A Model-Based Development Environment for Rapid-Prototyping of Latency-Sensitive Automotive Control Software

Sakthivel Manikandan Sundharam
FSTC/LASSY
University of Luxembourg
L-1359 Luxembourg
sakthivel.sundharam@uni.lu

Lionel Havet
RealTime-at-Work (RTaW)
615, rue du Jardin Botanique
54600 Villers-les-Nancy France
lionel.havet@realtimeatwork.com

Sebastian Altmeyer
CSA Group
University of Amsterdam
1098XH Amsterdam NL
altmeyer@uva.nl

Nicolas Navet
FSTC/LASSY
University of Luxembourg
L-1359 Luxembourg
nicolas.navet@uni.lu

***Abstract*—The innovation in the field of automotive embedded systems has been increasingly relying on software-implemented functions. The control laws of these functions typically assume deterministic sampling rates and constant delays from input to output. However, on the target processors, the execution times of the software will depend on many factors such as the amount of interferences from other tasks, resulting in varying delays from sensing to actuating. Three approaches supported by tools, namely TrueTime, T-Res, and SimEvents, have been developed to facilitate the evaluation of how timing latencies affect control performance. However, these approaches support the simulation of control algorithms, but not their actual implementation. In this paper, we present a model interpretation engine running in a co-simulation environment to study control performances while considering the run-time delays in to account. Introspection features natively available facilitate the implementation of self-adaptive and fault-tolerance strategies to mitigate and compensate the run-time latencies. A DC servo controller is used as a supporting example to illustrate our approach. Experiments on controller tasks with injected delays show that our approach is on par with the existing techniques with respect to simulation. We then discuss the main benefits of our development approach that are the support for rapid-prototyping and the re-use of the simulation model at run-time, resulting in productivity and quality gains.**

Keywords- Model-Based Development; Control software; Controller model; Task release jitters; Varying execution-times; Co-simulation; Real-time scheduling; Control system performance.

I. Introduction

Digital controllers are interfaced with sensors and actuators. They are real-time systems since they require inputs and outputs to occur at the right points in time. In automotive applications, for instance, digital controllers control engines, brakes, suspensions, airbags, etc., that are referred to as ”plants”. The plant is in the physical world and physical quantities are sensed by sensors interfaced to Analog to Digital Converters (ADC). An ADC has two steps, namely signal acquisition and sampling. On the other side, digital controllers with Digital to Analog Converters (DAC) are connected to actuators to control the plant. The real-time computing system is implemented with control algorithms as software functions i) to capture the current (reference) state of the plant using sensors, ii) to compare against the reference or the desired state iii) to control the actuators to reach the desired state of the plant.

The continuous-time signals from the sensors are periodically sampled, each sampled set of data is then processed by the real-time control functions. If the processing is not fast enough with respect to the sampling rate, then some samples of data will be lost and the frequency of the control algorithm cannot be respected. In practice, due to for instance varying task execution times and preemption delays, the input to output latencies will vary over time. These delays will directly impact the quality of the control functions, in the worst-case, possibly jeopardizing the safety of the system. Hence, it is important to consider these delays during the design phase of the control software. To achieve this, techniques and supporting tools based on simulated models have been developed. To the best of our knowledge, none of them however support the actual implementation, and software development work is required later in the development process with the risk that the developed software is not identical to the models.

This motivates the contributions of the paper: i) A model-interpretation based runtime environment, called Cyber-Physical Action Language (CPAL) [6], which provides a generic controller simulation engine. Figure 1 is an inverted pendulum control simulation using the CPAL control library in the Matlab/Simulink (MLSL) environment that can execute

978-1-5090- 2541-1/16/$31.00

any controllers written in CPAL. ii) A case-study servo control mechanism is developed to explain our co-simulation development approach. iii) We compare our synthesis approach to existing simulation-only approaches. As a result of this modeling approach integrating both functional and timing-related non-functional concerns, system designers achieve reduced development life cycle time.

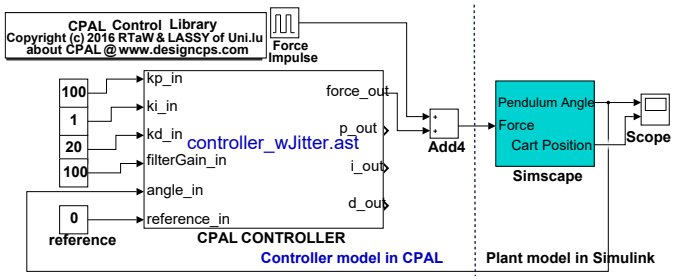

Fig. 1. A controller model for an inverted pendulum integrated within the Simulink Environment. The CPAL control library is used to design the controller model, input and output control data are visible in the design window and can be changed without the need to access CPAL model. The *ast file* format is the binary equivalent form of the source-code of the controller model.

The rest of the paper is organized as follows: Section II provides a survey of the state of the art practices. Then, Section III describes our co-simulation approach, where the controller model is designed in CPAL and the plant model is designed in Matlab/Simulink (MLSL). In Section IV, we present an automotive servo controller as the use-case, implemented in CPAL and running in the Simulink environment. In Section V, we compare our model-based co-simulation timing analysis against existing approaches.

II. State-of-the-art Practices for Co-simulation of Control and Real-time Aspects

Powerful Model-Based Development (MBD) tools such as MATLAB/Simulink (MLSL) and ASCET/MD are available for the design and development of control systems. On the other hand, there are dedicated tools such as MAST and PyCPA for analyzing and configuring real-time scheduling algorithms. Three approaches with associated tools that are presented hereafter go in the direction of a combined approach, *i.e.* to support control system design considering the influence of scheduling strategies.

SimEvents: MATLAB/Simulink [1] is a multi-domain industry standard for the design of control systems. Simulink Control Design supports the design and analysis of control systems. SimEvents is a discrete-event simulator that can be used as blocks in Simulink [2] to perform system-level simulation. It provides options to create tasks, and is able to inject network and scheduling delays with the support of the basic scheduling policies FIFO, LIFO and fixed priority scheduling. Other policies like EDF are left to the user to program. To the best of our knowledge, a significant drawback of MLSL SimEvents is that one cannot generate code from the system model consisting of SimEvents blocks. Hence, the actual realization of the controller model using SimEvents is not feasible. SimEvents is meant to model various applications where models are driven by events, starting from operation research and manufacturing processes to real-time systems. Because of this generality, it does not provide all domain-specific concepts needed for real-time systems like those available in TrueTime and T-Res.

TrueTime: The MATLAB/Simulink-based tool [3] enables the simulation of the temporal behavior of controller tasks executed on a multitasking real-time kernel. In TrueTime, it is possible to evaluate the performance of control loops subject to the latencies of the implementation. TrueTime offers a configurable kernel block, network blocks (CAN, Ethernet, etc.), protocol-independent send and receive blocks and a battery block. These blocks are Simulink S-functions written in C++. TrueTime is an event-based simulation using zero-crossing functions. Tasks are used to model the execution of user code. The release of task instances (jobs) is either periodic or aperiodic. For periodic tasks, the jobs are created by an internal periodic timer. Aperiodic tasks can be created in response to external trigger interrupts or network interrupts. The task code is written as code segments in a Matlab script or in C++. It models a number of code statements that are executed sequentially. All statements in a segment are executed sequentially, non-preemptively, and with a simulation time that can chosen by the developer through an annotation.

T-Res: This recent tool [5] is also developed using a set of custom Simulink blocks created for the purpose of i) simulating timing delays depending on the code execution, scheduling of tasks, and communication latencies, and ii) verifying their impact on the performance of control software. T-Res is inspired from TrueTime and provides a more modular approach to design the controller model enabling to define the controller code apart from the model of the task.

Besides these three tools introduced previously, the other tools developed in the past are in general, specialized to a certain aspect of the co-design problem. For example, Jitterbug [3] supports statistical control-performance analysis for a certain class of control systems. Also, these tools and methods focus only on the analysis and simulation level. They help the designer only with the study of system performance under the effects of timing delays, but not the system development. The system designer, then takes these analysis results into account to develop the actual embedded control algorithms in the next steps. This increases the possibility of distortions between the simulation model and the implementation.

In this paper, we propose a model-interpretation based runtime environment which can be used in a co-simulation environment to analyze the effects of delays on the performance of the control system, with the advantage that the controller model used in the simulation can be executed on the target hardware. This way, we eliminate the gaps between the simulation models and the executables. In the next section, we explain our co-design approach.

III. Co-simulation of CPAL and MLSL

CPAL is an embedded-system specific language designed jointly by our research group at the University of Luxembourg and the company RealTime-at-Work. CPAL is also a design-exploration platform to develop Cyber Physical Systems (CPS). It supports a Model-Driven Development (MDD) approach to model, simulate and verify systems.

CPAL can be used as a stand-alone simulation environment for CPS under development [4] or it can also be integrated with other simulation environments like MATLAB/Simulink (MLSL). The CPAL documentation, a graphical editor and the execution engine for Windows, Linux and Raspberry Pi platforms are available at http://www.designcps.com. The CPAL control library, needed to execute controller models written in CPAL in MLSL, and the models used in this paper are available at http://www.designcps.com/wp-content/uploads/cpal-simulink-control-library.zip.

In a control-system simulation, the controller model controls the plant model. In our proposed co-simulation approach a controller model is designed in CPAL, and the plant model is designed in MLSL. Controllers can easily be designed in Simulink too. But Simulink is not offering possibilities to study the behavior of control loops subject to scheduling and networking delays. Varying execution times, preemption delays, blocking delays, kernel overheads cannot be captured in the standard Simulink environment.

```
process sensor   : p1[10ms]();
  / * Periodic process with initial offset */
process control  : p2[15ms, 2ms]();
@cpal:sched:p1
{
 p1.execution_time = 2ms;
 p1.deadline  = 8ms;
 p1.priority  =  1;
 p1.jitter = time64.rand_uniform(0s,100us);
}
@cpal:sched:p2
{
 p2.execution_time = 3ms;
 p2.deadline  = 12ms;
 p2.priority  =  2;
 p2.jitter = time64.rand_uniform(0s,200us);
}
```

$\tau_i : (O_i, C_i, T_i, D_i)$,

τ is typically characterized by this tuple in real-time control systems

Fig. 2. CPAL real-time task model where periods, offsets, execution times, deadlines, and release jitters are specified.

In the case of the co-simulation CPAL/MLSL, Simulink acts as the primary simulator while CPAL executes the controller model as an S-function, and is being called by the Simulink engine. S-functions (system-functions) are high-level programming language description of a Simulink block written in C, C++ etc.. The CPAL control library is implemented as a *mex* (Matlab Executable) file which executes the CPAL controller model. This CPAL controller is a generic execution engine that can run any CPAL model. The CPAL source model is converted to a binary-equivalent representation using the CPAL parser. The Simulink engine interacts with the CPAL model through data flows and control flows.

Data flows are for the information exchange between the Simulink engine and the CPAL S-function, while the control flows define when Simulink invokes the CPAL S-function. Tasks and real-time schedulers are available natively in CPAL. Figure 2 shows the way to define the tasks using CPAL, called processes in CPAL. The default CPAL scheduling policy is FIFO, processes are executed in the order of their activation. Non-Preemptive Earliest Deadline First (NP-EDF) and Fixed Priority Non-Preemptive (FPNP) are also supported by CPAL.

Simulation of the plant's dynamic is done by computing model states at successive time steps over a specified duration. This computation is done by a solver provided by Simulink. Since our overall model is discrete, a variable step size solver is used in our co-simulation approach. The rationale behind this choice is that for the timing analysis of real-time control systems, it is necessary to reduce the step size (when needed) to increase the accuracy when model states are changing rapidly during zero crossing events. In the next section, our technique is exemplified on a use-case.

IV. Case-study: Servo control in an automotive function

Idle Air Control Actuator also called as Idle Air Control valve (IAC actuator/valve) is a device commonly used in fuel-injected vehicles to control the engine idle RPM. This actuator is essential because during idling (when the driver is not pressing the accelerator pedal), the throttle valve is completely closed, while the engine still needs some air to prevent the engine from stalling.

Engine ECU controls this IAC actuator electrically. The actuator is fitted such that it either bypasses the throttle or operates the throttle valve directly. The actuator consists of a servo motor that controls a plunger which varies the amount of air flowing through the throttle body. The position of the servomotor and hence the amount of air bypass is controlled digitally by the engine ECU. More air means an increase in the idle speed and less air indicates the reduction in idle speed. Servo motors, by definition, run using a control loop and require feedback of some kind to attain a desired state (position, velocity, and so on).

Though different types of control loops exist, PID (Proportional, Integral, Derivative) control loops are commonly used in servo motors. Figure 3 shows the CPAL controller model which controls three servos where the controller task is activated with input-to-output delays. The interfaces of the CPAL controller model, basically the inputs and outputs data, are exposed in the co-design model. The input parameters can be easily changed interactively in the design model itself. When using a PID control loop, tuning of the servo motor becomes necessary.

Tuning is the process of making a motor respond in the desired way. Tuning a motor can be a challenging process, but tuning has an advantage that, it lets the users have more control over the behavior of the motor. Adaptive step size for controlling the differential (D) part and integral (I) part of PID is easily achievable in CPAL by using the introspection

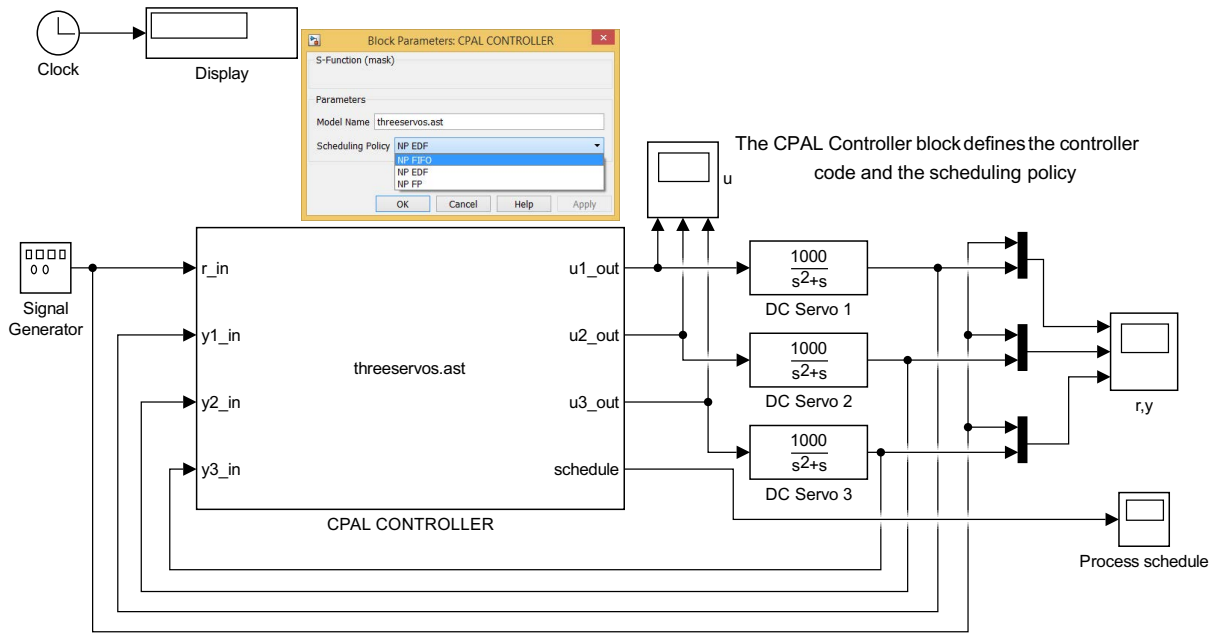

Fig. 3. CPAL controller model which controls the three servos. The controller tasks are activated with input to output delays. The scheduling policy can be selected interactively and the servo control is specified as a transfer function. The process schedule scope displays the activation pattern of tasks, and the r,y-scope displays the control performances.

mechanisms. Next section explains such features available in our approach in comparison to existing techniques.

V. Comparison of TrueTime, T-Res with CPAL co-simulation approach

A. Scheduling and task model

Unlike TrueTime and T-Res, a real-time scheduling kernel is inbuilt with the CPAL interpreter. Because our ultimate research goal [8] is to abstract all the low-level details from the system designer, where designer would define the initial task specifications and performance objectives such as stability, power-consumption, etc.. This paper is in the direction of system synthesis automation, where acceptable scheduling configurations are identified by evaluating the control performance with the scheduling delays.

The control task model in TrueTime is written in the MATLAB file format or in C++. In T-Res, the task model is a modular and triggered subsystem, executed on the occurrence of a function-call event. Outputs are latched to control the plant. A task in TrueTime and T-Res is a sequence of segments. Every segment is identified by an execution time. Task description parameters such as the task type (periodic or aperiodic), the inter-arrival time (period), the deadline and the execution time are declared in MATLAB which needs to be launched before running the controller model.

In our approach, a process, also called a task, consists of the functioning logic described in the form of a Finite State Machine (FSM). FSMs, possibly reduced to a single state, is repeatedly executed. Several tasks can be executed in parallel in which case the order of the process executions depends on the scheduling policy. The first step is to define the process, that is its list of parameters and the code itself. Then, one or several instances of the process can be created. These instances will be automatically executed at run-time by the interpreter according to the defined activation pattern. CPAL supports all three task types periodic, aperiodic and sporadic.

B. Decoupling of timing definitions from control code

In TrueTime, control functionality is combined with timing definitions. There is no separation of concerns and experts in those domains have to work together to evaluate the system performance. On the other hand, T-Res provides a segregated approach, where the control aspects of the controller model are implemented using Simulink blocks and scheduling aspects are dealt with using T-Res blocks. T-Res blocks are again not meant for code-generation and the subsequent steps in development life cycle. As shown in Figure 4, CPAL also employs a segregated approach where control aspects are separated from timing aspects, so that both domain designers can work independently and seamlessly.

C. Influence of scheduling choices to control performance

The Figure 4 is a snippet of code of the servo-control example described in previous section, which is originally an example from the TrueTime distribution. This example is here re-used with the same parameters as in [5] so as to check that same results are achieved with our approach. It should be noted that the CPU utilization factor with this parameter set is 1.23, the system is thus overloaded and not all task instances can be executed.

```
/* Control code */
processdef servo_controller(out PID_data:data,in float32:r,in float32:y,out float32:u)
{   state Main {
    var float32 : P; var float32 : I; var float32 : D;
    P = data.K*(data.beta*r-y);
    I = data.Iold;
    D = data.Td/(data.N*data.h+data.Td)*data.Dold
    +data.N*data.K*data.Td/(data.N*data.h+data.Td)*(data.yold-y);
    u = P + I + D;
    data.Iold = data.Iold + data.K*data.h/data.Ti*(r-y);
    data.Dold = D;
    data.yold = y;   }
/* Timing code -  process instance defines default period, and required parameters */
process servo_controller  :servo_controller3[4ms](data_control3,r_in, y3_in, u3_out);
process servo_controller  :servo_controller2[5ms](data_control2,r_in, y2_in,  u2_out);
process servo_controller  :servo_controller1[6ms](data_control1,r_in, y1_in,  u1_out);
/* CPAL process timing annotations */
/* CPAL annotations do not require any control code modifications */
@cpal:time{
/*process priority and execution time, larger the number lower the priority*/
    servo_controller1.priority = 3;
    servo_controller1.execution_time =  2ms;
    servo_controller2.priority = 2;
    servo_controller2.execution_time =  2ms;
    servo_controller3.priority = 1;
    servo_controller3.execution_time =  2ms;   }
 /* Task 3 is overwritten for execution time and jitters - to show CPAL capabilities*/
 /* for the experiment in the paper below feature is not necessary and not used */
 /* this annotation gives varying (random) execution time between 1 and 2 ms */
 /* jitter annotation gives variable task scheduling jitter between 0 an 100us */
@cpal:sched:servo_controller3
{   servo_controller3.execution_time = time64.rand_uniform(1ms,2ms);
    servo_controller3.jitter = time64.rand_uniform(0s,100us);
}
```

Fig. 4. Separation of control and timing aspects in a controller model of the servo motor example. Three servo controllers tasks are defined with associated task parameters. Here, task 3 has highest priority with an execution time and release jitter which are varying. Task 1 has lowest priority with constant execution time. Control code is decoupled from timing definitions given as annotations.

Figure 5 shows the task activation pattern of the controller tasks which control the three different servo motors under the Fixed-Priority Non-Preemptive (FPNP) policy. Task 1 with the lowest priority is activated less frequently than the two others which translates into an unstable control output (see Figure 6 top). When NP-EDF is used, all tasks are activated as frequently and the system tends to stable.

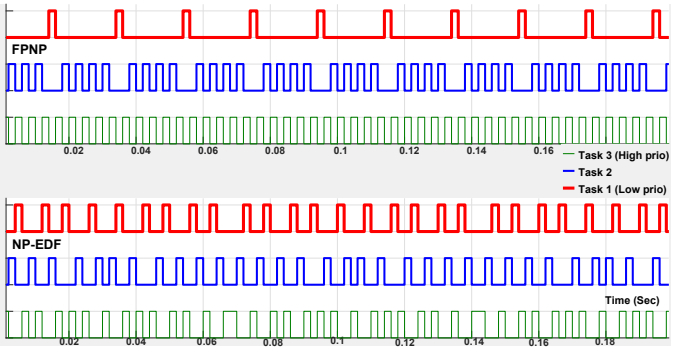

Fig. 5. Task activations of the three tasks of servo control use-case. Task 1 is the lowest priority when FPNP is selected (top diagram) and many instances are not executed. When NP-EDF becomes the scheduling option, the number of instances is evenly balanced between tasks.

D. Self-adapting mechanisms

In CPAL, it is possible at run-time to query execution characteristics such as process id, period, offset, jitter, priority, deadline and activation time of the current and previous instance of any task. This feature is typically used to implement control algorithms that must adapt to their frequency of execution or their execution jitters by compensating them.

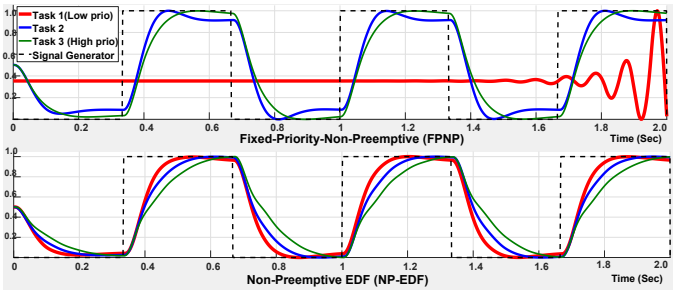

Fig. 6. Control system performance with the two scheduling options. Under FPNP, unstability happens due to less number of execution of the lowest priority task 1. Under NP-EDF, system performance is improved (no oscillations) due to an equally balanced number of execution accross tasks.

Using these introspection mechanisms, it is possible to get the actual run-time period to calculate the effective step-size that will influence the differential and integral component of PID control. Introspection is also useful when unstable behavior is observed due to a less number of low priority task executions as in the example above. By reducing at run-time the frequency of execution of the two highest priority tasks using the @cpal:sched:period annotation, we were able to achieve a stable system in the previous example.

E. Benefits of re-using controller code on target hardware

A key advantage of our co-simulation approach is that the same controller model used to evaluate control performance in the design phase can be re-used directly to target hardware to implement the final system. As discussed in [7], this development cycle with less steps allows reduced interactions between control and software engineers.

TrueTime and T-Res only supports simulation mode and not real-time execution on target. In CPAL, the same code can be re-used on target with the difference that timing annotations (e.g., jitters, execution times) are ignored. On the other hand, the annotations that define the scheduling (e.g., priorities, deadlines) are considered as run-time parameters for the execution engine. Table I summarizes the comparison of the approaches discussed in this paper to evaluate the performance of latency-sensitive control systems.

VI. Conclusions and Outlook

The timing behaviour of control tasks is a critical concern in real-time digital controllers. These delays, such as start-of-execution jitters, or missed executions, affect the system performance and need to be considered during the design phase. The approaches developed in the past to study the performance of the control system due to run-time delays are simulation approaches, not supporting the implementation of the system. In this paper, a model-driven co-simulation based development approach is proposed and illustrated with a servo control example. This federated approach provides the designer with both simulation and execution capabilities to define and validate functional and non-functional behaviors. The benefits of the proposed technique over the state-of-the-art are discussed in this paper, amongst which a good support for rapid-prototyping to shorten the development cycle.

TABLE I
COMPARISON OF CPAL CO-SIMULATION IN MATLAB/SIMULINK WITH EXISTING APPROACHES.

Characteristics	TrueTime in MLSL	T-Res in MLSL	SimEvents in MLSL	CPAL in MLSL
Scheduling kernel	Separate kernel	Separate kernel	Built-in kernel	Built-in kernel
Task model	.m file / C++	Graphical model	Graphical model	CPAL model(C alike)
Control and timing separation	No	Yes	Yes	Yes
Scheduling policies supported	Almost all	Almost all	Only basic policies	FIFO, FPNP, NPEDF
Target Execution environment	No	No	No	Yes (Raspberry Pi, Freescale FRDM)
Simulation strategy	Discrete event simulation with variable step-size with zero crossing	Discrete event simulation with variable step-size with zero crossing	Improved Discrete event simulation	Discrete event simulation with variable step-size with zero crossing
Control system compensation	Possible	Possible	Not possible	Introspection mechanism enables tuning of control law
Rapid-prototyping	Not possible	Not possible	Not possible	Easily achievable
Code-generation	Not possible, code cannot be generated from TrueTime blocks	Not possible, code cannot be generated from T-Res blocks	Not possible, code cannot be generated from SimEvents blocks	Not necessary, controller model can run directly on target
Model – code gaps	may happen	may happen	may happen	Model is code, no gaps

ACKNOWLEDGMENT

This research is supported by FNR (Fonds National de la Recherche), the Luxembourg National Research Fund (AFR Grant n°10053122).

REFERENCES

[1] MATLAB/Simulink, SimEvents. http://nl.mathworks.com/products/simevents/. Accessed: 2016-08-26.

[2] C. G. Cassandras, M. I. Clune, and P. J. Mosterman. Hybrid system simulation with simevents. In *2nd IFAC Conference on Analysis and Design of Hybrid Systems*, June 2006.

[3] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3), June 2003.

[4] L. Fejoz, N. Navet, S. M. Sundharam, and S. Altmeyer. Applications of the cpal language to model, simulate and program cyber-physical systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016.

[5] M. Morelli, F. Cremona, and M. Di Natale. A system-level framework for the evaluation of the performance cost of scheduling and communication delays in control systems. In *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, July 2014.

[6] N. Navet, L. Fejoz, L. Havet, and S. Altmeyer. Lean model-driven development through model-interpretation: the CPAL design flow. In *Embedded Real-Time Software and Systems (ERTSS2016)*, January 2016.

[7] S. M. Sundharam, S. Altmeyer, and N. Navet. Model interpretation for an AUTOSAR compliant engine control function. In *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2016.

[8] S. M. Sundharam, S. Altmeyer, and N. Navet. An optimizing framework for real-time scheduling. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2016.