Bhave Toolset: Modeling and Analysis of Electronic System Design and System Control

K.L. Man[*] T. Krilavičius[†] Š. Valaškevičius[‡] Yanyan Wu[§] and J.K. Seon[¶]

Abstract— **Behavioral Hybrid Process Calculus (BHPC) is a formalism for Modeling and analysis of hybrid systems combining process algebras and the behavioral approach for Modeling of instantaneous changes and continuous evolution. BHPC is supported by Bhave toolset, containing a tool for a novel way of visualization of hybrid systems simulations msp-svg and a new version of hybrid simulator. In this paper, we present the latest developments of Bhave toolset and apply it for case studies of system control and electronic system design.**

Keywords: formal methods, electronic system design, hybrid systems, simulation

1 Introduction

Process algebras/calculi [4, 13, 22] are formal languages in *Computer Science* that have formal syntax and semantics for specifying and reasoning about different systems. In simple words, process algebras are theoretical frameworks for formal specification and analysis of the behavior of various systems. Serious efforts have been made in the past to deal with various systems (e.g. *discrete event systems* [24, 29], *real-time systems* [12, 14, 30] and *hybrid systems* [3, 5–8, 19, 31]) in a process algebraic way. Over the years, process algebras have been successfully used in a wide range of problems and in practical applications in both academia and industry for analysis of many different systems.

Hybrid systems are systems that exhibit both discrete and continuous behavior. Such systems have proved fruitful in a great diversity of engineering application areas including air-traffic control, automated manufacturing, chemical process control and system control. On the other hand, mathematically, the behavior of electronic system design (e.g. digital, analogue and mixed-signal design) can be described by discrete variables, continuous variables and a set of differential equations, whereas switching-modes can be used for modeling mixed models (i.e. mixed-signal design). Due to all these, digital, analogue and mixed-signal design can be mathematically described as hybrid systems (with various level of abstraction) by nature.

Computer simulation is a powerful tool for analyzing and optimizing real-world systems with a wide range of successful applications. It provides an appealing approach for the analysis of dynamic behavior of processes and helps decision makers identify different possible options by analyzing enormous amounts of data.

Behaviural Hybrid Process Calculus (BHPC) [19] is a hybrid process algebra which was specifically designed for the description of the dynamic behavior of hybrid systems along with a powerful simulator called BHAVE TOOLSET. Currently, simulation results obtained by means of the BHPC simulator can also be visualized and analyzed via *Message Sequence Plots* (MSP) [19].

In this paper, we first present the latest development of BHAVE TOOLSET. Through case studies, we show the use of BHAVE TOOLSET for addressing several aspects of system control and mixed-signal design. Related work of the research activities presented in this paper can be found at [19–21, 34].

2 Behavioral Hybrid Process Algebra

One of the useful techniques for simulation of hybrid systems that includes continuous evolution and discrete changes, is Behavioral Hybrid Process Calculus (BHPC) [6, 19], an extension of classical process algebra that is suitable for the modeling and analysis of continuous and hybrid dynamical systems and can be seen as a generalization of the behavioral approach [26] in a hybrid setting. The main strengths of the BHPC are the following.

Sound mathematical foundations. BHPC has sound mathematical foundations. It means that rigorous reasoning can be applied to investigate diverse properties of models.

Behavioral approach. In BHPC continuous evolution

[*]Xi'an Jiaotong-Liverpool University (XJTLU), 111 Ren'ai Road, Suzhou, Jiangsu 215123, China. E-mail: ka.man@xjtlu.edu.cn. Tel: +86 512 8816 1509. Fax: +86 512 8816 1899.

[†]Vytautas Magnus University (VDU), Vileikos 8, Kaunas, LT-44404, Lithuania. E-mail: t.krilavicius@if.vdu.lt.

[‡]Vytautas Magnus University (VDU), Vileikos 8, Kaunas, LT-44404, Lithuania. E-mail: rakatan@gmail.com.

[§]Xi'an Jiaotong-Liverpool University (XJTLU), 111 Ren'ai Road, Suzhou, Jiangsu 215123. E-mail: yanyan.wu@xjtlu.edu.cn.

[¶]LS Industrial Systems, 1026-6,Hogye-dong,Dongan-gu,Anyang-si,Gyeonggi-do 431-848, South Korea. E-mail: jkseon@lsis.biz.

is defined in the behavioral setting [26] making it more general in contrast to other hybrid process algebras (Hybrid χ [31], HyPA [8], $\mathrm{ACP}_{\mathrm{hs}}^{\mathrm{srt}}$ [5]), i.e. it is defined using trajectories (solutions of differential equations is one of ways of defining trajectories), not just (solutions of) differential equations.

Separation of concerns. Continuous and discrete behaviors are specified orthogonally, therefore they can be changed and analyzed separately as well as in hybrid setting.

Bisimulation is congruence in BHPC, i.e. substituting bisimilar (processes, that exhibit the same observable behavior up-to the branching structure) does not change behavior of the system.

Tools support. BHPC is supported by Bhave toolset, see Section 3.

We present main ideas of the BHPC in this section, see [19] for the details.

2.1 Trajectories

We define trajectories over bounded time intervals $(0, t]$, and map them to a *signal space* $\mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \ldots, q_n))$. Components of the signal space $W \in \mathcal{W}$ correspond to the different aspects of the continuous-time behavior, such as current or voltage, and are associated with *trajectory qualifiers* $q_i \in \mathcal{T}$ identifying them. A *trajectory* in signal space $\mathbb{W}$ is a function $\varphi : (0, t] \to W_1 \times \cdots \times W_n$, where $t \in \mathbb{R}_+$ is the duration of the trajectory. We define conditions on the *endpoints* of trajectories or the *exit conditions*. $\Downarrow$ denotes such conditions, as the restrictions on the set of trajectories: $\Phi \Downarrow \mathcal{P}red_{\mathrm{exit}} = \{\varphi : (0, u] \to W_1, \ldots, W_n \in \Phi \mid \mathcal{P}red_{\mathrm{exit}}(\varphi(u))\}$, where u is a time parameter, Φ is a set of trajectories and $\mathcal{P}red_{\mathrm{exit}}(\varphi(u))$ is a predicate that defines restrictions. The set of trajectories Φ can be defined in different ways, e.g. by ODE/DAE. See [19] for the formal treatment.

2.2 Hybrid transition system

All behaviors of BHPC specification are defined by a *hybrid transition system* $HTS = \langle S, \mathcal{A}, \to, \mathbb{W}, \Phi, \to_c \rangle$

- S is a state space.
- $\mathcal{A}$ is a *finite set of (discrete) actions names.*
- $\to \subseteq S \times \mathcal{A} \times S$ is a *discrete transition relations*, where $\mathsf{a} \in \mathcal{A}$. We will denote it $s \xrightarrow{\mathsf{a}} s'$.
- $\mathbb{W}$ is a *signal space.*
- Φ is a *set of trajectories.*
- $\to_c \subseteq S \times \Phi \times S$ is a *continuous transition relation*, where $\varphi \in \Phi$ are trajectories. We will denote continuous transitions $s \xrightarrow{\varphi} s'$ for the convenience.

2.2.1 Bisimulation

One of the main tools to compare systems is *strong bisimulation.* Strong bisimulation requires both subsystems to be able to imitate each other at every step. A strong bisimulation for hybrid transition systems requires both systems to be able to execute the same trajectories and actions and to have the same branching structure. A binary relation $\mathcal{R} \subseteq S \times S$ on the states is a *hybrid strong bisimulation*, if for all $p, q \in S$, such that $p \,\mathcal{R}\, q$, holds

$$
\begin{aligned}
p \xrightarrow{\mathsf{a}} p' &\Longrightarrow \exists q' \text{ such that } q \xrightarrow{\mathsf{a}} q' \text{ and } p' \,\mathcal{R}\, q' \\
q \xrightarrow{\mathsf{a}} q' &\Longrightarrow \exists p' \text{ such that } p \xrightarrow{\mathsf{a}} p' \text{ and } p' \,\mathcal{R}\, q' \\
p \xrightarrow{\varphi} p' &\Longrightarrow \exists q' \text{ such that } q \xrightarrow{\varphi} q' \text{ and } p' \,\mathcal{R}\, q' \\
q \xrightarrow{\varphi} q' &\Longrightarrow \exists p' \text{ such that } p \xrightarrow{\varphi} p' \text{ and } p' \,\mathcal{R}\, q'.
\end{aligned}
$$

The first two statements define bisimulation requirements for the discrete actions, and the last two for the continuous-time transitions.

States p and q are *bisimilar* (denoted $p \sim q$), if there exists a hybrid strong bisimulation $\mathcal{R}$, containing the pair (p, q).

2.3 Language

A core language is used for defining evolution and interaction of systems

$$
B ::= \mathbf{0} \;\Big|\; \mathsf{a} \,.\, B \;\Big|\; [f \mid \Phi] \,.\, B \;\Big|\; \sum_{i \in I} B_i \;\Big|\; B \parallel_A^H B \;\Big|\; P
$$

We will require a *consistent signal flow*, i.e. only the parallel composition is allowed to change the set of trajectory qualifiers in the process.

Only a subset of complete language is presented in this paper, see [19] for auxiliary operators, such as renaming or hiding. Moreover, other operators can be defined on top of the core language for convenience. We demonstrate it by introducing *parametrized action prefix* and *guard.*

Deadlock 0 *Deadlock* is the process that does not exhibit any behavior.

Action-prefix $\mathsf{a} \,.\, B$ *Action prefix* performs a and continues as B. A special *silent action* τ defines directly unobservable behavior, and is usually used to specify a non-determinism (e.g. as *internal actions* in [22, p. 37–43]).

$$\mathsf{a} \,.\, B \xrightarrow{\mathsf{a}} B \tag{1}$$

We will use parametrization of action prefix as in [22, p. 53–58]

$$\mathsf{a}(v : V) \,.\, B(v) = \sum_{v \in V} \mathsf{a}(v) \,.\, B(v) \tag{2}$$

Trajectory-prefix $[f \mid \Phi] \;.\; B\,(f)$ *Trajectory prefix* $[f \mid \Phi] \;.\; B\,(f)$, where f is a trajectory variable, starts with a trajectory or a prefix of a trajectory from the set of trajectories Φ. If a trajectory or a part of it was taken and there exists a continuation of the trajectory, then the system can continue with a trajectory from the set of such continuations. If a whole trajectory (e.g., as defined by exit conditions) was taken, then the system can continue with B.

Let ; be a trajectories concatenation function such that

$$\varphi \,;\psi(t') = \begin{cases} \varphi(t'), & 0 < t' \leq t \\ \psi(t' - t), & t < t' \leq t + u \end{cases}$$

Let $\setminus$ denotes *continuation* of trajectory after taking an initial part of it. Let φ and ψ be trajectories of duration t_φ and t_ψ, respectively, such that $t_\varphi > t_\psi$, and up-to t_ψ both trajectories coincide. Then $\varphi \setminus \psi$ is a continuation of φ after taking ψ with duration $t_\varphi - t_\psi$ and coincides with a part of φ from t_ψ to t_φ shifted to the left (zero) by t_ψ. See [19] for the formal treatment of trajectories.

Then we can define trajectory prefix as follows

$$[f \mid \Phi].B\,(f) \xrightarrow{\varphi} [f' \mid \Phi \setminus \varphi].B\,(\varphi \,;\, f') \text{ for all } \varphi \in \overline{\Phi}^{+} \tag{3}$$

where Φ is a set of trajectories such that both φ and ψ contain the same trajectory qualifiers, and $\overline{\Phi}^{+}$ is a set of all possible non-empty trajectories and trajectory prefixes from Φ. Notice that ϵ can be in Φ, because it complies with any qualifier type. f, f' are trajectory variables. Let ψ be a trajectory and φ is taken from (3), then $\varphi;\psi \in \Phi$ or $\varphi \in \Phi$ such that φ is finite and non-empty. If a trajectory or a part of it was taken and there exists a continuation of the trajectory, then the system can continue with a trajectory from the trajectory continuations set. However, if a whole trajectory was taken, then the system may continue with the consecutive process with the substituted trajectories (see (4) and (5)). $(\varphi \,;\, f')$ defines substitution of the taken trajectories in the following processes, i.e., all instances of f in B are substituted by the taken trajectory φ concatenated with its follow-up f', or if it is finished, by the whole taken trajectory φ.

After defining concatenation (4) and (5) we present a derived rule that explains behavior of the empty trajectory ϵ in trajectory prefix.

We will extend notation to make use of trajectory prefix more convenient

$$[q_1, \ldots, q_m \mid \Phi \downarrow \mathcal{P}red \Downarrow \mathcal{P}red_{\text{exit}}]$$

where

- $q_1, \ldots, q_m$ are trajectory qualifiers, which can be used to access corresponding parts of trajectories.
- The set of trajectories can be defined in several different ways. We will allow such notation in the trajectory prefix definition to bring out conditions on the set of trajectories.

Furthermore, we will allow to define the set of trajectories directly in the definition of trajectory prefix, where commas will be used to separate conditions. We will use $\Downarrow$ to separate exit conditions, when it is required.

Concatenation *Concatenation* extends definition of trajectory prefix. It formalizes behavior after taking a complete trajectory. The process can choose to continue with another trajectory or an action prefix, depending on the successive process.

Concatenation is formalized by the following derivation rules.

$$\frac{B\,(\varphi) \xrightarrow{\psi} B'}{[f \mid \Phi] \;.\; B\,(\varphi) \xrightarrow{\varphi;\psi} B'} \qquad \varphi \in \Phi \tag{4}$$

$$\frac{B\,(\epsilon) \xrightarrow{\mathsf{a}} B'}{[f \mid \Phi] \;.\; B\,(f) \xrightarrow{\mathsf{a}} B'} \qquad \epsilon \in \Phi \tag{5}$$

In (4) it is shown, how to concatenate two trajectories. While (5) defines a situation, where after taking a whole trajectory process continues with an action prefix.

For a convenience we derive an equation from the concatenation and trajectory prefix rules. If $\epsilon \in \Phi$ then $[f \mid \Phi] \;.\; B(f)$ behaves in the same way as $[f \mid \Phi \setminus \epsilon] \;.\; B(f) + B(\epsilon)$

Choice $\sum\{B(v) \mid v \in I\}$ *Choice* is a generalized *nondeterministic* choice of processes (I is an arbitrary index set). It chooses before taking an action prefix or trajectory prefix. Binary version of choice is denoted by $B_1 + B_2$.

$$\frac{B\,(w) \xrightarrow{\mathsf{a}} B'}{\sum\limits_{v \in I} B\,(v) \xrightarrow{\mathsf{a}} B'} \qquad w \in I \tag{6}$$

$$\frac{B\,(w) \xrightarrow{\varphi} B'}{\sum\limits_{v \in I} B\,(v) \xrightarrow{\varphi} B'} \qquad w \in I \tag{7}$$

Choice for action prefix is defined in 6 the choice for action prefix is defined, it is the same as in usual process algebras. Rule 7 tells that choice for trajectories is made before taking a trajectory.

Parallel composition $B_1 \parallel_A^H B_2$ *Parallel composition* models concurrent evolution of several processes. During the evolution they may interact with each other via synchronization on discrete and continuous-time transitions. In BHPC synchronization on identical names is assumed as the basic synchronization concept. In order to avoid context-dependent interpretations of operators, the set of action names A and the set of trajectory qualifiers H that are subject to synchronization, are made explicit in the parallel operator $\parallel_A^H$.

This form of synchronization implies that parallel components jointly execute identical actions or trajectories with common signal evolutions that occur in their transitions and are subject to synchronization.

The basic idea of synchronizing trajectories is not much different than that of synchronizing actions. Let B_1 and B_2 be the processes that can take trajectories φ and ψ, such that coinciding qualifiers of both trajectories belong to H and evolve in the same manner. Then the resulting trajectory is a synchronized trajectory of $B_1 \parallel_A^H B_2$ that simultaneously changes the states of B_1 and B_2 according to $\chi = \varphi \times_H \psi$, where $\times_H$ defines composition of trajectories in such a way that χ contains all qualifiers of φ and ψ.

We define the following deduction rules for parallel composition

$$\frac{B_1 \xrightarrow{\mathsf{a}} B_1', B_2 \xrightarrow{\mathsf{a}} B_2'}{B_1 \parallel_A^H B_2 \xrightarrow{\mathsf{a}} B_1' \parallel_A^H B_2'} \quad \mathsf{a} \in A \tag{8}$$

$$\frac{B_1 \xrightarrow{\mathsf{a}} B_1'}{B_1 \parallel_A^H B_2 \xrightarrow{\mathsf{a}} B_1' \parallel_A^H B_2} \quad \mathsf{a} \notin A \tag{9}$$

$$B_2 \parallel_A^H B_1 \xrightarrow{\mathsf{a}} B_2 \parallel_A^H B_1'$$

$$\frac{B_1 \xrightarrow{\varphi} B_1', B_2 \xrightarrow{\psi} B_2'}{B_1 \parallel_A^H B_2 \xrightarrow{\varphi \times_H \psi} B_1' \parallel_A^H B_2'} \tag{10}$$

Rules (8) and (9) define parallel composition for actions in a well known interleaving fashion. In (10) parallel composition for trajectories is defined. Because trajectories always synchronize, one rule is enough.

Recursion *Recursion* is a tool to define processes in terms of each other, as in the equation $P = B$, where P is the process identifier and B is a process expression that may only contain actions and signal types of B.

$$\frac{B \xrightarrow{\mathsf{a}} B'}{P \xrightarrow{\mathsf{a}} B'} \quad P = B \qquad \frac{B \xrightarrow{\varphi} B'}{P \xrightarrow{\varphi} B'} \quad P = B \tag{11}$$

Guard $\langle \mathcal{P}red \rangle$ *Guard* $\langle \mathcal{P}red \rangle$ operator evaluates $\mathcal{P}red$ conditions, and if they are not satisfied, stops the progress of the process.

$$\langle \mathcal{P}red(\mathbf{x}) \rangle \,.\, B(\mathbf{x}) = \sum_{\mathbf{w} \models \mathcal{P}red(\mathbf{w})} B\,(\mathbf{w}) \tag{12}$$

Here $\mathbf{x}$ are process parameters (variables).

Strong Bisimulation and Congruence. The *hybrid strong bisimulation relation* (equivalence) defined for the HTS is a congruence relation w.r.t. all operations defined above [19]. Hence, bisimilar components can be interchanged without changing systems behavior, and that can be effectively employed while building and improving systems (models).

3 Bhave toolset

BHPC is supported by Bhave toolset [15]. The toolset allows modeling, simulation and visualization of the hybrid models [16, 34]. It consists of several tools.

Discrete Bhave [18] allows discrete simulation of the BHPC specifications.

Hybrid simulator [33, 34] is a tool for modeling and simulation of hybrid process algebras. Basically, it is a framework to implement a simulator for a selected process algebra.

Bhave simulator [33,34] is an implementation of simulator for BHPC based on the hybrid simulator. It allows hybrid simulation of BHPC specifications. Current version supports a subset of BHPC. A snapshot of the system with examples is available from `bhpc-simulator.sourceforge.net`.

BHPC2Mod can translate a restricted set of BHPC models to Modelica [11] language, and then simulate them using Dymola [10] or OpenModelica [25]. However, because Modelica does not have formal semantics, translation does not necessary preserves all the properties. Moreover, parallel composition is not translated [32]. It is not supported.

MSP-SVG is a visualization tool that uses Message Sequence Plots (MSP) [19,28,33,34] approach for visualizing hybrid evolution. It is available at `http://msp-svg.sourceforge.net/`. See Section 3.2 for details about MSP-SVG.

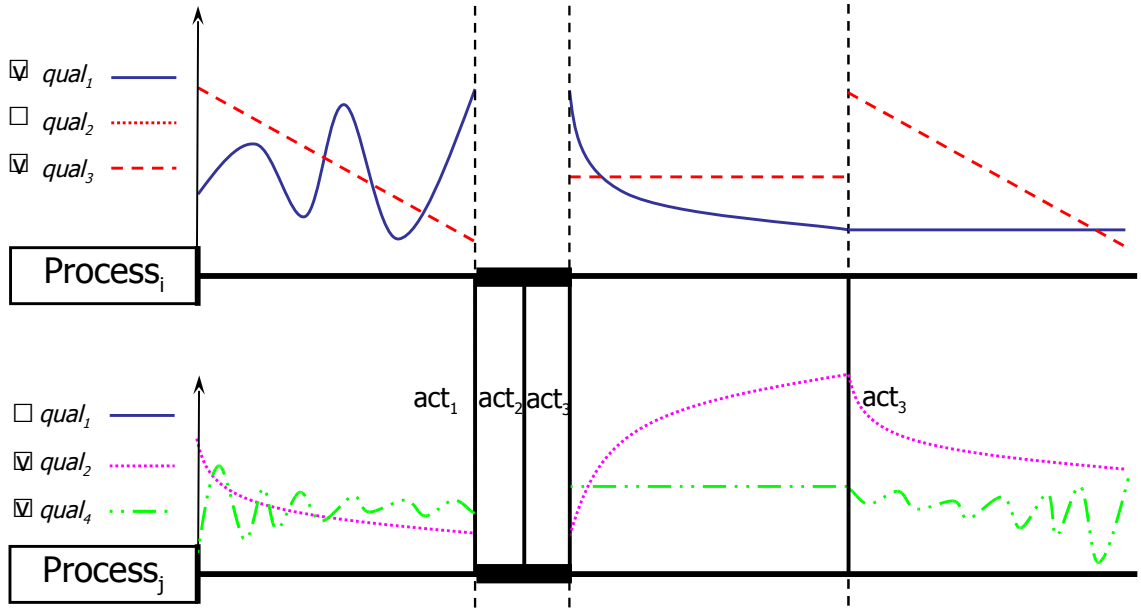

Figure 1: MSP example.

The current versions of tools (BHAVE and MSP-SVG) are built not just as prototypes, but also as a hybrid "sandbox", a place to experiment with BHPC and related developments. Architecture and implementation of the tools allow accommodating diverse changes and test the algorithms developed for BHPC or other (hybrid) process algebras or MSP-based visualization techniques comparatively easy.

Our plans include further development of the process algebra and BHAVE TOOLSET. We are planning to improve and extend BHAVE and integrate it with MSP-SVG.

3.1 Simulator for Hybrid Process Algebras

3.1.1 Generic architecture

Simulator for hybrid process algebras takes a specifications as an input and then interprets each statement according to the process algebra, it implements, semantics. Current implementation of uses a stack based approach [2, pp. 418-422] to store the current state, because it is a standard approach for executing recursive function calls. Moreover, abstract interfaces are used to define the bases operator types instead of using other operators directly.

The base of the simulator structure is a non-deterministic choice selector, a choice of all possible simulation flows at the current step, formally - a normal form [19, p.106]. Expansion laws [19, p.106-110] are used to transform processes to a /emphnormal form. *Execute* stage is primarily designed for execution of actions and trajectories, at this stage the operations are executed. Abstract algorithm is as follows.

1. **Do**:
2. *Expand* available choices.
3. Non-deterministically *select* one choice discarding others.
4. *Execute* expression.
5. **While** there are unfinished choices.

Simulation of Continuous Behavior. Usually, the trajectories of hybrid process algebras are expressed as a system of *differential algebraic equations* (DAE) or *ordinary differential equations* (ODE) describing the evolution of the system variables. We use SUNDIALS IDA library [1] to solve ODE/DAE. It is derived from DASPK and is designed to solve DAE systems in form $F(t, y, y') = 0$. IDA library was chosen because it provides a fast and elegant solution for the problem and allows using custom evaluation functions for the integration. Hence, the same calculations logic is applied both to *guard*, *discrete variables assignment* and *continuous-time evolution* solving. As SUNDIALS IDA library provides a clean API in C programming language, a C++ adapter class *DaeSolver* is written for the simulator. This class uses simulator's *Evaluable* class family functionality to solve equations and then passes the result back to IDA. Also, in order to use a different DAE solver library or tool, only the *DaeSolver* class should be changed.

3.2 Visualization of Hybrid Evolutions

Simulation results usually visualize an evolution of the system in time. Event traces or message sequence charts (MSC) [27] adequately represent discrete system behavior, and graphs are convenient for the ordinary continu-

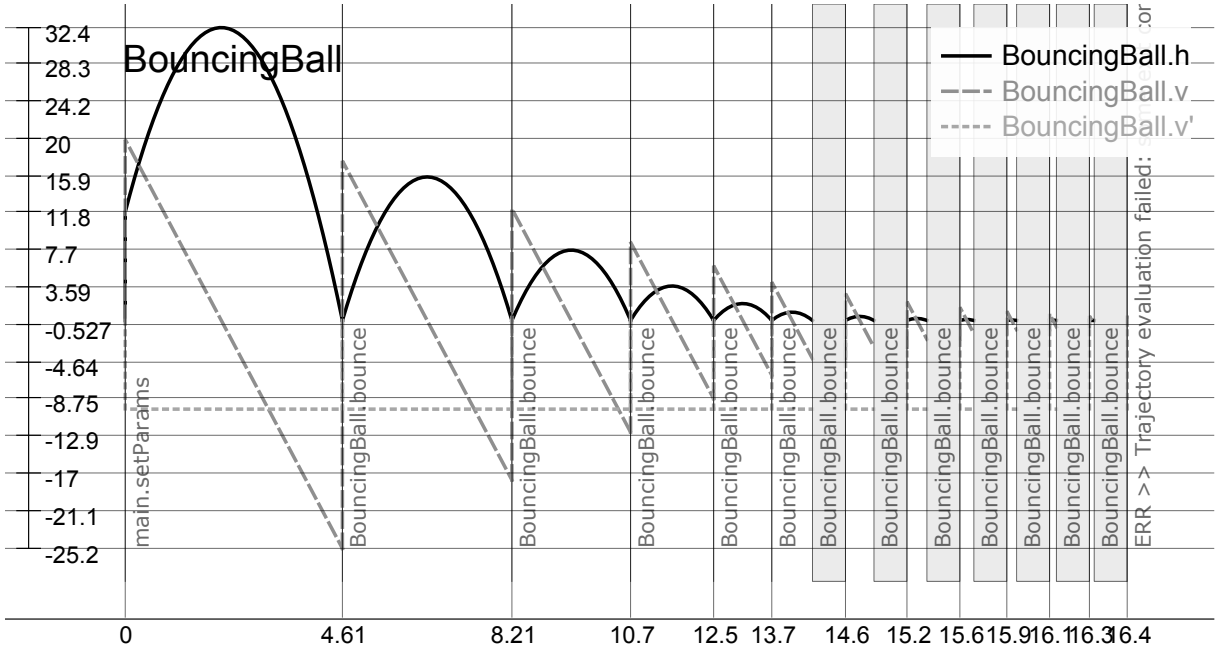

Figure 2: Simulation of the bouncing ball, Section 4.1.

ous systems. However, in hybrid systems we have both the evolution of system variables and events. Hence, a combined view is crucial to fully analyze hybrid system behavior. See [19, p. 118-124] for the details.

We believe, that Message Sequence Plots (MSP) [19, p. 118-124] contain all the necessary components for an adequate visualization of hybrid systems. It has two compounds: message-sequence charts rotated 90° combined with plots. We explain MSP by an example depicted in Figure 1.

Plots over time-lines show continuous-time evolution.

A legend allows selecting qualifiers of interest, that are depicted in the plot. If several processes evolve concurrently, the synchronizing qualifiers appear for both processes. In Figure 1 qualifiers $qual_1$, $qual_2$, $qual_3$ and $qual_4$ are depicted. $\mathbf{Process}_\mathrm{i}$ is related with qualifiers $qual_1$, $qual_2$ and $qual_3$, and only qualifiers $qual_1$ and $qual_3$ are selected to be visible. $\mathbf{Process}_\mathrm{j}$ is related with qualifiers $qual_1$, $qual_2$ and $qual_4$, and qualifiers $qual_2$ and $qual_4$ are selected to be visible.

Single horizontal lines connected to the corresponding boxes with process identifiers, represent processes and the *time-line* (or *life-line* in MSC terminology). Time is assumed to flow to the right along each time-line at the same speed. $\mathbf{Process}_\mathrm{i}$ and $\mathbf{Process}_\mathrm{j}$ are represented by the horizontal lines and boxes with processes identifiers in the example.

Labeled vertical lines going across time-lines represent communication, i.e. (parametrized) action prefixes in BHPC. Notice that we use simple lines instead of arrows, because communication in BHPC is not directed. However, the MSP (and related tool) can be easily modified to display direction of actions. Communication of $\mathbf{Process}_\mathrm{i}$ and $\mathbf{Process}_\mathrm{j}$ consists of actions $\mathsf{act}_1, \mathsf{act}_2$ and act_3.

Triple horizontal time-lines depict suspension of the time-flow. Single actions are placed on the time-line at the time that relates to their moment of occurrence. A sequence of actions occurs at one moment in time, when there is no continuous behavior between the actions. We suspend the flow of time to allow insight in the ordering of these actions. In the example, suspension of the time is depicted on the time-line as three parallel solid lines.

Figure 1 contains all information that would be available in an ordinary plot. Correspondingly, all information that is visible in message sequence charts, is also visible in MSP. Furthermore, in MSP all processes and communication between them are visualized. The proposed technique can be easily adopted to other hybrid system modeling frameworks with minimal changes, e.g., if communication is directed, arrows can be used to depict it.

Additional notation, such as decorating the results with process expressions (e.g. action and trajectory prefixes), adding recursive calls, providing information about renaming of qualifiers in the legend, forking of processes to depict parallelism. See [17] for details.

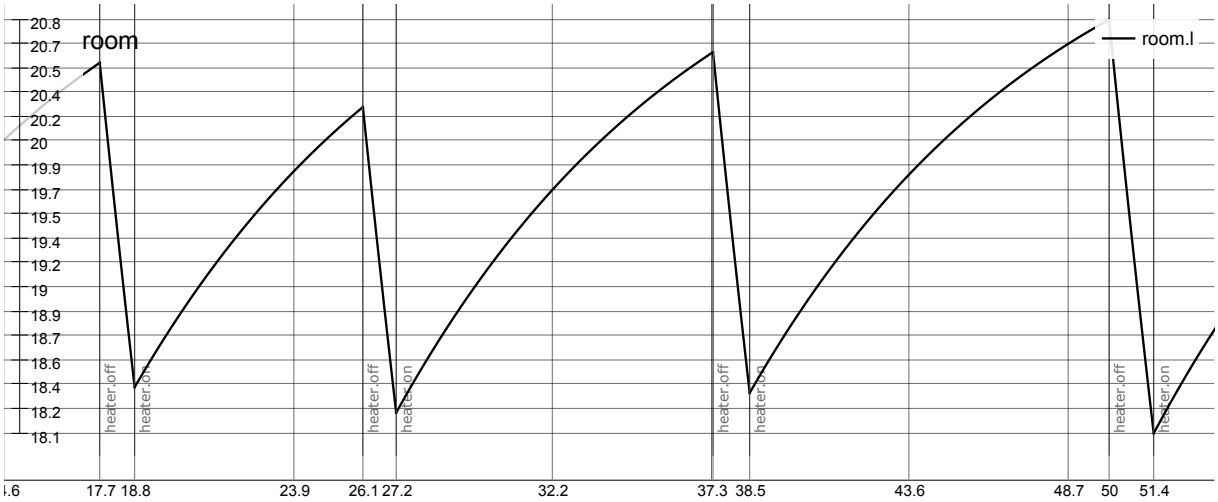

Figure 3: Evolution of thermostat, Section 4.2.

Figures 2, 3 and 5 depict evolution of the bouncing ball simulation from Section 4.1, the thermostat from Section 4.2 and the two tanks from Section 4.3, respectively, using a proof of concept tool MSP-SVG. Executable and source code of MSP-SVG are available at `sourceforge.net/projects/msp-svg/`, see [33, 34] for the details.

4 Application of BHPC

4.1 Bouncing Ball

Bouncing ball is a common example of hybrid process algebra systems. The system [19, pp. 12, 86] consists of one ball and a ground plane. The ball in the system is defined by its altitude h, vertical speed v and the constant c, which describes the energy that is lost on every bounce. Also, the ball is constantly affected by the gravitational acceleration $g = 9.81$.

The formal specification is rather simple.

```
BouncingBall(h, v) =
     [  h' = v ;    // dh/dt = v
        v' = -9.81  // dv/dt = -g
      | h ]         // when h = 0, stop
     . bounce{v' = 0;} // bounce action
      // bounce with 0.7 velocity loss
     . BouncingBall(h, -0.7 * v);

// set simulation parameters
main() = setParams{step=1e-1}
      // invoke the bouncing process
     .BouncingBall(12, 20);
```

The system consists of two processes:

- `BouncingBall` process defines the trajectory and the *bounce* action of the ball. The motion is described by the derivative of the altitude, which is the vertical speed v. The speed is affected by the acceleration $\dot{v} = -g$. This motion is executed until the ball touches the ground plane ($h = 0$) and a discrete *bounce* action is executed. As the ball bounces, a fraction $c = 0.7$ of its energy is lost, and the ball changes the direction upwards.

- The process `main` is the simulation entry point. A discrete action *setParams* is invoked, which changes the parameters for the simulation – the parameter *"step"* defines the integration interval length for the DAE solver. Then, a `BouncingBall` process is invoked with the initial parameters $h = 12$ and $v = 20$.

Figure 2 displays the results of bouncing ball simulation. It shows the speed and altitude change over time together with the discrete action `bounce`. Visualization of the model evolution is generated by using a prototype *Message Sequence Plot* (MSP) visualization application [17, 21, 23], proposed in [19, pp. 120-123]. MSP displays the changes of the system's process variables together with the actions that are performed.

The energy of the system is reduced by a fraction in every bounce, which results in a shorter timespan with every bounce. This leads to *Zeno* behavior [19, p. 124], where the system tries to execute an infinite amount of bounces in a finite amount of time. The simulator prevents such behavior by forcing a small fraction of the initial integration interval to simulate regardless of the trajectory exit conditions and checks the result after the first step. Such situation can be seen in the *16.4 second* of the simulation, when the error is printed.

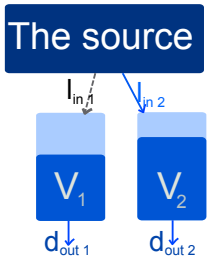

Figure 4: Two tanks scheme.

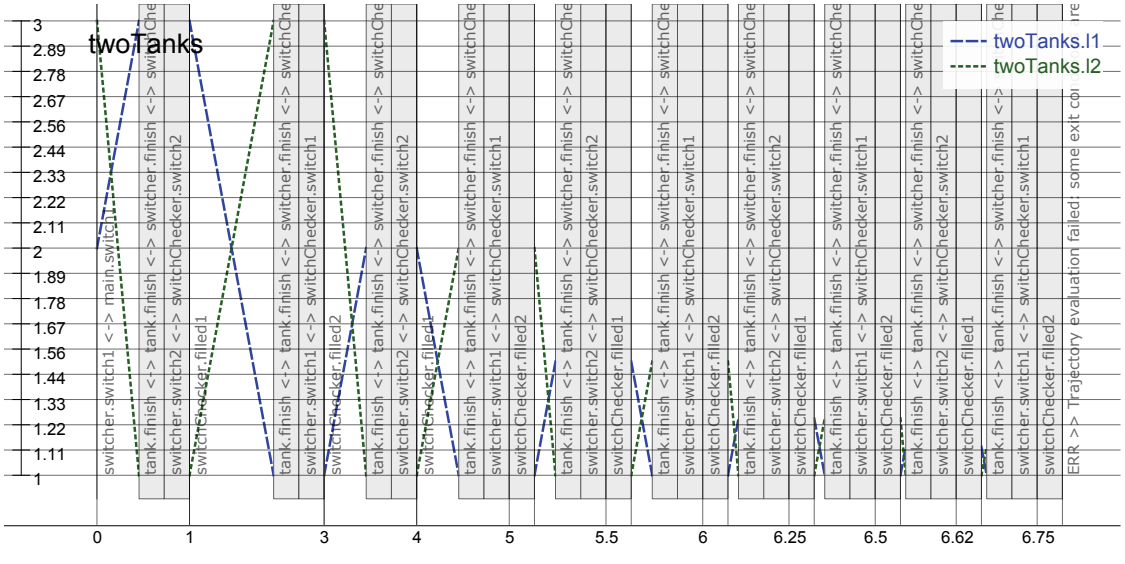

Figure 5: Simulation of the two tanks example, Section 4.3.

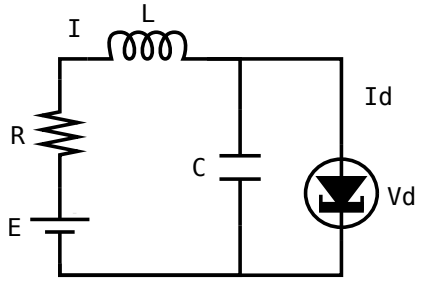

Figure 6: Tunnel diode circuit.

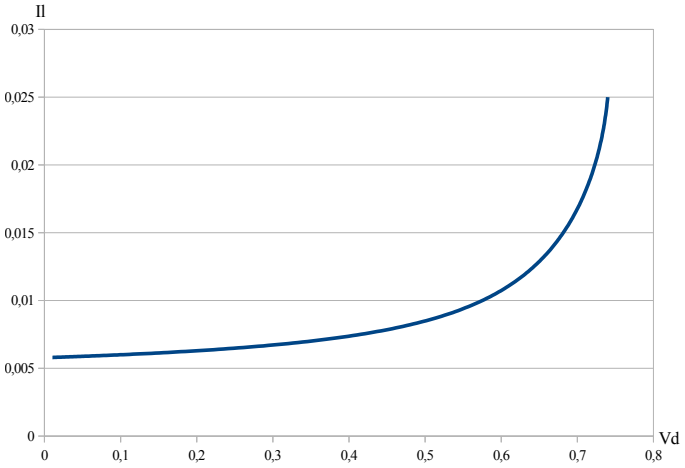

Figure 7: Non-oscillating tunnel diode circuit: current vs voltage

4.2 Thermostat

A thermostat is another widely known example of hybrid systems [19, p. 13]. It defines a system, where the thermostat controls the temperature of a room. The system consists of three elements – the room, which looses the temperature over time, a heater, and a thermostat, which senses the temperature of the room, and controls the heater. The goal of the controller is to maintain the temperature inside the interval of the required temperatures $[l_{min}, l_{max}]$.

The temperature of the room is defined by the function $l(t) = \Theta e^{-Kt} + h(1 - e^{-Kt})$. K in the given function is a constant, determined by the room, h is a constant, determined by the power of the heater and Θ denotes the initial temperature of the room. The derivative $\frac{d}{dt}l = K(h - l)$ of the given function is used to calculate the temperature changes over time during the simulation [19].

A BHPC thermostat model for the simulation is as follows.

```
const K = 0.1;
const H = 22;

heater(h) = [ h' = 0 | room.l - (18 + rand())]
          . on{h = H}
          . [ h' = 0 | (20 + rand())-room.l ]
          . off{h = 0}
          . heater(h);

room(l) = [ l' = K*(heater.h-l) |
            (l - 18)*(21 - l)]
```

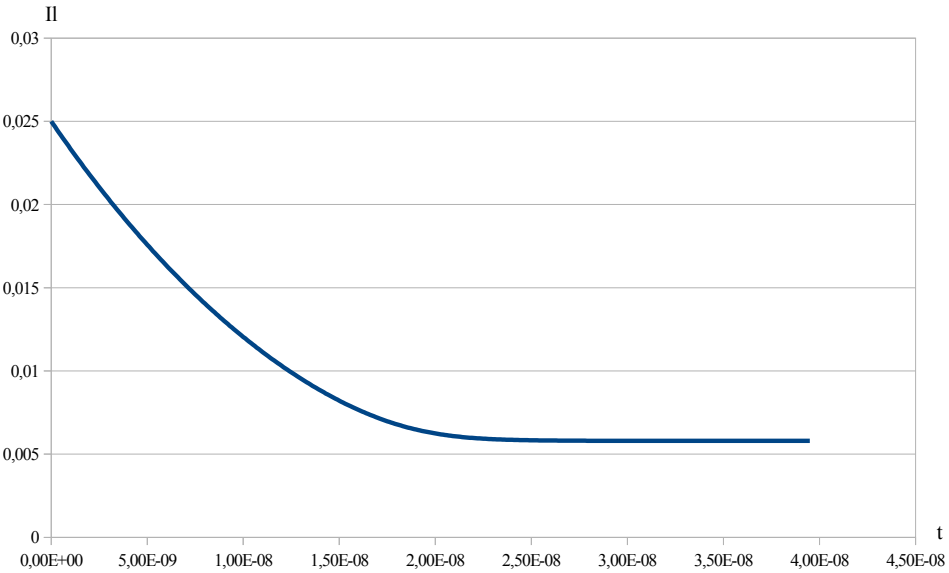

Figure 8: Non oscillating tunnel diode circuit: current

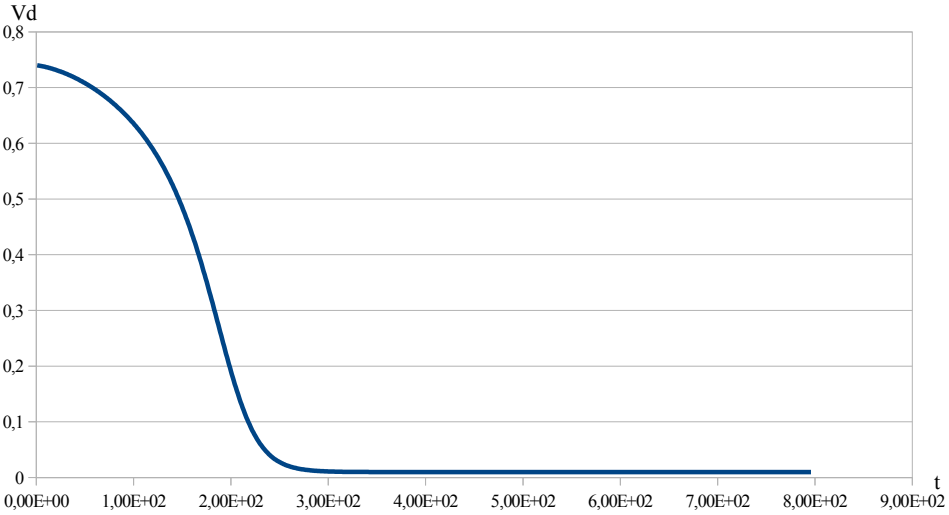

Figure 9: Non-oscillating tunnel diode circuit: voltage

```
        . error_temp_out_of_range;

main() = setParams{step=1e-1;tStop=100}
        . room(19) || _ ] heater(0);
```

The following processes are defined in the model:

`heater(h)` defines a heater with a thermostat. This process tracks the changes of the temperature in the room and modifies the power of the heater – turns it *off* or *on*.

`room(l)` represents the room and its temperature. The trajectory is calculating the temperature of the room during the whole simulation and the exit conditions of it are only the limits of the required temperature interval (using a parabola $(l - l_{min})(l_{max} - l)$ for specifying the range). If the trajectory is finished before the simulation end, it means only that the temperature was out of the required bounds and an error action is performed.

`main()` sets the parameters for the simulation and invokes both the *heater* and *room* processes in parallel.

The simulation result of the thermostat system model is presented in Figure 3. It shows how the thermostat and the heater are affecting the temperature of the room. When the heater is *off*, the temperature of the room drops rapidly and once it is in the range $[18, 19]$ (the lower bounds of the thermostat activation), the heater is turned *on*. The temperature then slowly rises (how fast it rises depends on the power of the heater – the constant `h`)

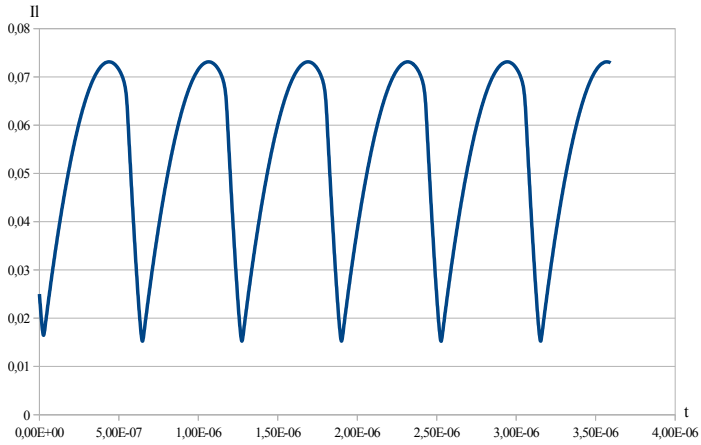

Figure 10: Oscillating tunnel diode circuit: current

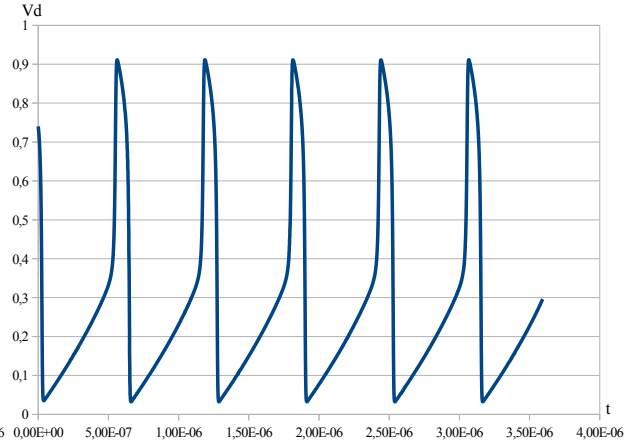

Figure 11: Oscillating tunnel diode circuit: voltage

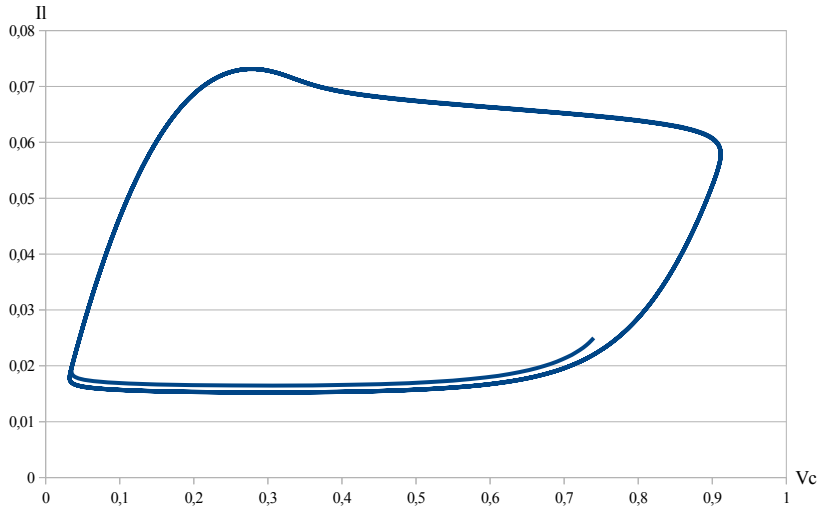

Figure 12: Oscillating tunnel diode circuit: current vs voltage

to the upper checking bounds of the thermostat [20, 21] and the heater is turned *off*.

The process continues until the time of the simulation reaches the limit `tStop` and then the simulator exits.

Simulation results are depicted in Figure 3.

4.3 Two Tanks

Two tanks model [19, p. 89] specifies a hybrid system, where a mutual fluid source fills two separate tanks (Figure 4). Only one tank is receiving the fluid input at a time. At the bottom, each of the tanks has an opening, through which the fluid flows out at a constant rate d_{out} (different for each tank). Switching the source pipe from the fluid source to any of the tanks happens instantly, and the pipes to each tank have different diameters (l_{in} rate, at which the tank is filled with the fluid). The objective of the system is to keep fluid level in each of the tanks in the interval $[l_{min}, l_{max}] = [1, 5]$. The system starts with initial fluid levels in the tanks given ($l_0^{left} = 2$ and $l_0^{right} = 3$). The BHPC code for the simulation of this model is as follows.

```
tank( new l, new d_out, new l_in) =
    [ l' = l_in - d_out; l_in'=0; d_out'=0;
      | (5 - l) ; (l - 1) ;  ]
  . <(l > 5) + (l < 1)>
  . tank(l, d_out, l_in);

twoTanks(l1, l2, out1, out2, in1, in2) =
  (tank(l1, out1, in1) [twoTanks.l1/l,
     twoTanks.out1/d_out, twoTanks.in1/l_in])
  ||_]
  (tank(l2, out2, in2) [twoTanks.l2/l,
     twoTanks.out2/d_out, twoTanks.in2/l_in]);

switcher(new in1, new in2) =
    switch1{ twoTanks.in1=in1; twoTanks.in2=0;}
  . [ | (5 - twoTanks.l1); (twoTanks.l2 - 1);
        (twoTanks.l1 - 1); (5 - twoTanks.l2); ]
  . finish
```

```
  . switch2 {twoTanks.in1=0; twoTanks.in2=in2;}
  . [ | (5 - twoTanks.l1); (twoTanks.l2 - 1);
        (twoTanks.l1 - 1); (5 - twoTanks.l2); ]
  . finish
  . switcher(in1, in2);

switchChecker() = [finish].
    (<(twoTanks.in1 == 0)*(twoTanks.in2 != 0)>
    .switch2.filled1.switchChecker())
    +
    (<(twoTanks.in1 != 0)*(twoTanks.in2 == 0)>
    .switch1.filled2.switchChecker());

main() = setParams{step=1e-1}.((
    twoTanks(2, 3, 1, 2)
  ||_]
    switcher(2, 3)
) ||finish,switch1,switch2]
    (switch1.switchChecker()));
```

The system starts with the process `tank` definition, which describes the trajectory of the evolution of the fluid level in the tank. This trajectory ends when the fluid level of this or *other* tank reaches $[1, 5]$ range limits. After the trajectory, there is a *guard* operator, which ensures, that the fluid level is in the required range − if the guard is evaluated to `true`, a *deadlock* occurs. The `tank` process then continues with *recursion* repeating the same operations.

`twoTanks` process describes a system of two tanks, where both tanks evolve in parallel. Also, the variables of each tank are renamed to use `twoTank` process variables.

A `switcher` models the pipe switching controller, which idles until the either of the tanks reaches the required limits and then instantly switches the pipe to the other tank. The behavior of this controller is checked by the `switchChecker` process. It ensures that the pipe switching operation is performed in the right process evolution step and issues *filled1* and *filled2* actions to notify the possible listeners about the last position of the pipe.

The process `main` in BHave is the starting point of the system. This process sets the required integration step parameter for DAE solver and invokes `twoTanks` in parallel to the `switcher`.

The visualization of the *two tanks* model evolution is presented in Figure 5. The system starts with the *switch1* action, which sets the inflow parameters for the tanks. Then the trajectories of fluid flow are executed. Here it can be seen, that *Tank2* output is faster than *Tank1* input, thus, when pipe is switched, tank1 contains only 3 units of fluid level. After the pipe is switched, the rates of inflow in the *Tank1* and the outflow of the *Tank2* are equal. After each full cycle of pipe switching, the levels of the fluids in the tanks are approaching the lower limit at 1, which leads to *Zeno* behavior [19, p. 124], where the system will try to execute an infinite amount of pipe switching actions in a finite amount of time. Like in the bouncing ball example (Section 4.1), the simulator detects the trajectory error and prints it, stopping the simulation.

4.4 Tunnel Diode Circuit

We present results of experiments with the tunnel-diode circuit from [9]. The circuit is depicted in Figure 6. State of the system is defined by two state variables: the current through the inductor I_l and the voltage across the diode V_d. Behavior is defined by the following differential equations

$$\dot{I}_l = \frac{1}{C}\left(-I_d(V_d) + I_l\right) \tag{13}$$

$$\dot{V}_d = \frac{1}{L}\left(E - RI_l - V_d\right) \tag{14}$$

where

$$I_d(V_d) = V_d^3 - 1.5V_d^2 + 0.6V_d$$

defines non-linear characteristics of the tunnel diode.

We provide a simple model of the system, that consists of one process, namely **TunnelDiode**, that contains a trajectory prefix defining evolution of the circuit over time. Exit conditions of the trajectory prefix define expected intervals for the current I_l and voltage V_d.

```
TunnelDiode(Il, Vd) =
     [ Vd' = (1 / C) * (Il -
      (Vd * Vd * Vd  -
       1.5 * Vd * Vd + 0.6 * Vd));
       Il' = (1 / L) * (E - R * Il - Vd)
      | (Vd - 0); (Vd - 0.92);
        (Il - 0); (Il - 0.08) ]
      . stop;
```

We performed simulation with two different sets of parameters.

	Non-oscillating	Oscillating
$C\ (F)$	10^{-9}	10^{-9}
$L\ (H)$	10^{-6}	10^{-6}
$E\ (V)$	0.3	10^{-9}
$R\ (\Omega)$	50	0.3

Initial values: $I_l = 0.025A$ and $V_d = 0.74V$. Notice, that only voltage differs.

As expected, in the first case simulation stabilizes in the equilibrium. Figures 8, 9 and 7 depict the current, voltage and current vs voltage of the non-oscillating tunnel diode circuit, respectively.

With the second set of parameters we get an oscillating system in the expected intervals. Simulation of the oscillating tunnel diode circuit is depicted in Figures 10, 11 and 12.

5 Conclusions

Modeling and analysis of the tunnel diode circuit shows that Behavioral Hybrid Process Calculus (BHPC) and its toolset can be used for the formal specification and simulation of electronic systems.

Application of the proof-of-concept MSP-SVG tool for visualization of other examples presented in Section 4 demonstrates advantages of the Message Sequence Plots over simple plots, because not only switching points are visible, but a cause (a related event) as well.

Our future work will focus on several aspects:

- Application of the toolset to complex analog and mixed-signal design.
- Modular specifications of diverse circuits, i.e. modeling of circuits as parallel components, e.g. modeling tunnel diode circuit as parallel interconnection of capacitor, inductor, resistor, tunnel diode and power source.
- Improvements of the modeling language. Currently we use the language, that consists only of basic constructs. We are planning to add some syntactical constructs for convenience.
- Improvements of tools. We are planning to improve simulator and MSP-SVG tools, integrate them.
- Simulation of HyPA [8] using Hybrid Simulator to compare BHPC and HyPA.
- Integration of Bhave toolset with other hybrid systems modeling, simulation and verification tools, directly or via an interchange format.

References

[1] SUite of Nonlinear and DIfferential/ALgebraic equation Solvers, `https://computation.llnl.gov/casc/sundials/description/description.html#descr_ida`, 2010.

[2] Abelson, H., Sussman, G.J., Sussman, J. *Structure and Interpretation of Computer Programs*, The MIT Press, January, 2007.

[3] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S., The algorithmic analysis of hybrid systems, *TCS* V138, pp.3–34, 1995.

[4] Baeten, J.C.M., Weijland, W.P., *Process Algebra*, V18, *Camb. Tracts in TCS*, Camb. Univ. Press, Cambridge, UK, 1990.

[5] Bergstra, J.A., Middelburg, C.A., Process Algebra for Hybrid Systems, *Theoretical Computer Science*, V335, pp.215–280, 2005.

[6] Brinksma, E., Krilavičius, T., Usenko, Y.S., Process Algebraic Approach to Hybrid Systems, *Proc. of 16th IFAC World Congress*, July, 2005 ,Prague, Czech Rep. pp.1–6.

[7] Carloni, L.P., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A.L., Language and Tools for Hybrid Systems Design, *Journal of Foundations and Trends*, V1, pp.1–177, 2005.

[8] Cuijpers, P.J.L., Reniers, M.A. Hybrid Process Algebra, *Journal of Logic and Algebraic Programming*, V62, pp.191–245, 2005.

[9] Denman, W., *Formal Verification of Analog and Mixed Signal Designs*, master thesis, Concordia Univ., 2009.

[10] Dynasim, `http://www.dynasim.se/`, last accessed: 2006 May 11, 2006.

[11] Fritzson, P., Engelson, V., *Modelica - A Unified Object-Oriented Language for System Modelling and Simulation*, 1998.

[12] Geilen, M., *Formal Techniques for Verification of Complex Real-time Systems*, phd thesis, Tech. Univ. of Eindhoven, 2001.

[13] Hoare, C.A.R., *Communicating Sequential Processes*, Prent.-Hall, 1985.

[14] Hooman, J., *Specification and Compositional Verification of Real-Time Systems*, Springer, 1991.

[15] Krilavičius, T., *Bhave :: Simulation of Hybrid Systems*, `http://fmt.cs.utwente.nl/tools/bhave/`, 2006.

[16] Krilavičius, T., Simulation of Mechatronic Systems Using Behavioural Hybrid Process Calculus, *Electron. and Elec. Eng.*, V1, pp.45–48, 2008.

[17] Krilavičius, T., Man, K., *Intelligent Automation and Computer Engineering*, chapter Behavioural Hybrid Process Calculus for Modelling and Analysis of Hybrid and Electronic Systems, Springer, 2009.

[18] Krilavičius, T., Schonenberg, H., Discrete Simulation of Behavioural Hybrid Process Calculus, *IFM2005 Doctoral Symposium*, Tech. Univ. of Eindhoven, Dept. of Math. and CS, November, 2005, Eindhoven, Netherlands, ISSN 0926-4515, pp.33–38.

[19] Krilavčius, T., *Hybrid Techniques for Hybrid Systems*, phd thesis, Univ. of Twente, 2006.

[20] Man, K.L., Schellekens, M.P., *Current Trends in Intelligent Systems and Computer Engineering*, chapter Interoperability of Performance and Functional Analysis for Electronic System Designs in Behavioural Hybrid Process Calculus (BHPC), Springer, 2008.

[21] Man, K., Krilavičius, T., Chen, C., Leung, H., Application of Bhave Toolset for Systems Control and Electronic System Design, *Lecture Notes in Engineering and Computer Science*: *Proceedings of The International MultiConference of Engineers and Computer Scientists 2010*, IMECS 2010, March, 2010, Hongkong, pp.1336-1341

[22] Milner, R., *Communication and Concurrency*, Pren.-Hall, 1989.

[23] *Prototype application for Message Sequence Plot visualisation*, `http://sourceforge.net/projects/msp-svg/`, March, 2010.

[24] Naumoski, G., Alberts, W., *A Discrete-Event Simulator for Systems Engineering*, phd thesis, Tech. Univ. of Eindhoven, 1998.

[25] OpenModelica System website, OpenModelica System, `http://www.ida.liu.se/\~{}pelab/modelica/OpenModelica.html`, `http://www.ida.liu.se/~pelab/modelica/OpenModelica.html`, 2009.

[26] Polderman, J.W., Willems, J.C., *Introduction to Mathematical Systems Theory: a behavioral approach*, Springer, 1998.

[27] Rudolph, E., Graubmann, P., Grabowski, J., Tutorial on Message Sequence Charts, *Comput. Netw. ISDN Syst.*, V28, pp.1629–1641, 1996.

[28] Schonenberg, M.H., *Discrete Simulation of Behavioural Hybrid Process Algebra*, master thesis, Univ. of Twente, 2006.

[29] van Beek, D.A., Gordijn, S.H.F., Rooda, J.E. Integrating Continuous-Time and Discrete-Event Concepts in Modelling and Simulation of Manufacturing Machines, *Sim. Pract. and Theory*, V5, pp.653–669, 1997.

[30] van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H., Syntax and Semantics of Timed Chi, technical report, CS-Report 05-09, Tech. Univ. of Eindhoven, Dept. of CS, The Netherlands, 2005.

[31] van Beek, D.A., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H., Syntax and Consistent Equation Semantics of Hybrid Chi, *JLAP*, V68, pp.129–210, 2006.

[32] van Putten, A., *Behavioural Hybrid Process Calculus Parser and Translator to Modelica*, master thesis, Univ. of Twente, 2006.

[33] Valaškevičius, Š., *Simulation Tool for Hybrid Process Algebras*, master thesis, Vytautas Magnus University, 2010.

[34] Valaškevičius, Š., Krilavičius, T., Hybrid process algebra simulation tool, *Information Society and University Studies XV, IVUS 2010*, Vytautas Magnus University, 6 May, 2010, Kaunas, Lithuania, pp.1-6.