Rational Time-Advance DEVS (RTA-DEVS)

Hesham Saadawi (hsaadawi@connect.carleton.ca)[1] Gabriel Wainer (gwainer@sce.carleton.ca)[2]

[1]School of Computer Science, Carleton University, Ottawa, ON, CANADA
[2]Departement of Systems and Computer Engineering, Carleton University, Ottawa, ON, CANADA

Keywords: Discrete event simulation, DEVS, UPPAAL, DEVS verification, Timed Automata, model checking.

Abstract

This paper introduces a new extension to the DEVS formalism, called Rational Time-Advance DEVS. The basic idea of this new formalism is to permit modeling the behavior of systems that can be modeled by classical DEVS; however, RTA-DEVS models could be formally checked with standard model-checking algorithms and tools. In order to do so, we introduce a procedure to create Timed Automata models that are behaviorally equivalent to the original RTA-DEVS models. This therefore, enables the use of the available TA tools and theories for formal model checking.

I. INTRODUCTION

Real-time embedded systems (RTS) are very advanced computer system applications with hardware and software components interacting in a tight fashion. RTS are highly reactive systems where the decisions taken can lead to catastrophic consequences for goods or lives; hence, correctness, and the timing of the executing tasks are critical.

An effective approach in studying such systems is to model the application by abstracting unnecessary details, obtaining a model that closely approximates the behavior of the real system. The model would then be studied to reason about the real system. A modeler would select a suitable formalism and tools depending on the nature of the system under study. For instance, continuous systems have a long tradition of being modeled with a algebraic, differential and partial differential equations, which can faithfully model the systems and have well-known methodologies and tools available. For discrete-event systems, *formal methods* can be used to determine the model's behavior to reason about the real system. Formal verification methods allow the model to be rigorously checked against a specification property, proving if the property is satisfied or not. For instance, *Timed Automata* (introduced in the 1990's as an extension to finite automata) can be used to model discrete-event systems and their timing aspects [1].

TA has an established theory of formal verification and analysis through *model-checking* [2][3]. This method tests a model against given specifications or properties using algorithms to formally verify and prove that a model meets a specification (coded as a logic formula), or if not, it can show the sequence of events to cause the defect. The main advantage of model-checking is that it is usually very difficult (or impossible) to test all possible executions of a model to find a defect. Instead, models checked formally are guaranteed to be free of errors (as opposed to simulation and testing based solutions, which gives a certain confidence in a model with no absolute guarantee). However, model-checking algorithms suffer from a state explosion problem, in which total number of states to be checked grows exponentially with the model size. Thus, for large size applications, model-checking is impractical.

In these cases, Modeling and Simulation (M&S) techniques and tools can be used for analyzing varied scenarios. In M&S based analysis, the model is executed with sample inputs, and the outputs are observed and these are compared to real system behavior. This validation is usually done with the help of a subject matter expert to check the logical assumptions and abstractions in the model to reflect the real system. Most M&S frameworks are not as robust formal methods due to their lack of a mathematical foundation (which poses difficulties when trying to prove properties about RTS). Instead, *DEVS* (*Discrete-EVent Systems specifications*) [4], provides a formal method (based on systems theory) to model and simulates discrete-event systems. DEVS has been widely used since its inception in the '70s to model discrete-event systems, and numerous DEVS tools have been created. However, DEVS does yet have a sound theory for formal verification (thus, DEVS models are mainly studied through simulation).

Both model verification techniques and M&S based methods need manual work for testing the model in an iterative cycle (an error-prone and expensive task). By enabling simulation models to be formally checked, a modeler can use formal verification to compose queries about the model behavior and submit it to a model verifier, which would be able to answer if the model satisfies these properties. Likewise, the model can be formally verified against any faults that may be introduced during model construction, making easier the testing tasks.

Our long-term goal is thus to define a theoretical framework and a practical environment that will combine the advantages of formal methods and simulation for RTS construction. At this stage, we are exploring methods to convert DEVS to TA to enable formal DEVS model checking. Converting from DEVS models to equivalent TA models and vice versa is a valuable addition to both worlds. TA could be used for DEVS model validation and verification through model-checking. On the other hand, TA could be modeled as DEVS to be simulated, complementing model-checking for very large models.

As would be shown in section II, previous work on formal verification of DEVS models has to work on restricted types of DEVS. This is because formally checking classic DEVS has its challenges and limitations. However, by using these restricted types of DEVS, a modeler would lose much of the modeling power of classic DEVS to model the system at hand. Ideally, a modeler would like to use some formalism that is as powerful as DEVS, yet this formalism can be formally verified. In this paper, we try to achieve this by introducing RTA-DEVS, a subset of DEVS still powerful for practical modeling applications. RTA-DEVS can be seen as a relaxation of some restriction of FD-DEVS [9] (the elapsed time cannot be used to determine the transition to next state). This makes the expressiveness of RTA-DEVS closer to classic DEVS, giving the modeler a more powerful formalism. Also, RTA-DEVS is defined in a way to make it possible to transform it to an equivalent TA that can be formally verified. We show a transformation from RTA-DEVS to TA that preserves all proper-

ties of RTA-DEVS models into TA (the resulting TA model is built to be behaviorally equivalent to the RTA-DEVS model). This enables complete formal validation and verification of RTA-DEVS models with available tools and theories. We finally show an example of RTA-DEVS and its transformation to TA, and its verification using UPPAAL [1].

II. Related work

There have been several proposals to verify DEVS models, ranging from formal model-checking of restricted classes of DEVS, the generation of traces from DEVS models for testing, the specification of high-level system requirements in TA (and verifying DEVS model against those requirements), or introducing clock constructs to DEVS to conform with TA.

In [5], the Real-Time DEVS formalism (RT-DEVS) introduces a time advance function that maps each state to a range with maximum and minimum time values. In [8], RT-DEVS was used to model a real-time system of train-gate-controller. It introduced an algorithm to build a timed reachability tree to be used for safety analysis. Further work on verifying RT-DEVS is done in [6][7] using TA and UPPAAL, and a transformation from RT-DEVS to UPPAAL is shown. This transformation allows weak synchronization between components of TA model as RT-DEVS semantics uses weak synchronization. The transformation given did not show formally timed behavior equivalence between RT-DEVS and TA models.

Other approach [9] was to introduce a subclass of DEVS (called Finite-Deterministic DEVS) in which the time advance function is maps states to rational numbers, and the external transition function cannot use the elapsed time to determine its result. In [9] the authors introduced verification through reachability analysis similar to TA algorithms and techniques.

In [10], the authors show how to map DEVS models to TA. The conversion method mapped DEVS model through its components and its simulator. The approach suggests trace equivalence as the basis for parallel DEVS and TA model equivalence. In [11], high-level system specifications are done in TA and then modeled with DEVS; system requirements are verified through simulation of the DEVS model.

A result in [12] showed that verification of general DEVS models through reachability analysis is undecidable. The authors based their deduction on building a DEVS simulation Turing machine. Since in Turing machines the halting problem is undecidable (i.e. with analysis only, we cannot know in which state a Turing machine would be), they concluded that this is also true for DEVS models: we cannot know if we reach a particular state starting from initial state, and hence reachability analysis for general DEVS is impossible. They argue that reachability analysis maybe possible only for restricted classes of DEVS. This result however was based on introducing state variables into DEVS formalism with infinite number of values. In [13], the authors introduced a new class of DEVS called Time Constrained DEVS (TC-DEVS) that expanded DEVS atomic model definition with the introduction of multiple clocks incremented independently of other clocks. Classic DEVS atomic models can be seen as having only one clock that keeps track of elapsed time in a state, and is reset on each transition. TC-DEVS also added clock constraints similar to TA (to function as guards on external and internal transitions). However, it allows clock constraints in states as invariants that contain clock differences. TC-DEVS is then transformed to an UPPAAL TA model. The paper however, did not explain a transformation of TC-DEVS state invariants to UPPAAL TA when the model has invariants with clock differences.

Other DEVS verification techniques use testing to verify the DEVS [14][15]. Testing sequences generated from model specifications are applied against the model implementation to verify the conformance of implementation to specifications.

Our work differs from the above approaches in that it defines a new class of DEVS, called RTA-DEVS, which is very close to classic DEVS in semantics and expressive power. We then define a transformation to obtain a TA that is behaviorally equivalent to RTA-DEVS. The advantage of doing so is that many classic DEVS models would satisfy the semantics of RTA-DEVS models. Thus, they could be simulated with any DEVS simulator. Likewise, it can be transformed to TA to validate desired properties formally. RTA-DEVS has followed FD-DEVS in restricting the time advance function to nonnegative rational numbers, but also relaxed the restriction of FD-DEVS on external transition functions. This makes RTA-DEVS closer to general DEVS and adds expressiveness. However, RTA-DEVS still restricts the elapsed time in a state used in the external transition function to be nonnegative rational number. This restriction translates to having nonnegative rational constants in guards in the transformed TA model, and ensures termination of reachability analysis algorithm implemented in UPPAAL [1], as irrational constants in TA guards renders reachability analysis undecidable [16].

III. RTA-DEVS

As in classical DEVS, we need to define RTA-DEVS atomic model. RTA-DEVS changes the definitions of time advance function *ta* and the external transition function δ_{ext} as follows. The Atomic Rational Time-Advance is defined as:

$$AM_{TC} = < X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

- X : The set of external inputs.
- Y : Set of external outputs.
- S: set of system states.
- δ_{int}: $S \rightarrow S$ is the internal transition function (the same as in classic DEVS).

– δ_{ext}: $TxX \rightarrow S$ with $T=\{(s,e)/s\ 0 \leq e \leq ta(s),\ e \in Q_{0,+\infty}\}$ is the external transition function (e is the time elapsed since the last transition, which takes a positive rational value).

- λ: $S \rightarrow Y \cup \varnothing$ is the output function.
- *ta*: $S \rightarrow Q_{0,+\infty}$ is the time advance function that maps each state to a positive rational number.

Coupled RTA-DEVS model are defined exactly as in classic DEVS. Coupled RTA-DEVS models are composed of atomic or other coupled RTA-DEVS models:

$$CM \equiv \left\langle X, Y, D, \{M_i\}, C_x, C_y, Select \right\rangle$$

X: Set of external input events.

Y: Set of external output events.

D: Finite index of sub-components.

$\{M_i\}$: The set of sub-components. A sub-component may be an atomic or coupled. $i \in D$ is the index of the component.

C_x: Set of input couplings.

C_y: Set of output couplings.

Select: $2^D \rightarrow D$ is a tie-breaking function, which defines how to select an event from asset of simultaneous events.

A coupled RTA-DEVS model M can be simulated with an equivalent atomic RTA-DEVS model, whose behavior is defined as follows [18]:

$$M = < X,Y,S,s_0,\delta_{ext},\delta_{int},\lambda,ta >$$

- X and Y are the input and output event sets, respectively. X is the set of all input events accepted and Y is the set of all output events generated by coupled model M.
- $S = \times_{i \in D} V_i$ is the model state. It is expressed as the Cartesian product of all component states, where V_i is the total state for component *i*, $V_i = \{(s_i, t_{ei}) \mid s_i \in S_i, t_{ei} \in [0, ta(s_i)]\}$. Here, t_{ei} denotes the elapsed time in state s_i of component *i,* and S_i is the set of states of component *i*.
- $s_0 = \times_{i \in D} v_{0i}$ is the initial system state, with $v_{0i} = (s_{0i}, 0)$ is the initial state of component $i \in D$.
- $ta : S \rightarrow \mathrm{T}$ is the time advance function. It is calculated for the global state $s \in S$ of the coupled model as the minimum time remaining for any state among all components, formally: $ta(s) = \min\{(ta(s_i) - t_{ei}) \mid i \in D\}$ where $s = (...,(s_i, t_{ei}),...)$ is the global total state of coupled model at some point in time, s_i is the state of component *i*, t_{ei} is elapsed time in that state.
- $\delta_{ext} : X \times V \rightarrow S$ is the external transition function for the coupled model. Where *V* is total state of the coupled model: $V = \{(s, t_e) \mid s \in S, t_e \in [0, ta(s)]\}$.
- $\delta_{\mathrm{int}} : S \rightarrow S$ is the internal transition function of the coupled model.
- $\lambda : S \rightarrow Y$ is the output function of the coupled model.

IV. TIMED SAFETY AUTOMATA (TA)

A Timed Automaton can be defined as [19]:

$$A = (N, l_o, E, I)$$

– *N* is a finite set of locations (or nodes),
– $l_o \in N$ is the initial location,
– $E \subseteq N \times \beta(C) \times \Sigma \times 2^C \times N$ is the set of edges and
– *I:* $N \rightarrow \beta(C)$ assigns invariants to locations

Here, *C* is a set of clock variables (with *x, y, etc.* representing clock variables from the set *C*). We use *a, b, etc.* to represent actions from a set of finite alphabet Σ. Assume a finite set of real-valued variables *C* ranged over by clocks *x, y, etc.* and a finite alphabet Σ (with actions *a,b, etc.*). Let us call a *clock constraint* a conjunctive formula of atomic constraints of the form $x \sim n$ or $x\text{-}y \sim n$ where *x,y* are clock variables, ~ is one of $\{\leq, <, =, >, \geq\}$ and *n* is a natural number. Clock constraints can used on transitions, where they are called a *guard*; or in a location (state), were they are called *invariant*. Invariants are constraints on the form $x \leq n$, or $x < n$ to restrict time spent in a TA location. $\beta(C)$ denotes set of clock constraints.

TA are Timed Transition Systems where states are pairs <*L,u*>, where *L* is a location, and *u* is a clock valuation. We write $l \xrightarrow{g,a,r} l'$ when $(l, g, a, r, l') \in E$ with *g* a clock constraint, *a* an action and *r* a set of clocks to be reset to zero. TA use two types of transitions:

- *Delay Transition*: $< L,u > \xrightarrow{d} < L,u+d >$: the time passage *d* causes a transition from the start location, to an end location.
- *Action transition*: $< L,u > \xrightarrow{a} < L',u' >$: an action *a* causes A transition from start location, to end location.

V. EQUIVALENCY OF RTA-DEVS AND TA

To verify RTA-DEVS applying existing theory and tools existing for TA, we need to construct TA models from RTA-DEVS models that are behaviorally equivalent.

Generally there are two methods to check if two Labeled Transition Systems (LTS) are behaviorally equivalent, namely *Trace Equivalence* and *Bisimulation*. In [17], it was shown that, for RTS, trace equivalence is not enough to show the complete equivalence of two LTS. Although one can show the trace equivalency of two LTS (based on their acceptance or the generation of event traces), RTS usually have multiple concurrent components working together. Those components may go into a deadlock state in which no external event is observable. Due to these subtle errors, bisimulation is a better notion for behavior equivalence, and the same reason is valid for Timed LTS (TLTS), of which RTA-DEVS and TA are two examples.

Bisimulation is a relation between two TLTS (e.g., systems A and B), which establishes a relation between every state in A, and a corresponding one in B. It also relates every observable transition in A to a corresponding one in B. In [17], the concepts of strong and weak timed bisimilarity were defined for behavioral equivalence of systems. In our case, we used timed weak bisimilarity equivalence, as it is more general in its application than the conditions for strong bisimulation.

In next sections, we define behavior equivalency based on timed weak bisimulation. Then following the conditions of this bisimulation, we construct a TA model for the basic behavior elements of RTA-DEVS, namely internal and external transitions. Then, we deduce the required constant values on the TA model to complete the bisimulation equivalence.

Definition: Eventual transition relation "$\Rightarrow$"

In a TLTS with states, *s* and *t*, the *eventual transition relation* defines a transition from state *s* to state *t* that may contain one or more of direct transitions labeled with non-observable events to the outside world. If we have an observable action *a*, a non-observable action τ, and a transition label α, then, for any TLTS [17], the *eventual transition relation* $\Rightarrow$ between *s* and *t* on action α (written $s \overset{\alpha}{\Rightarrow} t$) is defined if any of the following conditions is true:

1. $s \overset{\tau}{\Rightarrow} t$: there is a transition from *s* to *t* only composed of transitions labeled with non-observable actions. E.g., for the non-observable action $\alpha=\tau$, there is a transition $s(\xrightarrow{\tau})^* t$ (* defines one or more occurrences of these transitions).
2. $s \overset{a}{\Rightarrow} t$: there is a transition from *s* to *t* composed of one transition labeled with an observable action $\alpha=a$, and one or

more eventual transitions labeled with non-observable actions. E.g., $s \overset{\tau}{\Rightarrow} s_1 \xrightarrow{a} s_2 \overset{\tau}{\Rightarrow} t$ for some states s_1 and s_2;

3. $s \overset{d}{\Rightarrow} t$ **:** there is an eventual transition relation from s to t with total delay d (called *eventual delay transition*), which is composed of one or more direct delay transitions combined with some non-observable action transitions. This represents a sequence of transitions with no observable actions whose total delay amounts to d. E.g., for action $\alpha = d \in \Re_{\geq 0}$ and $s \overset{\tau}{\Rightarrow} s_1 \xrightarrow{d_1} t_1 .....t_{n-1} \overset{\tau}{\Rightarrow} s_n \xrightarrow{d_n} t_n \overset{\tau}{\Rightarrow} t$ (with $n \geq 0$), for some intermediate states $s_1 ... s_n$, $t_1 ... t_n$ and delays $d_1 ... d_n$ with $d = \sum_{i=1}^{n} d_i$ (d is the total delay for the eventual transition from s to t; by convention, d=0 when n= 0).

Definition: weak timed bisimulation.

The *weak timed bisimulation* is a binary relation $\boldsymbol{R}$ over a set of states of a TLTS. For example if we have states s_1, s'_1, s_2 *and* s'_2, then $\boldsymbol{R}$ is a weak timed bisimulation $s_1 \, \boldsymbol{R} \, s_2$ [17] *iff*:

- $s_1 \xrightarrow{d} s'_1$, then there is a transition $s_2 \overset{d}{\Rightarrow} s'_2$ such that $s'_1 \, \boldsymbol{R} \, s'_2$ as shown in Figure 1.
- $s_1 \xrightarrow{a} s'_1$, then there is a transition $s_2 \overset{a}{\Rightarrow} s'_2$ such that $s'_1 \, \boldsymbol{R} \, s'_2$ as shown in Figure 2.
- $s_2 \xrightarrow{d} s'_2$, then there is a transition $s_1 \overset{d}{\Rightarrow} s'_1$ such that $s'_1 \, \boldsymbol{R} \, s'_2$ as shown in Figure 3.
- $s_2 \xrightarrow{a} s'_2$, then there is a transition $s_1 \overset{a}{\Rightarrow} s'_1$ such that $s'_1 \, \boldsymbol{R} \, s'_2$ as shown in Figure 4.

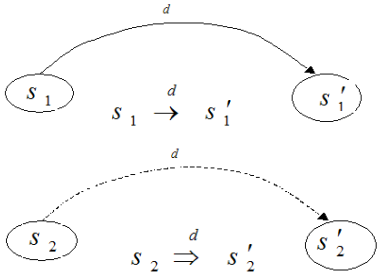

Figure 1: Direct delay transition from s_1 to s'_1 and corresponding eventual delay transition from s_2 to s'_2.

We choose the weak bisimulation relation to transform from RTA-DEVS to TA and vice versa, as this relation allows two models to be in bisimulation relation even with if one of them has some different transitions from the other, provided that these extra transitions are labeled with non-observable action τ [17]. This relaxation over strong bisimulation allows more flexibility to tune the TA model for performance while keeping the bisimulation relation to the RTA-DEVS model.

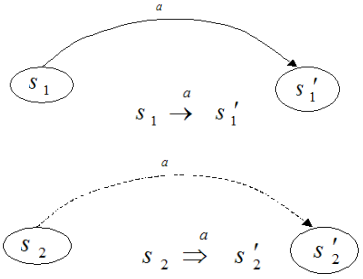

Figure 2: Direct action transition from s_1 to s'_1, and corresponding eventual action transition from s_2 to s'_2.

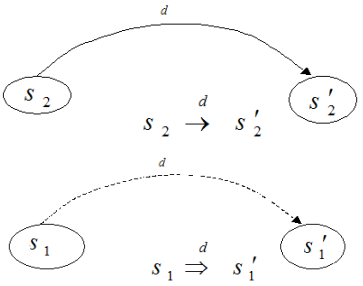

Figure 3: Direct delay transition from s_2 to s'_2 and corresponding eventual delay transition from s_1 to s'_1.

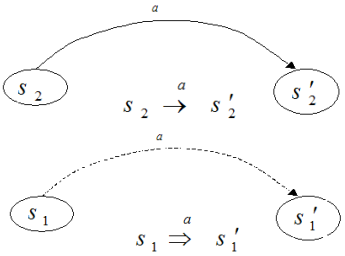

Figure 4: Direct action transition from s_2 to s'_2, and corresponding eventual action transition from s_1 to s'_1.

V. A) RTA-DEVS model internal transition semantics

TA expresses the notion of time through clock variables and constraints on them, as shown in the previous section. Here, we present the RTA-DEVS behavior in terms of its transition functions, and we discuss how to obtain a behaviorally equivalent TA according to the previous definition. In doing so, we determine the value of the constant in TA invariants and guards, in order to keep the same behavior of RTA-DEVS.

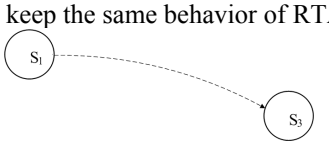

Figure 5: RTA-DEVS Internal Transition

The RTA-DEVS internal transition semantics is shown as a DEVS graph in Figure 5, and is defined as:

1. $\delta_{\text{int}}(s_1, e) = s_3$ if $e = ta(s_1) = T$

In RTA-DEVS semantics, this transition means that we move to state s_3 when the elapsed time e in s_1 equals the time advance value of s_1. In a TLTS, this can be defined as a time elapsed transition with delay d with the form:

$$s_1 \xrightarrow{d} s_3 \text{ if } 0 \leq e < ta(s_1) \text{ and } d = ta(s_1) - e$$

which means that if we start at s_1 with time spent e, we

need to delay d time units before changing to state s_3.

2. $\delta_{\text{int}}(s_1, e) = s_1$ if $0 \le e < ta(s_1)$

From the RTA-DEVS semantics, this transition means that we stay in the same state s_1 as long as the elapsed time e in that state does not equal or exceed the time advance value for s_1. This can be defined as a time elapse transition on the form:

$s_1 \xrightarrow{d} s_1$ if $0 \le e < ta(s_1)$ and $0 \le d < ta(s_1) - e$

which means that if we start at s_1 with time spent e, as long as time delay d is constrained as above, we stay at s_1.

From now on, we will use the operational semantics of UPPAAL for TA as defined in [19]. For the graphs in the rest of the paper, we will name RTA-DEVS states as s_i, and the corresponding TA locations as L_i (with i an integer number).

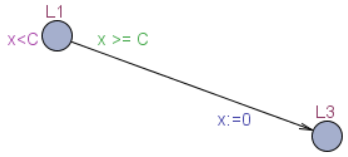

Figure 6: TA model for an Internal RTA-DEVS transition.

The delay transition for the TA in Figure 6 is defined as:

$(L_1, clock = x) \xrightarrow{d} (L_1, clock = x + d)$ for any $d \ge 0$

This defines the RTA-DEVS delay transition above, in which we stay in the same state with a total delay d less then time advance of s_1. We will show that the TA in Figure 6 is behaviorally equivalent to the DEVS graph model shown in Figure 5 through a timed bisimulation relation. To do this, we will show that the state s_1 is bisimilar to location L_1, and state s_3 is bisimilar to location L_3. This is done showing a weak timed bisimulation relation (from RTA-DEVS to TA, and also from TA to RTA-DEVS). We do these in steps 1 and 2 below.

In this model, two locations are defined (L_1 and L_3), along with a transition from L_1 to L_3. L_1 has an invariant on clock x ($x<C$) that allows the TA to stay in that location as long as the invariant is *true*. The transition from L_1 to L_3 has a guard ($x \ge C$) that must be true for the transition to be enabled, and C is a rational number. The transition also has an update rule for clock variable x to reset it to zero before entering location L_3. We apply the condition above to weak timed bisimulation, first from RTA-DEVS to TA and then from TA to RTA-DEVS. In doing so, we determine a value for the constant C to preserve the bisimulation relation.

1) Step1: from RTA-DEVS to TA

For the bisimulation of the states shown in Figure 5 and Figure 6, we have the following requirements:

- $S_1 \; \boldsymbol{R} \; L_1$: This is a delay transition from s_1 to itself. If $s_1 \xrightarrow{d} s_1$ for some value of d where $0 \le e < ta(s_1)$ and $0 \le d < ta(s_1) - e$; then to satisfy the bisimulation relation we should have: $(L_1, x = e) \xrightarrow{d} (L_1, x = e + d)$ for the same value of d. As we have from the invariant of state L_1, $x < C$ then by substituting for x and d, we get

  $ta(s_1) \le C$ (Rule 1)

- $s_3 \; \boldsymbol{R} \; L_3$: Bisimilarity between s_3 and L_3:

For the delay transition from s_1 to s_3, let us consider the execution of the DEVS internal transition $(s_1, e) \xrightarrow{d} (s_3, 0)$, where $0 \le e < ta(s_1)$ and $d = ta(s_1) - e$ by the TA transition $(L_1, x = e) \xrightarrow{d} (L_3, x = e + d)$, with a reset statement on the transition $x:=0$.

In order for these two delay transitions to be equivalent, they need to start from bisimilar states, and after same delay, they reach two bisimilar states. To achieve this, we use the same value of delay d on both transitions, and we deduce the constant C in the TA clock constraint to give us that condition above for bisimulation. The TA transition above starts from location L_1, with the clock x equals same value of elapsed time e at the RTA-DEVS transition above; then after same delay d of the RTA-DEVS transition, it transitions to L_3, and the value of clock x increases by d.

We apply the conditions for this delay transition on TA transition above. The TA guard on the transition out of location L_1: ($x \ge C$) with the value of clock x: ($x = e + d$), we get:

$ta(s_1) \ge C$ (Rule 2)

This rule means that as long we use a constant C in the guard of the TA transition with value greater or equal to the time advance value of s_1, the previous TA transition executes the RTA-DEVS transition above.

These two rules give the condition $ta(s_1) = C$ for the TA shown in Figure 6 to execute the DEVS graph as in Figure 5.

This condition guarantees the timed weak simulation relation from the RTA-DEVS model internal transition of Figure 5 to delay transitions of TA in Figure 6. We would show the condition of timed weak simulation relation from TA to RTA-DEVS in step 2 below.

2) Step 2: From TA to RTA-DEVS

To satisfy the other direction of the bisimulation relation, we convert the TA in Figure 6 with RTA-DEVS in Figure 5.

Case 1: TA delay transition $(L_1, x = e) \xrightarrow{d} (L_1, x = e + d)$.

Here, we need the value of clock x to be less than C in order for the L_1 invariant to be true and TA to stay in L_1, i.e.

$e + d < C$ (Rule 3)

For the RTA-DEVS time delay transition $(s_1, e) \xrightarrow{d} (s_1, e + d)$ to stay in s_1 after d, we need the sum of the elapsed time and delay to be less than the lifetime of s_1:

$e+d < ta(s_1)$ (Rule 4)

Case 2: TA transition $(L_1, x = e) \xrightarrow{d} (L_3, x = 0)$.

We start from location L_1 with a clock x equal to some elapsed time e in L_1. After d, we change to L_3 and clock x is reset. To exit from L_1, the invariant would be false and the guard on the TA transition would need to be true, which gives:

$e + d = C$ (Rule 5)

This is defined as in RTA-DEVS as $(s_1, e) \xrightarrow{d} (s_3, 0)$, in which we need elapsed time e in s_1 and a delay equal to the time advance of s_1 to trigger the internal transition:

$e + d = ta(s_1)$ (Rule 6)

From rules 3 and 4, we determine $C=ta(s_1)$. With this value, we have a timed simulation relation from TA (shown in Figure 6) to RTA-DEVS (shown in Figure 5). By having a simulation relation in both directions, the RTA-DEVS internal transition shown above is timed bisimilar and behaviorally equivalent to the TA timed transitions shown above if we have the constant C equal to the lifetime of corresponding state in RTA-DEVS model. This concludes that $s_1 \; \boldsymbol{R} \; L_1$ and $s_3 \; \boldsymbol{R} \; L_3$ by the bisimulation relation $\boldsymbol{R}$.

When we use the previous method could be used to map internal transitions from RTA-DEVS model to transitions at a TA model and vice versa, we guarantee the resulting transitions to be behaviorally equivalent. We will show the same for RTA-DEVS external transitions in the following section.

V. B) RTA-DEVS external transitions

The RTA-DEVS external transition function is defined as: $\delta_{ext} : V_D \times X \to S$, where: $V_D = \{(s,e) : s \in S, 0 \le e \le ta(s) \}$

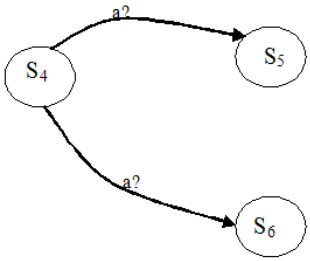

Figure 7: RTA-DEVS External transitions on action a.

Figure 7 represents the following definitions for the RTA-DEVS external transition function:

$\delta_{ext}(s_4, a, e) = s_5$ For $0 < e < 3$

$\delta_{ext}(s_4, a, e) = s_6$ For $3 \le e < ta(s_4)$

Each of these transitions can be expressed as a time passage and action transitions as:

1. $s_4 \xrightarrow{d < 3} s_4$ and $s_4 \xrightarrow{a} s_5$
2. $s_4 \xrightarrow{3 \le d < ta(s_4)} s_4$ and $s_4 \xrightarrow{a} s_6$

From these expressions, we can represent the external transitions as the TA transitions shown in Figure 8.

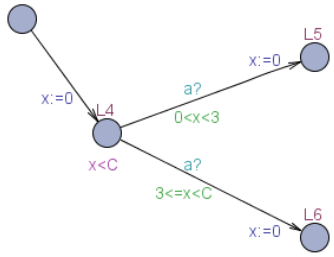

Figure 8: TA model for RTA-DEVS external transition.

From TA semantics, each of these transitions is expressed as a time and action transitions, that is:

1. $(L_4, x = 0) \xrightarrow{d < 3} (L_4, x = 3)$ and

$(L_4, x = 3) \xrightarrow{a} (L_5, x = 0)$ for the first RTA-DEVS transition. That is, we stay in L_4 while elapsed time is less than 3 units, and then with action a, the TA takes transitions to L_5.

2. $(L_4, x = 0) \xrightarrow{3 \le d < ta(s_4)} (L_4, x = 3)$ and

$(L_4, x) \xrightarrow{a} (L_6, x = 0)$ for the second RTA-DEVS transition. That is, if the elapsed time in L_4 exceeds 3 units and is less than lifetime of L_4, with action a, the TA transitions to L_6.

This gives us the relation $\boldsymbol{R}$ between RTA-DEVS and TA model states: $s_4 \; \boldsymbol{R} \; L_4$, $s_5 \boldsymbol{R} \; L_5$, and $s_6 \boldsymbol{R} \; L_6$.

Conversely, we can show the simulation in the other direction from each of TA transitions and the RTA-DEVS external transitions above. Hence, this shows a bisimulation relation $\boldsymbol{R}$ between the corresponding DEVS and TA models above.

VI. OBSERVATIONS: RTA-DEVS TO TA TRANSFORMATION.

The transformation example in the previous section introduces the methodology to transform RTA-DEVS models to TA models. The resulting TA models are a subset of deterministic safety automata used in the UPPAAL model checker. The transformation methodology can be summarized as follows:

1. Define a clock variable for each atomic RTA-DEVS model, for example x.
2. Replace every state in RTA-DEVS with a corresponding one in TA, i.e L_1 for source s_1 and L_2 for destination s_2.
3. An RTA-DEVS internal transition is modeled in TA as follows:
   - A source state L_1 and a destination state L_2.
   - Reset the clock variable on the entry to each state (x:=0).
   - Put an invariant in the source state derived from the time advance function for that state as shown above, i.e. $x<ta(s_1)$.
   - Define a transition with a guard. This guard should be the complement to the invariant in the source state as shown in the example transformation above, i.e. $x \ge ta(s_1)$.
4. The RTA-DEVS external transition is modeled in TA with the following items:
   - A source state and some destination state(s), i.e L_1 for source s_1 and L_2 for destination s_2.
   - A clock reset on the entry to each state.
   - An invariant in the source state that corresponds to time advance function for that state, i.e. $x<ta(s_1)$.
   - For the external transition(s) with guards of clock constraints, these constraints should be disjoint to obtain a deterministic TA model.
   - The action label on TA transitions for each RTA-DEVS input event to source state s_1.

By applying the previous steps, we obtain a TA model that executes every transition defined in the RTA-DEVS model under study. As we know, the RTA-DEVS behavior is completely defined by its transition functions, which defines all transitions in RTA-DEVS model. Thus, the resulting TA model executes the RTA-DEVS. This gives us a TA model that is behaviorally equivalent to the RTA-DEVS model and can be formally verified with tools like UPPAAL to infer properties about the original RTA-DEVS model. In the following

lowing section, we look into a full example of applying this methodology.

VII. Transformation Example

From the previous section, we can see that any RTA-DEVS model can be transformed to a behaviorally equivalent TA if we follow the steps shown. To show this process in further detail, we introduce an example of an elevator system [20]. The system is composed of an elevator and its controller, which interacts with the user to receive button requests from each floor. Then, it makes the elevator to respond to user requests. This is an example of a (soft) RTS with safety and bounded response time requirements. To check for these requirements, we applied the previous transformation rules to the DEVS graph models of the elevator system [21].

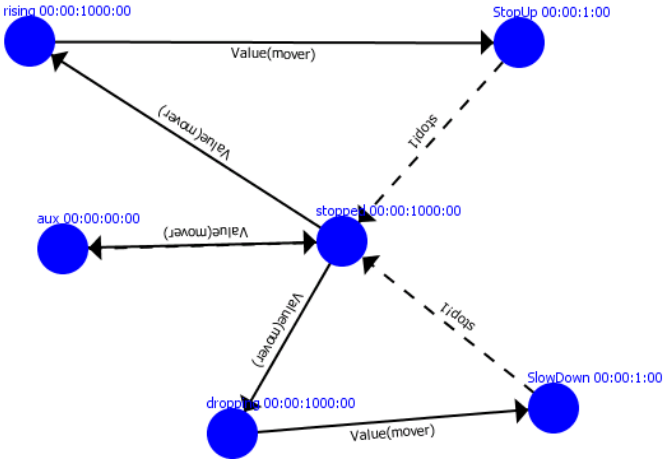

Figure 9: Elevator RTA-DEVS Model

The elevator DEVS graph model in Figure 9 has 5 external transitions shown with solid arrows, and three internal transitions shown with dotted arrows. By taking each transition from the RTA-DEVS model and applying the previous steps, we get the converted TA model shown in Figure 10. Note that an external transition is enabled whenever the expression on that transition evaluates to *true* in RTA-DEVS model. The expression *Value(mover)* evaluates to *true* whenever the elevator model receives a value in *mover* variable equals to *1*. This expression is translated to a channel reception *move?* as shown in TA model in Figure 10. In this case, whenever a value is transmitted on that channel, the transition synchronized on that channel is enabled.

Also note that time advance values for each state in RTA-DEVS model, has been substituted with an equivalent clock invariant in the corresponding state in the TA model, and the constant in that invariant equals state lifetime as indicated on the RTA-DEVS model.

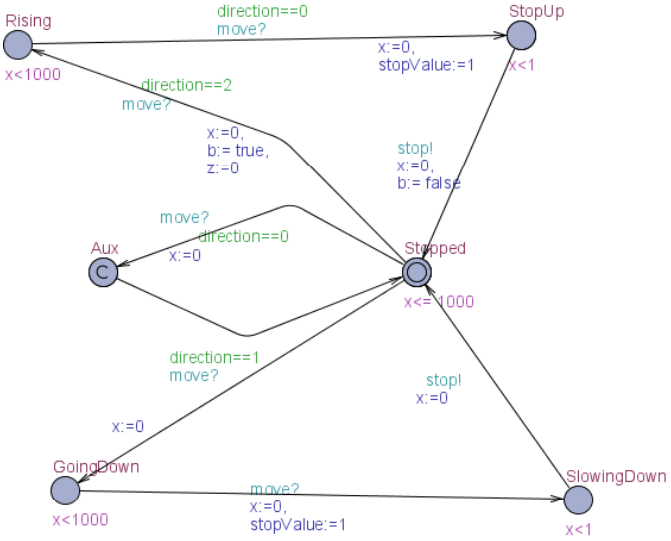

Figure 10: Elevator TA model

The elevator a controller is also responsible to interact with the user and send commands to the elevator to satisfy the user requests. The controller RTA-DEVS model is shown in Figure 11 and represented in DEVS graphs notation.

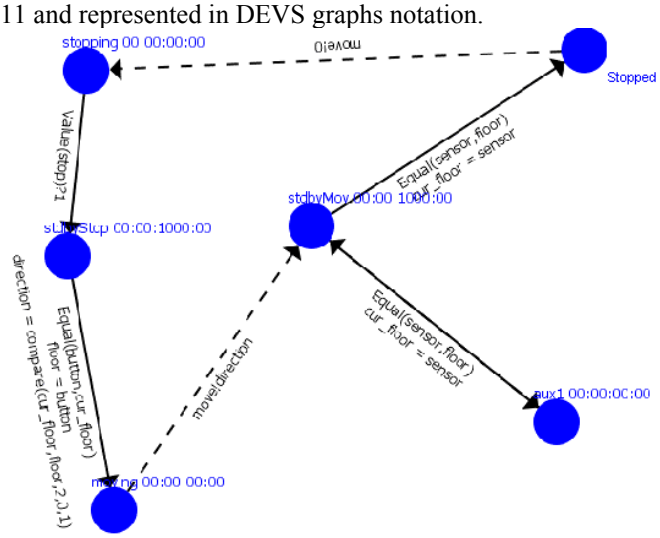

Figure 11: Elevator Controller Model as DEVS Graphs.

We applied the transformation steps above to the elevator controller RTA-DEVS model to obtain TA in Figure 12.

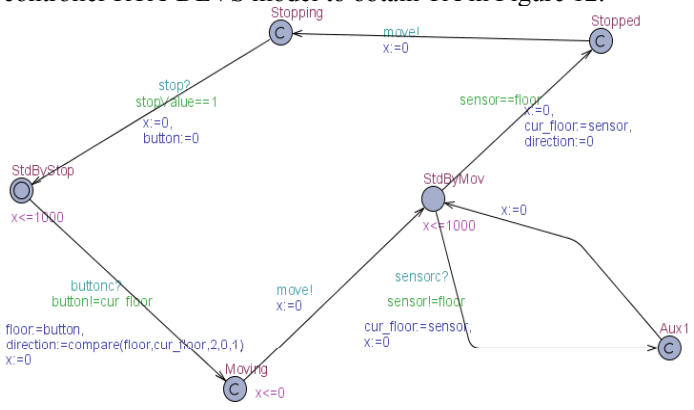

Figure 12: TA Controller model in UPPAAL.

The RTA-DEVS Elevator coupled model is composed of the elevator atomic model and the Controller model. The coupled Elevator model takes as an input a number between 1 and 3 that represents number of floor with button pressed. During coupled Elevator model simulation in DEVS simulator input is read from an input file representing button press.

In [20], we showed how to verify a number of desired properties for the RTA-DEVS model such as deadlock freedom, bounded response time, and safety properties for the elevator coupled model. We used Computational Tree Logic (CTL) to construct queries with the requirements and submitted it to UPPAAL to get an answer and hence verify that requirement. One such requirement is the freedom of deadlocks and expressed in CTL as `A[] not deadlock`. This means for all paths, there should be no deadlocks.

After running the checker, it shows that this property is satisfied, i.e. there is no deadlock in the DEVS model:

```
UPPAAL version 4.0.6 (rev. 2986), March 2007 --
server.
A[] not deadlock
Property is satisfied.
```

Another important check on correct model semantics is to check a user request against the model behavior. For example, if a user presses a specific floor button, the elevator should move to that floor. This is expressed as follows:

```
button == 3 --> ElevatorController.cur_floor == 3
```

Which means if 3rd floor button is pressed; the elevator would *eventually* reach that floor. This query returned true and thus the property is satisfied in our model. However the query

```
button == 3 --> ElevatorController.cur_floor == 4
```

was not satisfied as expected for a correct model, as whenever 3rd floor button is pressed; our simple model should not deliver the elevator to the 4th floor.

VIII. CONCLUSION

We introduced some of the problems that prevents general classic DEVS model from being modeled and verified by TA. We introduced RTA-DEVS to overcome these problems and to offer a subset of DEVS that is still expressive enough for modeling practical problems and that can be transformed to TA. We then could use the transformed TA to reason about the original RTA-DEVS model, and hence about the real problem being modeled. We also showed a methodology to convert RTA-DEVS to TA in a systematic way.

We can compare the expressiveness of RTA-DEVS with TA. As our transformation showed, the equivalent TA to an RTA-DEVS has no diagonal clock constraints of the form $x - y < C$. This is a limitation of general TA with diagonal constraints. However, this does not reduce the expressiveness of RTA-DEVS from general TA formalism as any TA with diagonal constraints can be expressed with a TA without diagonal constraints as shown in [22]. However general TA with diagonal constraints is more concise as shown in [23] than RTA-DEVS (as represented by a TA without diagonal constraints).

We envision that this methodology could influence the M&S in a number of ways, producing models more accurate, and better simulations with less cost and effort. The validation and verification of simulation models, which are done currently in a manual, error-prone procedure and usually needs a domain expert; this technique can improve such activities.

REFERENCES

[1] R. Alur, D. Dill. “Theory of Timed Automata". Theoretical Computer Science, volume 126, pg. 183-235, 1994.

[2] R. Alur. "Model Checking: From Tools to Theory", LNCS 5000, pg. 89–106, 2008.

[3] E. Clarke,"The Birth of Model Checking", Book: 25 Years of Model Checking, pg.1-26 , Springer Berlin / Heidelberg 2008.

[4] BP Zeigler, H. Praehofer, T.G. Kim (2000) Theory of modeling and simulation, 2nd edn. Academic Press, New York.

[5] J.S. Hong, H.S. Song, T.G. Kim, K.H. Park. 1997. “A realtime Discrete Event System Specification formalism for seamless real-time software development”. Discrete Event Dynamic Systems 7 (4): 355-75.

[6] A. Furfaro, L. Nigro, "Embedded Control Systems Design based on RT-DEVS and temporal analysis using UPPAAL", Computer Science and Information Technology, 20-22 Oct. 2008, IMCSIT 2008, p.601-608.

[7] A. Furfaro, L. Nigro."A development methodology for embedded systems based on RT-DEVS", Innovations in Systems and Software Engineering, vol 5, P. 117-127, June 2009.

[8] H.S. Song, T.G. Kim, "Application of Real-Time DEVS to Analysis of safety-Critical Embedded Control Systems: Railroad Crossing Control Example", SIMULATION, 81(2), February 2005. 119-136.

[9] M.H. Hwang, B. P. Zeigler, "Reachability Graph of Finite and Deterministic DEVS Networks", IEEE Transactions On Automation Science And Engineering, 6 (3), July 2009.

[10] S. Han, K. Huang. "Equivalent Semantic Translation from Parallel DEVS Models to Time Automata", ICCS 2007, LNCS 4487, 2007.

[11] N. Giambiasi, J-L. Paillet, F. Châne, "From Timed Automata To DEVS Models". Proceedings of the 2003 Winter Simulation Conference,.

[12] A. Hernandez, N. Giambiasi, "State Reachability for DEVS Models", Proc. of Argentine Symposium on Software Engineering (2005),.

[13] H.Dacharry, N.Giambiasi. "Formal Verification Approaches for DEVS". Proceedings of Summer Computer Simulation Conference. 2007.

[14] K.J. Hong T. G. Kim, "Timed I/O Test Sequences for Discrete Event Model Verification", AIS 2004, LNAI 3397, pp. 275–284, 2005.

[15] Y.Labiche, G. Wainer. “Towards the Verification and Validation of DEVS Models”. *Proceedings of the 1st Open International Conference on Modeling & Simulation*. Clermont-Ferrand, France. 2005.

[16] J. Miller. "Decidability and complexity results for timed automata and semi-linear hybrid automata", Hybrid Systems: Computation and Control, LNCS Vol. 1790, 2000.

[17] L. Aceto, A. Ingólfsdóttir, K. Guldstrand Larsen, J. Srba. “Reactive Systems: Modelling, Specification and Verification”. Cambridge University Press 2007.

[18] Wikipedia. DEVS behavior. http://en.wikipedia.org/wiki/ Behavior_of_Coupled_DEVS [Accessed: Aug. 2009].

[19] J. Bengtsson, W. Yi. “Timed Automata: Semantics, Algorithms and Tools”. Lectures on Concurrency and Petri Nets, Vol. 3098. 2004.

[20] H. Saadawi, G. Wainer. “Verification of Real-Time DEVS Models”, SpringSim'09, San Diego, CA March 2009.

[21] G. Christen, A. Dobniewski, G. Wainer. "Modeling State-Based DEVS Models in CD++ ". Proceedings of MGA, Advanced Simulation Technologies Conference. Arlington, VA. U.S.A. 2004.

[22] B. Bérard, V. Diekert, P. Gastin, A. Petit. “Characterization of the expressive power of silent transitions in timed automata”, Fundamenta Informaticae, vol. 36, num. 2–3, p. 145–182, 1998.

[23] P. Bouyer, F. Chevalier. “On conciseness of extensions of timed automata”, Journal of Automata, Languages and Combinatorics, vol. 10, num. 4, p. 393–405, 2005.