Behavior-Preserving Simulation-to-Animation Model and Rule Transformations

Claudia Ermel[1] and Hartmut Ehrig[2]

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin,
Germany

Abstract

In the framework of graph transformation, simulation rules define the operational behavior of visual models. Moreover, it has been shown already how to construct animation rules from simulation rules by so-called *S2A*-transformation. In contrast to simulation rules, animation rules use symbols representing entities from the application domain in a user-oriented visualization. Using animation views for model execution provides better insights of model behavior to users, leading to an earlier detection of model inconsistencies. Hence, an important requirement of the animation view construction is the preservation of the behavior of the original visual model. This means, we have to show on the one hand semantical correctness of the *S2A*-transformation, and, on the other hand, semantical correctness of a suitable backwards-transformation *A2S*. Semantical correctness of a model and rule transformation means that for each sequence of the source system we find a corresponding sequence in the target system. *S2A*-transformation has been considered in our contribution to GraMoT 2006. In this paper, we give a precise definition for *animation-to-simulation* (*A2S*) backward transformation, and show under which conditions semantical correctness of an *A2S* backward transformation can be obtained. The main result states the conditions for *S2A*-transformations to be behavior-preserving. The result is applied to analyze the behavior of a Radio Clock model's *S2A*-transformation.

Keywords: graph transformation, model and rule transformation, semantical correctness, simulation, animation, behavior-preserving transformation

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. [3]). The objective of *simulation rules* (graph transformation rules for simulation) is their application to the states of a visual model, deriving subsequent model states, thus characterizing

[1] Email: lieske@cs.tu-berlin.de
[2] Email: ehrig@cs.tu-berlin.de

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

system evolution. A *simulation scenario*, i.e. a sequence of simulation steps, can be visualized by showing the states before and after each rule application as graphs.

For validation purposes, simulation may be extended to a domain specific view, called *animation view* [8], which allows one to define scenario visualizations which are closer to the application domain than the abstract, graph-based model. Such an 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 so-called *animation rules* conforming 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. This visualization of animation steps in the animation view is called *animation*, in contrast to *simulation*, where simulation steps are shown as changes of the underlying abstract graph model.

S2A transformation is defined by a set of graph transformation rules, called *S2A* rules, and an additional formal construction allowing for applying *S2A* rules to simulation rules in order to obtain *animation rules*, which define the model behavior in the animation view. An important requirement of *S2A* transformation is that the behavior of the model is preserved in the animation view to ensure that validation results can be conferred to the original model. This means, on the one hand, semantical correctness of the *S2A* transformation, and, on the other hand, semantical correctness of a suitable backward transformation *A2S*. Semantical correctness of *S2A* means that for each simulation sequence of the model we find a corresponding animation sequence in the animation view, and has been considered in [6]. Semantical correctness of *A2S* means that for each animation sequence in the animation view we find a corresponding simulation sequence in the original model.

In this paper, we give a precise definition for *animation-to-simulation* (*A2S*) model and rule backward transformation, and show under which conditions semantical correctness of *A2S* backward transformation can be obtained, thus giving criteria for *S2A*-transformations to be behavior-preserving. In our approach, an *S2A* transformation generates one animation step for each simulation step, and the corresponding *A2S* transformation relates each animation step to a simulation step. 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 [1]. 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 continual changes of object properties. Since animation operations leave the states before and after a rule application unchanged, they do not influence behavior-preserving *S2A* transformations. Our approach has been implemented in the generic visual modeling environment GenGED [5,9]. The implementation includes an animation editor to define animation operations visually, and to export animation scenarios to the SVG format [18].

The paper is organized as follows: In Section 2, our running example, an animation view for a Radio Clock Statechart, is introduced. Section 3 reviews the basic concepts of simulation, animation, and model and rule transformation. In Section 4,

the main result on semantical correctness of *S2A* transformation is reviewed. As new contribution in this paper, it is shown that for each *S2A* transformation there exists a corresponding *A2S* backward-transformation. Semantical correctness of *A2S* transformations is shown for the case without negative application conditions (NACs). Extensions to cope with NACs are discussed. Section 5 discusses related work, and Section 6 concludes the paper.

2 Case Study: Radio Clock

In this section, we illustrate the concepts simulation and animation along the well-known Radio Clock case study from Harel [10]. The behavior of a radio clock is modeled by the nested Statechart shown in Fig. 1.

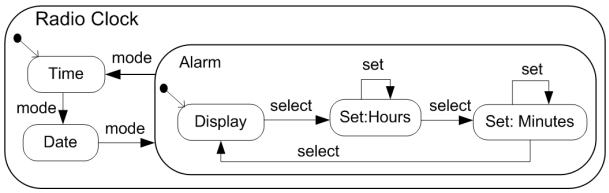


Fig. 1. Radio Clock Statechart

The radio clock display can show alternatively the time, the date or the alarm time. The changes between the modes are modeled by transitions labeled with the event mode. The nested state Alarm allows one 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.

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

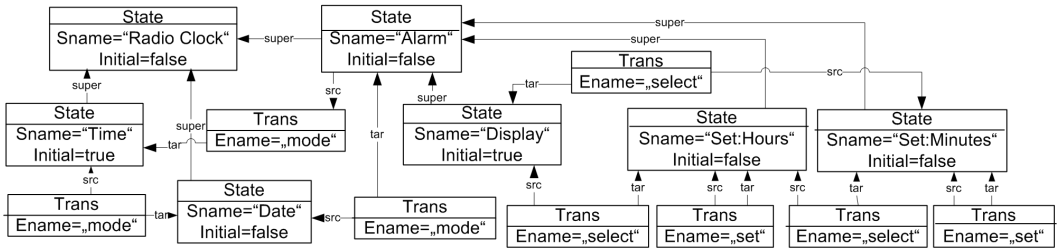


Fig. 2. Abstract Syntax Graph G_I of the Radio Clock Statechart

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 an object node with initial attribute values, 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 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 transitions's source state to its target state and removes the triggering event from the queue. The full set P_S of simulation rules is given in [7]. 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 alarm time.

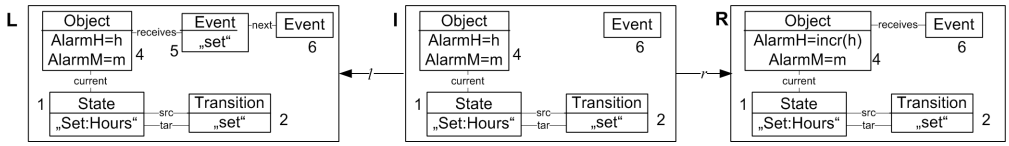


Fig. 3. A Simulation Rule p_{setH}

A domain-specific animation view of the Radio Clock is illustrated in Fig. 4. The two snapshots from a possible simulation run of the Statechart in Fig. 1 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, 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).

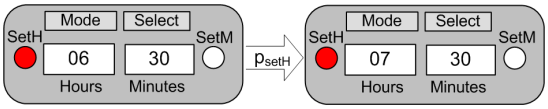


Fig. 4. Animation View Snapshots for the Radio Clock

3 Basic Concepts of Simulation and Animation

We use typed algebraic graph transformation systems (TGTS) in the double-pushout-approach (DPO) [3] 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 graphics for their visualization. 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, defined by simulation type graph TG_S. A simulation rule $p = (L \leftarrow I \rightarrow R) \in P_S$ is a TG_S-typed 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'. Such a graph transformation step is denoted by $G \overset{(p,m)}{\Longrightarrow} G'$, or simply by $G \Rightarrow G'$. 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)* [3], 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 regard 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 of the same model during simulation. Based on VL_S, the operational semantics of a model is given by a *simulation specification*.

Definition 3.1 (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) such that 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*.

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. 5, 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.

In order to visualize the model behavior, an animation view type graph TG_A is defined, which is a disjoint union of the simulation alphabet TG_S and the new visualization alphabet TG_V. Fig. 5 shows the animation view type graph TG_A for the Radio Clock, where TG_V consists of visualization symbols for a domain-specific view of the radio clock modes. The abstract syntax symbols of TG_V are connected to their concrete representation graphics by *layout* arcs. The graphics are part of the type graph, but they are not needed in the animation rules since layout arcs express a 1-to-1-correspondence between abstract symbols and their graphics.

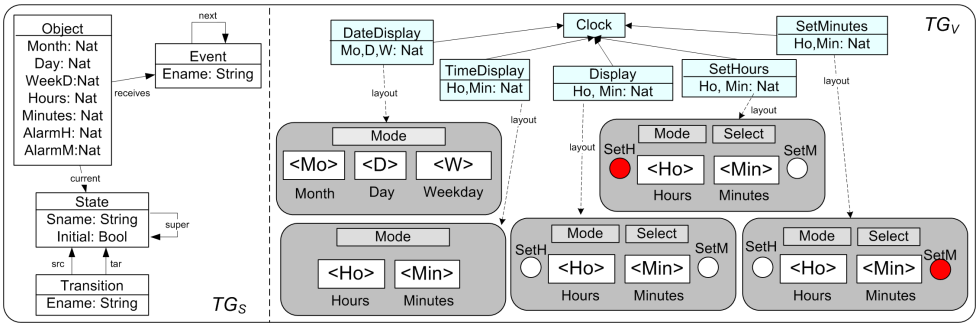


Fig. 5. Simulation and Animation Type Graphs for the Radio Clock

Three radio clock modes 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). A state in the Statechart corresponds to a display in the animation view. Thus, during animation, the display for the current active state is shown and displays the corresponding attribute values of the object pointer's attributes.

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.

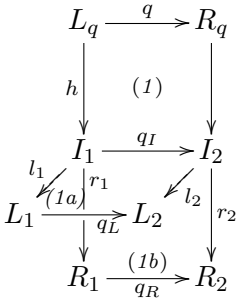
Definition 3.2 (*S2AM*-Transformation) Given a simulation specification $SimSpec_{VL_S} = (VL_S, P_S)$ with VL_S typed over TG_S and a type graph TG_A, called animation type graph, with $TG_S \subseteq TG_A$, a *simulation-to-animation model transformation*, short *S2AM*-transformation, $S2AM : VL_S \to VL_A$ 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 $VL_A = \{G_A |\ \exists\ G_S \in VL_S\ \wedge\ G_S \overset{Q\ !}{\Longrightarrow} G_A\}$. 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, resulting in a set of animation rules. The following definition reviews the construction for rewriting rules by rules from [6].

Definition 3.3 (Transformation of Rules by Non-Deleting Rules) Given a non-deleting rule $q = (L_q \to R_q)$ and a rule $p_1 = (L_1 \overset{l_1}{\leftarrow} I_1 \overset{r_1}{\to} R_1)$, 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 $p_2 = (L_2 \overset{l_2}{\leftarrow} I_2 \overset{r_2}{\to} R_2)$ is defined according to the corresponding construction.

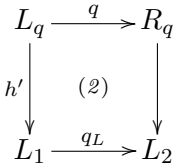
Case (1)

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.



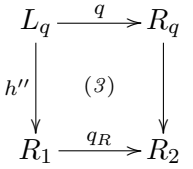
Case (2)

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 $r_2 = q_L \circ r_1$.



Def. 3.3 extends the construction for rewriting rules by rules given by Parisi-Presicce in [14], where a rule q is only applicable to a rule p if it is applicable to the interface graph I of p. This means, q cannot be applied if p deletes or generates objects which q needs. 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: Case (1) in Def. 3.3 corresponds to the notion of rule rewriting in [14], adapted to non-deleting *S2A* transformation rules. In Case (2), the *S2A* transformation rule q is not applicable to the interface I, but to the left-hand side of a rule p_1, and in Case (3), q is not applicable to I, but to the right-hand side of p_1. Note that it is possible that both Case (2) and Case (3) can be true for different matches of q. Then, q is applied in a first step to L_1 according to (2), and in a second step to R_1 according to (3).

Def. 3.3 now allows us 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 3.4 (*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. 3.3) 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*.

In our Radio Clock example, 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 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 visualization graphics themselves.

The full set Q of *S2A* rules is given in [7]. The top row of Fig. 6 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 $L_q \to N_q$, such that N_q and R_q are isomorphic. A sample *S2AR* transformation step $p'_{setH} \overset{q_{setH}}{\Longrightarrow} p^A_{setH}$ is shown in Fig. 6. Here, *S2A* rule $L_q \overset{q_{setH}}{\longrightarrow} R_q$ is applied to the rule p'_{setH}, according to Case (1) of Def. 3.3. Rule $p'_{setH} = (L' \leftarrow I' \rightarrow R')$ in Fig. 6 corresponds to rule $p_1 = (L_1 \leftarrow I_1 \rightarrow R_1)$ in Def. 3.3. The result of the rule rewriting step in Fig. 6 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. 3.3. Rule p^A_{setH} is a completely transformed animation rule, since no more *S2A*

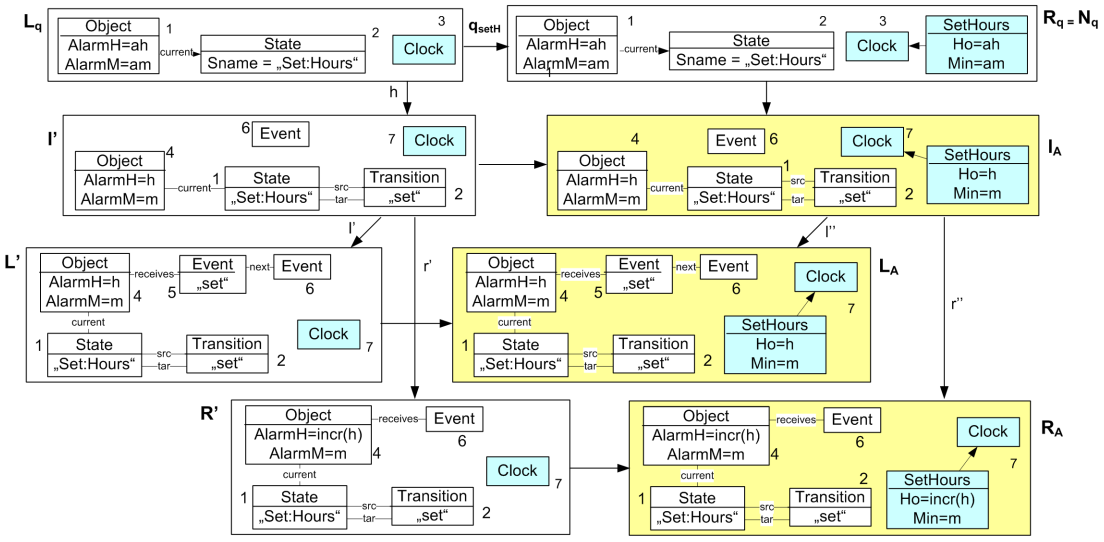


Fig. 6. *S2A* Transformation Step $p'_{setH} \overset{q_{setH}}{\Longrightarrow} p^A_{setH}$

rules are applicable to it. Note that variables for node attributes can be assigned to other variables or to expressions. For instance, in Fig. 6, 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.

Definition 3.5 (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

(i) $AnimSpec_{VL_A} = (VL_A, P_A)$ 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*.

(ii) $S2A : SimSpec_{VL_S} \to AnimSpec_{VL_A}$, defined by $S2A = (S2AM, S2AR)$ is called *simulation-to-animation model and rule transformation*, short *S2A* transformation.

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 [7].

Fig. 7 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. 7 is obtained by applying the initial animation rules setting the attribute values, initializing the event queue with the events mode, mode, select, set, mode, and processing the first mode event. The subsequent animation steps result from applying animation rules for processing the remaining events or for moving up and down the state hierarchy.

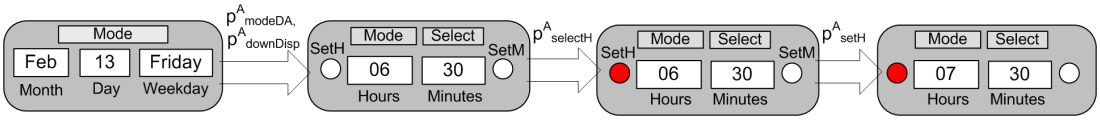


Fig. 7. Radio Clock Animation Scenario

4 Behavior-Preserving *S2A* Transformations

In this section, we continue the general theory of Section 3 and study properties of behavior-preserving *S2A*-transformations, i.e. *S2A*-transformations which are semantically correct and where a semantically correct *A2S*-backward-transformation exists. After reviewing semantical correctness of *S2A* transformation (which has been treated in depth in [6]), we define the construction of an *A2S*-backward-transformation for a given *S2A*-transformation, and give requirements for the semantical correctness of *A2S*. The main result in Theorem 4.14 states the conditions for *S2A* transformations being behavior-preserving.

4.1 Semantical correctness of S2A-transformations

In our case, semantical correctness of an *S2A*-transformation means that for each simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ there is a corresponding animation step $G_A \overset{p_A}{\Longrightarrow} H_A$ 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 [1], where instead of a single step $G_A \overset{p_A}{\Longrightarrow} H_A$ more general sequences $G_A \overset{*}{\Longrightarrow} H_A$ and $H_S \overset{*}{\Longrightarrow} H_A$ are allowed.

Definition 4.1 (Semantical Correctness of *S2A* Transformations)
An *S2A*-transformation $S2A : SimSpec_{VL_S} \to AnimSpec_{VL_A}$ given by $S2A = (S2AM : VL_S \to VL_A, S2AR : P_S \to P_A)$ is *semantically correct*, if we have for each simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ with $G_S \in VL_S$ and each *S2AR*-transformation sequence $p_S \overset{Q\ !}{\Longrightarrow} p_A$ (see Def. 3.4):

(i) *S2AM*-transformation sequences $G_S \overset{Q\ !}{\Longrightarrow} G_A$ and $H_S \overset{Q\ !}{\Longrightarrow} H_A$, and

(ii) an animation step $G_A \overset{p_A}{\Longrightarrow} H_A$

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

In [6,7], it is shown that the following properties have to be fulfilled by an *S2A*-transformation in order to be semantically correct:

Definition 4.2 (Termination of *S2AM* and Rule Compatibility of *S2A*)
An *S2AM* transformation $S2AM : VL_S \to VL_A$ is *terminating* if each transformation $G_S \overset{Q\ *}{\Longrightarrow} G_n$ can be extended to $G_S \overset{Q\ *}{\Longrightarrow} G_n \overset{Q\ *}{\Longrightarrow} G_m$ such that no $q \in Q$ is applicable to G_m anymore. An *S2A*-transformation $S2A = (S2AM : VL_S \to VL_A, S2AR : P_S \to P_A)$ 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 $G \overset{p_A}{\Longrightarrow} H$ with $G_S \overset{Q\ *}{\Longrightarrow} G$ and $H_S \overset{Q\ *}{\Longrightarrow} H$ for some G_S, $H_S \in VL_S$ and each $G \overset{q}{\Longrightarrow} G'$ (resp. $H \overset{q}{\Longrightarrow} H'$) we have parallel (resp. sequential) independence of $G \overset{p_A}{\Longrightarrow} H$ and $G \overset{q}{\Longrightarrow} G'$ (resp. $H \overset{q}{\Longrightarrow} H'$).

Without giving the proof (which can be found in [6]), Theorem 4.3 states the main result from [6], concerning semantical correctness of *S2A*-transformation.

Theorem 4.3 (Semantical Correctness of *S2A*)
Each S2A-transformation $S2A = (S2AM, S2AR)$ *is semantically correct, provided that S2A is rule compatible, and S2AM is terminating.*

4.2 Construction of A2S-Backward-Transformations

In this section we consider the relation between an animation specification $AnimSpec_{VL_A}$ and the corresponding simulation specification $SimuSpec_{VL_S}$ related by *S2A* transformation. We show in Theorem 4.10 that for each *S2A* transformation there is a backward transformation $A2S : AnimSpec_{VL_A} \to SimSpec_{VL_S}$, i.e. we get $A2S \circ S2A \subseteq Id_{VL_S}$

Definition 4.4 (Characterization of Backward Transformations)

(i) Given an *S2AM* transformation $S2AM : VL_S \to VL_A$, then a transformation $A2SM : VL_A \to VL_S$ is called *backward transformation of S2AM* if we have

$$A2SM \circ S2AM \subseteq Id_{VL_S},$$

i.e. $\forall G_S, G'_S \in VL_S, G_A \in VL_A : [(G_S, G_A) \in S2AM, (G'_S, G_A) \in S2AM \Longrightarrow G_S = G'_S]$

(ii) Given an *S2AR* transformation $S2AR : P_S \to P_A$, then the transformtion $A2SR : P_A \to P_S$ is called *backward transformation of S2AR* if we have

$$A2SR \circ S2AR \subseteq Id_{P_S}.$$

(iii) Given backward transformations *A2SM* of *S2AM* and *A2SR* of *S2AR*, then $A2S = (A2SM, A2SR)$ is called *backward transformation of* $S2A = (S2AM, S2AR)$.

Remark 4.5 All transformations in Def. 4.4 are considered as relations, and $\circ$ is the relational composition. If *S2AM* is total, we also require *A2SM* to be total and $A2SM \circ S2AM = Id_{VL_S}$, and analogously for *S2AR* and *A2SR*.

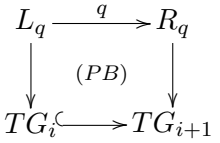
For an *S2A* transformation, we define an *A2S* backward transformation by restriction of graphs and rules to TG_S in Def. 4.6. and show in Theorem 4.10 (using the propositions Prop. 4.7 and 4.8) that *A2S* has the desired property $A2S \circ S2A \subseteq Id_{VL_S}$. If *S2AM* is total, we even get $A2S \circ S2A = Id_{VL_S}$.

Definition 4.6 (*A2S* Transformation) Given an *S2A* transformation $S2A = (S2AM : VL_S \to VL_A, S2AR : P_S \to P_A) : SimSpec_{VL_S} \to AnimSpec_{VL_A}$, then the transformation $A2S : AnimSpec_{VL_A} \to SimSpec_{VL_S}$ is defined by $A2S = (A2SM, A2SR)$ is called *animation-to-simulation model and rule transformation*, short *A2S* transformation, where

(i) $A2SM : VL_A \to VL_S$ is the *animation-to-simulation model transformation*, short *A2SM* transformation, defined by restriction to TG_S, i.e. $A2SM(G_A) = G_A|_{TG_S}$, and

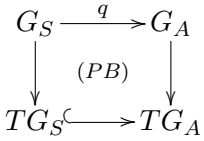
(ii) $A2SR : P_A \to P_S$ is the *animation-to-simulation rule transformation*, short *A2SR* transformation, defined by restriction to TG_S, i.e. $A2SR(p_A) = p_A|_{TG_S}$.

In the subsequent propositions Prop. 4.7 and 4.8, we use the notion of *layered type-increasing TGTS* to denote a typed graph transformation system with rule layers, where elements generated by a rule $q \in Q$ belonging to rule layer i generate only elements typed over new types in TG_{i+1} which do not occur already in L_q, such that $R_q|_{TG_i} = L_q$, i.e. the diagram to the right is a pullback for all $q \in Q$.

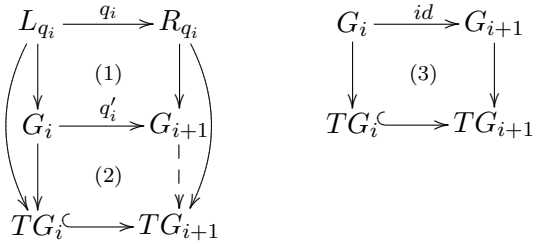


This property allows us to construct a parallel rule q_i from all rules q belonging to rule layer i, such that for $q = q_i$ we also have the pullback (PB).

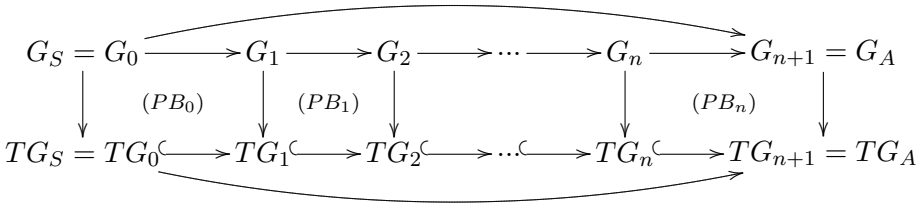
Proposition 4.7 (Restriction of *S2AM* to TG_S)
Given an S2AM transformation $S2AM : VL_S \to VL_A$ based on a layered type-increasing TGTS (TG_A, Q) with $TG_S \subseteq TG_A$, then we have: $G_S \overset{Q\ !}{\Longrightarrow} G_A$ with $G_S \in VL_S$ implies $G_A|_{TG_S} = G_S$, i.e. the diagram to the right is a pullback.



Proof. Given $G_S \overset{Q\ !}{\Longrightarrow} G_A$, we can assume to have a sequence $G_S = G_0 \overset{q_0}{\Longrightarrow} G_1 \overset{q_1}{\Longrightarrow} .. \overset{q_n}{\Longrightarrow} G_{n+1} = G_A$ where each q_i is either a parallel rule, composed of all $q \in Q$ with rule layer i, or an identity step. In each single step we have in the first case pushout (1) and the commutative square (2), where the typing $G_{i+1} \to TG_{i+1}$ is induced from $G_i \to TG_i$ and pushout (1), and $L_{q_i} \to TG_i, R_{q_i} \to TG_{i+1}$ are given by our layered type-increasing GTS (TG_A, Q), such that the outer diagram $(1+2)$ is a pullback and all horizontal morphisms are monomorphisms.

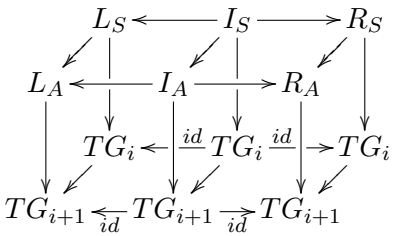


Hence, by pushout-pullback-decomposition property (see e.g. [3]), we get that (2) is a pullback. In the case that $q_i = id$, diagram (3) is a pullback because $TG_i \hookrightarrow TG_{i+1}$ is monomorphism. This leads to the following sequence of pullbacks, which can be composed to one pullback:

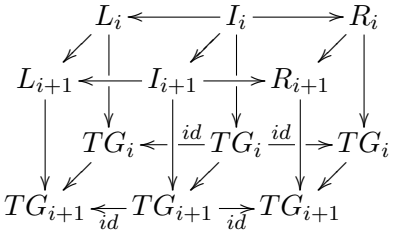


□

Proposition 4.8 (Restriction of *S2AR* to TG_S)
Given an S2AR transformation $S2AR : P_S \to P_A$ based on a layered type-increasing TGTS (TG_A, Q) with $TG_S \subseteq TG_A$, then we have: $p_S \overset{Q\ !}{\Longrightarrow} p_A$ with $p_S \in P_S$ implies $p_A|_{TG_S} = p_S$, i.e. for $p_S = (L_S \leftarrow I_S \to R_S), p_A = (L_A \leftarrow I_A \to R_A)$ the double cube to the right commutes with pullbacks in the diagonal squares.



Proof Sketch. (for a full proof see [5])
Given $p_S \overset{Q\,!}{\Longrightarrow} p_A$, we consider the subsequences according to the layers Q_i of Q, $p_S = p_0 \overset{Q_0!}{\Longrightarrow} p_1 \overset{Q_1!}{\Longrightarrow} p_2 ... p_n \overset{Q_n!}{\Longrightarrow} p_{n+1} = p_A$ and show that for each $i = 0, .., n$ the double cube to the right exists with pullbacks in the diagonal squares, which can be composed to the required double cube with p_S and p_A.



□

Remark 4.9 Proposition 4.7 implies that there exists a TGTS embedding $f : SimSpec_{VL_S} \to AnimSpec_{VL_A}$ given by $f = (TG_S \xrightarrow{f_{TG}} TG_A, P_S \xrightarrow{f_P} P_A)$, where f_{TG} is the type graph inclusion, and f_P maps each simulation rule p_S to the rule p_A resulting from the *S2AR* transformation. TGTS embeddings are morphisms between typed graph transformation systems, defined categorically via so-called retyping functors between categories $\mathbf{Graphs_{TG}}$ and $\mathbf{Graphs_{TG'}}$ of typed graph transformation systems (see [5], Sect. 2.1.3).

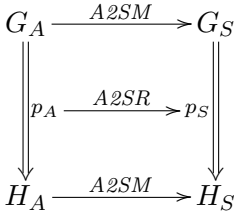
Theorem 4.10 (*A2S* is Backward Transformation of *S2A*) *Given an S2A transformation* $S2A = (S2AM : VL_S \to VL_A, S2AR : P_S \to P_A)$ *based on a layered type-increasing GTS* (TG_A, Q) *with* $TG_S \subseteq TG_A$*, then the transformation* $A2S : AnimSpec_{VL_A} \to SimSpec_{VL_S}$ *defined according to Def. 4.6, is a backward transformation of S2A in the sense of the characterization of backward transformations given in Def. 4.4.*

Proof. $A2SM : VL_A \to VL_S$ for $G_A \in VL_A$ with $G_S \overset{Q\,!}{\Longrightarrow} G_A$ for $G_S \in VL_S$ maps G_A to G_S, because $G_A|_{TG_S} = G_S$ by Prop. 4.7. This implies $A2SM \circ S2AM \subseteq ID_{VL_S}$. Analogously, $A2SR : P_A \to P_S$ for $p_A \in P_A$ with $p_S \overset{Q\,!}{\Longrightarrow} p_A$ for $p_S \in P_S$ maps p_A to p_S, because we have $p_A|_{TG_S} = p_S$ by Prop. 4.8. This implies $A2SR \circ S2AR \subseteq ID_{P_S}$. Hence, $A2SM, A2SR$ and $A2S = (A2SM, A2SR)$ are backward transformations of $S2AM, S2AR$ and $S2A$, respectively, according to Def. 4.4. □

4.3 Semantical Correctness of A2S-Backward-Transformations

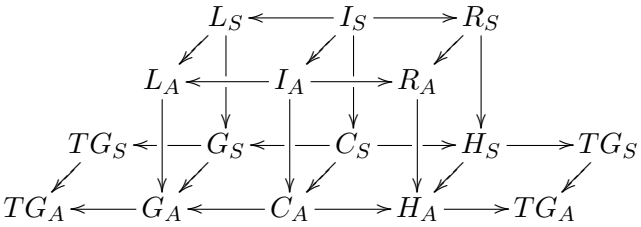
Given an *A2S* backward transformation of *A2S* with $A2S = (A2SM, A2SR) : AnimSpec_{VL_A} \to SimSpec_{VL_S}$ such that $A2SR(p_A) = p_S$ for $p_A \in P_A, p_S \in P_S$ and $A2SM(G_A) = G_S$ for $G_A \in VL_A, G_S \in VL_S$, then the graph H_S resulting from the simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ and the graph H_A resulting from the animation step $G_A \overset{p_A}{\Longrightarrow} H_A$ should be related by *A2SM* backward transformation, i.e. $A2SM(H_A) = H_S$.

Definition 4.11 (Semantical Correctness of *A2S* Transformation)
An *A2S* transformation $A2S : AnimSpec_{VL_A} \to SimSpec_{VL_S}$ given by $A2S = (A2SM : VL_A \to VL_S, A2SR : P_A \to P_S)$ is *semantically correct* if for each animation step $G_A \overset{p_A}{\Longrightarrow} H_A$ with $G_A, H_A \in VL_A$ and $A2SM(G_A) = G_S$ and $A2SR(p_A) = p_S$, there is a corresponding simulation step $G_S \overset{p_S}{\Longrightarrow} H_S$ with $A2SM(H_A) = H_S$.



Theorem 4.12 (Semantical Correctness of *A2S* Backward Transformation) *Each A2S backward transformation $A2S = (A2SM, A2SR)$ of an S2A transformation $S2A = (S2AM, S2AR)$ is semantically correct.*

Proof Sketch. (for a full proof see [5])
The semantical correctness of *A2S* backward transformation holds due to the fact that the *S2A* transformation induces a TGTS embedding from $SimSpec_{VL_S}$ to $AnimSpec_{VL_A}$ (see Remark to Fact 4.8). TGTS embeddings reflect the behavior in the sense that if we have a transformation $G_A \overset{p_A,m_A}{\Longrightarrow} H_A$ in $AnimSpec_{VL_A}$, we get the transformation $G_S \overset{p_S,m_S}{\Longrightarrow} H_S$ in $SimSpec_{VL_S}$, where the matches are related by $m_A|_{TG_S} = m_S$. Basically, the proof works by construction of the double cube shown below, where the front squares are pushouts corresponding to a rewriting step $G_A \overset{p_A}{\Longrightarrow} H_A$ in the DPO approach, applying the animation rule $p_A = (L_A \leftarrow I_A \rightarrow R_A)$ to graph G_A. It is shown that the diagonal squares are all pullbacks. Thus, the Van-Kampen property (see [3]) can be used to prove that the back squares are also pushouts, which correspond to the rewriting step $G_S \overset{p_S}{\Longrightarrow} H_S$ in the DPO approach, applying the rule $p_S = (L_S \leftarrow I_S \rightarrow R_S)$ to G_S, where $G_S = G_A|_{TG_S}$.



□

4.4 Behavior-Preserving S2A Transformations

We now present the main result in Theorem 4.14, stating the conditions for *S2A* transformations being behavior-preserving, based on Theorems 4.3 and 4.12.

Definition 4.13 (Behavior-Preserving *S2A* Transformations)
Given an *S2A* model and rule transformation $S2A = (S2AM, S2AR) : SimSpec_{VL_S} \rightarrow AnimSpec_{VL_A}$ (Def. 3.5), and the corresponding *A2S* backward-transformation $A2S = (A2SM, A2SR) : AnimSpec_{VL_A} \rightarrow SimSpec_{VL_S}$ (Def. 4.6), we say that *S2A* is *behavior-preserving*, if

(i) *S2A* is semantically correct (acc. to Def. 4.1), and

(ii) *A2S* is semantically correct (acc. to Def. 4.11)

Theorem 4.14 (Behavior-Preserving *S2A* Transformation) *An S2A transformation $S2A = (S2AM, S2AR) : SimSpec_{VL_S} \rightarrow AnimSpec_{VL_A}$ is behavior-preserving if*

(i) *S2A is rule-compatible, and S2AM is terminating (Def. 4.2),*

(ii) *A2S is constructed according to Def. 4.6,*

Proof. By Theorem 4.3 we have that *S2A* is semantically correct for rule-compatible *S2A* and terminating *S2AM*. By Theorem 4.10 and Theorem 4.12 we know that *A2S* is a valid backward-transformation of *S2A* and that *A2S* is seman-

tically correct. Hence, according to Def. 4.13, *S2A* is a behavior-preserving model and rule transformation. □

Finally, we can consider semantical equivalence of $SimSpec_{VL_S}$ and $AnimSpec_{VL_A}$, which requires behavior-preserving *S2A*, such that *S2A* and *A2S* are inverse to each other, i.e. $A2S \circ S2A = Id$ and $S2A \circ A2S = Id$. It is shown in [5] that we have semantical equivalence if *S2A* is behavior-preserving, and the *S2A* transformation rules in Q are confluent.

4.5 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. 3.3 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 $NAC_q = (L_q \xrightarrow{n} N_q)$. To extend rule compatibility (Def. 4.2), 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 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 $p_i \overset{q}{\Longrightarrow} p_{i+1}$ satisfies NAC_q, and $G_i \overset{p_i}{\Longrightarrow} H_i$ satisfies $NAC(p_i)$ then $G_{i+1} \overset{p_{i+1}}{\Longrightarrow} H_{i+1}$ 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 4.3 and Theorem 4.14, where now rule compatibility and termination have to be required with NACs (for the complete extended theorems see [7,5]).

Moreover, the proofs of Prop. 4.7 and 4.8 can be extended to NACs in a straightforward way. An additional property has to be required to get semantical correctness of *A2S* (Theorem 4.12), namely NAC-compatibility of *A2S*. Fortunately, NAC-compatibility can be shown in general for all *A2S*-transformations (see [5] for the complete proof of Prop. 4.15).

Proposition 4.15 (NAC-Compatibility of *A2S* Transformations) *An A2S transformation* $A2S = (A2SM : VL_A \rightarrow VL_S, A2SR : P_A \rightarrow P_S)$ *is* NAC-compatible *in the following sense: Let* $G_S = A2SM(G_A)$ *and* $p_S = A2SR(p_A)$. *Then, if* $G_A \overset{p_A}{\Longrightarrow} H_A$ *satisfies* $NAC(p_A)$ *then* $G_S \overset{p_S}{\Longrightarrow} H_S$ *satisfies* $NAC(p_S)$.

4.6 Behavior Preservation in the Radio Clock Case Study

Using the extended theorems, we can show behavior preservation of our Radio Clock *S2A* transformation (see [5,7]). Termination is shown to be fulfilled for general *S2A* transformation systems based on layered type-increasing TGTS. Moreover, it is

shown that each *S2AR* transformation is NAC-compatible provided that we have layered type-increasing TGTS, as our case study has. NAC-compatibility of *S2AM* has been shown explicitly for the Radio Clock in [7]. For the Radio Clock case study, we even have semantical equivalence of $SimSpec_{VL_S}$ and $AnimSpec_{VL_A}$, since the Radio Clock *S2A* transformation is shown to be confluent in [5].

5 Related Work

To ensure the correctness of model transformations, Varró et al. [15,17] 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 [13], 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. [4].

Backward transformations are also of interest in the context of bidirectional model transformations based on triple graph grammars [16]. In [2], it has been investigated under which conditions a given forward transformation has an inverse backward transformation, but semantical correctness has not yet been considered.

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 [11], 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 [12]. Hence, the mapping from simulation to animation views happens at the implementation level and is neither specified formally, nor shown to be behavior-preserving. Analogously, different Petri net tools also offer support for customized Petri net animations 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.

6 Conclusion and Ongoing Work

In this paper we have reviewed the definition for simulation-to-animation (*S2A*) model and rule transformations, and defined a corresponding *A2S*-backward transformation $A2S : SimSpec_{VL_S} \to AnimSpec_{VL_A}$, essentially given by restriction of all graphs and rules to the simulation type graph TG_S. The main results show under which conditions an *A2S* transformation is semantically correct, in the cases without and with negative application conditions. Having semantical correctness both of *S2A* and of *A2S*, we have a behavior-preserving simulation-to-animation (*S2A*) model and rule transformation system. The results have been used to show that the *S2A* transformation of our radio clock case study preserves the behavior.

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 [3].

Future work is planned to generalize our approach formalizing behavior-preserving model and rule transformations from *S2A* transformations to other kinds of model transformations based on graph transformation, especially to triple graph grammar specifications.

References

[1] Ehrig, H. and K. Ehrig, *Overview of Formal Concepts for Model Transformations based on Typed Attributed Graph Transformation*, in: *Proc. International Workshop on Graph and Model Transformation (GraMoT'05)*, ENTCS **152** (2005).

[2] Ehrig, H., K. Ehrig, C. Ermel, F. Hermann and G. Taentzer, *Information preserving bidirectional model transformations*, in: M. B. Dwyer and A. Lopes, editors, *Fundamental Approaches to Software Engineering*, LNCS **4422** (2007), pp. 72–86.

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

[4] Ehrig, H. and B. Koenig, *Deriving bisimulation congruences in the dpo approach to graph rewriting*, in: *Proc. FOSSACS 2004*, LNCS **2987** (2004), pp. 151–166.

[5] Ermel, C., "Simulation and Animation of Visual Languages based on Typed Algebraic Graph Transformation," Ph.D. thesis, TU Berlin, Fak. IV (2006).

[6] Ermel, C., H. Ehrig and K. Ehrig, *Semantical Correctness of Simulation-to-Animation Model and Rule Transformation*, in: *Proc. Workshop Graph and Model Transformation (GraMoT'06)*, Electronic Communications of the EASST **4** (2006).

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

[8] Ermel, C., 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)* (2005).

[9] *GenGED Homepage*, http://tfs.cs.tu-berlin.de/genged.

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

[11] Harel, D., S. Efroni and I. Cohen, *Reactive Animation*, in: *Proc. Formal Methods for Components and Objects (FMCO'02)*, LNCS **2852** (2003), pp. 136–153.

[12] Macromedia, Inc., "Macromedia Flash MX 2004 and Director MX 2004," (2004), http://www.macromedia.com/software/.

[13] Narayanan, A. and G. Karsai, *Towards Verifying ModelTransformations*, in: *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'06)* (2006).

[14] Parisi-Presicce, F., *Transformation of Graph Grammars*, in: *5th Int. Workshop on Graph Grammars and their Application to Computer Science*, LNCS **1073** (1996).

[15] Schmidt, Á. and D. Varró, *Checkvml: A tool for model checking visual modeling languages.*, in: *Proc. UML, Modeling Languages and Applications (UML'03)*, LNCS **2863** (2003), pp. 92–95.

[16] Schürr, A., *Specification of Graph Translators with Triple Graph Grammars*, in: *Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS **903** (1994), pp. 151–163.

[17] Varró, D., *Automated formal verification of visual modeling languages by model checking.*, Software and System Modeling **3** (2004), pp. 85–113.

[18] WWW Consortium, "Scalable Vector Graphics Specification." (2003), http://www.w3.org/TR/svg11/.