Proceedings of the Federated Conference on Computer Science and Information Systems pp. 757–766
DOI: 10.15439/2016F24
ACSIS, Vol. 8. ISSN 2300-5963

Method for Approaching the Cyber-Physical Systems

Tiberiu S. Letia
Dept. of Automation
Technical University of Cluj-Napoca
Romania
Email: Tiberiu.Letia@aut.utcluj.ro

Attila O. Kilyen
Dept. of Automation
Technical University of Cluj-Napoca
Romania
Email: ors.kilyen@accenture.com

***Abstract*—The main topics of the Cyber-Physical Systems (CPSs) cover the specification, modeling, control, design, verification and testing. The CPSs implementation consists of reactive programs conceived using models that are capable to sustain the mentioned activities. Component diagrams (introduced by Unified Modeling Language) are used here for the architecture design, with the goal to split the CPS complexity into smaller entities that are easier to tackle. All the components are modeled by Fuzzy Logic Enhanced Time Petri Nets (FLETPNs) that can simultaneously describe the discrete event and the time discrete features. This unique and compact approach facilitates the control synthesis, the software design, the verification and the testing.**

An example of application to the control of a system composed of a wind turbine generator, a photo-voltaic generator and loads is used to show the model utilization and its benefits.

I. Approaches of the Cyber-Physical Systems

CPSs integrate the dynamics of the physical processes with those of the software and communication. There are some surveys that present the main characteristics, the main domains where they are applied and the main topics of the CPSs [10], [11], [12]. The main topics of the CPSs cover the specification, the modeling, the control, the design, the verification and the testing. The CPSs implementation consists of reactive programs that are based on models that are capable to support the mentioned activities.

The main goal of the current research is to conceive a control system that concurrently reacts to discrete events and continuous modifications of plant state. The target is a set of interacting dynamic models capable to approach the following specification:

- the reaction to synchronous and asynchronous (plant) events that are signaled by continuous variables (instead of single level events)
- the continuous time reaction to modification of some (plant output) variables
- the reaction control signals that belong to continuous domains (the discrete domains, as the binary set, should be particular cases)

Some reactions require the execution of activities involving non-ignorable durations and could have real-time constraints that have to be fulfilled. This requirement leads to the conclusion that the target model has to be capable to describe concurrent behavior.

A relevant issue is to conceive a model that is capable to describe the controller behavior and its structure. A practical goal is to make possible the verification that the implemented model fulfills the specified requirements.

The implementation of controllers on digital computers supposes that the information of continuous variables can be represented with a limited and tolerated accuracy (due to the limited length of the number representation) and the calculus can add other losses of the precision. On the other hand, the continuous time reactions are not possible to be implemented on digital computers. For this reason, instead of continuous time models, the discrete time models are used. The loss of accuracy due to the conversion of the continuous time models into discrete time models is supposed to be tolerated too.

The OMG (Object Management Group) Unified Modeling Language could successfully fulfill the requirements using a set of state machines, but these dynamic models have to be endowed with many variables, equations and condition expressions to completely describe the desired behavior [1]. The verification that the obtained models fulfill the requirements needs the use of other complex methods (such as different kinds of Petri nets) or simulation tools.

Many authors emphasize that CPSs are hybrid systems [2], [10], [11]. A hybrid system is composed of a discrete event side and a continuous time side in an interaction that provides a complex behavior. The control of a hybrid system is a challenge due to the requirements of asynchronous reactions to the discrete events as well as to the continuous adjustment of some controlled outputs. The CPSs involve interdependencies between physical behavior and digital control [13]. The control system implementation should be based on asynchronous interrupts and synchronous discrete time reactions.

The controller asynchronous reaction involves the execution of rules of the form:

$$ON\ event\ IF\ condition\ THEN \\ action_1 \wedge action_2 \wedge \cdots \wedge action_k \quad (1)$$

The ordinary Petri nets can model the handling of events, the binary conditions, the concurrency and the controller structure. These models are not capable to model the cases when the involved reactions require input of continuous variables and outputs that signal continuous variables. These models are not appropriate to model continuous type operations.

The fuzzy logic controller (FLC) based on fuzzy logic provides a means of converting a linguistic control strategy based on expert knowledge into automatic control strategies. This approach was chosen (in the current research) for its capability to conceive controllers that tackle, beside the synchronous reactions (i. e. the periodic discrete time feature), the asynchronous reactions for the cases that require variable output control signals.

An overview of the possibility of implementing the fuzzy control systems as fuzzy rule-base systems is contained in [6]. Here it is justified that the conventional methods are good for simpler problems, while the fuzzy systems are suitable for complex problems or control applications that involve human descriptions or intuitive thinking. Lee presents a survey of the general methodology for constructing an FLC and the assessing of its performance [7].

In [9] another model that links the Petri nets with FLC is introduced.

II. Fuzzy Logic Enhanced Time Petri Net Models

A. Low Level Petri Nets

As it is well known, a Petri Net (PN) is a directed graph with two kinds of nodes. An *ordinary PN* is a 5-tuple

$$PN = (P, T, pre, post, \mathbf{M}) \tag{2}$$

with:

- a finite place set $P = \{p_1, p_2, ..., p_m\}, (m \geq 0)$
- a finite transition set $T = \{t_1, t_2, ..., t_n\}, (n \geq 0)$
- $pre : P \times T \rightarrow \mathbf{N}$ (natural number set) is the backward incidence function:
- $post : P \times T \rightarrow \mathbf{N}$ is the forward incidence function

In the current approach

- $pre(p, t) = 0$, if there is not an arc from p to t and $pre(p, t) = 1$, if there is an arc from p to t,
- $post(p, t) = 0$, if there is not an arc from t to p and $post(p, t) = 1$, if there is an arc from t to p.

$N = (P, T, pre, post)$ describes the structure without marking. $PN = (N, \mathbf{M}_0)$ is the structure with a marking $\mathbf{M}$ where $M : P \rightarrow \mathbf{N}$ is the marking specifying the number of tokens of each place. The marking $\mathbf{M} = [M(p_1), M(p_2), ..., M(p_m)]^T$ describes the PN state.

The lack of the PN capability to handle the time is removed in the models *Time Petri Nets (TPNs)*. The TPNs are suited for modeling the time-dependent systems with timing constraints [3] A timed Petri net can be defined with delayed transitions, or delayed tokens [4],[5]. The current approach uses the timed transitions. A TPN is a PN with each transition t_i delayed by an assigned delay d_i from a set of non-negative integers $D = \{d_1, d_2, ..., d_n\}$. That means, each transition t_i is delayed with d_i time units from the moment of time when it becomes enabled.

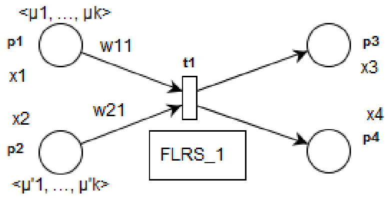

Fig. 1. Example of a FLETPN

The definition of TPN is:

$$TPN = (P, T, pre, post, D, \mathbf{M}) \tag{3}$$

where $P, T, pre, post$ and $\mathbf{M}$ have the previous meanings. $D : T \rightarrow \mathbf{N}$ is a mapping that assigns to each transition a delay.

An Enhanced Time Petri Net (ETPN) is a TPN endowed with an input place set *Inp* and an output place set *Out* [5]. In ETPN only the transitions with single input places can be delayed. The input places (Inp) are loaded with tokens by the plant. The ETPN injects tokens in the output places (Out) and these tokens are extracted immediately by the plant.

All these kinds of Petri nets have a single type of tokens.

B. High Level Petri Nets

Unlike the above defined Petri nets, the high level Petri nets have distinct tokens. The current approach is based on a particular case of high level Petri nets. There are some kinds of Petri Nets endowed with fuzzy features. A relevant review of fuzzy Petri nets and industrial applications can be found in [8].

A FLETPN is an ETPN extended with fuzzy logic rules that is capable of processing fuzzy information. Each place has a distinct token and its capacity is equal to one. Each place of the ETPN is assigned a variable and each transition has assigned a fuzzy logic rule set, but one fuzzy logic rule set could be assigned to more than one transition. A token injected into a place expresses the membership degrees of the (assigned) variable to the fuzzy sets. Figure 1 shows a FLETPN that has a transition with two input places and two output places. Each place p_i has assigned a variable x_i.

The definition of a FLETPN is:

$$FLETPN = (P, T, pre, post, D, W, X, EFS, \mathbf{FLRS}, \alpha, \beta, \mathbf{M}) \tag{4}$$

where $P, T, pre, post$ and D have the previous meanings. $X = \{x_1, x_2, \cdots, x_m\}$ is a set of variables with $x_i \in \mathbf{R}$ (with $\mathbf{R}$ a domain in the real number set). α is a bijective mapping $\alpha : P \rightarrow X$ that assigns to each place a variable from the set X. EFS is an extended fuzzy set of the fuzzy set $FS = \{A_1, A_2, \cdots, A_k\}$, $EFS = FS \bigcup \{\Phi\}$. The statement $x\, is\, \Phi$ means there is no information about the value of the variable x at the current moment of time.

The marking $M(p_i)$ of a place p_i is the vector of the membership degrees of the assigned variable to the fuzzy set

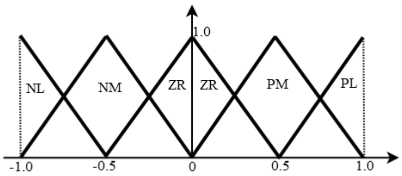

Fig. 2. Fuzzy logic membership functions.

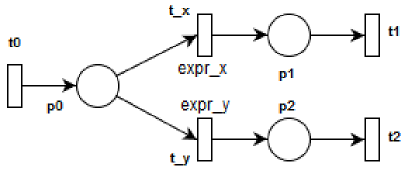

Fig. 3. Selection by expressions.

TABLE I
Fuzzy logic rules $FLRSx, y$ in Fig. 4

NL	NM	ZR	PM	PL
Φ,ZR	Φ,ZR	PL, PL	ZR,Φ	ZR,Φ

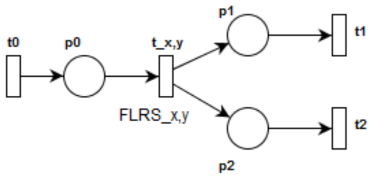

Fig. 4. Selection by FLRS.

FS. Any distinct token of the form

$$\mu =< \mu_1, \mu_2, \cdots, \mu_k > \quad (5)$$

inserted into a place p_i expresses the membership degree of the variable x_i to the fuzzy set FS. In the current case, a place corresponds to a set of statements and the information is available only when a token μ is contained.

Each input arc of a transition is endowed with a weighting coefficient: $W : P \times T \rightarrow \mathbf{R}$ (with **R** a domain in the real number set), $W(p_i, t_j) = w_{ij} \in \mathbf{R}$.

FLRS is a set of fuzzy rule sets. β is a mapping that assigns to each transition a fuzzy logic rule set $\beta : T \rightarrow$ **FLRS**. The fuzzy logic rules considered here have the form:

$$IF\ x_1 is A_1 \wedge x_2 is A_2 \wedge \cdots \wedge x_k is A_k\ THEN$$
$$x'_1 is A_1 \wedge x'_2 is A_2 \wedge \cdots \wedge x'_k is A_k \quad (6)$$

with $x'_1, x'_2, \cdots, x'_k$ belonging to the same set X and representing the consequences of the inference rules.

An example is $FS = \{NL, NM, ZR, PM, PL\}$ where the elements mean negative large, negative medium, zero, positive medium and positive large respectively. For simplicity reasons, the membership functions used for fuzzification and defuzzification are those presented in Figure 2. For practical reasons the values of the variables $x_i \in X$ were bounded to the real number set [-1,1].

An example of rule using FS is:

$$IF\ x_1 is ZR \wedge x_2 is NM\ THEN\ x_3 is PM \wedge x_4 is PL \quad (7)$$

In an earlier release of FLETPN model (see [9]) the selection of alternatives was implemented based on logical expressions assigned to transitions. For example, the selection to continue the execution with transitions t_1 or t_2 included in the partial FLETPN model represented in Figure 3 was chosen using the expressions $expr_x$ and $expr_y$. In the current release the logical expressions were removed and the selection is performed by appropriately conceiving the $FLRS_{x,y}$, as shown in Figure 4 and in Table I.

Supposing that all the rules have the same two inputs and two outputs (i. e. consequences) the fuzzy logic rule set can be described in a table such as Table II. There are represented the following rules:

$$IF\ x_1 is NL \wedge x_2 is NL\ THEN\ x_3 is NL \quad (8)$$

$$IF\ x_1 is PL \wedge x_2 is PL\ THEN\ x_4 is PM \quad (9)$$

The consequence of rule (7) injects a token into the place p_3 and another one into the place p_4. The consequence of rule (8) means that if only this rule is activated, the execution of the transition leads to a token in the place p_3 and no token in the place p_4. Unlike rule (8), rule (9) leads to a token in the place p_4 and no token in the place p_3. This manner allows the selection to continue the execution from the place p_3, the place p_4 or from the both of them.

An input x_i can belong to the fuzzy set A_j with a membership degree $\mu_j(x_i)$. For the given example, if the variable x_i is assigned to a place p_i a token injected into this place can be $< \mu_{NL}, \mu_{NM}, \mu_{ZR}, \mu_{PM}, \mu_{PL} >$ and it describes the membership degree of the variable x_i to the fuzzy set FS. All the rules included into a fuzzy rule set have the same inputs and outputs. The dimension (cardinal) of a fuzzy rule set of a transition t_i is $|FS|^l$ where $|FS|$ is the cardinal of the fuzzy set and l the number of the input places of the current transition.

The fuzzy rule set provides an output vector of the dimension equal to the cardinal of the transition output place set. The elements of this output vector are fuzzy sets.

The *execution of an enabled transition* t_j involves:

- the extracting of the tokens from the transition input places (denoted by ${}^o t_i$);

TABLE II
Example of inference rules

$x_1 \backslash x_2$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	NL, ϕ	NL, Φ	NL,PL	ZR,PL	PL,PL
NM	NM, Φ	PL, Φ	PL,PL	PL,PL	NL,PL
ZR	PM,PL	PM,PL	ZR,PL	NL,PL	PM,PL
PM	ZR,PL	PL,PL	NL,PL	ϕ, NM	ϕ, PL
PL	NL,PL	ZR,PL	ZR,PL	ϕ, PM	ϕ, PM

- the defuzzification of all input variables x_i;
- the multiplication of the variables with the corresponding weighting coefficients $x'_{ij} = w_{ij}x_i$;
- the fuzzification of the variables x'_{ij};
- the use of the FLRS with x'_{ij} as inputs;
- the normalization operation that reduces the previous consequences to a single one and leads to injection of a single token into the output places;
- the injecting of the resulted tokens into the transition output places (denoted by t_i^o) when the delay elapses.

Due to the fact that a variable number of rules can be involved (activated) for a transition execution, this could inject a variable number of tokens into the output places. To avoid this, a *normalization* operation is required. Let $r_l, l = 1, \cdots$ be the rules that are activated. The *strength* s_l of a rule is calculated with $s_l = \mu_1 \cdots \mu_k$ where μ_i is the membership of the input variable. Let z_l be the crude value provided as a consequence by the rule r_l. The value of the transition output variable x' is:

$$x' = \frac{\sum\limits_l z_l \cdot s_l}{\sum\limits_l s_l} \tag{10}$$

The regular fuzzification of the variable x provides the token that is injected into the output place. As a consequence, the execution of every transition leads to a single token or no token in each output place. The result of the normalization operation leads to a token the fulfills the relation:

$$\sum_i \mu_i = 1 \tag{11}$$

A FLETPN can model the synchronous and asynchronous reaction to a signal belonging to a continuous domain. The handling of the discrete events can be implemented by constraining the membership degree. For example, if a variable x_i assigned to a place p_i is of the discrete event type, the variable belongs (by convention) to PL set. That means all the tokens injected into the place p_i have the form $< 0, 0, 0, 0, 1 >$.

A discrete event variable is a particular case of a continuous variable. All the discrete event variables belong to the same set (could be a fuzzy set) with a membership degree $\mu = 1$. A transition could have input places corresponding to discrete event variables and places corresponding to continuous variables. The assigned fuzzy rule set has to be constructed according to this structure.

In conclusion, a FLETPN model can mix the continuous type tokens with the discrete event type tokens, but every place can contain only one type tokens. Using Petri nets with tokens integrating higher complex information simplifies the program structure, while the program functionality is moved to the associated FLRSs. A program with a simpler structure is easier to be synthesized and to be tested for fulfilling the real-time features. The FLRSs have to be found such that the program fulfills the functional requirements.

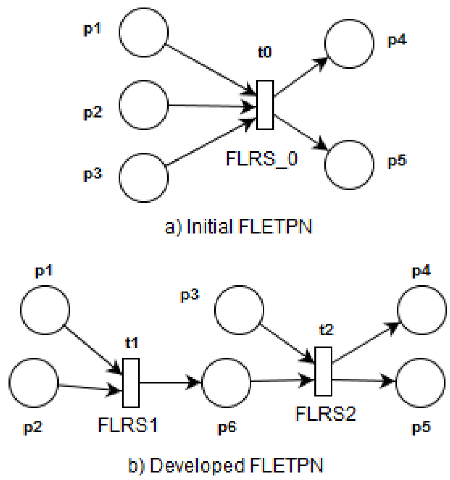

Fig. 5. Reducing the number of a transition's input places.

The FLETPN model should be free of conflicts. If conflicts exist in the ETPN model, the executor grants the execution to the transitions with shorter delays, and if multiple transitions with the same delays are simultaneously enabled, the transitions with lower indexes are chosen for fire. Even if an ETPN model has conflicts, these can be removed by the appropriate conceiving of the FLRSs using the method shown in Figure 4 and Table I.

According to [8] the reasoning process by using fuzzy PN can be implemented by algorithms involving reachability trees, algebra forms and high level PNs. The current approach concerns the modeling of dynamic control systems implemented by reactive programs. The TPNs describe the program structures, while the FLRSs implement their functionalities.

C. FLRS construction

For practical reasons it is convenient to have transitions with maximum two input places. In this case all the fuzzy logic rules have maximum two premises. If there are requirements to have transitions with more than two input places, each of them can be replaced by two transitions as shown in Figure 5. Consequently, all the FLRSs have maximum two premises, but the number of consequences of a rule is not limited.

Supposing that the control system synthesizer constructed the FLETPN, a remained relevant task for the current method consists of the construction of the fuzzy logic rule sets that are assigned to transitions. The proposed method uses the Genetic Algorithm (GA) to search a FLRS that is capable to control the given plant with a specified competence. The control system fulfills the competence requirements if the assessment of the system behavior exceeds a specified threshold. The control system synthesizer has to provide the set of relevant tests used for system evaluation.

The genome is composed of genes coding (by non-negative integers) the consequences of the all FLRSs and genes coding (by real numbers) the weighting coefficients. Table III shows

TABLE III
Example of a genotype.

$c_{1,1}$	$c_{1,2}$	...	$c_{4,5}$	$c_{5,5}$	w_{11}	w_{21}
000010	010100	...	100100	001100	$2,35$	$-3,28$

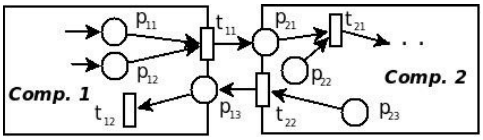

Fig. 6. Example of CPS component diagram.

an example of a genotype for the FLETPN presented in Figure 1. The notations $c_{i,j} (i = 1, \ldots, 5; j = 1, \ldots, 5)$ represent the pairs of consequences included in a table (e. g. Table II) considered as matrix.

Two kinds of mutation operators were used: one acting on the integer number part and another on the real number part. The synthesizer has to provide the domains of the parameters w_{ij}. The crossover operator splits the genotypes in one point. The selections are performed using the classical performance functions taking into account the plant set points and the constraints.

III. CPS Component Diagram

The design of a control system is based on a set of components that contain discrete event, discrete time or hybrid models. They are included into a component diagram. The models of the current approach are FLETPN.

Fig. 6 shows an example of a CPS component diagram. The two components Comp.1 and Comp.2 are connected through the ports represented on the frontiers. The component ports on the frontiers contain transitions and places. The transitions on the frontier describe outgoing actions (send tokens), and the places receive the tokens.

All the components have input and output ports implementing interfaces. In the given example the component Comp. 1 has the interfaces $Out = \{t_{11}\}$ and $Inp = \{p_{13}\}$. In a well designed component diagram the transitions of a set Out have input places only from the places included in the FLETPN model of the current component, and the places of a set Inp have output transitions only from the current component.

Each component has its own thread of execution. An interface Inp is implemented by an input port and an interface Out by an output port.When a transition of the set Out is executed, the component (output port) sends the corresponding tokens to the components that have in their input interfaces places of the current transition output set via linked component ports. This is denoted by $send(t_i^o, X_{i,out})$ where $X_{i,out}$ corresponds to the tokens (i.e. the variable values of the marking) that have to be sent and included in the transition output places. The destination component executes $receive(Inp, X_i)$ and updates its marking.

The component thread is executed cyclically or when an external event is signaled. The thread is awoke by the input port when a new token arrives or when the clock tic occurs. If the signaled event is tic, the delays of the activated transitions are decreased. If the signaled event is *new token*, the cycle of the thread starts with updating the information of its input place set. The FLETPN marking is updated with the newly received information.

Complex applications can be conceived including components in other components. The links between a component and its included components implement the same protocol outgoing port-ingoing port based on the transition-place connection.The proposed models partition the program structure and functionality at the component level in a compact manner.

IV. FLETPN executor

The executor algorithm of FLETPN is executed with the period of 1 time unit (t.u.) or when an external event is signaled loading an input place with a token. The algorithm updates the places of the input set and determines the transitions that are enabled taking into account the markings of the transition's input place set. If a transition is chosen to be executed, the tokens from its input place set are removed and injected into a temporal marking vector M_t. A time counter $Delay[t_i]$ is loaded with the transition assigned delay if it has any or zero. If the time counter is zero or it reaches the value 0 (after decreasing), the execution of the transition is finished. If a transition belongs to the output set *Out*, its execution is signaled to the linked output place and the corresponding token is loaded.

The counters of all the started transitions are decreased after each sample period.

FLETPN executor algorithm:
Input: ***Pre***, ***Post***, $X, M_0, P, T, D,$ ***FLRS***, Out, Inp;
Initialization: $M = M_0$, *execList* $= empty$;
* reorder the transition set T according to their delays;
repeat
  wait(event);
  if event is tic **then**
    * decrease the Delays of the transitions in *execList*;
  else
    $receive($*Inp*$, X_i)$;
    * update M;
  end if;
  repeat
    for all $t_i \in T$ **do**
      if all $p \in {}^o t_i$, $M(p) \neq \Phi$ **then**
        * move the tokens of ${}^o t_i$ from M to M_t;
        * add t_i to *execList*;
        $Delay[t_i] = d_i$;
      end if;
    end for;
    for all t_i in *execList* **do**
      if ($Delay[t_i]$ is 0) **then**
        * remove t_i from *execList*;
        * calculate and inject the tokens in M for all t_i^o;

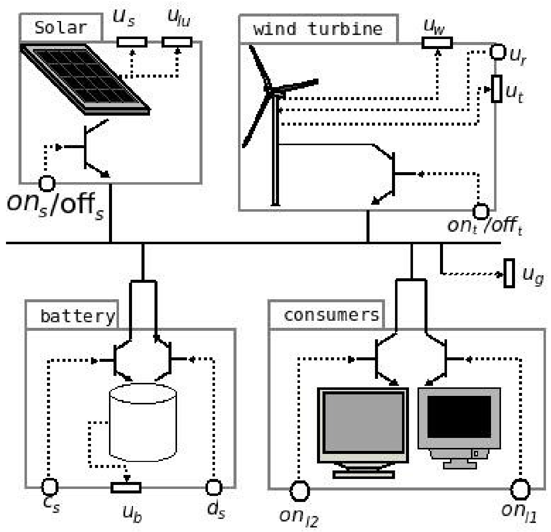

Fig. 7. Microgrid architecture.

* remove the tokens from M_t for all ${}^o t_i$;
end if
if $t_i \in Out$ **then**
$send(Out, X_{i,out})$;
end if
end for
until there is no transition that can be executed;
until the time horizon;
END algorithm;

V. Examples of Application

A. Energy microgrid specifications

The plant represented in Fig. 7 concerns an energy micro grid composed of a solar cell, a wind turbine, a battery and two loads. The plant has to be controlled according to the specification. The control system has to maintain the voltage (u_g) of the main bus between u_m and u_M. The turbine and solar generators asynchronously inject energy into the system. The consumers asynchronously demand the use of the energy as well. When the produced energy exceeds the demands, the surplus has to be discharged on the battery. If the level of the generated energy is lower than the demand, the battery should be used to increase the voltage to guaranty that the bus voltage level remains between the specified limits. The control system is composed of load controller (L-Controller), turbine controller (T-Controller), solar cell controller (S-Controller) and battery controller (B-Controller) as can be seen in Fig. 8.

The following notations are used:

- u_t the turbine output voltage
- u_w the wind force applied to the turbine
- u_n the main bus nominal voltage
- u_g the voltage of the grid main bus
- u_s the solar cell output voltage
- u_{lu} the luminosity on the solar cell
- on_s/off_s control signal to connect or disconnect the solar cell
- on_t/off_t control signal to connect or disconnect the turbine
- d_s control signal to connect the battery to increase the voltage
- c_s control signal to connect the battery to charge it
- u_b the battery level
- on_{l1} control signal to connect the first load
- on_{l2} control signal to connect the second load

The control system requirements are:

repeat
$u_g = read(u_{bus})$;
if $(u_g \geq u_M)$ **then**
* connect the battery to discharge energy;
end if;
if $(u_g \leq u_m)$ **then**
$u_b = read(u_{battery})$;
if $u_b > u_{minim}$ **then**
* connect the battery to increase the voltage;
end if;
end if;
if $(u_m \leq u_g \leq u_M)$ **then**
* disconnect the battery;
end if;
$u_t = read(u_w)$;
if $(u_t < u_n)$ **then**
* stop(turbine);
else
*start(turbine);
end if;
$u_s = read(u_{lu})$;
if $(u_s < u_n)$ **then**
* stop(solarCell);
else
* start(solarCell);
end if
wait(1 t.u.);
$u_g = read(u_{bus})$;
if $(u_g < u_m)$ **then**
* stop(load 2);
else
*allow(load 2);
end if;
wait(1 t.u.);
$u_g = read(u_{bus})$;
if $(u_g < u_m)$ **then**
* stop(load 1);
else
*allow(load 1);
end if;
wait(1 t.u.);
until (the time horizon);

B. Plant model

The loads are considered pure resistances. The photo voltaic solar cells produce energy proportionally with the environment luminosity. The most complex is the wind turbine model.

The discretization (by approximation) of the wind turbine model constructed of differential equations [14] leads to:

$$X_1(k+1) = A_1 \cdot X_1(k) + B_1 \cdot u_r(k) \tag{12}$$

$$Y_1(k+1) = C_1 \cdot X_1(k) \tag{13}$$

$$U(k) = u_w(k) \cdot \cos(Y_1(k)) \tag{14}$$

$$X_2(k+1) = A_2 \cdot X_2(k) + B_2 \cdot U(k) \tag{15}$$

$$u_t(k+1) = C_2 \cdot X_2(k) \tag{16}$$

The notations are:

- X_1 and X_2 are 3 dimensional state vectors,
- Y_1 is a mono dimensional output vector,
- u_r is the input control signal used for the positioning of the turbine,
- U is a combination of the output Y_1 and the wind force u_w.
- u_t is the turbine output voltage.

The corresponding matrix are:

$$A_1 = \begin{pmatrix} 0.050558 & 2.6979e-10 & 5.9355e-05 \\ 0.029825 & 1 & 0.77505 \\ 0.034992 & -4.1333e-06 & 0.58666 \end{pmatrix}$$

$$B_1 = \begin{pmatrix} 0.052746 \\ 0.0020924 \\ 0.0049453 \end{pmatrix}$$

$$C_1 = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 0.0021802 & -5.8872e-08 & 0.0062 \\ 0.029825 & 1 & 0.77505 \\ 0.034992 & -4.1333e-06 & 0.58666 \end{pmatrix}$$

$$B_2 = \begin{pmatrix} 0.0019624 \\ 0.010389 \\ 0.18356 \end{pmatrix}$$

$$C_2 = \begin{pmatrix} 300 & 0 & 0 \end{pmatrix}$$

The discretization of the continuous model of the wind turbine was performed for the aim of reducing the calculus volume involved by the GA.

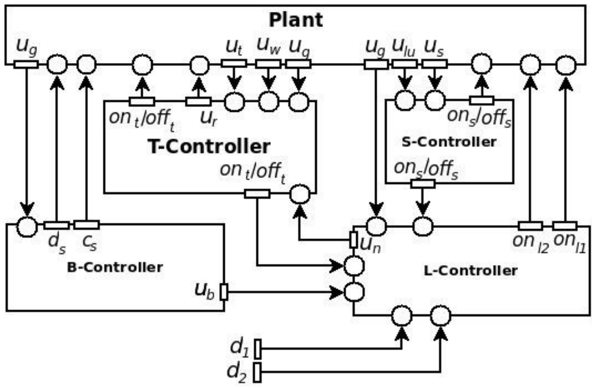

Fig. 8. Control component diagram.

TABLE IV
COEFFICIENTS OF THE FLETPN MODEL

w_1	w_2	w_3	w_4	w_5	w_6
-0.1829	-2.5079	-7.7209	0.1332	4.7337	0.0076

C. Control system architecture

The control system can be conceived as:

- independent controllers
- coordinated controllers or
- cooperative controllers.

Figure 8 shows the proposed component diagram for the control system. The component T-Controller solves the control problem related to the turbine, the component S-Controller concerns the solar cell and the component B-Controller connects or disconnects the battery. The L-Controller has the role to receive the user demands to connect *load 1* and *load 2*. The L-Controller decides to perform these actions taking into account the current energy produced and accumulated. The L-Controller can work independently, to coordinate the other controllers, or to cooperate with them according to the specifications.

D. Wind turbine control component

Figure 9 shows the FLETPN model synthesized for the turbine control. The zero delays of the transitions are not represented on the FLETPN for simplicity reason. The coefficients of the FLETPN are given in Table IV. Some transitions have associated FLRSs as they are mentioned in Tables V, VI, VII, VIII and IX. Other transitions perform simple transformations or store operations.

The T-Controller achieves a kind of fuzzy logic PID (Proportional Integrative Derivative) control function. The place p_0 is loaded with a token corresponding to u_n (nominal voltage, i.e. set point) and the place p_1 with a token corresponding to u_t (turbine output voltage; when it works, it is equally to the main bus voltage). The transition t_0 calculates (using the FLRS assigned to the current transition) the error $e(k) = u_n(k) - u_t(k)$. The resulted tokens are injected into the places

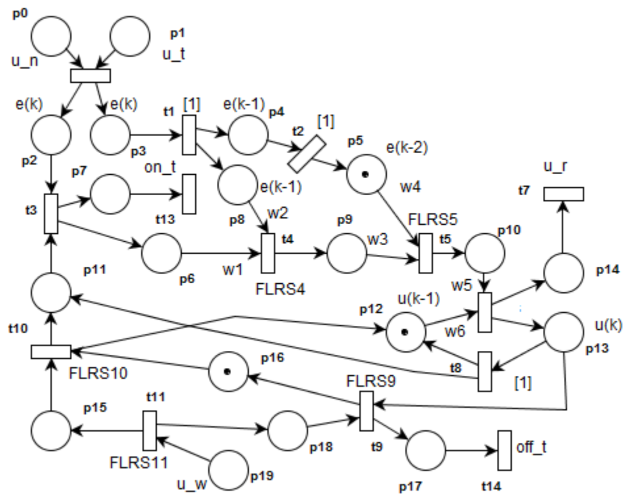

Fig. 9. Turbine control FLETPN.

TABLE V
Fuzzy logic rules of transition t_4 in Fig. 9

$x_6\backslash x_8$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	NM,PM	PM,ZR	NL,NM	NL,ZR	NL,NM
NM	NL,ZR	NM,NL	NM,ZR	ZR,ZR	PL,PM
ZR	NL.PM	NL,NM	NL,PM	NL,NL	NL,ZR
PM	ZR,NL	PM,NM	ZR,NM	ZR,PM	PL,PM
PL	PM,PM	NM,ZR	NL,NM	PM,NL	NM,NM

TABLE VI
Fuzzy logic rules of transition t_5 in Fig. 9

$x_9\backslash x_5$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	PM,ZR	PL,ZR	ZR,PL	ZR,NM	PM,PL
NM	PL,NL	NM,ZR	NM,PM	NL,PL	ZR,PM
ZR	PL,NM	PL,NL	NM,PL	NL,NL	PM,NL
PM	PM,PM	NL,NM	PM,NM	PM,NM	PL,ZR
PL	NL,NL	NL,PM	ZR,ZR	PL,ZR	NL,ZR

TABLE VII
Fuzzy logic rules of transition t_9 in Fig. 9

$x_{13}\backslash x_{18}$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	ZR,NL	ZR,NL	Φ,Φ	Φ,Φ	Φ,Φ
NM	ZR,NM	ZR,NM	Φ,Φ	Φ,Φ	Φ,Φ
ZR	ZR,ZR	ZR,ZR	Φ,Φ	Φ,Φ	Φ,Φ
PM	ZR,PM	ZR,PM	Φ,Φ	Φ,Φ	Φ,Φ
PL	ZR,ZR	ZR,ZR	Φ,Φ	Φ,Φ	Φ,Φ

TABLE VIII
Fuzzy logic rules of transition t_{10} in Fig. 9

$x_{15}\backslash x_{16}$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	Φ,Φ	Φ,Φ	Φ,Φ	ZR,PM	ZR,PL
NM	Φ,Φ	Φ,Φ	Φ,Φ	ZR,PM	ZR,PL
ZR	Φ,Φ	Φ,Φ	Φ,Φ	ZR,PM	ZR,PL
PM	Φ,Φ	Φ,Φ	Φ,Φ	ZR,PM	ZR,PL
PL	Φ,Φ	Φ,Φ	Φ,Φ	ZR,PM	ZR,PL

TABLE IX
Fuzzy logic rules of transition t_{11} in Fig. 9

x_{19}	NL	NM	ZR	PM	PL
	Φ,ZR	Φ,ZR	Φ,Φ	ZR,Φ	ZR,Φ

p_2 and p_3. The transition t_1 injects tokens corresponding to the variable e(k-1) into the places p_4 and p_8 after 1 t.u. (time unit) delay. Similar injection performs the transition t_2 for the value e(k-2) into the place p_5. The transition t_3 calculates if the wind turbine can work properly and inject into the output places p_7 the token on_t. The transition t_4 calculates a function of the type $f(w_1e(k), w_2e(k-1))$ using the $FLRS_4$ presented in the Table V. A similar function is performed by the transition t_5 using the $FLRS_5$ given in Table VI. The place p_{10} contains the current variation $\Delta u(k)$ of the control signal. The transition t_6 calculates the current control signal, using the values $w_5\Delta u(k)$ and $w_6u(k-1)$, and injects it into the places p_{13} and p_{14}. The transition t_7 sends the control signal u_r to the turbine. The transition t_8 reloads the place p_{12} with the previous value of the control signal and permits a new execution loading the place p_{11}.

The input place p_{19} is loaded with the wind force u_w (speed) value. If the u_w is lower than a specified value, the turbine is stopped by injecting a token off_t into the place p_{18}. This allows the execution of the transition t_9. If the turbine was stopped (p_{16} has a token) and the wind speed is according to specification, the transition t_{11} allows the turbine to start injecting a token into the place p_{15}. The $FLRS_{11}$ assigned to the transition t_{11} discerns if the wind turbine can work properly injecting a token into the place p_{15} or not and as a consequence it injects a token into the place p_{18}. Table IX contains the $FLRS_{11}$ assigned to transition t_{11}.

E. Load control component

Figure 10 presents the FLETPN model of an independent L-Controller component. This receives in place p_1 as a continuous variable the bus voltage u_g and transforms it into a fuzzy logic value. The L-Controller receives the user's demand d_1 to connect *load 1* as a discrete input $< 0,0,0,0,1 >$ or disconnect as the value $< 1,0,0,0,0 >$. The controller uses these two pieces of information to accept or not the demand using the $FLRS_1$ and signals this by the port (transition) t_4 with the values $< 0,0,0,0,1 >$ or $< 1,0,0,0,0 >$. The information is passed further to the place p_4. The transition t_2 takes the user demand d_2 to connect or not *load 2*, calculates the controller behavior using the $FLRS_2$ and signals this by the transition t_5.

Figure 11 presents a FLETPN that correspond to a cooperative L-Controller. It added the information on_t and on_s to determine the connection or disconnection of the *load 1* and *load 2*. The L-Controller also sets the reference point u_n for the T-Controller to a better adjustment of the bus voltage u_g. Unlike the previous L-controller, the cooperative controller uses the information $E(k)$ denoting the current power (energy)

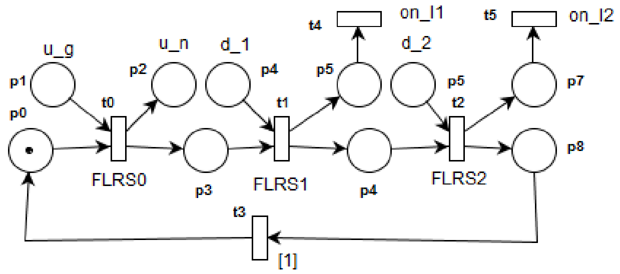

Fig. 10. FLETPN of the independent L-Controller.

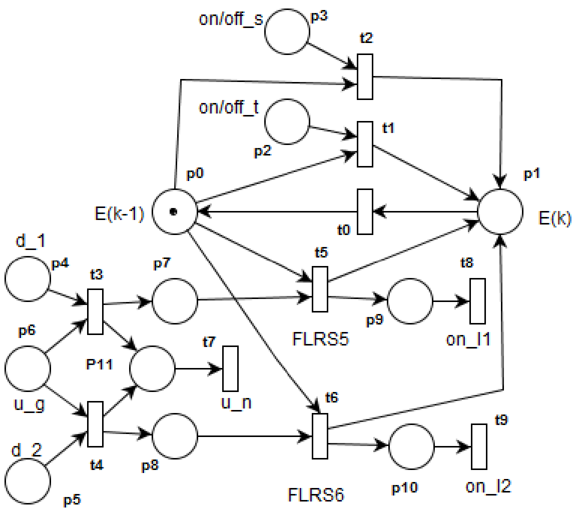

Fig. 11. FLETPN of the cooperative L-Controller.

introduced into the system. $E(k-1)$ stores the power available at the previous clock tic. The input place p_2 is injected with a token on/off_t signaling the event that the wind turbine is working or not working respectively. The transition t_1 is used to increase or decrease the information about the current power level. Similar function performs the transition t_2 for the solar cell using the token on/off_s for that purpose. The demand for connecting *load 1* or *load 2* is granted according to the current available power and the voltage u_g. The transitions t_5 and t_6 permit or not the connections and modify the power level. The transitions $t_1, t_2, ..., t_6$ have assigned the necessary FLRSs. Table X shows $FLRS_5$ and $FLRS_6$ assigned to transitions t_5 and t_6.

VI. Tests and Results

All the tests were performed by simulations using standard Java language. Figure 12 presents the test results for the turbine generator. The weighting coefficients $w_i, i = 1, 2, \cdots, 6$ and the FLRSs are calculated using a genetic algorithm. The genome contains the rows of the FLRSs and the weighting coefficients. The fitness function assesses the response to perturbations as shown in Figure 12. The searching process was stopped when a competent solution was obtained, that

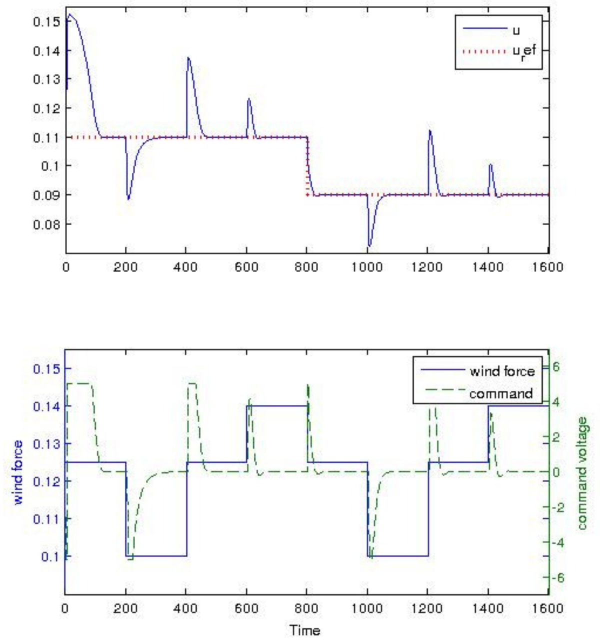

Fig. 12. Turbine signals.

TABLE X
Fuzzy logic rules of transitions t_5 and t_6 in Fig. 11

$x_0 \backslash x_5, x_6$	**NL**	**NM**	**ZR**	**PM**	**PL**
NL	NL, Φ	NM, Φ	ZR, Φ	PM, Φ	PL, Φ
NM	NL, Φ	NM, Φ	ZR, Φ	PM, Φ	PL, Φ
ZR	NL, Φ	NM, Φ	ZR, Φ	PM, Φ	PL, Φ
PM	NL, Φ	NM, Φ	ZR, Φ	ZR, ZR	PM, ZR
PL	NL, Φ	NM, Φ	ZR, Φ	ZR, ZR	PM, ZR

is the control performances exceed the specified values. The FLRSs obtained by GA are given in Table V, Table VI, ... and Table IX.

Adding the empty set Φ to the fuzzy logic set permits the deterministic selection of the execution on the different paths as can be seen in the FLETPN presented in Figure 9. The conflict between the transitions t_8 and t_9 was solved by the rule *execute the earliest possible transition*. The conflict between the transitions t_9 and t_{10} is solved by transition t_{11} injecting tokens into the places p_{15} or p_{18} according to the token introduced into the input place p_{19}.

In Figure 11 the conflict between the transitions t_3 and t_4 is solved by the rule *execute the transition with the lowest index*. The conflict between t_5 and t_6 is solved by the previous selection.

VII. Conclusions

The proposed method can be easily used to conceive the hybrid control system for different kinds of hybrid plants. It needs the use of knowledge from the same field combining the Petri nets capabilities to implement the discrete event systems requirements with fuzzy logic models suitable for continuous systems.

There are some benefits of the proposed method: Constructing the tokens with the membership degrees of a variable to all the fuzzy set and assigning to any transition an entire fuzzy rule set leads to a smaller Petri net, and this increases the capability of the model to be used in more complex applications.

The FLETPN models are capable to include the discrete event part and discrete time part. The distinct tokens injected into the corresponding discrete event type places and continuous type places make possible to comprise in the same model the discrete event and discrete time behavior. The FLETPN models can describe the concurrent, synchronous and asynchronous behavior. The reactions to asynchronous events are taken into account when the event occur. These models can easily be implemented, and if a TPN executor is used, the need of a real-time operating system can be avoided.

The structure of the model can be verified using the TPN analyses methods. The proposed method can be used for the verification of the discrete event behavior.

The verification (i. e. the performance evaluation) of the continuous side behavior can be performed by simulation. The weighting coefficients added to input arcs increase the continuous control capabilities enhancing the fuzzy logic rules with the possibility to amplify the relative significance of some variables.

REFERENCES

[1] OMG Unified Modeling Language V2.5, 2015, http://www.omg.org/spec/UML/2.5/PDF/

[2] R. Banach, M. Butler, S. Qin, N. Verma and H. Zhu. Core Hybrid Event-B Machine, Science of Computer Programming, Elsevier, vol. 105, pp. 92-123, 2015.

[3] F. Cicirelli, C. Nigro and L. Nigro, "Qualitative and Quantitative Evaluation of Stochastic Time Petri Nets", Proc. of the IEEE Federated Conference on Computer Science and Information Systems, Vol. 5, 2015, pp. 763–772, doi: 10.15439/2015F69

[4] E.Y.T. Juan, J.P. Tsai and T. Murata, Y. Zhou. Reduction methods for real-time systems using delay time Petri nets, IEEE Transactions on Software and Engineering, volume: 27, no. 5, pp. 422-448, 2001.

[5] T. S. Letia and A. O. Kilyen, Enhancing the time-Petri nets for automatic hybrid control synthesis. Proc. of ICSTCC IEEE Conference, Sinaia, Romania, pp. 627-633, 2014, doi:10.1109/ICSTCC.2014.6982486.

[6] T. Munakata and Y. Jani. Fuzzy Systems: An Overview, Comm. ACM, vol. 37, no. 3, pp. 69-76, Mar. 1994.

[7] C.C. Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part I1, Trans. on Systems, Man and Cybernetics, vol. 20, no. 2, pp. 419-435, 1990.

[8] K.-Q. Zhou and A. M. Zain, "Fuzzy Petri nets and industrial applications: a review", Artificial Intelligence Review, Springer, 2016, doi: 10.1007/s10462-015-9451-9.

[9] T. S. Letia and A. O. Kilyen, Fuzzy Logic Enhanced Time Petri Net Models for Hybrid Control Systems . Proc. of AQTR 2016 IEEE Conference, Cluj-Napoca, Romania, 2016, doi: 10.1109/AQTR.2016.7501322.

[10] J. Shi, J. Wan, H. Yan and Hui Suo. A Survey of Cyber-Physical Systems, WCSP - International Conference on Wireless Communications and Signal Processing, pp. 1-6, 2011.

[11] I. Horvath and B. H. M. Gerritsen. Cyber-Physical Systems: Concepts, Technologies and Implementation Principles, Proceedings of TMCE 2012, Karlsruhe, pp.19-36, 2012.

[12] S. N. Krishna and A. Trivedi. Hybrid Automata for Formal Modeling and Verification of Cyber-Physical Systems, Journal of the Indian Institute of Science VOL 93:3 Jul.Sep. 2013

[13] R. Banach, P. Van Schaik and E. Verhulst, "Simulation and Formal Modelling of Yaw Control in a Drive-by-Wire Application ", Proc. of the IEEE 2015 Federated Conference on Computer Science and Information Systems, 731–742, 2015, doi:10.15439/2015F132,

[14] J. de J. Rubio, L. A. Soriano and W. Yu, Dynamic Model of a Wind Turbine for the Electric Energy Generation, Mathematical Problems in Engineering, vol. 2014, Article ID 409268, 8 pages, 2014. doi:10.1155/2014/409268