Multi-Domain Virtual Prototyping in a SystemC SIL Framework: A Heating System Case Study

Nikolaos Ilieskou*†, Marijn Blom*†, Lou Somers*†, Michel Reniers* and Twan Basten*
*Eindhoven University of Technology, Eindhoven, The Netherlands
Email: nilieskou@tue.nl, m.blom@student.tue.nl, {l.j.a.m.somers, m.a.reniers, a.a.basten}@tue.nl
†Océ Technologies, Venlo, The Netherlands
Email: {nick.ilieskou, marijn.blom, lou.somers}@oce.com

***Abstract*—This paper presents a proof-of-concept for a modular SystemC SIL (Software-in-the-Loop) simulation environment, using a blackboard-like architecture. The proposed SIL framework integrates embedded control software with simulators developed in SystemC/SystemC-AMS or external tools, like MATLAB. The environment has been validated by a heating application for a professional printer, as example of an MDVP (Multi-Domain Virtual Prototyping) application. Our goal is to evaluate the use of SystemC/SystemC-AMS and to address the challenges in developing multiple-domain prototypes and blackboard-like SIL frameworks using this technology.**

I. Introduction

The significance of time to market is constantly increasing in many industries. Especially in the development of complex high tech electromechanical systems, where the development proceeds through a number of subsequent phases, each embodied in a new prototype or lab model, it is important to be able to perform quick iterations. Usually, such products consist of elements that have to be designed and developed simultaneously and then integrated. An example of such a system is a professional printer, which contains electronics, mechanical parts and embedded software for controlling and interacting with actuators and sensors. To be able to get early feedback on the performance of a new prototype, software engineers have to develop and test embedded software before the plant elements that it is governing are available.

In order to do that, there is a need for a simulation framework that enables communication between simulators of the plant elements and the embedded software controlling them. The communication has to be synchronized and handled by the framework. Moreover, each of the plant element simulators can be a combination of models from multiple domains (chemistry, physics, mechanics, etc.). Thus, it is also necessary to develop multiple-domain virtual prototypes, which can be coupled with the simulation framework.

Software-In-the-Loop (SIL) environments provide a framework in which some or all plant elements are simulated. Embedded software modules are integrated into a SIL environment in order to test their functionality and explore their limitations [1]. As a result, software engineers can identify potential bugs and test software earlier in the development process without waiting for mechanical or electronic parts to be available.

In order to implement a SIL framework, two elements have to be determined. The first is a software architecture that ensures the quality attributes of flexibility, modularity and heterogeneous systems cooperation. A blackboard-like architecture is a suitable candidate. It provides a way of interaction between different modules, known as knowledge sources, through a global database. Each knowledge source is independent and can provide different expertise to the system. The interaction between knowledge sources takes place by writing to or reading values from the database. When knowledge sources need to exchange data, one will write data to the database and the other will consequently read that data from the database. [2].

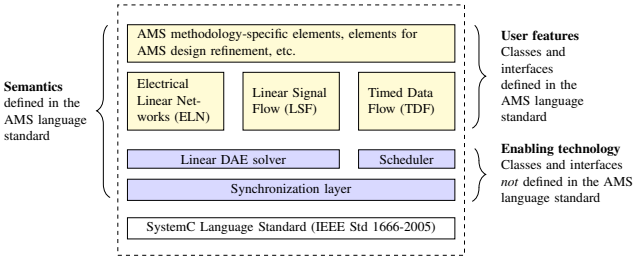

Fig. 1: SystemC-AMS language architecture. From [3].

The second element of the SIL framework is a simulator for handling and scheduling multiple-domain virtual prototypes, which can be developed at different levels of abstraction. SystemC is a suitable candidate to provide this element since it has a discrete simulation kernel. SystemC extensions, like SystemC-AMS, provide a variety of Models of Computation (MoC) that enable the development of multiple-domain simulators at a variety of abstraction levels [4].

In this paper, the SystemC extension for Analog and Mixed-signal Systems (SystemC-AMS) is used as it provides the proper MoCs for the target application. The SystemC-AMS architecture builds upon and interacts with the existing pure discrete SystemC environment (Figure 1) and uses both discrete-time static non-linear and continuous-time dynamic linear model abstractions to provide three MoCs, namely timed data flow (TDF), linear signal flow (LSF) and electrical linear networks (ELN) [4]. These MoCs or modelling formalisms have different use cases as can be seen in Figure 2, but they

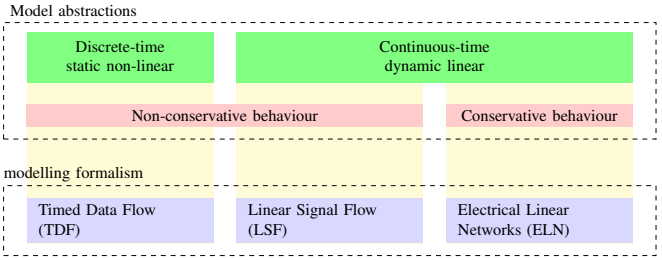

Fig. 2: SystemC-AMS modelling formalisms and their use cases. Adapted from [3].

can be used alongside each other to describe different system components.

The H-INCEPTION project [5] aims to provide a unified design environment for Multiple-Domain Virtual Prototyping (MDVP), and as partners of this project we present a proof of concept for a SystemC SIL framework able to connect multiple-domain physical system simulators and embedded software. This is a flexible and modular simulation environment for testing embedded control software for complex multiple-domain systems. SystemC provides the backbone for this implementation. The SIL framework is validated by means of a printer heating model, which has to be precise enough to test the embedded software, but does not need more accuracy. Our goal is to evaluate and address the challenges of using SystemC/SystemC-AMS for developing multiple-domain prototypes and a blackboard-like SIL framework.

The rest of the paper is organized as follows. The next section provides the heating system as motivating example for a multiple-domain virtual prototype. Section III discusses related work. Sections IV and V analyse the implementation of the SIL framework and the heating system model. Challenges encountered are presented in Section VI and conclusions and future work in Section VII.

II. Motivation Example

In order to motivate a multiple-domain virtual prototyping process, we choose as example the heating system of a professional printer. The modelled system simulates the movement and temperatures of the paper sheets as well as the temperature of some printer heating components. The heating system (Figure 3) consists of four main components: the paper path, the heatXchange unit (1), the preheater (2) and the toner transfer belt (3) [6].

The heatXchange unit transfers heat from the hot paper at the end of the paper path to the cold paper at the start of the paper path. This is done by a thin foil that acts as a conductor. After being heated up by the heatXchange unit, the paper passes the preheater, a component with a metal piece inside that heats up (due to a coil receiving power from the power supply) and transfers heat to the paper. After passing the preheater, the paper comes in contact with the toner transfer belt. This is where the image is transferred to the paper from the imaging drum (the cylinder labeled 4 in Figure 3). In

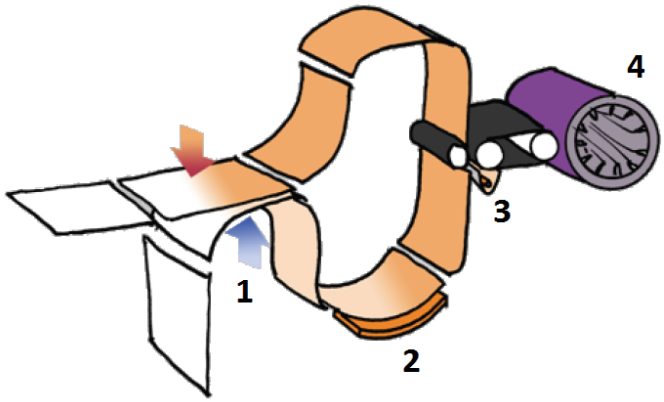

Fig. 3: Paper path and heating components.

order to transfer the image correctly, both the paper and the toner transfer belt need to be at the correct temperature (more precise: within a certain temperature range). The toner transfer belt is heated by a halogen lamp (the heat-on-demand unit) receiving power from the power supply. After merging with the toner, the (still hot) paper will continue down the paper path and reach the hot side of the heatXchange unit.

The heating system is modelled at an abstract level. This is done for several reasons. The main goal is to evaluate and identify potential challenges of modelling in SystemC and SystemC-AMS, which does not require a detailed model. Also, due to the modularity of the SIL framework, it is possible to model a subsystem in higher detail and couple it with the SIL framework separately, communicating the values of shared variables through the database. Moreover, the purpose of the simulator is to test the embedded software and the model does not require an accuracy higher than the embedded software needs.

III. Related Work

Various Software-in-the-loop frameworks have been proposed in literature [1], [7]–[13]. Depending on the target application and requirements, different development methods have been used. A SIL framework is commonly composed of simulators and a SIL engine which consists of various components such as a scheduling engine, a database, a synchronization layer, and others, depending on the target application and its complexity.

MATLAB/Simulink is a popular choice for developing models of actuators, sensors, or plant elements and has been used in many cases [7], [10]–[12]. SIL engines have been developed in many different ways: [10], [11] implemented a SIL engine as a .NET application, while [7] used MATLAB and [12] wrapped to a MATLAB S-function together with a real time OS. A discrete-event simulation engine named DEUS was used in [9] and a custom C/C++ SIL implementation was deployed in [8].

Unlike most of these implementations, we provide a modular SIL framework with which we can couple models from var-

ious modelling environments and languages. Our SIL framework was developed in SystemC and it enhances modularity by enabling the coupling of multi-domain models created with different tool sets. In order to achieve the modular behaviour, we were inspired by the functionality of Functional Mock-up Interface (FMI) for Co-Simulation [14].

There is a large variety of modelling languages and simulation tools available, both open source as well as commercial packages, which are often designed specifically for particular domains [15]–[18]. Most of them, however, are only suited for a single domain, or cannot combine discrete-event simulators with continuous-time simulators [19]–[23]. Most of them also do not offer the ability to model a system with different levels of abstraction.

IV. SIL ARCHITECTURE

In this section the proposed SIL architecture is discussed. We first present the basic structure and requirements of the SIL environment. Then we discuss scheduling of models and framework elements. Scheduling is the process by which we determine the execution order of the simulators and the embedded control software, in order to correctly simulate the behaviour of the real-world system. By scheduling the elements of the simulation correctly we ensure correct simulation results. Finally, we explain the model structure and behaviour that can be integrated into the SIL framework.

A. SIL structure and requirements

The proposed SystemC SIL framework supports integration of SystemC, SystemC-AMS and external physical simulators. The latter are simulators generated from different environments, like MATLAB. In order to embed external models and embedded software, created outside of the SystemC environment, into a SystemC SIL framework, we transform them into a Dynamic-link Library (DLL). External model coupling is enabled through a custom-defined external interface.

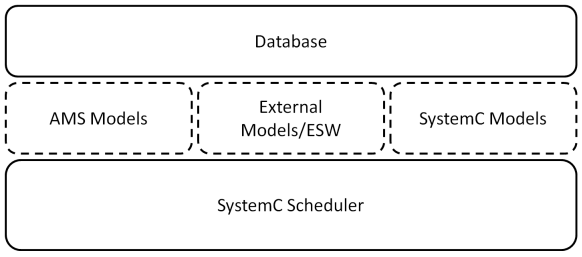

Fig. 4: SystemC SIL framework structure.

Figure 4 illustrates the structure of the SystemC SIL framework. The SystemC Scheduler, which is the bottom layer, proceeds in discrete time instants and is responsible for calling simulators and embedded software. For simplicity we refer to both simulators and embedded software modules as model(s). The execution order of models is defined in a configuration file. This file contains information regarding which models will take part in the simulation, their dependencies and frequencies. In order for the SIL framework to correctly incorporate these models, the framework has to adhere to the following requirements, which are explained below:

- **R1** The SIL should be able to handle sequential execution of models.
- **R2** The SIL should be able to simulate the parallel execution of independent models.
- **R3** The SIL should support models with independent frequencies.

Sometimes a model has been split in two parts that are executed sequentially within one time instance. This can be because of model development by two different teams or because of distinct models that give a more precise result in combination with each other. **R1** covers this case.

Figure 5 shows an example illustrating the reasons why models have to execute in parallel (**R2**). The old output of model A is the input of model B and C for any time instance. All of them have to execute at the same time. In reality, models A, B and C have to execute in parallel. However, SystemC serializes parallel execution by updating the signal values in a later time. Because the SIL framework is based on the blackboard architecture, models that execute in parallel do not communicate directly through signals but through the database. The SIL framework has to ensure that model A will not feed models B and C with a new value of X2 at the same time instance. To ensure correct results, all the models have to use the old input values.

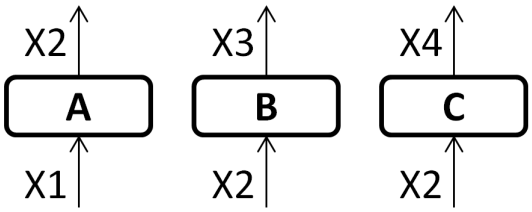

Fig. 5: Possible dependency between models.

To achieve such behaviour, each model is executed in two distinct phases. In the first phase, a model reads its input variables. In the second phase, the model executes and outputs the generated values. For the example illustrated in Figure 5, models A, B and C first have to read their inputs and then execute and write their outputs. The order in which the models will read or execute-write is arbitrary for both phases. In order to achieve correct behaviour, we place models with such dependencies or models that have the same frequency in the same *group*. Groups are specified in the configuration file.

Finally, **R3** is a crucial requirement for a complex system simulation environment with multiple models. In such an enviroment each entity may have its own frequency according to its needs. For instance, an embedded control software module might be called every 1ms, while a slow mechanical part can be called every 100ms.

As mentioned before, plant elements have to communicate with each other or with embedded software modules. Our SIL framework enables model communication through a database

according to the blackboard architecture. Figure 4 shows at the top the database which is responsible for interaction between models. In order for a model to read and write values to the database, an interface has been implemented by means of a SystemC module DB_i, which has access to the database and is responsible for providing simulators with data and writing back their outputs. Each model is linked bidirectional to a DB_i instance through two ports of a custom data structure <packet_type>. We refer to these ports as db_in and db_out on both the DB_i side and the model side. The <packet_type> is a map data structure between a double variable and a string representing the name of the variable. It also includes additional information such as the action a particular model wants to perform on the database, like read or write. The DB_i modules are sensitive to their input: they will execute when input is available on the db_in port.

B. Scheduling in SIL

The SIL framework uses the SystemC scheduler to schedule the execution of the models. The configuration file is parsed before the simulation starts. The information extracted from the configuration file, i.e. model dependencies and frequencies, is used to set up the simulation environment in such a way that the SystemC scheduler will call the models in the correct execution order. For each model a clock is created. This clock is linked to the respective model. The structure of the clock is the following:

$$Clock(Period, Delay)$$

Period is used for calling each model at the correct time instance, while the *Delay* attribute takes care of sequential dependencies (**R1**). Assume that a number of models have to be executed in the same time instance in a sequential fashion. The SystemC scheduler will execute them in an arbitrary order. To force a specific execution order, we introduce very small delays between the execution of these models and thus achieve the desired sequential execution order. To keep *Delay* small, we define it as three orders of magnitude smaller than the smallest period. Suppose we have models X and Y with the same period and a sequential dependency. Model Y will follow X after an insignificant amount of time, equal to the Delay attribute. As example, consider the following configuration:

```
Group0:    A,B,C   Period:  1ms
Group1:    D,E     Period:  3ms
Group2:    H       Period:  3ms
...
Group600:  P,Q,R   Period:  yms
```

The group numbers represent the delay attribute. We assume that 1ms is the smallest period. At t = 3ms models A, B and C will execute in an arbitrary order, at 3,001ms models D and E will execute, at 3.002ms model H will execute, and so on. This simulates the sequential execution of these groups at 3ms. All these models are calculating values for t = 3ms, ignoring the execution delay.

If there are many groups, the delay used to simulate sequential execution can become relatively large. In the example the smallest period is 1ms and the number of groups is 601. The delay for the last group is 60% of the smallest period. However, the results are still correct since the models in Group600 execute as if they were called without the additional delay. If the number of groups is larger than 1000, the delay should be set to a value that is more than three orders of magnitude smaller than the smallest period.

Bear in mind that the configuration file is responsible for a feasible scheduling. The SystemC scheduler, with the help of the configuration file, will call the models in the correct order. This is ensured by first properly creating modules related to models and then linking them to SIL related modules, such as DB_i.

The SIL framework supports three categories of models: SystemC, AMS, and external simulators or embedded control software modules in a DLL form. In the following we explain how these three categories are structured. Each model has the same behaviour, as can be seen in Figure 6. They start with a reading phase, in which their input variables are requested from the database. During this phase they fill a <Packet_type> structure with the data they intend to read and then they output them to the port which is connected to their DB_i instance. Then they wait until their requested data are available. After the DB_i instance delivers the data, the models proceed to the execute-write phase during which they execute and then output the results through the <Packet_type> structure towards their DB_i instance. All model entities are sensitive to a unique clock and to the db_in port. The former is used to trigger a model in the first delta cycle of a time instance. A delta cycle is a step inside a particular simulation time instance. A simulation time instance can have one or more delta cycles [24].

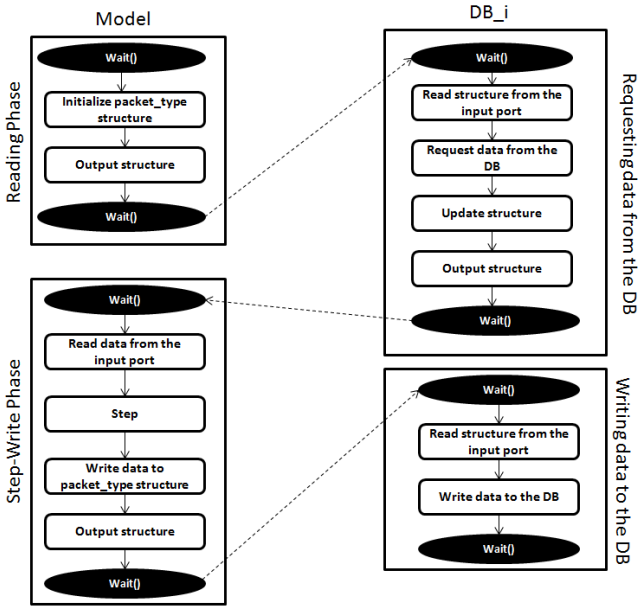

Fig. 6: Flow chart of model and DB_i modules.

TABLE I: Example of SystemC execution queue.

	Delta Cycle	Module	Description
t = t_1	1	A	Module A request to read
	1	B	Module B request to read
	2	DB_ia	DB_ia sends data to A
	2	DB_ib	DB_ib sends data to B
	3	A	A gets data, executes and request to write
	3	B	B gets data, executes and request to write
	4	DB_ia	DB_ia writes data from A
	4	DB_ib	DB_ib writes data from B

To make the scheduler functionality more clear, we consider the following example. Table I illustrates the execution order of two models A and B that are in the same group and thus have the same period. At time t_1 the SystemC scheduler inserts models A and B in the execution queue. During the execution of the first model (A), a request will be sent towards DB_ia and thus DB_ia will be placed in the execution queue for the next delta cycle. The same will happen for DB_ib when model B executes. During the second delta cycle, DB_ia and DB_ib will execute to serve the read requests from A and B. The outcome of their execution gives the values of the requested variables back to the models. Because both A and B are sensitive to their In_db port they will be placed in the execution queue again. During the third delta cycle, A and B will read the variables, execute, and write their results to db_out, causing the execution of the DB_i instances to be scheduled for the next delta cycle.

C. AMS and SystemC models

AMS and SystemC models are coupled to the SIL framework with a generic adapter as presented in Figure 7. The adapter is responsible for providing communication between the database and the model. It uses the In_db and Out_db ports to communicate with a DB_i instance. It also provides the two phase functionality of models, i.e. the read and execute-write phases. The adapter is sensitive to the the In_db and to the On port, which is connected to the clock that triggers the model. It has n signals towards the model, where n is the number of inputs plus the number of outputs of the model.

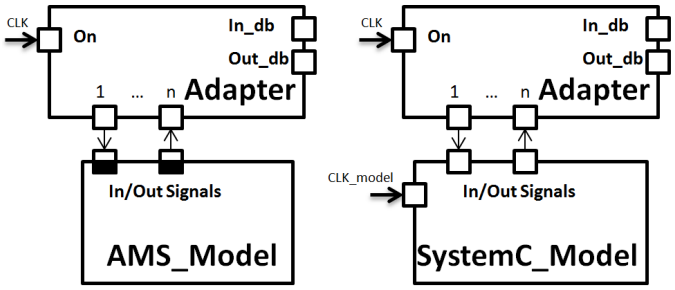

Fig. 7: SystemC/AMS models and adapters basic block diagram. An AMS_Model uses converter ports.

In order to integrate an AMS model into the SIL framework, we need to create a SystemC hierarchical module that contains an AMS module cluster, which represents the actual simulator. We refer to hierarchical modules that contain AMS clusters as AMS_Models. Because an AMS_Model is a continuous model, it needs to have converter ports to convert discrete signals coming from and going to the adapter. The modules inside an AMS_Model execute based on a schedule which is automatically generated by the AMS scheduler based on the set time step of the modules.

The adapter feeds the AMS_Model with data and then it reads the output of the AMS_Model at the same time instance. This input data will be processed by the AMS_Model at the next trigger time, while the output data read by the AMS_Adapter is the result from the previous trigger time. Trigger time refers to the time instance at which a model is called. Physical attributes like position or speed cannot change instantaneously, and the SIL framework is using AMS to model physical systems. Consequently, the fact that values calculated by the AMS_Model are based on past input values does not result in incorrect output data.

SystemC models are also coupled to the SIL framework using an adapter. A SystemC model is a hierarchical SystemC module containing interconnected discrete modules. In Figure 7 we refer to this model as SystemC_Model. The SystemC_Model can be triggered by different events. One of the possibilities is triggering by a clock. If this is the case, then all clock-triggered modules inside the hierarchical module have to be triggered by the same clock and thus have the same period. In case a SystemC model has modules that are triggered by clocks with different periods, they have to be grouped in different hierarchical modules.

D. External models / embedded software

A DLL entity uses a DLL_Adapter to connect to the SIL framework in a modular way. Figure 8 shows the basic block diagram of a DLL_Adapter. The DLL_Adapter is responsible for the communication with the database through the DB_i module, the implementation of the read-execute-write functionality and is triggered by the On and In_db input ports. The communication between the DLL_Adapter and the DLL is achieved using an external SIL interface.

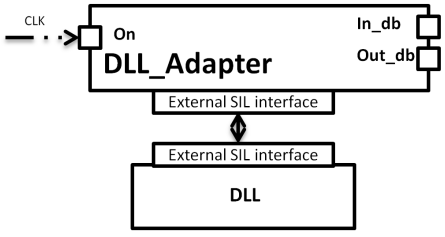

Fig. 8: DLL model basic block diagram.

Summarizing, the SIL framework can integrate plant element simulators developed in SystemC, SystemC-AMS, or external tools like MATLAB with embedded control software. We have chosen a blackboard-like architecture, where models communicate through a database, because of the modularity

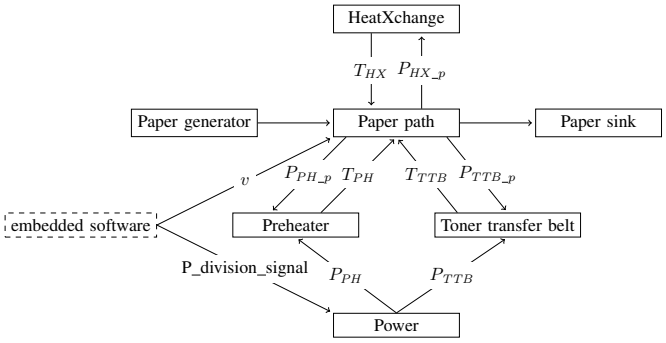

Fig. 9: Abstracted heating system

and flexibility it offers. The execution order and the dependencies of the models are defined in a configuration file which is used to properly set up models and SIL framework elements, like DB_i, so that a correct scheduling can be achieved.

V. Heating system modelling

In this section the heating system model is discussed. We explain the governing equations, discuss the control, and address constants and values. The heating system is modelled at a high abstraction level. This, and the non-linear behavior in the model's equations, makes the TDF MoC a suitable choice as modeling formalism.

A. Abstracted heat system model

The heating system is modelled with several modules (Figure 9). The *Paper generator* module signals the *Paper path* module every time a new sheet of paper is put into the system. The paper path module contains the position and temperature values for every sheet of paper in the system. The position of the sheets along the paper path is governed by the simple formula $\dot{x} = v$, where v is a parameter sent by the embedded software. The embedded software also sends a signal to the *power* module that determines how much power is sent to the *preheater* module and how much is sent to the *toner transfer belt* module. The preheater module receives two signals: the amount of power added to the preheater and the amount of power absorbed by the paper path. It sends its current temperature to the paper path module. The physical preheater element has a heat capacity C_{PH}, will heat up when power is supplied to the element and will cool down due to heat transfer to the environment and paper. The toner transfer belt is modelled in the same way as the preheater, but with heat capacity C_{TTB}. The heatXchange unit receives the amount of power drawn or added by the paper path and sends out its temperature to the paper path. The thin foil is modelled as an object with a small heat capacity C_{HX}. Finally, if a sheet of paper leaves the paper path, the paper path module sends a signal to the *paper sink* module.

B. Control

The embedded software is responsible for the power management and the speed control of the paper path. In our abstracted model of the heating system, that is intended as proof of concept of our SIL architecture, the embedded software is not used for providing a proper control of the heating system. It is simply in place to test the functionality the SIL framework and the heating system model. The implemented control switches the power supply between the preheater and the toner transfer belt with a constant frequency $f_{switching}$. The speed is set to a constant 0.5m/s.

C. Temperature equations

All components are modelled to be of uniform temperature. The temperature of the preheater is governed by Equation 1 where T_{PH} [°C] is the temperature of the preheater, C_{PH} [J/°C] is the heat capacity of the preheater, η_{PH} [-] is the efficiency, P_{PH} [W] is the power added to the preheater, k_{env} [W/°C] is the heat transfer coefficient between the system components and the environment, T_{env} [°C] is the environment temperature and P_{PH_p} [W] is the power transferred to the paper path.

$$\dot{T}_{PH} = \frac{1}{C_{PH}}(\eta_{PH} \cdot P_{PH} - k_{env}(T_{PH} - T_{env}) - P_{PH_p}) \quad (1)$$

The temperature of the toner transfer belt is governed by Equation 2, which is very similar to Equation 1.

$$\dot{T}_{TTB} = \frac{1}{C_{TTB}}(\eta_{TTB} \cdot P_{TTB} - k_{env}(T_{TTB} - T_{env}) - P_{TTB_p}) \quad (2)$$

Equation 3 governs the temperature of the heatXchange unit. This equation is similar to Equations 1 and 2, but without the added power from the power supply.

$$\dot{T}_{HX} = \frac{1}{C_{PH}}(-k_{env}(T_{HX} - T_{env}) - P_{HX_p}) \quad (3)$$

The temperature of the paper is governed by Equation 4. The paper sheets are considered to have a uniform temperature.

$$\dot{T}_{p,i} = \frac{1}{C_p}(P_{PH,i} + P_{HX,i} + P_{TTB,i} - k_{env}(T_{p,i} - T_{env})) \quad (4)$$

In this equation $T_{p,i}$ [°C] is the temperature of paper sheet i, C_p [J/°C] is the heat capacity of a single sheet of paper, $P_{PH,i}$ [W] is the power added to sheet i from the preheater, $P_{HX,i}$ [W] is the power added to sheet i from the heatXchange unit and $P_{TTB,i}$ [W] is the power added to sheet i from the toner transfer belt.

Equation 5 is used to calculate the power added from the preheater used in Equation 4.

$$P_{PH,i} = h_{PH} \cdot A_{PH,i} \cdot (T_{PH} - T_{p,i}) \quad (5)$$

Here, h_{PH} is the conductivity [W/m^2°C] between the preheater and the paper and $A_{PH,i}$ is the contact area between the preheater and sheet i. This area is calculated with Equation 6 if $d_p \leq d_{PH}$ or Equation 7 if $d_p > d_{PH}$, where d_p is the paper width and d_{PH} is the width of the preheater. These formulas can easily be deduced from Figure 10.

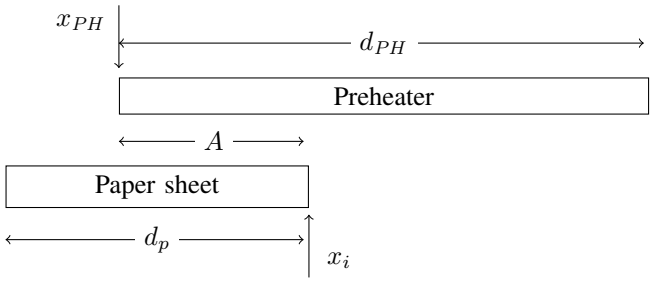

Fig. 10: Paper sheet passing under the preheater

$$A_{PH,i} = l \cdot \begin{cases} 0, & \text{if } x_i \leq x_{PH} \\ x_i - x_{PH}, & \text{if } x_{PH} < x_i \leq x_{PH} + d_p \\ d_p, & \text{if } x_{PH} + d_p < x_i \leq x_{PH} + d_{PH} \\ d_p - x_i + x_{PH} + d_{PH}, & \text{if } x_{PH} + d_{PH} < x_i \leq x_{PH} + d_{PH} + d_p \\ 0, & \text{if } x_i > x_{PH} + d_{PH} + d_p \end{cases} \tag{6}$$

$$A_{PH,i} = l \cdot \begin{cases} 0, & \text{if } x_i \leq x_{PH} \\ x_i - x_{PH}, & \text{if } x_{PH} < x_i \leq x_{PH} + d_{PH} \\ d_{PH}, & \text{if } x_{PH} + d_{PH} < x_i \leq x_{PH} + d_p \\ d_p - x_i + x_{PH} + d_{PH}, & \text{if } x_{PH} + d_p < x_i \leq x_{PH} + d_{PH} + d_p \\ 0, & \text{if } x_i > x_{PH} + d_{PH} + d_p \end{cases} \tag{7}$$

In these equations, x_i is the position of paper sheet i, x_{PH} is the position of the preheater, d_p and l are the width and length of the paper sheets respectively and d_{PH} is the width of the preheater.

The total power drawn from the preheater by the paper sheets is simply $P_{PH_p} = \sum_{i=1}^{n} P_{PH,i}$, where n is the total number of paper sheets in the paper path.

The heatXchange unit and toner transfer belt are modelled in the same way as the preheater, with the exception that the heatXchanger unit has two separate positions at which it makes contact with the paper and has no added heat from the power supply.

All values for constants and coefficients are determined by taking reasonable estimates from real world examples.

VI. SystemC modelling and evaluation

In this section we address the challenges we encountered during the implementation of the SIL framework. We also discuss the advantages and the disadvantages of using SystemC in the context of this paper. Finally, challenges related to developing multiple-domain virtual prototypes using the SystemC-AMS extension are presented.

A. SIL framework challenges

We use a blackboard-like architecture for the SIL simulation framework and the greatest challenge was to use the SystemC scheduler in accordance with our requirements and in combination with the basic structure elements of the architecture, e.g. the database. In the SystemC environment, elaboration is the phase before the simulation starts. It involves the connectivity and the initialization of the data structures [25]. In order to correctly schedule model executions, we implemented a smart elaboration phase which introduced additional delays during the simulation. In particular, to satisfy requirement **R2** we had to divide model execution in two phases: a read phase and an execute-write phase. After each phase, a DB_i instance provides or stores data from/to the database. A pure SystemC implementation could avoid all these additional steps. However, having a blackboard-like architecture has its advantages. It has a centralized data storage which enables us to have an overview of the values of the simulation variables at any time and use this data for other functions, like visualization of the whole system. Without the database, it would be more difficult to extract useful information from the framework, because SystemC works with distributed data and the useful data would be located in signals which connect all models of the simulation.

Moreover, the blackboard-like architecture provides the modularity quality attribute for the SIL environment, as integrating a new model with our framework is easy because we just have to connect the new model with a new DB_i instance. Implementing this in a SystemC oriented architecture would be more difficult as the model would have to be connected with one or more other models.

To summarize, the combination of the blackboard architecture and SystemC was a choice which is a tradeoff between SystemC capabilities, like MDVP, performance and modularity. As mentioned above, the greatest challenge was to schedule the models in accordance with the correct execution order. This difficulty resulted in a small performance penalty.

B. SystemC language evaluation

SystemC provides adequate documentation. However, it requires good programming skills to develop models of plant elements. This fact may cause difficulties to people from different domains such as electrical engineering, mechanical engineering or even physics that have to develop models which will be integrated into the SIL. Another disadvantage of SystemC is performance. Although our performance penalty was caused by to the blackboard-like architecture, SystemC could be much more efficient if it would offer real parallel execution of modules.

On the other hand, SystemC provides a lot of abstraction levels. It is possible to develop models from the register transfer level to transaction level modeling and hence provide abstract models depending on the requirements of the simulation. SystemC also provides an event based scheduler that fits the general needs of a general implementation of a SIL. SystemC is an open source language with increasing popularity. Also, the SystemC-AMS extension and the MDVP framework offer the ability to model multiple-domain systems and thus are a perfect match for model development for the SIL framework. Furthermore, SystemC-AMS is an extension of SystemC and as a result one can develop all models and

the SIL framework in the same language. More precisely, SystemC and its scheduler can be used for developing the SIL framework, and SystemC and SystemC-AMS can be used for developing multiple-domain virtual prototypes. This leads to a simple SIL framework structure including one main technology, i.e. SystemC. By using just one language we can also avoid coupling between MATLAB and C-like simulators. Consequently, there is no need for any external interface as all the models can, in principle, be developed inside the MDVP framework and the embedded software can be written in plain C++ and easily integrated into a SystemC environment.

C. SystemC-AMS modelling challenges

1) Interpolation: Contrary to Simulink, the SystemC-AMS MoCs handle incoming signals as continuous signals, even if they come from a discrete-event module. The SystemC-AMS modules will interpolate the input signal between the samples taken at each time step. This means that if one would send samples representing a block signal from a SystemC discrete event module to a TDF module, the TDF module will not interpret this as a block signal, but rather as a sawtooth signal. Of course, this depends on the time step and the length of the block signal. An example of this is illustrated in Figures 11 and 12. The red marks represent the input signal samples as provided to the system representing a block signal. The green and blue lines represent the signal as it is interpreted by the Simulink and SystemC-AMS system respectively.

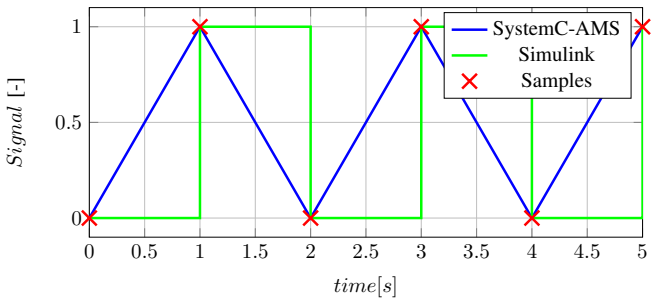

Fig. 11: The sampled block signal as interpreted by SystemC-AMS and Simulink with a low sampling rate.

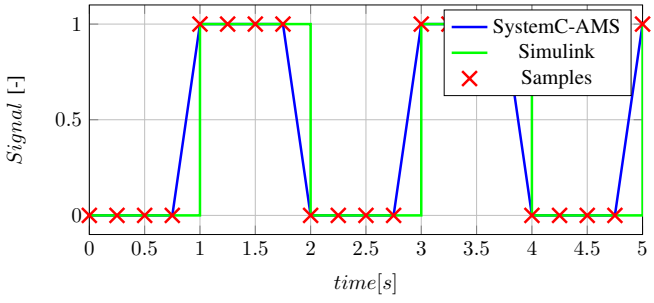

Fig. 12: The sampled block signal as interpreted by SystemC-AMS and Simulink with a higher sampling rate.

2) Cyclic dependencies: The heating system model has several cyclic dependencies in the equations. The paper path module has an input signal coming from the preheater module, which in turn has an input signal that originates at the paper path module. The same holds for the heatXchanger module and the toner transfer belt module, which both have cyclic dependencies with the paper path module. Cyclic dependencies do not cause problems with Simulink. This is because it uses the value from the previous time step for each signal. SystemC-AMS however, handles these dependencies differently.

If two modules are connected with a single signal, SystemC-AMS will first execute the first module and then use the value that module sends over the signal for the execution of the second module. This means that the execution of the second module can only start after the execution of the first module has finished. If there is a cyclic dependency, both modules will be waiting for each other. In order for the SystemC scheduler to be able to calculate an execution schedule for all connected modules, a delay has to be implemented for one of the signals in each cyclic dependency.

For the first execution, SystemC-AMS only needs an initial value for the signal on which the delay is implemented, whereas Simulink requires an initial value for each signal. After the first execution, all modules will have calculated a value for their respective outgoing signals, and therefore all incoming signals will have values assigned to them for the second execution.

3) Differential equations: Each modelling language offers different ways to implement the differential equations governing the heating system. TDF supports the use of single-input single-output state-space equations. These equations must have the form of the following equation system:

$$\frac{ds(t)}{dt} = \mathbf{A} \cdot s(t) + \mathbf{B} \cdot x(t - delay) \tag{8}$$

$$y(t) = \mathbf{C} \cdot s(t) + \mathbf{D} \cdot x(t - delay) \tag{9}$$

where $s(t)$ is the state vector, $x(t)$ is the input vector, $delay$ is the continuous time delay in seconds applied to the values available at the input, and $y(t)$ is the output vector. $\mathbf{A}$ is an n-by-n matrix, where n is the number of states, $\mathbf{B}$ is an n-by-m matrix, where m is the number of inputs, $\mathbf{C}$ is an r-by-n matrix, where r is the number of outputs and $\mathbf{D}$ is an r-by-m matrix.

As an example, the matrices for the state-space equation governing the temperature of a paper sheet are given below. These matrices are found by rewriting the differential equation for the paper temperature:

$$\dot{T}_{p,i} = \frac{1}{C_p}(P_{PH,i} + P_{HX,i} + P_{TTB,i} - k_{env}(T_{p,i} - T_{env})) \tag{10}$$

The state vector is the temperature of the paper sheet:

$$s(t) = \left[T_{p,i}\right] \tag{11}$$

The input vector has the temperature of the toner transfer belt, preheater, heatXchange unit and environment as elements.

$$x(t) = \begin{bmatrix} T_{TTB} \\ T_{PH} \\ T_{HX} \\ T_{env} \end{bmatrix} \tag{12}$$

The matrices then become:

$$\mathbf{A} = \left[-\frac{k_{env} + h_{TTB} \cdot A_{TTB}(x_i) + h_{PH} \cdot A_{PH}(x_i) + h_{HX} \cdot A_{HX}(x_i)}{C_p} \right] \tag{13}$$

$$\mathbf{B} = \begin{bmatrix} \frac{h_{TTB} \cdot A_{TTB}(x_i)}{C_p} & \frac{h_{PH} \cdot A_{PH}(x_i)}{C_p} & \frac{h_{HX} \cdot A_{HX}(x_i)}{C_p} & \frac{k_{env}}{C_p} \end{bmatrix} \tag{14}$$

$$\mathbf{C} = \begin{bmatrix} 1 \end{bmatrix} \tag{15}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

As can be seen in Equation 13 and 14, matrices $\mathbf{A}$ and $\mathbf{B}$ are not constant, but dependent on the position of the paper sheets. Since this position changes constantly, the matrices have to be updated every time step.

The state-space equation solver, however, assumes the matrices to be constant. This means that the input vector for the state-space equation is interpolated (see Section VI-C1), but the values for the matrix elements are not. This can cause a difference in accuracy, but only if the time step is large.

VII. Conclusions & future work

We presented a proof-of-concept for a modular SIL simulation environment in SystemC in which we can integrate SystemC, SystemC-AMS, and external simulators of plant elements with embedded software. The main challenge of the SystemC SIL framework was to use the SystemC scheduler in compliance with our requirements. In order to test the SIL functionality we developed a model of a heating system using SystemC-AMS.

We discussed the challenges of developing a multiple-domain virtual prototype and a SIL framework using SystemC/SystemC-AMS. Moreover, we illustrated the advantages and disadvantages of these languages.

We intend to continue working on the MDVP framework. The TDF modules related with the heating system model will be reconstructed in a generic way so that they can be reused by developers that want to develop similar simulations in the same or a different context. We intend to contribute to the MDVP library by providing basic infrastructure for heating system models. We will also investigate different SIL architectures that depend only on the SystemC capabilities. We predict that such an architecture will provide a better performance because the overhead of a blackboard architecture is avoided.

References

[1] S. Han, S. Choi, W. H. Kwon, *Real-Time Software-in-the-Loop Simulation for Control Education*. Int. Journal of Innovative Computing Information and Control Vol. 7, Number 11, 2011. pp. 6369-6382.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-Oriented Software Architecture, Vol. 1, A System of Patterns*. Wiley, 1995.

[3] *SystemC AMS extensions Users Guide*, In Open SystemC Initiative (OSCI) SystemC AMS extensions 1.0. Last accessed from www.accellera.org on March 2015.

[4] A. Banerjee, B. Sur, *SystemC and SystemC-AMS in Practice*. Springer, 2013.

[5] Catrene (CA701) H-Inception: *https://www-soc.lip6.fr/trac/hinception*. Last accessed March 2015.

[6] C. Cochior, *Model-Based Control for Professional Printing Systems*. PhD thesis, Eindhoven University of Technology, 2014.

[7] M. Zoppi, C. Cervone, G. Tiso, F. Vasca, *Software in the loop model and decoupling control for dual clutch automotive transmissions*. 3rd Int. Conf. on Systems and Control, 2013. pp. 349-454.

[8] R. C. B. Sampaio, M. Becker, A. A. G. Siqueira, L. W. Freschi, M. P. Montanher, *Optimal H∞ Controller on the Stability of MAVs in a Novel Software-in-the-Loop Control Platform*. IEEE Int. Conf. on Industrial Technology (ICIT), 2013. pp. 146-151.

[9] G. Brambilla, A. Grazioli, M. Picone, F. Zanichelli, M. Amoretti, *A cost-effective approach to software-in-the-loop simulation of pervasive systems and applications*. IEEE Int. Conf. on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014. pp. 207-210.

[10] V. Osadcuks, A. Pecka, A. Lojans, *Hardware and Software Environment for Evaluation of Control Algorithms and Strategies of Hybrid Power Systems*. Engineering for Rural Development - 10th Int. Scientific Conf., 2011. pp. 311-316.

[11] V. Osadcuks, A. Galins, *Software In The Loop Simulation Of Autonomous Hybrid Power System Of An Agricultural Facility*. Engineering for Rural Development - 11th Int. Scientific Conf., 2012. pp. 500-505.

[12] D. Pitica, *Software in the Loop Environment Reliability for Testing Embedded Code*. 18th IEEE Int. Symposium for Design and Technology in Electronic Packaging (SIITME), 2012. pp. 325-328.

[13] S. van der Hoest, *The development of a software-in-the-loop simulation framework for testing real-time control software*. SAI technical report, Eindhoven University of Technology, 2006.

[14] L. Exel, G. Frey, G. Wolf, M. Oppelt, *Re-use of existing simulation models for DCS engineering via the Functional Mock-up Interface*. 19th IEEE Int. Conf. on Emerging Technology and Factory Automation (ETFA), IEEE, 2014. pp. 1-4.

[15] P. Fishwick, *Handbook of Dynamic System Modeling*. Chapman and Hall, 2007.

[16] C. Sonntag, *Modeling, simulation and optimization envoronments*. In Handbook of Hybrid Systems Control - Theory, Tools, Applications, Cambridge University Press, 2009, pp. 328-362.

[17] P. Mosterman, *An overview of hybrid simulation phenomena and their support by simulation packages*. In Hybrid Systems: Computation and Control, LNCS 1569, Springer, 1999. pp. 165-177.

[18] F. Breitenecker, N. Popper, G. Zauner, M. Landsiedl, M. Roessler, B. Heinzl and A. Koerner, *Simulators for physical modelling - classification and comparison of features (revision 2010)* EUROSIM - 7th Congress on Modelling and Simulation, Vol. 2, 2010. pp. 1051-1061.

[19] A. Borshchev, Y. Karpov, V. Kharitonov, *Distributed Simulation of Hybrid Systems with AnyLogic and HLA*. Future Generation Computer Systems, Vol. 18, Issue 6, 2002. pp. 829-839.

[20] J. Nutaro, *ADEVS (A Discrete EVent system Simulator)*. Arizona Center for Integrative Modeling & Simulation (ACIMS), University of Arizona, Tucson. Available at http://www.ece.arizona.edu/nutaro/index.php, 1999.

[21] K. Jensen, *Coloured Petri Nets, Chapter 6: Computer tools for coloured Petri nets*. EATCS Monographs in Theoretical Computer Science, Springer, 1992. pp. 155-203.

[22] F. Bergero, E. Kofman, *PowerDEVS. A Tool for Hybrid System Modeling and Real Time Simulation*. Simulation: Transactions of the Society for Modeling and Simulation International, Vol. 87, Number 1-2, 2011, pp. 113-132,

[23] N. Matloff, *Introduction to discrete-event simulation and the SimPy language*. University of California at Davis, Dept. of Compter Science, 2008. Last accessed March 2015.

[24] *Functional Specification for SystemC 2.0*, Version 2.0-Q, in SystemC install folder SystemC-2.3.1, 2002. Last accessed from www.accellera.org on March 2015.

[25] D.C. Black, J. Donovan, B. Bunton, A. Keist, *SystemC: From the Ground Up*, Second Edition, Springer, 2010.