Hybrid Systems Modeling and Verification with DEVS (WIP)

Hesham Saadawi[1] **Gabriel Wainer**[2]
HeshamSaadawi@cmail.carleton.ca gwainer@sce.carleton.ca
[1]School of Computer Science, Carleton University, Ottawa, ON, CANADA
[2]Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, CANADA

Keywords: Hybrid Systems Verification, DEVS, Timed Automata, Quantized State Systems QSS

Abstract
Hybrid systems (where continuous and discrete phenomena interact) are found in many natural and artificial systems. An important example, real-time embedded systems usually include discrete-event controllers interacting with a continuous plant. Verifying these real-time systems for correct behavior is of utmost importance, as results of incorrect behavior are usually catastrophic. To complement the use of Modeling and Simulation study of such hybrid real-time systems, we extend here the verification method, based on RTA-DEVS, hybrid Timed Automata and the QSS that was introduced in [1], which allows verifying real-time hybrid systems modeled by DEVS formalism. This extension allows the transformation of the QSS model into overapproximation TA model using interval arithmetic to solve the limitation introduced by the purely integer arithmetic available in UPPAAL.

I. Introduction

Real-Time (RT) systems are complex computer systems with hardware and software components with timing constraints. These constraints can be "soft" timing (i.e., a deadline can be missed without serious consequences), or "hard" timing constraints (and a missed deadline can result in catastrophic consequences). In these highly reactive systems, correctness is concerned with both system behavior and its ability to execute its tasks in a timely manner. These systems are usually hybrid in nature as they are often composed of a digital computer controller interacting with the physical environment. This implies the need for an analysis methodology that can deal with both discrete and continuous models.

Formal analysis is growing as a methodology to enable the full verification of RT systems. One of the most successful methods of formal verification is *model checking* which enables full automation of the verification process. RT systems have benefited the most from *timed* model checking as with the case of Timed Automata TA [2].

As DEVS is formalism for M&S, modeling the RT system with this formalism, enables testing real life scenarios, even for those cases in which real-life testing might be too costly or impossible to achieve [3]. If the executable models used for M&S are formal, their correctness would also be verifiable, and a designer could see the system evolution and its inner workings even before starting a simulation [3], then these models can be deployed to the target platform, thus giving the opportunity to use the model not only for simulations, but also as the actual implementation deployed on the target hardware. This avoids any new errors that would appear during the implementation from transformation of the verified models into an implementation, thus guaranteeing a high degree of correctness and reliability.

The objective of this paper is to introduce an advanced methodology enabling formal verification of hybrid RT systems modeled with DEVS formalism. This methodology would add the benefit of rigorous formal correctness checking to the current practice of simulating RT hybrid systems. The main contribution is to extend on the transformation we presented in [1] to enable the verification of any general continuous system modeled with QSS through proper transformation to an equivalent TA model. Then, TA can be formally verified with tools such as UPPAAL.This method would deal with issues of infinite continuous state space and the limitations of integer arithmetic in UPPAAL by the use of interval arithmetic to obtain an over-approximated TA model that preserves safety and reachability properties of the original RT system. DEVS ability to be executed on embedded DEVS simulators provides the opportunity to use the verified controller component, from the verified hybrid model, to be the actual implementation code executing on the embedded platform.

II. Background

A. Hybrid DEVS Models

A Major problem in hybrid systems verification is the lack of a unified theory to model and solve both continuous and discrete components together [4]. As a result, modeling and simulation is still one of the most useful methods to verify this kind of systems [5-7]. Hybrid systems simulation was enabled within DEVS formalism by using a method, called Quantizes State Systems (QSS) that will be covered in section B, which allows modeling continuous components [8-10] as discrete event systems.

To use the model checking method to verify hybrid systems, the focus would be to find a suitable *finite abstraction* of the hybrid system that can be verified and hence reachability algorithm is guaranteed to terminate. Different types of labeled transition systems were proposed to model hybrid systems abstractions including Petri Nets [11], hybrid automata and TA [12].

However, as Henzinger et al. shows in [13], Hybrid TA verification through reachability analysis is not decidable in general. For this reason, it would be an advantage to model the hybrid system in some form with a decidable verification such as TA. Many techniques have been proposed to approximate continuous-time systems into a discrete representation of TA [14-17].

This paper uses another technique to represent the continuous system in discrete format using DEVS formalism based on Quantized State Systems (QSS).

B. Quantized State Systems (QSS) Method

In this section, we introduce the QSS method as was explained in [8,9]. The QSS is an approximation method to model and simulate continuous systems, which are usually modeled with Ordinary Differential Equations (ODE) and Algebraic Equations. Traditional method to obtain a detailed description of a system behavior entails solving these equations simultaneously. To do this, a technique of numerical integration is used to solve ODEs such as Euler, Runge-Kutta, etc. All these methods rely on discrete-time integration of ODEs. In this way, time progresses in small steps, and at each step, an approximation is computed for the ODEs solution. When a system modeled by ODEs has a discontinuity (i.e. sudden jumps in its variables values with regard to time), the numerical integration method may produce unacceptable errors [10]. However, these kinds of discontinuity are normal properties in hybrid systems, which can be seen as operating in different modes, each described with a specific ODE, for example, a heating system with an on-off thermostat controller.

Quantized State Systems QSS is a different method for approximation. This is a quantization-based method that models hybrid systems as discrete-event systems and not as discrete-time. This solves the above problem around discontinuities while solving hybrid system as discussed in [9]. We consider here a continuous system modeled by some time-invariant Ordinary differential equation (ODE) and it is in its State Equation System (SES) representation:

$$\dot{x}(t) = f[x(t), u(t)\,)\,] \qquad (Eq.II.1)$$

Where $x(t) \in \boldsymbol{R}^n$ represents the system state vector and $u(t) \in \boldsymbol{R}^m$ represents an input vector, which is a known piecewise constant function, and $\boldsymbol{R}$ is the set of Real numbers. With the QSS method, we simulate an approximate system, which is called Quantized State System:

$$\dot{x}(t) = f[q(t), u(t)\,)\,] \qquad (Eq.II.2)$$

Where *q(t)* is a vector of quantized variables which are obtained by a quantization function *q(t)* from the state variables *x(t)*. Each component of *q(t)* may be related with the corresponding component of *x(t)* by a hysteretic quantization function, as given in [9]. A hysteresis function approximates a continuous linear function $x_i(t)$ by outputting a number of discrete levels. Each level is called a quantization level Q_i. The difference between two successive quantization levels (Q_i, Q_{i+1}) is called the *quantum* (*dq*) and it is usually constant. The crossing of the continuous function to a quantization level generates an output.

A DEVS model that solves (*Eq.II.2*) by integration is called a quantized integrator and can be written as follows [9]:

$M_1 = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$, where: (Eq. II.3)

$X = \Re \times \{\text{inport}\}$

$Y = \Re \times \{\text{outport}\}$

$S = \Re^2 \times Z \times R^+_0 \infty$

$\delta_{int}(s) = \delta_{int}(x, d_x, k, \sigma) =$

$(x + \sigma \cdot d_x, d_x, k + sgn(d_x), \sigma_1)$

$\delta_{ext}(s, e, x_u) = \delta_{ext}(x, d_x, k, \sigma, e, x_v, port) =$

$(x + e \cdot d_x, x_v, k, \sigma_2)$

$\lambda(s) = \lambda(x, d_x, k, \sigma) = (Q_{k+\text{sgn}(dx)}, \text{outport})$

$ta(s) = ta(x, d_x, k, \sigma) = \sigma$

Where

{inport}: The set of input ports.

{outport}: The set of output ports

$$\sigma_1 = \begin{cases} \dfrac{Q_{k+2} - (Q_{k+1})}{d_x} & if\ d_x > 0 \\ \dfrac{Q_k - (Q_{k-1} - \varepsilon)}{|d_x|} & if\ d_x < 0 \\ \infty & if\ d_x = 0 \end{cases} \qquad \text{and}$$

$$\sigma_2 = \begin{cases} \dfrac{Q_{k+1} - (x + e.d_x)}{x_v} & if\ x_v > 0 \\ \dfrac{(x + e.d_x) - (Q_k - \varepsilon)}{|x_v|} & if\ x_v < 0 \\ \infty & if\ x_v = 0 \end{cases}$$

A static function $f(z_1, \ldots, z_p)$ can be represented by the DEVS model:

$M_2 = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$, where (Eq. II.4)

$X = \Re \times \{\text{inport}_1, \ldots, \text{inport}_p\}$

$Y = \Re \times \{\text{outport}\}$

$S = \Re^p \times \Re^+_0 \infty$

$\delta_{int}(s) = \delta_{int}(z_1, \ldots, z_p, \sigma) =$
$(z_1, \ldots, z_p, \infty)$

$\delta_{ext}(s, e, x) = \delta_{ext}(z_1, \ldots, z_p, \sigma, e, x_v, port)$
$= (\tilde{z}_1, \ldots, \tilde{z}_p, \infty)$

$\lambda(s) = \lambda(z_1, \ldots, z_p, \sigma) =$
$(f(z_1, \ldots, z_p, \sigma), outport)$

$ta(s) = ta(z_1, \ldots, z_p, \sigma) = \sigma$

Where

$$\tilde{z} = \begin{cases} x_v & if\ port = inport_j \\ z_j & otherwise \end{cases}$$

As indicated in [9], this combined DEVS model of M_1 and M_2 simulates the QSS system.

Figure 1 shows a DEVS coupled model for a QSS model as defined by (Eq. II.3) and (Eq. II.4). The Model M_1 defines the integrator and the quantization function.

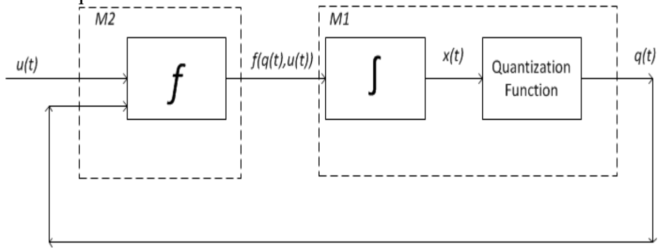

Figure 1. QSS Block Diagram Model

An example of simulating a continuous system with QSS can be shown by using the exponential decay formula which is modeled as follows, using an ODE:

$$dx/dt = -x(t) \qquad (Eq.II.5)$$

Which has the analytical solution $x(t) = e^{-t}$, with the initial condition x(0) =1. Figure 2 shows the exact analytical solution of the exponential decay formula $x(t) = 10\ e^{-t}$ where $x(0) = 10$. This exact solution of *(Eq.II.5)* is approximated with linear segments in a discrete-event form by the following QSS DEVS model:

$AMD = <X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>$ *(Eq.II.6)*

$X = \emptyset$; $S=\{s \mid s=(q,\sigma)\}$; $ta(s) = ta(q,\sigma) = \sigma$

$\delta_{int}(s) = \delta_{int}(q,\sigma) = (q-0.1, 0.1/q)$; $\lambda(q,\sigma) = q$

q: is a quantized variable related to the *x(t)* system variable by a quantization function.

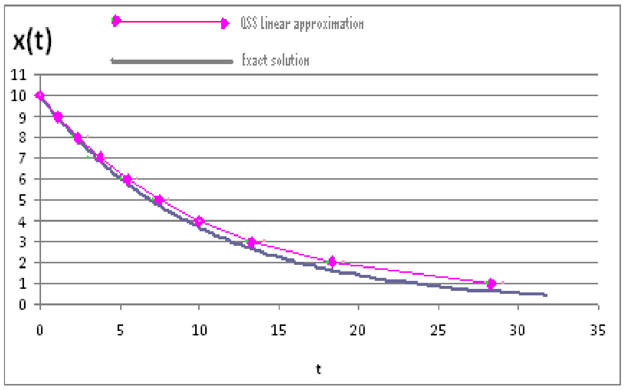

Figure 2. Linear approximation of Decay formula

The QSS linear approximation for the decay formula is shown in Figure 2. Figure 3 shows the quantized representation of the decay formula as a result of simulating this QSS model.

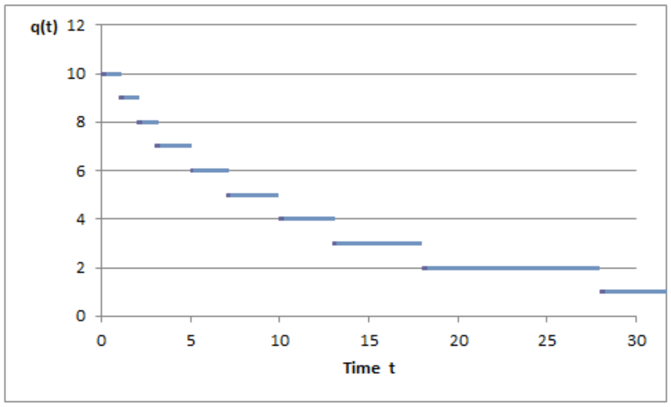

Figure 3. Quantized representation of Exponential Decay.

III. OVER-APPROXIMATING QSS WITH INTEGER INTERVALS

The main contribution of this paper is a novel approach to transform general QSS models to TA by using interval arithmetic [18-20] to solve the limitation of UPPAAL in using only integer arithmetic. To transform a QSS DEVS model (Eq. II.3), (Eq. II.4) to a TA, we need to solve the following two issues:

1. The TA variables can only be of bounded integer type, in order to guarantee the finiteness of state space and the termination of the model-checking algorithm. However, in QSS, state variables are real numbers and thus have infinite values.
2. Time (σ) of next quantum event is approximated to an integer number when transforming to TA. However in doing so we need to preserve the original behavior of QSS and hence the properties we need to formally verify.

First, We can obtain a finite system state-space by incorporating practical system bounds on variable values and identifying the least precision possible when measuring these variables. This would produce the finite set of quantization levels Q. Second, these above two issues are handled by approximating all Real-valued variables in QSS to some equivalent Integer values. When faced with obtaining an Integer approximation to a Real value, we need to round the Real value to the nearest Integer. However, in doing so, we lose the actual value as it was in the QSS model, and this may render TA verification results not applicable to the original QSS system.

To include the actual values of the QSS variables in our approximation, we replace any QSS Real-value with a closed interval between two Integers [L,U], where L is the mathematical floor function of the Real-value, and U is the mathematical ceiling. This would constitute an overapproximation that contains the true trajectory produced by the QSS. This overapproximation, when transformed to a TA model, would contain all possible behaviours in the QSS model. This way, the Overapproximation preserves safety properties, i.e. any proof of a safe overapproximation implies the original system is also safe. However as the overapproximation contains more behaviors than the original system, its verification may produce safety violations that does not exist in the original system. In this case, any violation scenario should also be checked against the original system to confirm it is a real safety violation [21]. An example of applying this overapproximation to the quantized decay formula is shown in Figure 4. In this figure, the line marked with x represents the QSS trajectory, the line marked with c represents the trajectory obtained by taking the ceiling $\lceil \sigma \rceil$ of the QSS trajectory, and the line marked with f represents the trajectory obtained by taking floor $\lfloor \sigma \rfloor$ of the QSS trajectory.

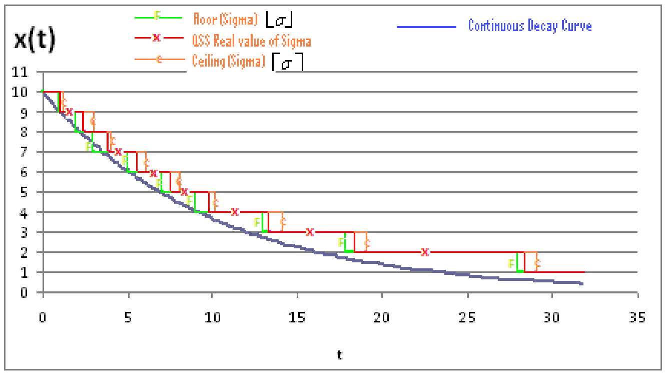

Figure 4. Overapproximation of QSS trace.

To get the overapproximated QSS model, all calculations would then use interval arithmetic. Thus, the QSS system of (Eq. II.3) may be written as:

$$M_{1OA} = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta), \text{ where} \qquad \text{(Eq.III.1)}$$

$X = Z^2 \times \{\text{inport}\}$

$Y = Z \times \{\text{outport}\}$

$S = Z^4 \times Z \times Z^{2\,+}_{\;0}\infty$

$\delta_{int}(s) = \delta_{int}([x_L, x_U],\ [d_{xL}, d_{xU}],\ k,\ [\sigma_L, \sigma_H]) =$
$\quad (Q_{k+1}, [d_{xL}, d_{xU}], k + sgn(d_{xL}), [\sigma_{1L}, \sigma_{1H}])$

$\delta_{ext}(s, e, x_u) = \delta_{ext}(x, d_x, k, \sigma, e, x_v, port) =$
$\quad ([x_L, x_U], [x_{vL}, x_{vL}], k, [\sigma_{2L}, \sigma_{2H}])$

$\lambda(s) = \lambda(x, d_x, k, \sigma) = (Q_{k+sgn(dx)}, \text{outport})$

$ta(s) = ta(x, d_x, k, \sigma) = [\sigma_L, \sigma_H]$

$$x_L = x + e \bullet d_{xL}$$
$$x_U = x + e \bullet d_{xU}$$

$$\sigma_{1L} = \begin{cases} \dfrac{Q_{k+2} - (Q_{k+1})}{d_{xU}} & \text{if } d_{xU} > 0 \\ \dfrac{Q_k - (Q_{k-1} - \varepsilon)}{|d_{xU}|} & \text{if } d_{xU} < 0 \\ \infty & \text{if } d_{xU} = 0 \end{cases} \qquad \text{(Eq. III.2)}$$

$$\sigma_{1U} = \begin{cases} \dfrac{Q_{k+2} - (Q_{k+1})}{d_{xL}} & \text{if } d_{xL} > 0 \\ \dfrac{Q_k - (Q_{k-1} - \varepsilon)}{|d_{xL}|} & \text{if } d_{xL} < 0 \\ \infty & \text{if } d_{xL} = 0 \end{cases}$$

$$\sigma_{2L} = \begin{cases} \dfrac{Q_{k+1} - (x_L + e.d_{xL})}{x_{vL}} & \text{if } x_{vL} > 0 \\ \dfrac{(x_L + e.d_{xL}) - (Q_k - \varepsilon)}{|x_{vL}|} & \text{if } x_{vL} < 0 \\ \infty & \text{if } x_{vL} = 0 \end{cases} \qquad \text{(Eq. III.3)}$$

$$\sigma_{2U} = \begin{cases} \dfrac{Q_{k+1} - (x_U + e.d_{xU})}{x_{vL}} & if\ x_{vU} > 0 \\ \dfrac{(x_U + e.d_{xU}) - (Q_k - \varepsilon)}{|x_{vU}|} & if\ x_{vU} < 0 \\ \infty & if\ x_{vU} = 0 \end{cases}$$

and from (Eq.II.2) we calculate current function slope as:

$$d_{xL} = \lfloor f[q(t), u(t)] \rfloor \qquad \text{(Eq. III.4)}$$
$$d_{xU} = \lceil f[q(t), u(t)] \rceil$$
$$x_{vL} = \lfloor f[x_L(t), u(t)] \rfloor$$
$$x_{vU} = \lfloor f[x_U(t), u(t)] \rfloor$$

Another obstacle in representing general QSS models with TA is the nature of the function *f* of (Eq.II.2). In UPPAAL, users can define functions with syntax close to that of the C language. However, the available operators are limited to the primary mathematical operators. Furthermore, operands and expression results are all integers. This puts a limit on the functions that can be expressed in UPPAAL TA. Similar restrictions on the system dynamics also exist with other formalisms for hybrid systems verification such as Linear Hybrid Automata [22], where derivatives are limited by linear constraints. To model QSS systems whose dynamics are described by complex function, the modeller would need to approximate the complex function with a polynomial formula that uses only preliminary operators of UPPAAL. This approximation may use one of the series expansion methods such as the Maclaurin Series, and can be done up to the desired precision. This approximation is similar to Taylor models as defined in [23].

IV. TA MODEL OF GENERAL QSS.

The integrator automaton representing the QSS model of (Eq.III.1) is shown in figure 5. The structure of this automaton is fixed for any QSS system, and thus can be considered as a template representing the DEVS model of (Eq.III.1). This template has the following parameters that are part of the QSS model definition:

- A set of defined quantization levels Q = {$Q_0, Q_1, \ldots Q_r$}, where Q_0 is the initial level, Q_r is the final level.
- The quantum $dQ = Q_{k+1} - Q_k$, such that $0 \leq k < r$
- Hysteresis value ε
- System defined functions: $f[x(t), u(t)]$, $x(t)$, and $u(t)$.

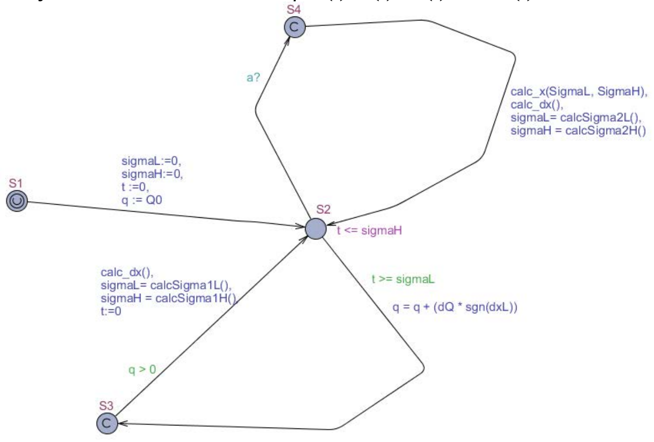

Figure 5. TA model of a QSS general Integrator

This automaton starts in S1, and on the transition from S1 to S2, it initializes the variables sigmaL, sigmaH, *q*, and clock t. The internal transition function $\delta_{int}(s)$ of (Eq.III.1) is simulated with transitions S2 → S3, and S3 → S2. At S2, the automaton waits for a time *t*, where $\sigma_{1L} \leq t \leq \sigma_{1H}$, then transits to S3. On this transition, value of *q* is updated as $Q_{k+sgn(dx)} = q + (dQ * sgn(d_x))$, which is equivalent to the expression $(x + \sigma \cdot d_x)$ of (Eq. II.3). Then, on transition S3 → S2, an updated values of d_{xL}, d_{xU} are calculated according to (Eq. III.4), then values [σ_{1L}, σ_{1H}] are calculated according to (Eq. III.2). Clock t is also reset to zero on the transition S3 → S2 before waiting in S2 for the next event. The output function $\lambda(s)$ is implicit in the TA model, as the value of shared variable *q* can be read by other models coupled with this TA model. The definitions of UPPAAL functions of these calculations are shown in table 1. These functions are described as follows:

- calc_dx(): Calculates lower & upper bound of dx
- f(int x,int v): Calculates the slope from the system de-fined function.
- calcSigma1H() , calcSigma1L(): These functions calculate the new upper & lower bounds of Q based on its current value and the quantum dQ in the internal transition.

Table 1. User defined functions for the UPPAAL TA model of figure 5

int sgn(int dx){
if (dx > 0)
  return 1;
else
  return -1;
}
	void calc_dx(){
	 dxL = f(xL, v);
	 dxU = f(xU, v);
	}
		int f(int x,int v){
		int f = x + v;
		return f;
		}

int calcSigma1L(){
int sigmaL;
if (dxU > 0)
  sigmaL = roundDownDiv (dQ, dxU);
else if (dxU < 0)
      sigmaL = roundDownDiv (dQ - epsilon, -dxU);
     else
      sigmaL = 32767; //infinity in UPPAAL int type.
return sigmaL;
}		

int calcSigma1H(){
int sigmaH;
 	
if (dxL > 0)
  sigmaH = roundUpDiv (dQ, dxL);
else if (dxL < 0)
      sigmaL = roundUpDiv (dQ - epsilon, -dxL);
     else
      sigmaL = 32767; // this represents infinity
                        //in UPPAAL int type.
return sigmaH;
}		

To simulate the external transition function $\delta_{ext}(s, e, x_u)$ of (Eq.III.1), we use the transitions S2 → S4 → S2. Transition S2 → S4 is enabled only if the automaton receives a synchronization input on channel a. This synchronization comes from the automaton generating the function *u(t)* as shown in figure 6, and would be described later. When this synchronization happens, the automaton moves to the committed state S4, and then, without delay, it executes the transition S4 → S2. On this latter transition, the new values of [x_L, x_U] are calculated based on the shared variables SigmaL, SigmaH which are passed from the automaton of figure 6. New values of [x_{vL}, x_{vL}] are also calculated to represent a new slope value. Finally, new values for [σ_{2L},σ_{2H}] are calculated. The UPPAAL functions that we defined for these calculations are shown in table 2. The functions in this table are described as follows:

- calc_x(int SigmaL, int SigmaH): calculates current value of x from elapsed time which is in the interval [SigmaL,SigmaH] passed from the external input, and the slope dx.

- calcSigma2L(), calcSigma2H(): calculates the lower, and upper bounds of sigma in the external transition.

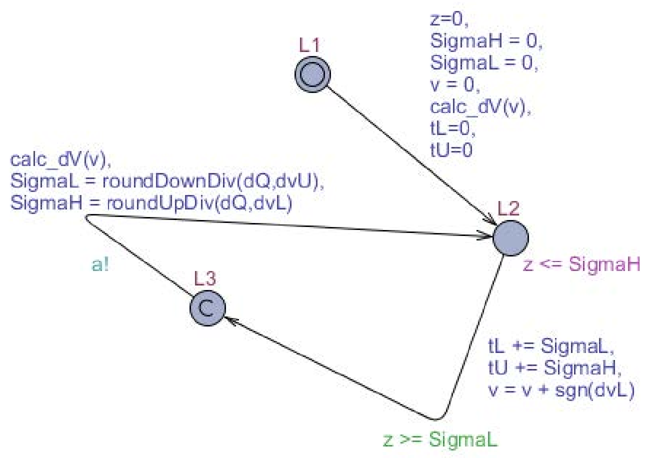

Figure 6. TA Model of a QSS Input function generation

For a general QSS system where $u(t) \neq \varphi$, the QSS model would have an external transition function to process the input $u(t)$. $u(t)$ is a constant input step function as defined in QSS [9]. To simulate a system described with ODE's as (Eq.II.1), the input $u(t)$ needs to be generated by a DEVS QSS model. This DEVS QSS model does not have external inputs, and it generates a sequence of quantized events to represent the step function $u(t)$. Such a system can be described by an overapproximated QSS model Mu_{OA} where we use closed integer intervals instead of Real numbers, for the same reason we did with the integrator QSS model above. The overapproximated model of the input function is shown in (Eq. IV.1) - (Eq. IV.3). As this model has no external input, then we know, from the semantics of QSS method, that the state variable u would always take a value from the set Q of the quantization levels. Thus the term $[u_L, u_U]$, of the state, would be the current quantization level Q_{k+1}.

$Mu_{OA} = (X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta)$, where (Eq. IV.1)

$X = \varphi$

$Y = Z \times \{outport\}$

$S = Z^4 \times Z \times Z^{2}\ {}^{+}{}_{0}\infty$

$\delta_{int}(s) = \delta_{int}([u_L, u_U],\ [d_{uL}, d_{uU}],\ k,\ [\sigma_L, \sigma_H]) = (Q_{k+1}, [d_{uL}, d_{uU}], k + sgn(d_{uL}), [\sigma_{1L}, \sigma_{1H}])$

$\delta_{ext}(s, e, x_u) = \varphi$

$\lambda(s) = \lambda(x, d_u, k, \sigma) = (Q_{k+\mathrm{sgn}(du)}, \text{outport})$

$ta(s) = ta(x, d_u, k, \sigma) = [\sigma_L, \sigma_H]$

$$\sigma_{1L} = \begin{cases} \dfrac{Q_{k+2} - (Q_{k+1})}{d_{uU}} & if\ d_{uU} > 0 \\ \dfrac{Q_k - (Q_{k-1} - \varepsilon)}{|d_{uU}|} & if\ d_{uU} < 0 \\ \infty & if\ d_{uU} = 0 \end{cases} \qquad \text{(Eq. IV.2)}$$

$$\sigma_{1U} = \begin{cases} \dfrac{Q_{k+2} - (Q_{k+1})}{d_{uL}} & if\ d_{uL} > 0 \\ \dfrac{Q_k - (Q_{k-1} - \varepsilon)}{|d_{uL}|} & if\ d_{uL} < 0 \\ \infty & if\ d_{uL} = 0 \end{cases}$$

and current function slope as:

$$d_{uL} = \lfloor u(t) \rfloor \qquad \text{(Eq. IV.3)}$$
$$d_{uU} = \lceil u(t) \rceil$$

Table 2. User defined functions for the UPPAAL TA model of figure 6

```
void calc_x(int SigmaL, int
SigmaH ){

//xL: Lower integer bound of
//x variable

 xL += SigmaL * dxL;

// xU:Upper integer bound of
//x variable

 xU += SigmaH * dxU;
}
```

```
void calc_dx(){

 dxL = f(xL, v);
 dxU = f(xU, v);

}
```

```
int calcSigma2L(){

int sigmaL, xvL;
xvL = calc_xv(xL);
if (dxL > 0)
  sigmaL = roundDownDiv ((q + dQ) - xL, xvL);
else if (dxL < 0)
  sigmaL = roundDownDiv ( xL - (q - epsilon) , -
xvL);
     else
      sigmaL = 32767;//infinity for int type.
return  sigmaL;
}
```

```
int calcSigma2H(){

int sigmaH, xvU;
xvU = calc_xv(xU);

if (dxL > 0)
  sigmaL = roundUpDiv ((q + dQ) - xL, xvU);
else if (dxU < 0)
    sigmaH = roundUpDiv ( xU - (q - epsilon) , -
xvU);
     else
      sigmaH = 32767;      //infinity for int type.
return  sigmaH;
}
```

In figure 6, the transition L1→ L2 initializes clock z, the lower and upper time constraints $[\sigma_{1L}, \sigma_{1H}]$, the quantized output variable v , and calculates the initial slope from the function definition of $u(t)$ by invoking user defined function *calc_dV(v)*. The slope is calculated as a closed integer interval [*dvL,dvU*] to represent the overapproximation of slope value, as defined in (Eq. IV.3). $\delta_{int}(s)$ is simulated by transitions L2→L3→L2. On these two transitions, new value of v is calculated, the function *calc_dV(v)* calculates the values of $u(t)$, then the new values of next event timing is calculated $[\sigma_{1L}, \sigma_{1H}]$. We note here that to calculate $u(t)$ we need the current value of time t. At any time during the model execution, the total elapsed time t after j number of internal transitions is given by:

$$t_j = \sum_{0 \le i \le j-1} \sigma_i$$

And as we have an estimate of σ in the interval $[\sigma_{1L}, \sigma_{1H}]$, we get:

$$t_{j_L} = \sum_{0 \le i \le j-1} \sigma_{iL} \quad , \quad t_{j_U} = \sum_{0 \le i \le j-1} \sigma_{iU}$$

This gives an estimate of time at current iteration j as shown on transition L2→L3 with the integer interval $[t_L, t_U]$.

On the transition from L3 to L2, after the automaton calculates the next values of shared variables SigmaL and SigmaH, it synchronizes on channel "a" to the integrator automaton shown in figure 5 , so the latter can read the values of v, SigmaL, and SigmaH which are used in the external transition definition to recalculate a new function slope x_v.

V. Conclusion

We showed a methodology to verify hybrid DEVS models. This is an extension of previous results verifying discrete DEVS [1][24-26], and this was obtained by using QSS method to model continuous components in a discrete representation. Enabling approximation of system dynamics in timed automata models using QSS opens the door to more complex verification queries and better controller designs. For example, the following research directions can be built on results of hybrid systems verification using QSS:

- System dynamics are presented in a fine-grain in the TA model. This would enable verification of more advanced types of controllers than an On-Off controller. For example, advanced control algorithms could use the information about the state variable change with respect to time.
- TA controller synthesis techniques could use the fine-grained information of system dynamics to synthesis advanced controllers based on QSS environment models.

Some limitations, however, for this method of overapproximation is that for systems described with nonlinear derivatives, the overapproximation can lead to a wide flow pipe around the actual system trajectory. This can lead to more spurious safety violations because of this wide overapproximation. Other limitation is the inherit problem with model checking technique of state-space explosion that limits the ability to scale verification to larger models.

References

[1] H.Saadawi, G. Wainer. 2012. “On the verification of hybrid DEVS models”. In Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium (TMS/DEVS '12), Orlando, FL,USA. March 26-28.

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

[3] G. Wainer, E. Glinsky, and P. MacSween. 2005. “A Model-Driven Technique for Development of Embedded Systems Based on the DEVS Formalism”. In *Model-driven Software Development - Volume II of Research and Practice in Software Engineering*, edited by S. Beydeda and V. Gruhn. Springer-Verlag.

[4] M. S Branicky. 2005. “Introduction to Hybrid Systems” D. Hristu-Varsakelis and W.S. Levine (eds.), *Handbook of Networked and Embedded Control Systems*, 91-116. Boston: Birkhauser.

[5] A. Donzé, O. Maler. 2007. “Systematic simulation using sensitivity analysis”. In *Proceedings of the 10th international conference on Hybrid systems: computation and control (HSCC'07)* :174-189.

[6] A. Donzé. 2007. “Trajectory-Based Verication and Controller Synthesys for Continuous and Hybrid Systems”. PhD thesis, University Joseph Fourier.

[7] A. Donzé, B. Krogh, and A. Rajhans. 2009. “Parameter synthesis for hybrid systems with an application to simulink models”. In *Proceedings of the 12th International Conference on Hybrid Systems : Computation and Control (HSCC'09)*, San Francisco, CA, USA, April 13-15, 2009.

[8] E. Kofman, S. Junco. 2001."Quantized State Systems. A DEVS Approach for Continuous Systems Simulation". Transactions of SCS. 18(3): 123-132.

[9] E. Kofman. 2004. "Discrete Event Simulation of Hybrid Systems". SIAM Journal on Scientific Computing 25(5): 1771-1797.

[10] M. Otter, F. Cellier. 1996. The Control Handbook, chapter Software for Modeling and Simulating Control Systems, 415–428. CRC Press, Boca Raton, FL.

[11] G Decknatel, R. Slovák, E. Schnieder. 2002. “Definition of a Type of Continuous-Discrete High-Level Petri Nets and Its Application to the Performance Analysis of Train Protection Systems In S. Engell, G. Frehse, and E. Schnieder (Eds.), *Modelling, Analysis, and Design of Hybrid Systems, Lecture Notes in Control and Information Sciences* 279: 355–367.

[12] S. Kowalewski. 2002. “Introduction to the Analysis and Verification of Hybrid Systems”. *Modelling, Analysis, and Design of Hybrid Systems*. Lecture Notes in Control and Information Sciences, 279: 153-171.

[13] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. 1995. “What's decidable about hybrid automata?”. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, STOC ’95,* New York, NY, USA, 373–382.

[14] J. Lunze and J. Raisch. 2002. “Discrete Models for Hybrid Systems. Modelling, Analysis, and Design of Hybrid Systems”. *Lecture Notes in Control and Information Sciences*, 279: 67-80.

[15] R. Alur, T.A. Henzinger, G. Lafferriere, G.J. Pappas. 2000. “Discrete abstractions of hybrid systems”. *Proceedings of the IEEE,* 88(7): 971-984.

[16] E. Barke, D. Grabowski, H. Graeb, L. Hedrich, S. Heinen, R. Popp, S. Steinhorst, and Y. Wang. 2009. “Formal approaches to analog circuit verification”. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '09),* European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 724-729.

[17] Oded Maler and Grégory Batt. 2008. “Approximating Continuous Systems by Timed Automata”. In *Proceedings of the 1st international workshop on Formal Methods in Systems Biology (FMSB '08)*, Cambridge, UK, Jasmin Fisher (Ed.). Springer-Verlag, Berlin, Heidelberg, 77-89.

[18] B. Hayes, 2003."Lucid Interval", Scientific American, November-December 2003, 91(6): 484.

[19] J. G. Rokne. 2001." Interval arithmetic and interval analysis: an introduction". In Granular computing, Witold Pedrycz (Ed.). Physica-Verlag GmbH, Heidelberg, Germany.

[20] R. E. Moore, R. B. Kearfott, and M. J. Cloud, 2009."Introduction to Interval Analysis". Society for Industrial and Applied Math., Philadelphia, PA, USA.

[21] B. Berard, , M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen and P. McKenzie. 2001. Systems and Software Verification: Model-Checking Techniques and Tools. Springer Verlag.

[22] T. Henzinger. 1996. “The Theory of Hybrid Automata”. *Lecture Notes in Computer Science* 278.

[23] X. Chen, E. Abraham, S. Sankaranarayanan, 2012, "Taylor Model Flowpipe Construction for Non-linear Hybrid Systems," In *Proceedings of Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd* , 183-192, San Juan, PR, USA, Dec. 4-7.

[24] H. Saadawi, G. Wainer. 2009. “Verification of real-time DEVS models”. In *Proceedings of DEVS Symposium 2009*. San Diego, CA, March 22 – 27.

[25] H. Saadawi, G. Wainer. 2010. “Rational time-advance DEVS (RTA-DEVS). In *Proceedings of DEVS Symposium 2010*, Orlando, FL., April 11-15.

[26] H. Saadawi, G. Wainer. 2010. “From DEVS to RTA-DEVS”. In Proceedings of the 2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications (DS-RT '10). IEEE Computer Society, Washington, DC, USA, 207-210.