A Modelica IBM Implementation for Fast Simulation of Digital Controllers in Power Systems

Mehran Jafari
Dept. of Electrical Eng.
Cyprus University of Technology
Limassol, Cyprus
mm.jafari@edu.cut.ac.cy

Gautier Bureau
Réseau de Transport d' Électricité (RTE)
Paris, France
gautier.bureau@rte-france.com

Marco Chiaramello
Réseau de Transport d' Électricité (RTE)
Paris, France
marco.chiaramello@rte-france.com

Adrien Guironnet
Réseau de Transport d' Électricité (RTE)
Paris, France
adrien.guironnet@rte-france.com

Patrick Panciatici
Réseau de Transport d' Électricité (RTE)
Paris, France
patrick.panciatici@rte-france.com

Petros Aristidou
Dept. of Electrical Eng.
Cyprus University of Technology
Limassol, Cyprus
petros.aristidou@cut.ac.cy

2024 Open Source Modelling and Simulation of Energy Systems (OSMSES) | 979-8-3503-8468-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/OSMSES62085.2024.10668990

***Abstract*—The behaviour of modern power systems is more and more dictated by smart digital controllers responsible for ensuring the security, optimal operation, and stability of the systems. For the development, testing, and validation of these controllers, it is necessary to model and simulate different scenarios in power system dynamic analysis. However, digital controllers introduce discontinuities during the dynamic simulation that force the solver to reduce the time step taken to land on the discontinuity and restart the simulation. Therefore, the simulation of systems containing many digital controllers becomes very time-consuming. The Interpolation-based method (IBM) provides a fast but accurate approach for the dynamic simulation of power systems with multiple digital controllers without the need to reduce the time steps. This paper presents a fixed-step implementation of IBM in Modelica that allows to embed this method in the Modelica controller model without the need to modify the solver algorithm. The performance of the method is showcased on a single-machine infinite-bus test system.**

***Index Terms*—interpolation-based method, discrete events, time-domain simulations, digital controllers, Modelica.**

I. Introduction

Digital controllers can be found in every corner of modern power systems today, ensuring the efficient and reliable operation of the system. While power systems in time-domain simulations are modeled using *large-scale, stiff, differential-algebraic equations (DAEs)* [1], digital controllers are modeled using difference equations, defined only at the discrete sampling times, [2]. This transforms the system to be solved into a hybrid (continuous and discrete) model and introduces many challenges for its time-domain simulation [3].

Discontinuities making a system hybrid can be categorized into state events and time events [4]. State events occurrence depends on the state variables of the system. For example, a transformer may change the tap due to the voltage reaching a level. The time of tap changing cannot be predicted before starting the simulation. On the other hand, the time of the time events is known beforehand. Digital controllers introduce a time event per sampling action, having a discrete nature. Regardless of the discontinuity type, each discrete event must be handled to ensure accuracy.

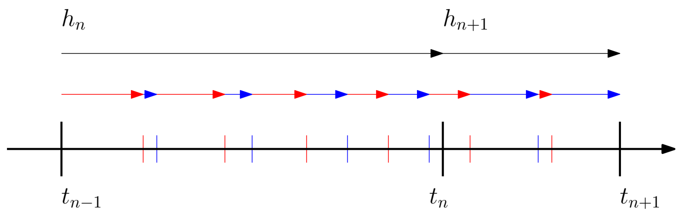

Fig. 1. Time steps size for a system with two digital controllers

The most accurate approach to treat the time events of digital controllers is to reduce the simulation step size to "land" on the event, then apply the changes defined by the controller, and restart the simulation [2]. This method, known as the Step-Reduction Method (SRM), leads to slow simulations for systems with many digital controllers since the step size is limited between the controllers' samples. Fig 1 shows the time steps taken for a system to be solved with and without the presence of digital controllers based on the SRM method. The black arrows are the time steps if there are no digital controllers, and the red and blue arrows are forced by the sampling periods of two digital controllers.

To accelerate the simulation process, a simplified method can be used that shifts the events found in the time step to the end of it [5]. In this way, the need for reducing the time step to land on the events is relieved, allowing a large time step to be taken. Although the accuracy will be compromised, and there is the possibility of the simulation cycling between states, the performance increases.

Alternatively, IBM can be used for the simulation of systems with digital controllers, which leads to an increase in performance similar to the simplified simulation method while keeping the same accuracy as SRM [6]. The reason is that IBM doesn't reduce the time steps and continues with the black arrows, including the impact of the controllers on the system by correcting their output in each Newton iteration. However, IBM implementation is more complex than SRM.

Modelica provides a powerful open-access tool for modeling and simulating the dynamics of systems described with DAEs. It is an object-oriented, equation-based programming language that permits causal modeling, accounting for ease of use and reuse of components [3]. Similar to other simulation tools available, Modelica also uses the SRM while facing discontinuities [7]. Although SRM may be fast enough for many applications if there are a few discontinuities, it becomes significantly slow when there are too many, e.g. systems with multiple digital controllers. Therefore, implementing IBM in Modelica can benefit the user by providing better performance while maintaining all the comforts of Modelica language.

In this paper, a fixed-step IBM implementation using Modelica is proposed for the dynamic simulation of systems containing digital controllers. The method is devised in a way that all the changes required are implemented in the controller block without a need to modify the solver. Therefore, the remaining of the system remains the same as before and the user only needs to replace the controller with its IBM equivalent.

The rest of the paper is organized as follows. Section II briefly introduces the basics of IBM for treating discrete events. Then, the challenges of implementing IBM in Modelica and the proposed controller modeling are discussed in Section III. Two case studies in Section IV are used to demonstrate the accuracy and performance of the proposed controller modeling in comparison to regular modeling. Finally, the conclusions are drawn in Section V.

II. Interpolation-Based Method

A power system dynamic system can be modeled as an Initial Value Problem of Differential-Algebraic Equations (IVP-DAE):

$$\begin{aligned} \mathbf{0} &= \boldsymbol{F}(\dot{\boldsymbol{y}}(t), \boldsymbol{y}(t), \boldsymbol{e}(t)) \\ \boldsymbol{y}(0) &= \boldsymbol{y}_0 \end{aligned} \tag{1}$$

where $\boldsymbol{y}(t)$ is the state variable vector (both differential and algebraic), and $\boldsymbol{e}(t)$ controllers outputs vector are subjects of the simulation.

Let's also assume the digital controller is modeled with difference equations [8]:

$$e_k = \zeta(e_{k-1}, \boldsymbol{y}(kT)) \tag{2}$$

where ζ is the function of the controller. e_{k-1} and $\boldsymbol{y}(kT)$ are the inputs to the controller denoting the controller output at the previous sampling action and the feedback from the dynamical system at k-th sampling, respectively.

To solve the system, an integration method is employed to discretize the IVP-DAE at the n-th time step (with a step $h_n = t_n - t_{n-1}$) and a Newton method is used to solve the dicretized equations [9]. By employing the SRM method (i.e., the time-steps are reduced to coincide with the discrete controller events), the controllers' outputs are considered constant between the sampling times (zero-order-hold approach). However, as mentioned above, this approach makes the time steps very small and the simulation computationally intensive.

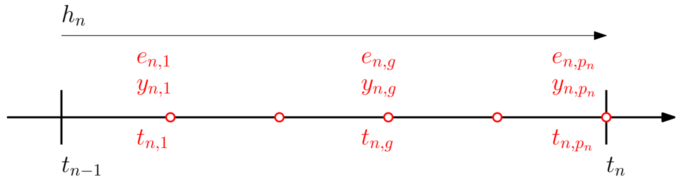

Fig. 2. The variables and sampling actions of the controllers within a time step

IBM allows simulating the system (1) without the need of reducing the time steps. This is achieved by defining the controller outputs at each sampling time as new variables and including them in the Newton formula, as detailed in [10]. The new state variables vector is defined as follows:

$$\boldsymbol{z}_n = \begin{bmatrix} \boldsymbol{z}_{n,1} \\ \boldsymbol{z}_{n,2} \end{bmatrix} \tag{3}$$

with:

$$\boldsymbol{z}_{n,1} = \boldsymbol{y}_n \tag{4}$$

$$\boldsymbol{z}_{n,2} = \begin{bmatrix} e_{n,1} & e_{n,2} & \dots & e_{n,g} & \dots & e_{n,p_n} \end{bmatrix}^T \tag{5}$$

where $\boldsymbol{z}_{n,1}$ is the vector of system state variables at the n-th time step and $\boldsymbol{z}_{n,2}$ is the vector of the controller outputs at each sampling time within the same time step.

Fig. 2 shows the controller sampling times $t_{n,g}$ and controller outputs $\boldsymbol{z}_{n,2} = e_{n,g}$ within a time step where p_n denotes the last sampling of the controller within the time step h_n. To solve for the new extended variables vector, the mismatch vector is also extended to include the controller response at each sampling time:

$$\tilde{\boldsymbol{g}} = \begin{bmatrix} \tilde{\boldsymbol{g}}_1 \\ \tilde{\boldsymbol{g}}_2 \end{bmatrix} \tag{6}$$

where the dynamic system mismatch is:

$$\tilde{\boldsymbol{g}}_1(\boldsymbol{z}_n) = \boldsymbol{g}(\boldsymbol{y}_n, e_{n,p_n}) \tag{7}$$

and for the controllers, it can be formulated as:

$$\tilde{\boldsymbol{g}}_2(\boldsymbol{z}_n) = \begin{bmatrix} e_{n,1} - \zeta(e_{n,0}, \boldsymbol{y}_{n,1}) \\ \dots \\ e_{n,g} - \zeta(e_{n,g-1}, \boldsymbol{y}_{n,g}) \\ \dots \\ e_{n,p_n} - \zeta(e_{n,p_{n-1}}, \boldsymbol{y}_{n,p_n}) \end{bmatrix} \tag{8}$$

To form $\tilde{\boldsymbol{g}}_2$, the system's state variable at the sampling time $\boldsymbol{y}_{n,g}$ is required without reducing the time-step. Thus, an interpolation formula is employed between the solutions at $n-1$ and n:

$$\boldsymbol{y}_{n,g}^{(m)} = \boldsymbol{w}_n^{(m)}(t_{n,g}), \quad \forall g \in [1, p_n] \tag{9}$$

where m denotes the Newton iteration and w denotes the employed interpolation function.

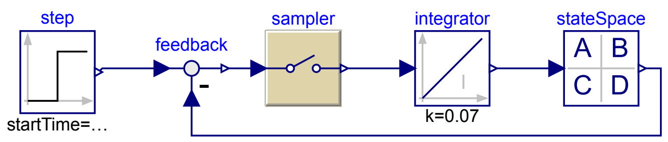

Fig. 3. A continuous system under control by a digital integral controller modeled in Modelica

Finally, the combined system is solved using a Newton method:

$$\boldsymbol{J}_n^{(m)}(\boldsymbol{z}_n^{(m+1)} - \boldsymbol{z}_n^{(m)}) = -\tilde{\boldsymbol{g}}(\boldsymbol{z}_n^{(m)}) \tag{10}$$

with the extended Jacobian matrix:

$$\boldsymbol{J}_n^{(m)} = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_1}{\partial \boldsymbol{z}_{n,2}} \\ \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,1}} & \frac{\partial \tilde{\boldsymbol{g}}_2}{\partial \boldsymbol{z}_{n,2}} \end{bmatrix} \tag{11}$$

This approach allows simulating the system with large time steps that are not constrained by the digital controller sampling times.

III. Modelica Implementation of IBM

In this section, a fixed-step implementation of this method in Modelica is proposed. Modelica simulation environments usually decouple the models with the simulation algorithms and libraries to enhance modularity. Therefore, implementing the IBM to accelerate the simulation performance requires to either modify the solver and create a link to communicate the discrete controller parameters to the solver structures and methods or to implement the IBM within the controller Modelica block in a solver-agnostic manner. The latter has been chosen for this work, allowing to switch between solvers and simulation environments seamlessly.

To describe the method, let's consider a continuous system under control by an integral controller, as illustrated in Fig. 3. The goal is to replace both the sampler and the controller (integrator in here) with a single block which implements the IBM version of the controller.

A. Interpolation

The first challenge to tackle is the interpolation formula implementation. As an example, let's assume a first-order interpolator as follows:

$$\boldsymbol{y}_{n,g}^{(m)} = \boldsymbol{w}_n^{(m)}(t_{n,g}) = h_{n,g}\dot{y}_{n-1} + y_{n-1} \tag{12}$$

where $h_{n,g} = t_{n,g} - t_{n-1}$ is the step of the g-th sample within the time step h_n from the beginning of the time step.

The feedback from the system $\dot{y}_{n-1}$ and its derivative y_{n-1} are required for the interpolation function. The IBM version of the integral controller controlling the continuous system of Fig. 3 is illustrated in Fig. 4. It should be noted that only the controller is replaced, while the remaining system and simulation environment is the same.

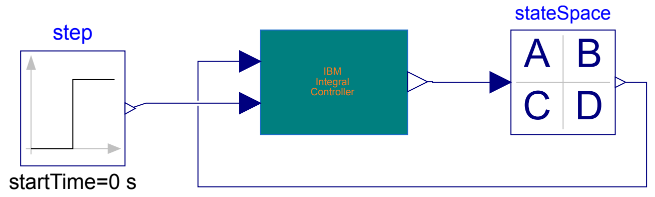

Fig. 4. A continuous system under control by the IBM version of digital integral controller modeled in Modelica

B. Time stepping

The time step size can be chosen by the user by modifying the parameter of a `Clock` function. This function halts the solver to provide the opportunity of running the IBM code inside the IBM controller at each time step. It should be noted that there is no need for the user to select a time step size that is an integer multiplication of the controller sampling rate to make sure the time step lands on the samplings. Employing a zero-order-hold approach, the output of the controllers remains constant between the samplings.

It should be noted that the time events forced on the solver are different from the time events imposed by the digital controller. For example, simulating a digital controller with sampling time $T = 10$ ms using SRM will impose time events every 10 ms. On the other hand, with the IBM controller, the only time event happen at the time step. In other words, the IBM version of the controller is handled as a continuous model, and no time events are created in between the time steps because of the controller's dicrete nature.

C. IBM-Modelica implementation

Based on the technicalities discussed previously, the IBM version of the integral controller can be modeled as the following listing:

```
1  Clock c1=Clock(1,20);
2  algorithm
3     when Clock(c1,"ImplicitTrapezoid") then
4     i:=0;
5     prett:=pre(tt);
6     ttfloor:=floor(tt/T);
7     prettfloor:=floor(prett/T);
8     ttfloordiff:=ttfloor - prettfloor;
9     while i < ttfloordiff loop
10      i:=i + 1;
11      w:=(prettfloor*T+i*T-prett)*pre(z1)+pre(z2)+((
       prettfloor*T+i*T-prett)^2/(tt-prett)^2)*(z2-pre(
       z2)-(tt-prett)*pre(z1));
12      e:=e + Ki*T*(zf - w);
13      earray[1,i]:=e;
14      if i==ttfloordiff then
15        earray[1,maxstep]:=earray[1,i];
16      end if;
17    end while;
18    equation
19  y=earray[1,maxstep];
```

Listing 1. The IBM-Modelica implementation of the digital integral controller

The first and third lines of the code impose the time events at fixed time steps to provide the opportunity to run the algorithm containing the IBM approach. The variable `tt` holds the value of time t_n at each time step. Therefore, the fifth line defines the previous time step t_{n-1}. Lines 6 to 8 have the duty of counting the number of controller samples in the time step to set the number of times the loop has to repeat. Depending on the fixed time step size, there may be a different number of controller actions in each time step.

Line 11 accounts for a second-order interpolation function of the feedback from the dynamical system at the sampling points of the controller where $z1$ and $z2$ address the derivative of the feedback and the feedback itself, respectively. As can be noted from the code, the size of $h_{n,g}$ is considered variable which depends on the number of samplings that fall into the time step. Line 12 has the integral controller equation and the next line saves the calculated controller output in an array which represents $z_{n,2}$ mentioned in the previous section.

It should be noted that the size of the array `earray` defined by the parameter `maxstep` for saving the controller outputs is larger than it needs to be since it relieves the user from modifying it every time based on the sampling rate of the controller and fixed time step. Therefore, lines 14 to 16 have the task of saving the last controller output in the last cell of the array to be fed to the output of the controller block using line 19. In this approach, the user can actually use any solver with any settings and the code itself forces the time events to the solver at the times required and runs the IBM algorithm.

IV. Case studies

In this section, two test cases are used to showcase the accuracy and performance of IBM implementation in Modelica versus the regular Modelica modeling of the digital controller[1]:

1) An integral controller controlling a state space system with two ordinary differential equations (ODE).
2) A single-machine infinite-bus (SMIB) system under control by digital excitation and digital governor systems.

All models are simulated using Modelica language in Dymola [11].

A. Integral controller

For the first test system, a continuous state-space system represented by the following set of ODEs is considered:

$$\underbrace{\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix}}_{\dot{\boldsymbol{y}}} = \underbrace{\begin{bmatrix} a & b \\ -b & 0 \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}}_{\boldsymbol{y}} + \underbrace{\begin{bmatrix} -b \\ 0 \end{bmatrix}}_{\boldsymbol{B}} e(t) \tag{13}$$

where an integral controller is used to put the system under control:

$$e_k = e_{k-1} + K_I T(u(kT) - y_2(kT)) \tag{14}$$

where it has the gain of $K_I = 0.07$ and sampling time of $T = 0.01$ s. The schematic of the test system is sketched in Fig. 3. The Modelica-IBM equivalent of the system can be modeled as shown in Fig. 4. The results of the simulation of both systems for the second output of the continuous state-space system using the RK2 solver are shown in Fig. 5.

[1]github.com/SPS-L/OSMSES2024

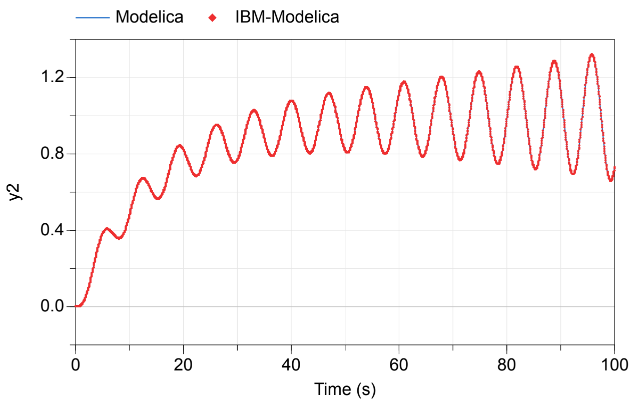

Fig. 5. Simulation results for regular Modelica modeling and IBM-Modelica modeling of y_2 using the RK2 solver

TABLE I
Test System 1: Performance results using the RK2 solver

Model	CPU Runtime (s)
Modelica	0.022
IBM-Modelica	0.007

TABLE II
Test system 1: performance results using the Dassl solver

Model	CPU Runtime (s)
Modelica	0.16
IBM-Modelica	0.005

The CPU runtime for the simulation of both controller models using the RK2 solver is listed in Table. I. The fixed time-step for IBM-Modelica simulation is considered 0.05 s. The fixed time step size for the regular Modelica controller model is limited to 0.01 s, which is equal to the controller sampling rate. It can be seen that IBM-Modelica is 3 times faster than regular Modelica modeling.

The same simulation is performed again, this time with the variable step solver Dassl with the default settings. The accuracy is the same as before as can be deducted from Fig. 6. The performance results are listed in Table. II. As can be seen, solving the regular Modelica system with the Dassl solver is significantly slower compared to RK2. The reason is that the solver is constantly trying to reduce the time steps to land on the time events of the controller while it wasn't the case for the RK2 solver since the fixed time steps were chosen to land perfectly on the controller samplings. However, the same solver has great performance results for the IBM-Modelica model.

B. SMIB

For the second test system, a fourth-order synchronous generator connected to an infinite bus is considered as illustrated in Fig. 7. The open-source library OpenIPSL developed using Modelica language for power system dynamic studies is utilized to construct the test system [12]. The generator is

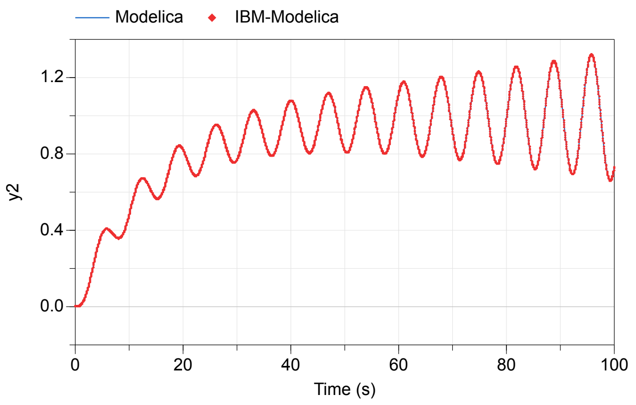

Fig. 6. Simulation results for regular Modelica modeling and IBM-Modelica modeling of y_2 using the Dassl solver

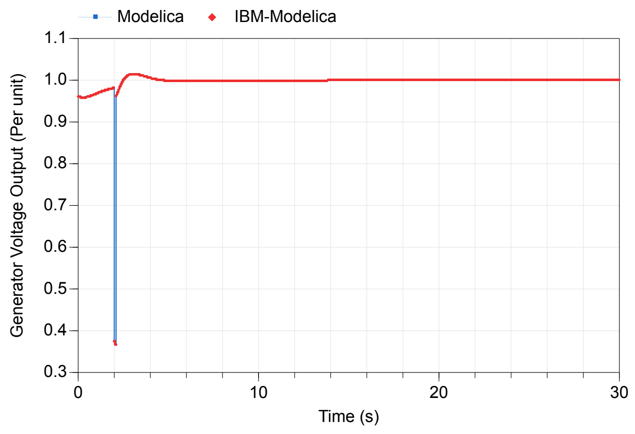

Fig. 9. Generator voltage output results for both models

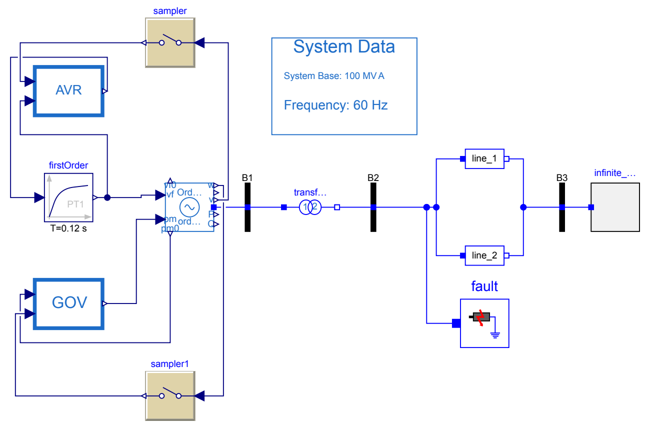

Fig. 7. The schematics of the SMIB test system modeled in Modelica

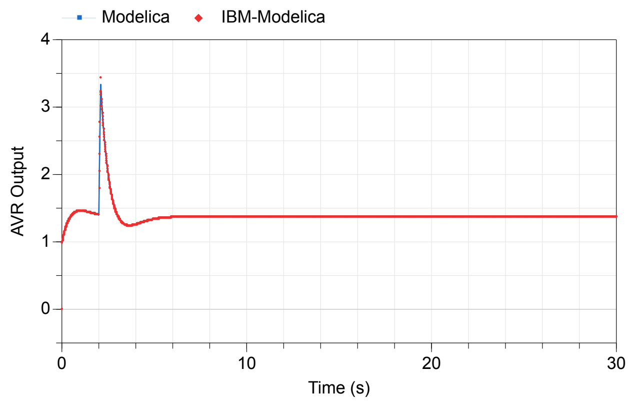

Fig. 10. AVR output results for both models

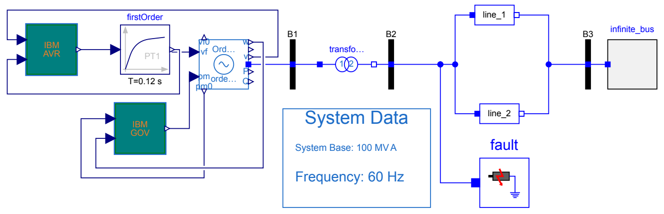

Fig. 8. The schematics of the SMIB test system with IBM controllers modeled in Modelica

under control by a digital AVR and a digital governor [2], both having a sampling rate equal to 1 ms. The same system with IBM modeling is shown in Fig. 8. It should be noted that the output of the digital AVR first goes to an exciter (continuous device), modeled by a first-order block, and then to the generator.

The simulation is performed using the Dassl solver and the generator voltage, AVR output, and governor output are depicted in Figs. 9, 10, and 11, respectively. As shown, the dynamic response is identical for both models.

The number of controller samples falling within each time step during the simulation is shown in Fig. 12. This emphasizes the fact that the user doesn't need to satisfy any condition for choosing the time step size.

The performance results of the simulation of the SMIB system are summarized in Table. III. As can be seen, the IBM-

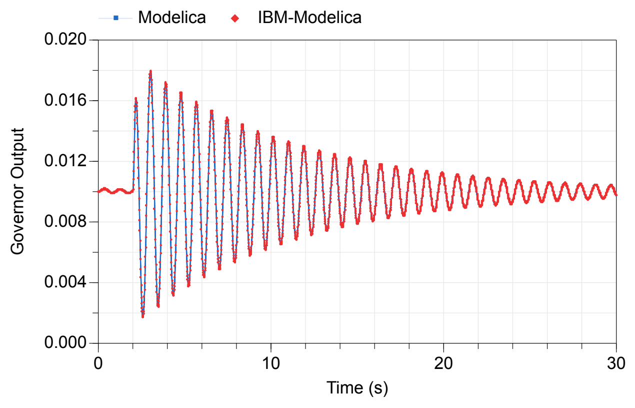

Fig. 11. Governor output results for both models

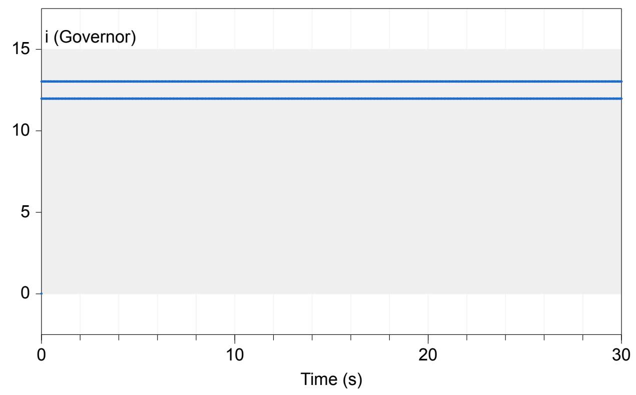

Fig. 12. Number of Sampling actions of governor in each fixed time step

TABLE III
TEST SYSTEM 2: PERFORMANCE RESULTS USING THE DASSL SOLVER

Model	**Step size (s)**	**CPU Runtime (s)**
Modelica	**0.001**	**0.39**
IBM-Modelica	**0.0125**	**0.151**

Modelica system is solved almost 3 times faster.

V. CONCLUSION

In this paper, an IBM-based Modelica implementation is proposed for the fast and accurate simulation of systems with digital controllers. Two case studies were used to showcase the accuracy and the performance of the method compared to regular Modelica modeling. It was shown that the IBM-Modelica method has similar accuracy to the regular Modelica molding while the performance increases.

For future work, we will focus on the simulation of large-scale systems with more controllers. Also, we will try to make the method a variable-step method so the flexibility and performance of the method can be further improved.

REFERENCES

[1] F. Milano, *Power System Modelling and Scripting*, ser. Power Systems. Springer Berlin Heidelberg, 2010.

[2] M. Jafari, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, and P. Aristidou, "Modeling of digital controllers in electric power system dynamic simulations," in *2023 IEEE Belgrade PowerTech*, 2023, pp. 01–06.

[3] P. Fritzson, *Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach*. John Wiley & Sons, 2014.

[4] D. Ellison, "Efficient automatic integration of ordinary differential equations with discontinuities," *Mathematics and Computers in Simulation*, vol. 23, no. 1, pp. 12–20, 1981.

[5] D. Fabozzi, A. S. Chieh, P. Panciatici, and T. Van Cutsem, "On simplified handling of state events in time-domain simulation," in *17th Power System Computation Conference*, 2011.

[6] M. Jafari, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, and P. Aristidou, "An Interpolation-based Method for Numerical Simulation of Digital Controllers in Power System Dynamic Studies," *TechRxiv. Preprint*, 2023. [Online]. Available: 10.36227/techrxiv.23579574

[7] S. E. Mattsson, H. Elmqvist, and M. Otter, "Physical system modeling with Modelica," *Control engineering practice*, vol. 6, no. 4, pp. 501–510, 1998.

[8] M. Otter, H. Elmqvist, and S. Mattsson, "Hybrid modeling in modelica based on the synchronous data flow principle," in *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No.99TH8404)*, 1999, pp. 151–157.

[9] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*. Siam, 1998, vol. 61.

[10] M. Jafari, G. Bureau, M. Chiaramello, A. Guironnet, P. Panciatici, and P. Aristidou, "Decoupled interpolation-based method for numerical simulation of digital controllers," in *2024 IEEE International Systems Conference (SysCon)*, 2024, pp. 1–6.

[11] *Dymola version: 2023x*. Dassault Systemes, 2023. [Online]. Available: https://www.3ds.com

[12] M. De Castro, D. Winkler, G. Laera, L. Vanfretti, S. A. Dorado-Rojas, T. Rabuzin, B. Mukherjee, and M. Navarro, "Version [OpenIPSL 2.0.0] - [iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations]," *SoftwareX*, vol. 21, p. 101277, 2023.