A Methodology for a Complete Simulation of Cyber-Physical Energy Systems

Youssef Driouich
Dip.to di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata (DIEM)
Università degli Studi di Salerno, Italy
ydriouich@unisa.it

Mimmo Parente
Dip.to Scienze Statististiche & Innovation of Systems (DISA-MIS)
Università degli Studi di Salerno, Italy
parente@unisa.it

Enrico Tronci
Dip.to di Informatica
Università degli Studi di Roma,
"La Sapienza", Italy
tronci@di.uniroma1.it

Abstract— **The number of computation cycles used for simulation-based Verification of Cyber Physical Energy Systems is outpacing the available throughput of simulation resources. In this paper, a methodology for the verification of the CPES at hand with the aim of full coverage of the system's states is proposed. This approach relies on representing the unpredictable behaviour of the environment in order to cover all feasible possible scenarios. Processed by JModelica, the simulation results are covering the system's complete dynamic behaviour. Simulation by complete state space covering guarantees the verification results to be sound for every possible state of the system under verification. The application to Photovoltaic circuits, specifically the Distributed Maximum Power Point Tracking, shows the feasibility of the approach.**

Keywords-component: Cyber-Physical Energy Systems; Simulation; Simulation-based Verification; JModelica; Distributed Maximum Power Point Tracking; Photovoltaic circuit; System Under Verification

I. INTRODUCTION

The complexity of the simulation models of the dynamic systems is scaling exponentially, and hence the amount of computation resource required to explore all of the states of these type of systems is scaling exponentially. As consequence, even the simplest designs of today are impossible to completely simulate. Given that simulation resources are more or less limited, the verification of each system is becoming less and less exhaustive.

In this work, we consider the system modeled as state machines. Intuitively, this modeling scheme is based on the assumption that each run of the system can be described by a (possibly infinite) sequence of discrete state changes. The model then consists of a finite amount of information defining the initial state of the system, as well as all the possible state changes. A simple way of checking the correctness of such a model is to explore its state space. Harshly speaking, the idea is to check precisely all the possible situations that can arise during the possible executions of the model. For instance the works introduced in [3, 4, 5] to formalize system requirements and like those in [6, 7, 8] to define admissible operating scenarios.

To this end, we present an approach for performing the state-space exploration of systems with an infinite state space with a relevant case study of Cyber Physical Energy System (CPES) for the Distributed Maximum Power Point Tracking (DMPPT) system built out of the Perturb and Observe (P&O) based Maximum Power Point Tracking (MPPT) circuit, the model of the system is presented in [9].

The paper is organized as follows: Section II presents our system's model. Section III describes the adopted approach to explore the state-space of the system at hand. The section IV gives the experimentation results. The last section offers a summary of the realised work and the future enhancement that can be accomplished.

II. CASE STUDY :

The models we study are primarily about dynamics, the evolution of the DMPPT [1, 2] system state in time. Our purpose is to verify if our system is able to minimize the loss of produced power when the irradiance of the panels is changing frequently in one hand, in the other hand, to check if the system converges to a desired behaviour under the actions of the controller. The system is illustrated in Fig.1.

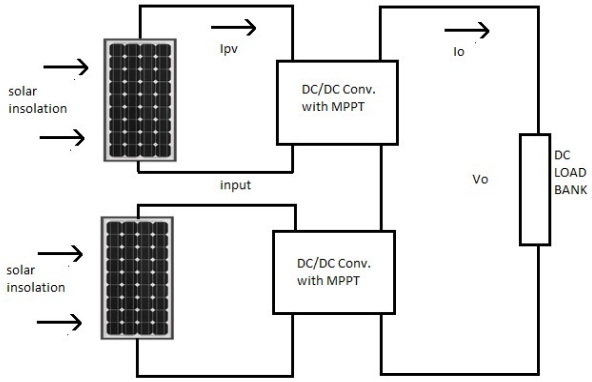

FIGURE 1: THE MODEL OF THE CPES

978-1-5386-6405-6/18/$31.00 ©2018 IEEE

A. *Physical system:*

Physical modeling deals with the expression of the physical system consisting of plant and actuators in mathematical or logical terms. Continuous dynamics modeling is required since we want to know how the PS regulates the production of energy in function of the irradiance variability. In the modeling of the DMPPT system, the model of the overall circuit is composed by a set of physical components: the solar cells, the photovoltaic panel, a dc/dc converter and a dc/ac inverter (see Fig. 2). We use an equation-based modeling approach to express the physical dynamics of the circuit. As usual, the output current (*I*) of each solar cell is computed by solving the Kirchhoff's law:

$$I = Iph - Ir \cdot q \cdot exp\left((V + I \cdot Rs)/\eta \cdot k \cdot T\right) - 1 - V + I \cdot Rs\ Rp$$

where *V* is the output voltage of the cell, *Iph* is the photocurrent, *Ir* is the saturation current, *Rs* is the resistance in series, *Rp* is the resistance in parallel, *q* is the electrical charge ($1.6 \times 10{-}19$ C), η is the p-n junction quality factor and *k* is the Boltzmann constant. The figure below exhibit the physical system

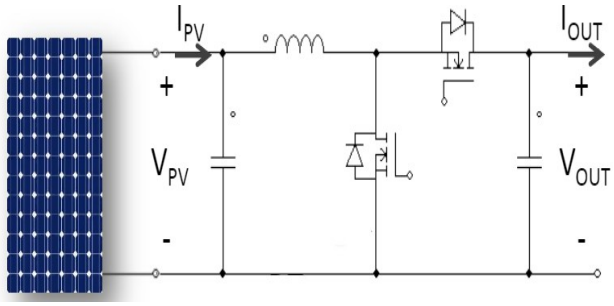

FIGURE 2: THE PHYSICAL MODEL OF THE CPES.

B. *Control algorithm:*

In our approach to model the CPES, two controls are adopted: the first control is the Maximum Power Point Tracking (MPPT) based on the P&O method. This operates by periodically incrementing or decrementing the output terminal voltage of the photovoltaic cell and comparing the power obtained in the current cycle with the power in the previous cycle (in [1] the MPPT control is discussed in details). However, the main control of our design consists in adapting the switching mode (duty cycle) of the dc/dc converter according to the current of the connected panel (I_pan) and the output current of the converter (I_out). We have modelled each plant in such a way that the software modules of the converter communicates the I_out value to the other modules. In this way, each software module of the converter can detect the state in which the others are into and adapt its state accordingly. This is crucial for our verification purposes. More precisely, this control depends on the computation of I_out and of I_pan and depending on the entered values, the switching mode is determined. The output voltage of the converter (V_out) is computed which modifies the switching frequency of the converter by calculating a new value of the MPPT duty cycle.

C. *Cyber system:*

As said in the previous section, the objective of the controller is to provide two main parameters to harvest the maximum energy under different environment conditions, the switching mode and the computation of the output voltage. According to [2] there are four possible states of the DMPPT system.

First the MPPT state, which is the desired working way of the dc/dc converter. In this case, the output voltage is in the range between (V_max) and (V_min). This protection measure is realized by modeling a blocking diode. The converter in this mode switches according to the computed value of the duty cycle. The second state is the CUT-OFF: this typically occurs when the converters are connected to un-shaded panels and generates much more power than shaded ones; the converter still switches and the output voltage is computed as follows: *V_out = V_max*. The third state is the PASS-THROUGH: to boost converters that are connected to shaded panels that generates much less power than un-shaded ones; the converter in this mode stops switching and the panel is directly connected in series to the output. The output voltage is computed as follows: *V_out = Vpv – Rs * I_out*. The fourth state is the BY-PASS: when the panel is heavily shaded and it cannot be connected to the string because it would sink rather than source power. In this mode, the converter bypasses the panel and here too stops switching. The output voltage equation describing this mode is *V _out = - Rs* I_out*. In particular, once the control algorithm detects the mode, according to the values of the currents, the formula for computing V_out is communicated to the plant, which modifies the value of the duty cycle, this way determining the switching mode. If the control does not detect a change in the values of the currents, then the output voltage remains unmodified.

III. SIMULATION METHODOLOGY:

The simulation methodology presented in this paper is aimed at improving the efficiency of the system level simulation using environment scenarios generation. Efficiency may be observed as the level of coverage divided by the simulation work required to achieve the exploration of all system's state space.

A. *Coverage methodology:*

The coverage requires detailed modeling of the dynamics of the environment and a clear understanding of the interaction between the dynamics of the system at hand and its environment (Sun's irradiance). We have represented the environment model as an infinite transition graph that allows us to perform a Breadth-first search (BFS) algorithm to traverse all the nodes of the environment graph in a well-defined order exploring all likely irradiance scenarios used in the parametrization of the simulations.

B. *Estimation of the simulation coverage:*

The transition graph of the state machine defining the environment model can be used to generate all the admissible irradiance-turnover sequences. Let *n* be the number of panels,

k the number of *changes* of the irradiance values during the simulation process, Z the set of the possible values assigned to the irradiance, and *p* the cardinality of Z. The number of traces is $(p^n)^k$. We implemented such an exhaustive simulation scenario generator within the environment model described in the previous section.

C. Computing the steady states:

After the step response duration, the circuit reaches an equilibrium (stationary state) where the effects of transients periodic are no longer important. The dynamics of the system's state variables, described by I-V relationships and Kirchhoff's law, are a periodically varying functions assuming values around a so called operating point. The steady state solution of the system is obtained by integrating from an appropriate initial value till the transients get stabilized, generally after the *warm-up* duration of the simulation. Our technique to reach the steady state consists on executing the simulations of a given environment scenario inside a feedback loop initializing the state variables of the current simulation X_i with the values where the previous simulation X_{i-1} was stopped. At the end of every execution, we quantize the obtained results via a moving average filter then we compare the resulting values with the already done simulations, if a match reveals that at least two simulation's runs lead to identical states, the loop is interrupted saving the state of the last simulation as the steady state.

D. Exploring the system's state space:

To enable an effective coverage of all the system's states, we split the simulation process into two main phases. First, an offline phase, where the environment model is generated and it is given in input to the simulation function. Second, since the size of a simulation state can easily take many GB of main memory space, it is not possible to store too many states, even resorting to the secondary storage device. Thus, a clever strategy is needed to decide what are the states that have to be kept.

The adopted technique consists to compare the set of evidenced states during the steady state phase, if a match indicates that two corresponding scenarios have identical steady states, the system's state is saved once and the environment parameters defining the state are saved into a data structure in which irradiance values and states machine are associated defining new sequences of states (state paths).

Using this association, the number of the states defining the system decreases significantly thus reducing the system's state space complexity. Our approach withstands scenario explosion by dodging as much as possible revisiting already explored system's states.

The algorithm bellow describes the overall simulation process:

Algorithm 1 : Simulation process

```
Env_model = Generate environment model with the BFS
foreach scenario in Env_model do
        res = simulate(scenario)
        state = steady_state(res)
        if (state.match(state_machines)) then
                Store (state)
                Associate(scenario , state_machine)
        else
                Store_new(state)
end
```

For example, let us suppose that our system described in the figure below is defined by the set of states s_i and the transition from a state to another is done by the changement of the irradiance of the photovoltaic panels value represented as a percentage of the nominal irradiance.

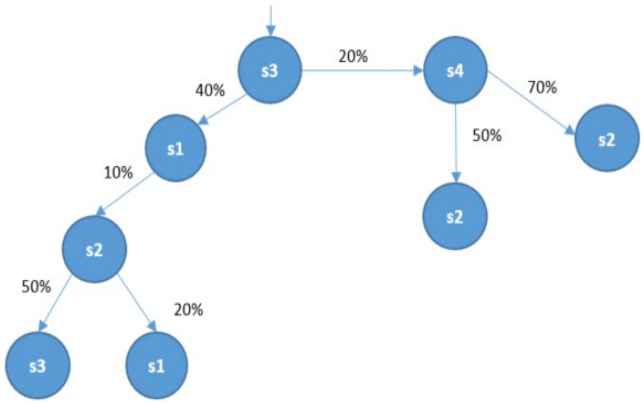

FIGURE 3: EXAMPLE OF SYSTEM'S STATES

After the execution of our simulation process, a new correspondence between states has been established by removing identical states and associating the irradiance values to the remaining states. It stops when no further reduction can be applied. The following figure shows the resulted state machine:

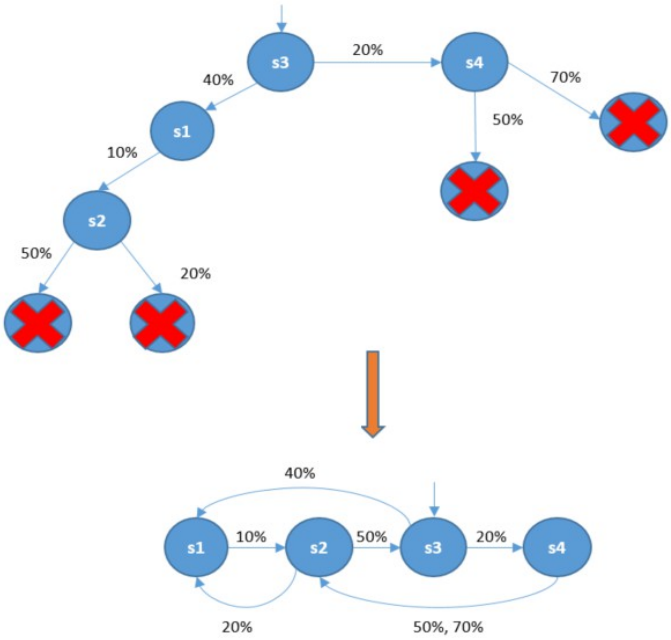

FIGURE 4 : THE SYSTEM'S STATES AFTER THE TRANSFORMATION

IV. EXPERIMENTS

The simulations were conducted on JModelica.org [10], which is an extensible Modelica-based open source platform for optimization, simulation and analysis of complex dynamic systems. The JModelica.org offers a Python interface that enables users to use Python scripting to interact with Modelica models. The parameters used to configurate our simulations are defined in the table below. We created the executable model of our CPES in Modelica instantiating the photovoltaic panel equations, the dynamic irradiance of each cell of the panels, the feedback control of the MPPT and the discrete control of the converter modes.

Parameters	Values
Nominal Irradiation of the panels	1000 W/m2
Rs (series resistance)	0.11 Ω
Rp (parallel resistance)	148 Ω
T_amb	20 °C
I_ph_STC	7.7 A
T_STC	25 °C
T_NOCT	46 °C
Fs (Switching frequency if the converter)	5^4
Ta(Sampling time of the MPPT control)	0.01 s
Td(Sampling time of the mode control)	0.05 s
V_max	(0.8 * cell's number) V
V_min	(0.05 * cell's number) V

TABLE 1 : PARAMETERS OF THE SIMULATIONS CONFIGURATION

The state variables of our CPES are: *I_pan* the current generated by the photovoltaic panel, *I_l* the current traversing the inductance of the converter, *I_out_conv* the output current of the converter and *I_out* the global current produced by the circuit. Once the model has been compiled, we obtain the simulation executable code. The input to this code is the irradiance model that is generated in Python language. The irradiance values are defined as percentage of the nominal irradiance equal to 1000 W/m2; it varies with a step of 10% in an interval going from 0% to 100%. The first step of the execution is to calculate the steady state of the system under a given value of the irradiance by iterating the simulations and comparing the values of the current inside and outside the circuit, the points describing this process are:

- Simulate for the first iteration with (*I_pan, I_l, I_out_conv, I_out) = (0, 0, 0, 0).* The results are saved in a text file in a directory called */SteadyState*.
- The next iterations start the simulation by initializing the values of the circuit's currents with the values of the previous simulations by taking the values from the text file. (*I_pan, I_l, I_out_conv, I_out) = (prev_I_pan, prev_I_l, prev_I_out_conv, prev_I_out).*
- At the end of every simulation, we compare the resulted values with the values of the previous simulations:

Algorithm 2 : Steady state computation

```
for i in range(0,k) do
    if (I_pan = prev_I_pan [i] and I_l = prev_I_l [i]
      and I_out_conv = prev_ I_out_conv [i]
      and I_out = prev_ I_out [i] )then
                print "steady state reached ";
                save (/SteadyState /simulation_results.txt);
                break;
          else:
                next_simulation();
end
```

Where k is the number of all the simulations already done.
When the steady state is found, the loop is exited and the actual irradiance values and the results of the simulation containing all the components of the system (e.g. value of the duty cycle, switching frequency of the converter) are saved into a subdirectory */S* of the system's states directory called */SysState.*
The second step of the process is to explore the possible system's state, which means identify and store all the possible resulting values of the circuit's components under whatever value of the irradiance.
The */SysState* directory contains also a text file (*index.txt*) in which the state system's state table is described. This table is created when the first steady state is reached and stored. It contains an association between the results of every system's state and the irradiance values.
The steps of this process are:

Algorithm 3 : System's state exploration

```
for j in range(0,l) do
    if (I_pan = ss_I_pan [j] and I_l = ss_I_l [j]
      and I_out_conv = ss_ I_out_conv [j]
      and I_out = ss_ I_out [j]) then
              print "There is a match between states";
              save (/SysState/S/simulation_results.txt);
              update(/SysState/index.txt, S, S[j]);
      else:
             print "There is NO match between states";
            save (/SysState/S/simulation_results.txt);
            update(/SysState/index.txt, S)
end
```

Where *l* is the number of all the states found, *ss_I_pan* is the current panel, *ss_ I_l* is the inductance current, *ss_ I_out_conv* is the current outside the converter and *ss_ I_out* is the current outside the circuit on the system's states. The iteration will terminate when all the likely irradiance scenarios match with at least one state.

A part of the resulting CPES states table is presented below, the first line represents the irradiance values and the first column values are the initial states of each simulation. Each (i,j) entry is the value of the transition from state Si with irradiance of the system Sj. For example, the system starting in state S2 with an irradiance of 60% transits to the state S109.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
S0	S97	S8	S37	S69	S76	S92	S2	S39	S9	S91
S1	S49	S29	S82	S13	S117	S45	S121	S100	S14	S126
S2	S115	S62	S41	S71	S18	S109	S72	S111	S3	S93
S3	S77	S56	S89	S21	S37	S55	S61	S128	S77	S58
S4	S107	S72	S4	S0	S83	S130	S53	S25	S81	S104
S5	S46	S35	S18	S74	S113	S44	S85	S83	S65	S67
S6	S22	S75	S123	S127	S128	S44	S3	S21	S21	S41
S7	S124	S130	S73	S120	S36	S30	S16	S77	S55	S14
S8	S85	S42	S3	S80	S91	S46	S97	S80	S75	S92
S9	S110	S5	S90	S114	S86	S66	S71	S33	S95	S74
S10	S33	S125	S89	S102	S126	S15	S96	S13	S126	S8
S11	S88	S104	S63	S106	S83	S18	S5	S55	S41	S98
S12	S1	S43	S93	S13	S61	S35	S92	S62	S103	S92
S13	S51	S51	S58	S8	S98	S42	S119	S108	S55	S118
S14	S64	S39	S54	S69	S65	S37	S97	S122	S107	S13
S15	S107	S58	S89	S21	S97	S20	S112	S48	S66	S44
S16	S118	S2	S96	S123	S42	S101	S130	S28	S121	S57
S17	S105	S9	S33	S117	S50	S69	S119	S47	S117	S27
S18	S129	S63	S103	S27	S12	S85	S110	S18	S52	S14
S19	S118	S112	S32	S4	S31	S6	S97	S44	S42	S32
S20	S102	S105	S25	S9	S66	S68	S132	S55	S113	S82
S21	S82	S75	S52	S129	S8	S8	S130	S102	S16	S24
S22	S119	S24	S82	S93	S70	S88	S40	S42	S116	S4
S23	S81	S18	S63	S1	S66	S87	S80	S102	S33	S61
S24	S117	S20	S51	S112	S15	S64	S24	S42	S56	S69
S25	S89	S102	S73	S72	S10	S18	S86	S62	S1	S107
S26	S1	S88	S78	S81	S33	S71	S131	S12	S88	S111
S27	S125	S85	S87	S33	S98	S60	S80	S111	S83	S97
S28	S15	S102	S114	S130	S80	S76	S122	S30	S85	S115
S29	S104	S50	S126	S40	S121	S84	S88	S31	S131	S70
S30	S132	S96	S38	S52	S75	S58	S90	S54	S6	S54
S31	S42	S70	S132	S132	S68	S48	S51	S47	S22	S103
S32	S83	S24	S33	S6	S6	S24	S65	S73	S15	S54
S33	S47	S76	S112	S97	S67	S27	S6	S56	S92	S121
S34	S101	S6	S120	S29	S106	S117	S75	S98	S81	S10
S35	S130	S82	S99	S107	S58	S53	S76	S114	S33	S22
S36	S64	S63	S120	S99	S50	S93	S91	S81	S122	S23
S37	S44	S58	S90	S114	S87	S105	S36	S43	S43	S63
S38	S78	S6	S79	S82	S27	S73	S28	S13	S97	S24
S39	S60	S34	S120	S36	S16	S7	S60	S95	S33	S115
S40	S14	S120	S68	S37	S10	S131	S31	S91	S46	S16
S41	S58	S99	S37	S0	S63	S124	S108	S110	S86	S131

FIGURE 5 : CPES STATES TABLE

Generally, the simulation runtime is highly dependent on the computer's performance and specifications, where the simulation tool is run, and solver options. In this paper, a computer with a 16-core Intel Xeon@2.7GHz machine with 32-Gigabyte available memory with a fixed-time step is utilized as a computational medium for simulations.

The number of states explored using our approach is 132 states. The process took 65 hours to simulate for one second of the simulation time. The size of the results obtained is about 70 GB. The resources committed for simulating our approach are less important than simulating all the possible scenarios of the environment. Moreover, the platform JModelica.org permits us to use the Functional Mock-up Interface (FMI) which is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of xml-files and compiled C-code [11]. This approach allows us also the definition of methods and techniques to support the reuse and interoperability of simulation models and their execution on distributed computing environment.

V. CONCLUSION AND FUTURE WORKS

The central theme of this paper is a new approach for performing the state-space exploration of Cyber physical energy systems with an infinite state space modeled by the sun's irradiance. The future works concern the use of Temporal Logics to express the properties specification. Furthermore, we plan to develop a simulation-based algorithm that verifies whether all the system trajectories satisfy the desired temporal property.

REFERENCES

[1] M. De Cristofaro; N. Femia; M. Migliaro; G. Petrone "Minimum Computing Adaptive MPPT Control" ISIE 2014, Pages: 1384 - 1389, DOI: 10.1109/ISIE.2014.6864816.

[2] M. De Cristofaro; G. Di Capua; N. Femia; G. Petrone; G. Spagnuolo; D. Toledo "Models and Methods for Energy Productivity Analysis of PV Systems" INDIN'15, Pages: 1153 – 1158, DOI: 10.1109/INDIN.2015.7281898

[3] A. Murano, M. Napoli, M. Parente. "Program Complexity in Hierarchical Module Checking", LPAR'08, ISBN: 978-3-540-89438-4.

[4] A. Ferrante, M. Napoli, M. Parente, "Model-Checking for Graded CTL", Fundamenta Informaticae, 96(3), 323-339, 2009.

[5] A. Ferrante, M. Napoli, M. Parente, "Graded-CTL: Satisfiability and Symbolic Model Checking", Int.l Conf. on Formal Engineering Methods, (ICFEM) 2009: 306-325, Lecture Notes in Computer Science 5885, Springer 2009, ISBN 978-3-642-10372-8.

[6] T. Mancini; F. Mari; A. Massini; I. Melatti; E. Tronci "SyLVaaS: System Level Formal Verification as a Service", PDP 2015 Pages: 476 - 483, DOI: 10.1109/PDP.2015.119

[7] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, and E. Tronci "System level Formal Verification via Model Checking Driven Simulation", CAV 2013, DOI: 10.1007/978-3-642-39799-8_21

[8] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, "System Level Formal Verification via Distributed Multi-Core Hardware in the Loop Simulation", PDP 2014, DOI: 10.1109/PDP.2014.32

[9] Y. Driouich; M. Parente; E. Tronci, "Modeling cyber-physical systems for automatic verification", SMACD 2017, DOI: 10.1109/SMACD.2017.7981621

[10] http://www.jmodelica.org/

[11] http://fmi-standard.org/