S2AM = (VL_S, TG_A, Q)$ with animation language $VL_A = \{G_A | \exists G_S \in VL_S : G_S \overset{Q\ !}{\Longrightarrow} G_A\}$. The Radio Clock $S2AR$ transformation $S2AR : P_S \rightarrow P_A$ is given by $S2AR = (P_S, TG_A, Q)$ with animation rules $P_A = \{p_A | \exists p_S \in P_S : \ p_S \overset{Q\ !}{\Longrightarrow} p_A \}$.

A sample $S2AR$ transformation step $p'_{setH} \overset{q_{setH}}{\Longrightarrow} p^A_{setH}$ is shown in Fig. 5. Here, $S2A$ rule $L_q \overset{q_{setH}}{\longrightarrow} R_q$ is applied to the rule $p'_{setH}$, according to Case (1) of Def. 2.3. Rule $p'_{setH} = (L' \leftarrow I' \rightarrow R')$ in Fig. 5 corresponds to rule $p_1 = (L_1 \leftarrow I_1 \rightarrow R_1)$ in Def. 2.3. The result of the rule rewriting step in Fig. 5 is rule $p^A_{setH} = (L_A \leftarrow I_A \rightarrow R_A)$, which corresponds to rule $p_2 = (L_2 \leftarrow I_2 \rightarrow R_2)$ in Def. 2.3. Note that variables for node attributes can be assigned to other variables or to expressions. For

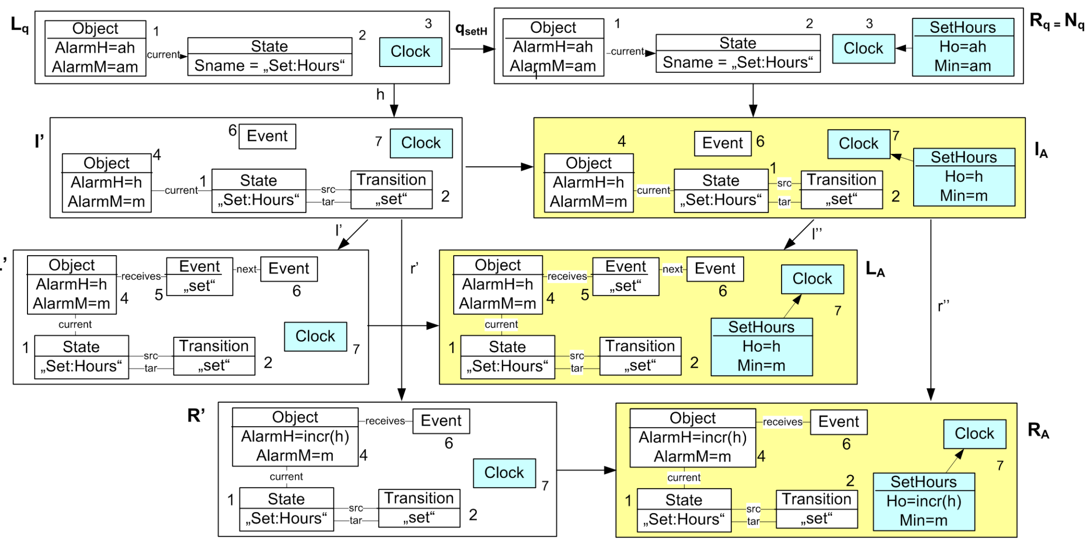



Figure 5: $S2A$ Transformation Step $p'_{setH} \xrightarrow{q_{setH}} p^A_{setH}$

instance, in Fig. 5, the variable h for attribute AlarmH in $I'$ is assigned to the expression incr(h) in $R'$ by the morphism $I' \xrightarrow{r'} R'$. Hence, a resulting animation rule can contain variables or expressions for attributes to be assigned to corresponding attribute values in graphs when the animation rule is applied. $p^A_{setH}$ is a completely transformed animation rule, since no more $S2A$ rules are applicable to it.

The Radio Clock animation specification $AnimSpec_{VL_A} = (VL_A, P_A)$ based on the $S2A$ transformation $S2A = (S2AM, S2AR)$ is given by the animation language $VL_A$, obtained by the Radio Clock $S2AM$ transformation, and the animation rules $P_A$, obtained by the Radio Clock $S2AR$ transformation. The full set $P_A$ of animation rules is given in [EEE06].

Fig. 6 shows a sample animation scenario in the concrete notation of the animation view, where animation rules from $P_A$ are applied. The first state of the scenario in Fig. 6 is obtained by applying the initial animation rules setting the alarm time and initializing the event queue with the events mode, mode, select, set, mode. The subsequent animation steps result from applying animation rules for event processing or for moving up and down the state hierarchy.

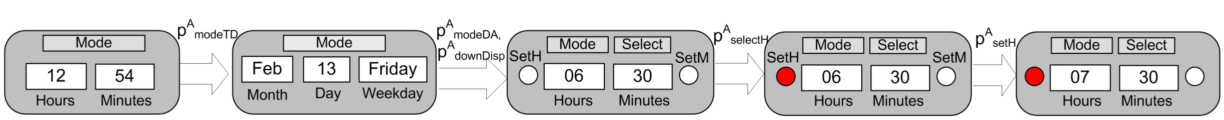



Figure 6: Animation Scenario

# 4 Semantical Correctness of $S2A$ Transformations

In this section, we continue the general theory of Section 2 and study semantical correctness of $S2A$-transformations. In our case, semantical correctness of an $S2A$-transformation means that for each simulation step $G_S \stackrel{p_S}{\Longrightarrow} H_S$ there is a corresponding animation step $G_A \stackrel{p_A}{\Longrightarrow} H_A$ where $G_A$ (resp. $H_A$) are obtained by $S2A$ model transformation from $G_S$ (resp. $H_S$), and $p_A$ by $S2A$ rule transformation from $p_S$. Note that this is a special case of semantical correctness defined in [EE05a], where instead of a single step $G_A \stackrel{p_A}{\Longrightarrow} H_A$ more general sequences $G_A \stackrel{*}{\Longrightarrow} H_A$ and $H_S \stackrel{*}{\Longrightarrow} H_A$ are allowed.

**Definition 4.1** (*Semantical Correctness of $S2A$ Transformations*)
An $S2A$-transformation $S2A : SimSpec_{VL_S} \rightarrow AnimSpec_{VL_A}$ given by $S2A = (S2AM : VL_S \rightarrow VL_A, S2AR : P_S \rightarrow P_A)$ is called *semantically correct*, if for each simulation step $G_S \stackrel{p_S}{\Longrightarrow} H_S$ with $G_S \in VL_S$ and each $S2AR$-transformation sequence $p_S \stackrel{Q\ !}{\Longrightarrow} p_A$ (see Def. 2.4) we have

1. $S2AM$-transformation sequences $G_S \stackrel{Q\ !}{\Longrightarrow} G_A$ and $H_S \stackrel{Q\ !}{\Longrightarrow} H_A$, and
2. an animation step $G_A \stackrel{p_A}{\Longrightarrow} H_A$

$$\begin{array}{ccc} G_S & \stackrel{Q\ !}{\Longrightarrow} & G_A \\ \Big\Downarrow p_S & \stackrel{Q\ !}{\Longrightarrow} & p_A \Big\Downarrow \\ H_S & \stackrel{Q\ !}{\Longrightarrow} & H_A \end{array}$$

△

Before we prove semantical correctness in Theorem 4.4, we first show local semantical correctness in Theorem 4.2 where only one $S2AM$-step (resp. $S2AR$-step) is considered.

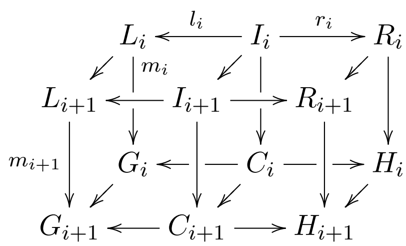
**Theorem 4.2** (*Local Semantical Correctness of $S2A$-Transformations*)
Given an $S2A$-transformation $S2A : SimSpec_{VL_S} \rightarrow AnimSpecVL_A$ with $S2A = (S2AM : VL_S \rightarrow VL_A, S2AR : P_S \rightarrow P_A)$ and an $S2AR$-transformation sequence $p_S \stackrel{Q\ !}{\Longrightarrow} p_A$ with intermediate $S2AR$-step $p_i \stackrel{q}{\Longrightarrow} p_{i+1}$ with $q \in Q$. Then for each graph transformation step $G_i \stackrel{p_i}{\Longrightarrow} H_i$ with $G_i, H_i \in \mathbf{Graphs_{TG_A}}$ we have

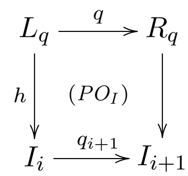
1. Graph transformation steps $G_i \stackrel{q_i}{\Longrightarrow} G_{i+1}$ in Cases (1) and (2), $G_i \stackrel{id}{\Longrightarrow} G_{i+1}$ in Case (3), $H_i \stackrel{q}{\Longrightarrow} H_{i+1}$ in Cases (1) and (3), and $H_i \stackrel{id}{\Longrightarrow} H_{i+1}$ in Case (2) of Def. 2.3.
2. Graph transformation step $G_{i+1} \stackrel{p_{i+1}}{\Longrightarrow} H_{i+1}$ with $G_{i+1}, H_{i+1} \in \mathbf{Graphs_{TG_A}}$

$$\begin{array}{ccc} G_i & \stackrel{q\ /\ id}{\Longrightarrow} & G_{i+1} \\ \Big\Downarrow p_i & \stackrel{q}{\Longrightarrow} & p_{i+1} \Big\Downarrow \\ H_i & \stackrel{q\ /\ id}{\Longrightarrow} & H_{i+1} \end{array}$$

△

**Proof:** We consider the respective pushout diagrams for $p_i \stackrel{q}{\Longrightarrow} p_{i+1}$ according to the three rule transformation cases in Def. 2.3, and show by pushout composition/decomposition that in each case we obtain the commuting double cube below where the two back squares comprise the given DPO for the transformation step $G_i \stackrel{p_i}{\Longrightarrow} H_i$, and in the front squares we get the required DPO for the transformation step $G_{i+1} \stackrel{p_{i+1}}{\Longrightarrow} H_{i+1}$. In Case (1) of Def. 2.3, we obtain the top squares as pushouts and then construct $G_{i+1}$, $C_{i+1}$, $H_{i+1}$ as pushouts in the diagonal squares, leading to unique induced morphisms $C_{i+1} \rightarrow G_{i+1}$ and $C_{i+1} \rightarrow H_{i+1}$ s.t. the double cube commutes. By pushout composition/decomposition also the front and the bottom squares are pushouts. Furthermore, we obtain pushouts for the transformation steps $G_i \stackrel{q}{\Longrightarrow} G_{i+1}$ and $H_i \stackrel{q}{\Longrightarrow} H_{i+1}$ by composing pushout $(PO_I)$ below with the respective pushouts from the double cube. Cases (2) and (3) are handled similarly, with the difference that some morphisms in the respective double cubes are identities.




□

The following notions are used for proving the main Theorem 4.4.

**Definition 4.3** (*Termination of $S2AM$ and Rule Compatibility of $S2A$*)
An $S2AM$ transformation $S2AM : VL_S \rightarrow VL_A$ is *terminating* if each transformation $G_S \stackrel{Q\ *}{\Longrightarrow} G_n$ can be extended to $G_S \stackrel{Q\ *}{\Longrightarrow} G_n \stackrel{*}{\Longrightarrow} G_m$ such that no $q \in Q$ is applicable to $G_m$ anymore.
An $S2A$-transformation $S2A = (S2AM : VL_S \rightarrow VL_A, S2AR : P_S \rightarrow P_A)$ with $S2AM = (VL_S, TG_A, Q)$ is called *rule compatible*, if for all $p_A \in P_A$ and $q \in Q$ we have that $p_A$ and $q$ are parallel and sequential independent.
More precisely for each $G \stackrel{p_A}{\Longrightarrow} H$ with $G_S \stackrel{Q\ *}{\Longrightarrow} G$ and $H_S \stackrel{Q\ *}{\Longrightarrow} H$ for some $G_S, H_S \in VL_S$ and each $G \stackrel{q}{\Longrightarrow} G'$ (resp. $H \stackrel{q}{\Longrightarrow} H'$) we have parallel (resp. sequential) independence of $G \stackrel{p_A}{\Longrightarrow} H$ and $G \stackrel{q}{\Longrightarrow} G'$ (resp. $H \stackrel{q}{\Longrightarrow} H'$). △

**Theorem 4.4** (*Semantical Correctness of $S2A$*)
Each $S2A$ transformation $S2A = (S2AM, S2AR)$ is semantically correct, provided that $S2A$ is rule compatible, and $S2AM$ is terminating. △

**Proof:**
Given $S2A = (S2AM : VL_S \rightarrow VL_A, S2AR : P_S \rightarrow P_A)$ with terminating $S2AM = (VL_S, TG_A, Q)$, a simulation step $G_S \stackrel{p_S}{\Longrightarrow} H_S$ with $G_S \in VL_S$, and an $S2AR$ transformation sequence $p_S \stackrel{Q\ !}{\Longrightarrow} p_A$ with $p_S = p_0 \stackrel{q_0}{\Longrightarrow} p_1 \stackrel{q_1}{\Longrightarrow} .. \stackrel{q_{n-1}}{\Longrightarrow} p_n = p_A$ with $n \geq 1$, then we can apply the Local Semantical Correctness Theorem 4.2 for $i = 0, .., n-1$, leading to the diagram below, which includes the case $n = 0$ with $G_S = G_0, H_S = H_0$ and $p_S = p_0 = p_A$, where no $q \in Q$ can be applied to $p_S = p_0 = p_A$.

$$
\begin{array}{ccccccccccc}
G_S = G_0 & \overset{q_0}{\Longrightarrow} & G_1 & \overset{q_1}{\Longrightarrow} & G_2 & \overset{q_2}{\Longrightarrow} & \cdots & \Longrightarrow & G_{n-1} & \overset{q_{n-1}}{\Longrightarrow} & G_n \\
\Big\Downarrow p_S{=}p_0 & & \Big\Downarrow p_1 & & \Big\Downarrow p_2 & & \overset{Q!}{\Longrightarrow} & & \Big\Downarrow p_{n-1} & & \Big\Downarrow p_n{=}p_A \\
H_S = H_0 & \underset{q_0}{\Longrightarrow} & H_1 & \underset{q_1}{\Longrightarrow} & H_2 & \underset{q_2}{\Longrightarrow} & \cdots & \Longrightarrow & H_{n-1} & \underset{q_{n-1}}{\Longrightarrow} & H_n
\end{array}
$$

If no $q \in Q$ can be applied to $G_n$ and $H_n$ anymore, we are ready, because the top sequence is $G_S \overset{Q\ !}{\Longrightarrow} G_n = G_A$, and the bottom sequence is $H_S \overset{Q\ !}{\Longrightarrow} H_n = H_A$.

Now assume that we have $q_n \in Q$ which is applicable to $G_n$ leading to $G_n \overset{q_n}{\Longrightarrow} G_{n+1}$. Then, rule compatibility implies parallel independence with $G_A \overset{p_A}{\Longrightarrow} H_A$, and the Local Church Rosser Theorem [EEPT06] leads to square $(n)$:

$$
\begin{array}{ccccccccc}
G_n & \overset{q_n}{\Longrightarrow} & G_{n+1} & \Longrightarrow \cdots \Longrightarrow & G_{m-1} & \underset{q_{m-1}}{\Longrightarrow} & G_m & = & G_A \\
\Big\Downarrow p_A & (n) & \Big\Downarrow p_A & & \Big\Downarrow p_A & & \Big\Downarrow p_A & & \\
H_n & \overset{q_n}{\Longrightarrow} & H_{n+1} & \Longrightarrow \cdots \Longrightarrow & H_{m-1} & \overset{q_{m-1}}{\Longrightarrow} & H_m & = & H_A
\end{array}
$$

This procedure can be repeated as long as rules $q_i \in Q$ are applicable to $G_i$ for $i \geq n$. Since $S2AM$ is terminating, we have some $m > n$ such that no $q \in Q$ is applicable to $G_m$ anymore, leading to a sequence $G_S = G_0 \overset{Q\ !}{\Longrightarrow} G_m = G_A$. Now assume that there is some $q \in Q$ which is still applicable to $H_m$ leading to $H_m \overset{q}{\Longrightarrow} H_{m+1}$. Now rule compatibility implies sequential independence of $G_m \overset{p_A}{\Longrightarrow} H_m \overset{q}{\Longrightarrow} H_{m+1}$. In this case, the Local Church Rosser Theorem would lead to a sequence $G_m \overset{q}{\Longrightarrow} G_{m+1} \overset{p_A}{\Longrightarrow} H_{m+1}$ which contradicts the fact that no $q \in Q$ is applicable to $G_m$ anymore. This implies that also $H_0 \overset{Q\ *}{\Longrightarrow} H_n \overset{Q\ *}{\Longrightarrow} H_m$ is terminating, leading to the required sequence $H_S = H_0 \overset{Q\ !}{\Longrightarrow} H_m = H_A$. $\square$

**Extension by Negative Application Conditions**

Considering rules with NACs both for the $S2A$ rules in $Q$ (now of the form $q = (N_q \leftarrow L_q \rightarrow R_q)$), and for the simulation rules in $P_S$ (now of the form $p_S = (N_i \leftarrow L \leftarrow I \rightarrow R)$), has the following consequences on the construction of the animation specification by $S2A$ transformation: Def. 2.3 has to be extended to deal with the additional transformation of NACs in Cases (1) and (2) (in Case (3), the NACs remain unchanged). Moreover, a new Case (4) has to be added covering the case that preconditions (1) - (3) are not satisfied, but there are matches into $N_i$. Furthermore, the preconditions for all cases now also require the satisfaction of $NAC_q = (L_q \overset{n}{\longrightarrow} N_q)$. To extend rule compatibility (Def. 4.3), in addition to parallel and sequential independence in the case without NACs, we have to require that the induced matches satisfy the corresponding NACs. The proof of local semantical correctness of $S2A$ transformations with NACs requires also *NAC-compatibility* of $S2AM$ and $S2AR$ for all $q \in Q$ and $G_i \overset{p_i}{\Longrightarrow} H_i$. NAC-compatibility of $S2AM$ means that if $q$ is applicable to a rule $p_S$, then each match of

$q$ in $G_i$ (resp. $H_i$) satisfies $NAC_q$. NAC-compatibility of $S2AR$ means that if $p_i \stackrel{q}{\Longrightarrow} p_{i+1}$ satisfies $NAC_q$, and $G_i \stackrel{p_i}{\Longrightarrow} H_i$ satisfies $NAC(p_i)$ then $G_{i+1} \stackrel{p_{i+1}}{\Longrightarrow} H_{i+1}$ satisfies $NAC(p_{i+1})$.

Considering these additional requirements, we can show that each $S2A$-transformation $S2A = (S2AM, S2AR)$ is semantically correct including $NACs$, provided that $S2A$ is rule compatible, $S2AM$ is terminating and $S2A$ is $NAC$-compatible. This extends Theorem 4.4, where now rule compatibility and termination have to be required with NACs (for the complete extended theorem see [EEE06, Erm06]). Using the extended theorem, we show the semantical correctness of our Radio Clock case study in [EEE06]. Termination is shown to be fulfilled for general $S2A$ transformation systems with suitable rule layers and applied to our case study in [EEE06]. Moreover, it is shown that each $S2AR$ transformation is NAC-compatible provided that we have suitable rule layers as in our case study. Thus, it suffices to show only NAC-compatibility of $S2AM$ explicitly for the Radio Clock.

# 5 Related Work

To ensure the correctness of model transformations, Varrò et al. [SV03, Var04] use graph transformation rules to specify the dynamic behavior of systems and generate a transition system for each instance model. Based on the transition system, a model checker verifies certain dynamic consistency properties by model checking the source and target models. In [NK06], a method is presented to verify the semantical equivalence for particular model transformations. It is shown by finding bisimulations that a target model preserves the semantics of the source model with respect to a particular property. This technique does not prove the correctness of the model transformation rules in general, as we propose in this paper for the restricted case of $S2A$ transformation rules. The formal background of bisimulations for graph transformations has been considered also in e.g. [EK04].

For the specification of model transformations, *triple graph grammars* [Sch94] have been frequently used. These grammars are based on a coupling of the syntax rules for the source and target language, which allows derivations in the source language to be translated into derivations of the target language. A third grammar in between source and target produces a mapping structure to keep track of the relation between the source and target structures. Triple graph grammars have been recently used also to model tool integration [KS06] and the integration of multiple views on a system [GDdL05]. Here, views are (possibly overlapping) parts of a global alphabet, and graph triples are made of one repository (the complete integrated model), one view and an intermediate graph that relates objects of both. The triple graph grammar specifies the gluing of the views in the repository. This approach has similarities to our approach concerning the relation of simulation and animation alphabets. But the restriction to subtypes of a VL type graph alone is usually not enough to define views which abstract from model details. Given a type graph for Petri nets, for example, it would not be possible to define a view which shows only the markings of particular states and hides the others. In this respect, our approach of $S2A$ transformation is much more flexible. Moreover, our notion of $S2AR$ transformation allows to relate views with behavior.

The animation specification resulting from an $S2A$ transformation provides a good basis for user interaction when defining scenarios in the animation view (e.g. by clicking on a radio clock button to apply an animation rule). Here lies the central advantage of coding the animation view information into the rules instead of translating directly simulation steps into animation steps (as realized e.g. in [HEC03]).

## 6 Conclusion and Ongoing Work

In this paper we have given a precise definition for simulation-to-animation ($S2A$) model and rule transformations. The main results show under which conditions an $S2A$ transformation $S2A : SimSpec_{VL_S} \to AnimSpec_{VL_A}$ is semantically correct in the cases without and with negative application conditions. The results have been used to show semantical correctness of a radio clock case study.

For simplicity, the theory has been presented in the DPO-approach for typed graphs, but it can also be extended to typed attributed graphs, where injective graph morphisms are replaced by suitable classes $M$ and $M'$ of typed attributed graph morphisms for rules and NACs, respectively [EEPT06]. Non DPO-based approaches have not yet been considered.

In addition to analyzing the semantical correctness of $S2A$, it may be interesting to construct also a backward model and rule transformation $A2S : AnimSpec_{VL_A} \to SimSpec_{VL_S}$, essentially given by restriction of all graphs and rules to the type graph $TG_S$. Semantical correctness of $A2S$ means that for each animation step $G_A \overset{p_A}{\Longrightarrow} H_A$ there is also a corresponding simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ using the restrictions $G_S, H_S$ and $p_S$ of $G_A, H_A$ and $p_A$, respectively. Finally, we can consider semantical equivalence of $SimSpec_{VL_S}$ and $AnimSpec_{VL_A}$, which requires existence and semantical correctness of $S2A$ and $A2S$, such that both are inverse to each other, i.e. $A2S \circ S2A = Id$ and $S2A \circ A2S = Id$.

## References

[EB04] C. Ermel and R. Bardohl. Scenario Animation for Visual Behavior Models: A Generic Approach. *Software and System Modeling: Special Section on Graph Transformations and Visual Modeling Techniques*, 3(2):164–177, 2004.

[EE05a] H. Ehrig and K. Ehrig. Overview of Formal Concepts for Model Transformations based on Typed Attributed Graph Transformation. In *Proc. Intern. Workshop on Graph and Model Transformation (GraMoT'05)*, ENTCS, Elsevier Science, 2005.

[EE05b] C. Ermel and K. Ehrig. View transformation in visual environments applied to Petri nets. In G. Rozenberg, H. Ehrig, and J. Padberg, editors, *Proc. Workshop on Petri Nets and Graph Transformation (PNGT)*, volume 127(2) of *ENTCS*, pages 61–86. Elsevier Science, 2005.

[EEE06] C. Ermel, H. Ehrig, and K. Ehrig. Semantical Correctness of Simulation-to-Animation Model and Rule Transformation: Long Version. Technical Report 2006/10, TU Berlin. http://iv.tu-berlin.de/TechnBerichte/2006-10.pdf.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[EHKZ05] C. Ermel, K. Hölscher, S. Kuske, and P. Ziemann. Animated Simulation of Integrated UML Behavioral Models based on Graph Transformation. In M. Erwig and A. Schürr, editors, *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, IEEE Computer Society, 2005.

[EK04] H. Ehrig and B. Koenig. Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting. In *Proc. FOSSACS 2004*, volume 2987 of *LNCS*, pages 151–166. Springer, 2004.

[Erm06] C. Ermel. *Simulation and Animation of Visual Languages based on Typed Algebraic Graph Transformation*. PhD thesis, Technische Universität Berlin, Fak. IV, 2006.

[GDdL05] E. Guerra, P. Diaz, and J. de Lara. A Formal Approach to the Generation of Visual Language Environments Supporting Multiple Views. In *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, IEEE Computer Society, 2005.

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

[Gra99] B. Grahlmann. The State of PEP. In M. Haeberer A. editor, *Proc. of AMAST'98 (Algebraic Methodology and Software Technology)*, Vol. 1548 of *LNCS*. Springer, 1999.

[Har87] D. Harel. Statecharts: a Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[HEC03] D. Harel, S. Efroni, and I.R. Cohen. Reactive Animation. In F.S. de Boer, M.M. Bonsangue, and S. Graf et al., editors, *Proc. Int. Symposium on Formal Methods for Components and Objects (FMCO'02)*, Vol. 2852 of *LNCS*, pages 136–153. Springer, 2003.

[KS06] A. Königs and A. Schürr. Tool Integration with Triple Graph Grammars - A Survey. In *Proc. SegraVis School on Foundations of Visual Modelling Techniques*. ENTCS Vol. 148, Elsevier Science, 2006.

[Mac04] Macromedia, Inc. *Flash MX 2004 and Director MX 2004*, http://www.macromedia.com/software/.

[NK06] A. Narayanan and G. Karsai. Towards Verifying ModelTransformations. In *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'06)*. ENTCS, Elsevier Science, 2006.

[PP96] F. Parisi-Presicce. Transformation of Graph Grammars. In *5th Int. Workshop on Graph Grammars and their Application to Computer Science*, Vol. 1073 of *LNCS*. Springer, 1996.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[Sch94] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *Workshop on Graph-Theoretic Concepts in Computer Science*, Vol. 903 of *LNCS*, pages 151–163, Springer, 1994.

[SV03] Ákos Schmidt and Dániel Varró. CheckVML: A Tool for Model Checking Visual Modeling Languages. In P. Stevens, J. Whittle, and G. Booch, editors, *Proc. Unified Modeling Language, Modeling Languages and Applications (UML'03)*, Vol. 2863 of *LNCS*, pages 92–95. Springer, 2003.

[Var04] Dániel Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, 2004.

[WWW03] WWW Consortium (W3C). *Scalable Vector Graphics (SVG) 1.1 Specification.*, 2003. http://www.w3.org/TR/svg11/.