Modeling and Analysis of Stage Machinery Control Systems by Timed Colored Petri Nets

Hehua Zhang, Ming Gu, Xiaoyu Song

Modeling and Analysis of Stage Machinery Control Systems by Timed Colored Petri Nets

Hehua Zhang
Dept. CST
School of Software
Key Laboratory of Information
System Security, Ministry of Education
Tsinghua National Laboratory
for Information Science and Technology
Tsinghua university, Beijing, China
Email: zhang-hh04@mails.tsinghua.edu.cn

Ming Gu
School of Software
Key Laboratory of Information
System Security, Ministry of Education
Tsinghua university
Beijing, China
Email: guming@tsinghua.edu.cn

Xiaoyu Song
Dept. ECE
Portland State University
Oregan, USA
Email: song@ee.pdx.edu

***Abstract*— This paper presents an approach and successful experience of applying timed colored Petri nets on modeling and analyzing a stage machinery control system. The programmable logic controllers (PLCs) based system is modeled with timing constraints. The compositionality is incorporated in the modeling process of the entire design. The PLC synchronization problem with the interactions of environment is analyzed by the state space analysis method. The case studies demonstrate the effectiveness of the approach.**

I. Introduction

Programmable Logic Controllers (PLC) are widely used in industry [1], [2]. In the stage machinery control system, PLCs are the control equipments. The main stages of developing a PLC-based system are requirement analysis, hardware/software design, equipment choosing, PLC programming and product debugging. There have been many research works which focus on the PLC programming [3], [4], [5], [6]. However, from the engineering point of view, logic errors inside PLC programs normally do not pose the key issues. Errors often occur due to deficient consideration of the environment conditions or misunderstanding the actions of the plants. In addition, synchronization among PLC and its controlled plants, called the environment of PLC, is critical to the correctness of a system.

Petri nets (PN) have been proven to be a powerful modeling tool for various kinds of discrete event systems [7], [8]. The typical characteristics exhibited by the activities such as concurrency, decision making, synchronization, can be modeled effectively by Petri nets. It provides strict mathematical basics which make the formal analysis as well as simulation possible. There are some Petri net based models like Synchronized PN [9], LCIPN [10], SIPN [11], NCES [12] in PLC field. They consider the synchronization between PLC and its environment. However, these research works concentrate on modeling the control algorithm. There is no plant model considered in their work and they cannot give a system-level model.

In this paper, the strategies and some experience of applying Timed Colored Petri Nets (TCPN) are proposed to model and analyze a PLC-based stage machinery control system[1]. It was responsible for the rigorous control of all kinds of stage equipments in the theater. The correctness of the stage machinery control systems is critical to the reliability of the whole theater. We explore the suitable systematic models for rigorous design and verification of PLC-based embedded systems. The proposed model can help understand actions inside the parts as well as the interactions. It can be used to describe and analyze the synchronization problem. The analysis results of the design can help the following equipments selection and PLC programming. The model is suitable for the system-level design of PLC systems as well as the detailed design of the plants in PLC systems. The compositional construction and the graphical representation of the TCPN model are understandable. The time information can be used to analyze whether the PLC system is correctly synchronized or not through simulation or formal analysis.

The rest of the paper is organized as follows. Section II addresses the formal definitions of TCPN model. In Section III, the stage machinery control system, especially the light control part is introduced. The system modeling procedure and method based on TCPN are presented in Section IV. In Section V, we formally analyze the synchronization problem and show how the analysis results can be applied after the design phase. Finally, some conclusions are drawn in Section VI.

II. Preliminaries of TCPN

The definition of TCPN follows the original definitions of Timed CPN by [13]. The main part of the TCPN definition is as follows. For detailed interpretation and the semantics of TCPN, see [13].

Definition 1. A timed colored Petri net is a tuple TCPN= (CPN, **R**, r_0) such that:

1) CPN = (Σ, P, T, A, N, C, G, D, E, I) is a colored Petri net, satisfying that:

[1]This research is sponsored by NSFC Program (60553002, 90718039) and 973 Program (2004CB719406) of China.

978-1-4244-1995-1/08/$25.00 ©2008 IEEE.

a) Σ is a finite set of non empty types, called color sets.
b) P is a finite set of places.
c) T is a finite set of transitions.
d) A is a finite set of arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$.
e) N is a node function. It is defined from A into $P \times T \cup T \times P$.
f) C is a color function. It is defined from P into Σ.
g) G is a guard function. It is defined from T into expressions such that: $\forall t \in T : [Type(G(t)) = BOOL \wedge Type(Var(G(t)) \subseteq \Sigma]$. $Type(expr)$ denotes the *type of an expression*; $Var(expr)$ denotes the *set of variables in an expression*.
h) D is a time delay function. It is defined from T into numbers such that: $\forall t \in T : [Type(D(t)) = \mathbf{R} \wedge Type(Var(D(t)) \subseteq \Sigma]$.
i) E is an arc expression function. It is defined from A into expressions such that: $\forall a \in A : [Type(E(a)) = (C(p(a))_{MS}) \wedge Type(Var(E(a)) \subseteq \Sigma]$, where p(a) is the place of N(a) and the suffix *MS* means a multiset.
j) I is an initialization function. It is defined from P into closed expressions such that: $\forall p \in P : [Type(I(p)) = (C(p))_{MS}]$.

2) **R** is the set of time values, called time stamps. It is a subset of REAL closed under + and contains 0.
3) r_0 is the start time, $r \in \mathbf{R}$.

III. The Stage Machinery Control System

In a PLC-based stage machinery control system, PLCs are introduced to independently control the moving of the presidium seats, fireproof curtains, screen shelves, lights, etc. Without loss of generality, the light control part of the stage machinery control system is introduced in this paper. The control process is carried out by PLCs. A motor draws the light to an appropriate location. The lights can move up and down and locate at any predefined height. When the motor rotates clockwise, the light rises up, otherwise it falls down. The motor rotation angle is measured by a sensor. In the model, each time the motor rotates one degree, the sensor will generate an impulse. To settle the light at the right location, PLC decides the motor actions through counting impulse number. The relationship of the height and the motor rotation angle(also recognized as the number of impulse) is:

$$Rotation\_angle = height * 17 \quad (1)$$

Fig. 1 shows the structure of the light controller with four components. The user can set the height and the moving direction of the light, and then confirm his(or her) settings by the start button on the control panel. The PLC control program instructs the motor to move according to the settings from control panel and the current status of the motor. There are three signals to control the motor: *CW* (to let the motor rotate clockwise), *ACW* (to rotate counter clockwise) and *STOP*. The status of the motor is captured by a sensor.

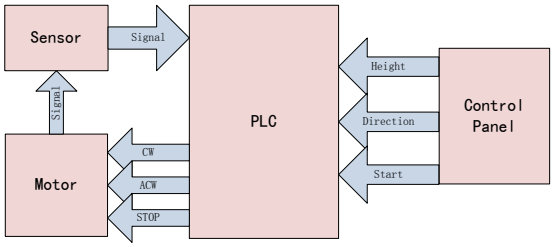

Fig. 1. Structure of a light controller

Note that the motor in a light control system is usually a DC motor, which will keep rotating once it begins until it receives the *STOP* command. It is different from the stepping motor which rotates one cycle with one command. In this case, the synchronization between the motor and PLC should be designed carefully to ensure the correctness.

IV. Modeling PLC Systems

In this section, a compositional method [14], [15] is proposed to construct the model of the light control system. We construct the local model of each part in the system, respectively. There are four parts in the PLC system: control panel, PLC, motor, and sensor. The sub model are constructed respectively and composed into a system model.

A. Control Panel

Users operate the system through the control panel. They can set the height and the moving direction of the light, and modify them arbitrarily before the *start* button on the control panel is pressed down. Once the *start* button is pressed down, the settings will be transmitted to the PLC. The user cannot modify the settings any more. The height to which the light moves up or down should not exceed 20m.

The model of the control panel is shown in Fig. 2. The left side of Fig. 2 gives the related declarations. There are four places and three transitions defined in this sub-model. The places named *init*, d_set, h_set and *userCmd* denote the initial state, the state after the direction has been set, the state after the height has been set, and the state after the user pressed the *start* button to confirm his (or her) settings, respectively. The transitions named *SetDir*, *SetH* and *Start1* describe the user actions of setting the direction, setting the height and pressing the *start* button, respectively.

The declarations of the color sets specify that tokens in place *init* have the same token color *e*, which is the only element of E. This means that these tokens carry no information apart from their presence/absence at the place *init*. The place d_set has a token color which can be *up*, *down* or *stop*, because the color set DIR is declared to be an enumeration type with these three elements. The token color of h_set is an integer, with the range from 0 to 20. The upper bound is set to 20 according to the request that the height should not exceed 20m. Finally, the tokens in place *userCmd* have the token color which is

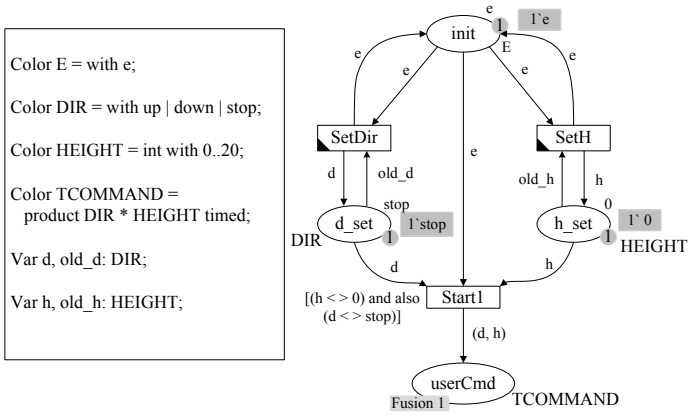

Fig. 2. The TCPN model of the Control Panel

a pair (because the color set TCOMMAND is declared to be the Cartesian product of two other color sets DIR and HEIGHT).The first element of the pair is the element from DIR. The second element is an integer defined by HEIGHT. Note that the tokens in *userCmd* are timed while the tokens in other places are not. From the definition of TCPN, timed and untimed tokens are permitted both in the TCPN model. The untimed tokens are viewed as they are always ready to be used.

To design the control panel in terms of the desired request, the initial marking of the model shown in the Fig. 2 is assigned as: one token with the value *e* in place *init*, one token with the value *stop* in place d_set and another token with the value 0 in place h_set. This means that the control panel is in the initial state and there is no setting up to now. The double arcs between the place *init* and the transition *SetDir* together with the double arcs between the place *init* and the transition *SetH* ensure that the user can set the direction and the height in an arbitrary order and can change his (or her) settings arbitrary many times. The two arcs between the place d_set and the transition *SetDir* with the respective inscription of variable *d* and old_d mean that each direction setting action will consume a token with the last setting value and generate a token with the new setting value. This ensures that the new moving direction can replace the old one for each direction setting action. The height setting is similar. In addition, the guard expression of the transition *Start1* guarantees that the start button can be pressed down only when the user has given the correct settings of both direction and height. Finally, a token with a time stamp of 0, carrying a value of a two-tuples, will be put in the place *userCmd*. And then the value will be transmitted to the PLC.

B. Programmable Logic Controllers

The PLC design is important in the whole system design. PLC catches the height and direction values from the control panel at the beginning of its scan cycle. It then calculates the desired cycles that the motor should rotate according to formula (1) and sends one CW/ACW command to let the motor start. There is a counter inside the PLC which counts the number of cycles the motor rotates. When the rotation number of the motor reaches the computed value, PLC issues the STOP command to the motor. By this process, the light should be located at the correct height.

The model of PLC and the corresponding declarations are shown in Fig. 3. The place *userCmd* represents the state that PLC has received a user command from the control panel. *PLCcmd* is used to denote the state that PLC has sent rotating or stopping command to the motor. The place *Getvalue* reflects that PLC has received the feedback signal about the motor movement from the sensor. The place *end* shows the ending of the whole PLC system and the place *counter* shows the state of the counter inside PLC. In addition, the place *goOn* and *FirstTime* help to decide what should do next according to the current status of the PLC system. The four transitions *Calculate*, *SendStop*, *SendCmd*, and *JudgeNext* represent the actions of calculating the number of motor rotating cycles, sending stop command to the motor, sending rotating command to the motor and judging what to do next, respectively.

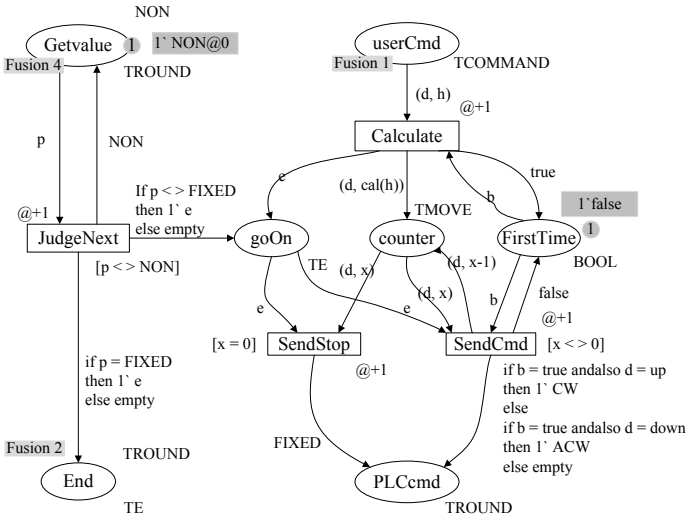

Fig. 3. The TCPN model of the PLC

The corresponding declarations of the PLC model are shown in Fig. 4. The color sets declarations show that each token on the place *PLCcmd* and the place *Getvalue* can has token with the value *CW*, *ACW*, *FIXED*, or *NON*. The first three token values represent clockwise rotation, counter clockwise rotation and stopping of the motor, respectively. The last token value is set specially to represent the initial state of the motor on which there is no rotation. The token color of the place *counter* is a pair which has the first element from DIR and the second element from CIRCLE. The color set CIRCLE contains integer values ranging from 0 to 400. The lower and upper bounds are specified to ensure that the value calculated by formula (1) is within the range. The explanation of other color sets is straightforward.

The execution of the model in Fig. 3 is as follows. Getting user command from the control panel gives a token to the place

```
Color TE = with e timed;
Color CIRCLE =  int with 0..400;
Color ROUND =   with CW | ACW | FIXED |NON;
Color TROUND = ROUND timed;
Color TMOVE =  product DIR * CIRCLE timed;
Var d: DIR;
Var x: CIRCLE;
Var p: ROUND;
Var b, b1, b2: BOOL;
fun cal(h:HEIGHT)  = h*17;
```

Fig. 4. The declarations of the PLC model

userCmd, which starts the PLC to work. Then the transition *Calculate* is enabled and when it fires a token with the token value represented by a pair (d, cal(h)) is generated to the place *counter*. The first element of the pair is just to transmit the direction information. The second element is a function *cal* which defines the relation between the height and the cycles according to formula (1). Function *cal* is defined in the last line of the declarations shown in Fig. 3. In this case, the number of cycles the motor should rotate is calculated and stored in the place *counter*. Each firing of the transition *SendCmd* will make the counter decrease by one. If the value in the place *counter* is not equal to zero, then the transition *SendCmd* will fire, otherwise the transition *SendStop* will fire and generate a token with the value *FIXED* to place *PLCcmd* to stop the motor. The choice is implemented by the exclusive guard on the two transitions. Notice that the CW/ACW command is sent to the motor only once, which is implemented in the model by the existence of place *FirstTime* and the inscriptions on the arc from the transition *SendCmd* to the place *PLCcmd*. Another entrance of this PLC model is when the sensor value is got. This means that there is a token in the place *Getvalue*. The transition *JudgeNext* is then applied to judge whether to finish the execution of the PLC system or go on the further steps.

Now, the time information is considered in the PLC model. Time information is added through making the tokens carry time stamps, adding time delay inscriptions on the transitions or on the arcs. From the declarations in Fig. 3, the tokens of all the places in this model are timed. To declare a timed color set, the keyword *timed* is appended to the declaration of ordinary untimed color sets. And the time delay added to the transitions has the form of @+ *delay-expr*. This means that the action execution consumes a period of time expressed by the expression *delay-expr*. For example, if the calculation action of the PLC needs one time unit to finish, @+*1* will be written as the time delay inscription of the transition *Calculate*. On the other hand, taking @+*0* or just omitting the time delay inscription shows that the time it takes can be neglected. Notice that the time unit is an abstract time, which can be microsecond, millisecond or others. The explanation can be given by the user according to the system in the real world.

The time information of the PLC model is added to analyze the synchronization between PLC and its environment, which will be explained in Section V.

C. Motor

The motor design is a key part in the whole system design, especially for the synchronization problem. It is known from Section III that the motor is requested to keep rotating once it receives a rotating command and will not stop until it receives a stop command. Then the motor will stop rotating and return to the initial state.

The TCPN model of the motor is shown in Fig. 5. No more declarations needed to construct the motor model, so the declarations are not shown in the figure. The place *PLCcmd* represents the state that the motor has received the command from the PLC. The place *Actioned* shows that the motor has finished an action which can be sensed by the sensor. Places named *startF*, *startB* and *beginS* are introduced to show the state that the motor has prepared to clockwise rotating, counter-clockwise rotating and stop, respectively. The place *endRotate* is to ensure that if there is a stop command, the rotating of the motor will stop at once. The transitions *setready1*, *setready2* and *setready3* are used to let the motor prepare to the corresponding actions. The transitions *FRotateOne*, *BRotateOne*, *StopF* and *StopB* represent the actions of clockwise rotating, counter-clockwise rotating, stopping the motor when it is clockwise rotating and stopping it when it is counter-clockwise rotating.

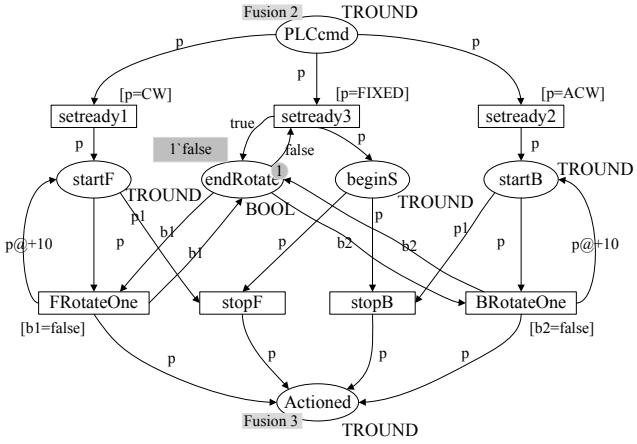

Fig. 5. The TCPN model of the Motor

Suppose there is a token with the value *CW* in the place *PLCcmd*, the transition *setready1* is the only one enabled according to the guard expressions on the transitions *setready1*, *setready2* and *setready3*. Notice that the transition *setready1* can be fired immediately because there is no time delay on it. An initial token is given to the place *endRotate* with the value *false*, so after a token is generated to the place *startF* by firing the transition *setready1*, the transition *FRotateOne* is enabled and can be fired immediately. The existence of the arc from the transition *FRotateOne* to the place *startF* ensures that once the motor starts to rotate, it can keep on rotating arbitrary number of cycles without the need for other commands. The

time delay *@+10* on this arc means that from the time when the transition *FRotateOne* is fired, it needs 10 time units to make the generated token to the place *startF* available. Among these time units, the token is unavailable and cannot fire any transition. This ensures that the time interval between the start of two cycles is 10 time units. In other words, it takes 10 time units that the motor rotates one cycle. Taking this mechanism, the motor model is totally consistent with the desired rotating requests. The case when there is a token with the value *ACW* in the place *PLCcmd* is similar.

Once there is a token with the value *FIXED* on the place *PLCcmd*, the transition *setready3* is enabled. When it fires, the token value of the place *endRotate* is replaced by *true*. This disables the transitions *FRotateOne* and *BRotateOne* according to the guard expressions of the two transitions. This ensures that once receiving the stop command, the motor will stop rotating immediately. In addition, the firing of the transition *StopF* will consume a token from the place *startF* and the transition *StopB* will consume a token from the place *startB*. This assures that when the motor stops after some kinds of rotation, it will not perform any self-motion in the later time unless it receives a rotating command from the PLC again. In other words, once the motor stops, it returns to its initial state.

D. Sensor

The sensor is to capture the signal when the motor rotates one cycle and when it stops. The TCPN model of the sensor is shown in Fig. 6. The place *Sensor* represents the sensor resource in the system and the transition *Capture* expresses the sensing action. Noting the fact that the PLC can only detect the signal from the sensor at the beginning of its scan cycle, only the newest sensor value can be detected. This case is expressed in the senor model: new sensor value will replace the old one. It is implemented by the double arcs between the transition *Capture* and the place *Getvalue* and assigning the initial marking of the place *Getvalue* with a token with the value *NON*. In this way, each firing of the transition *Capture* will consume a token with the old sensing value from the place *GetValue* and generate a token with the new sensing value to the same place. In addition, we specify that the capture action consumes one time unit to finish by adding the time delay inscription on the transition *Capture*.

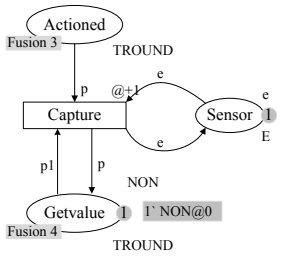

Fig. 6. The TCPN model of the Sensor

E. Model Composition

To get the whole system model, place fusion [16] techniques are applied to bind the four sub models together. Same names have been given for those places which needed to be fused for the four sub models. Through fusing the places named *user-Cmd* into one, the places named *PLCcmd* into one, the places named *Actioned* into one and the places named *Getvalue* into one, the whole model of the system is obtained. The place fusions are shown in Fig. 2 - Fig. 6 by the rectangle attached beside the places to be fused. There are four fusion sets, named by *Fusion1*, *Fusion2*, *Fusion3*, and *Fusion4*, respectively.

F. As a reactive system model

The model can represent the PLC system behaviors for one user command. However, the real case is a reactive system and it can continuously accept user commands and act accordingly. The TCPN model for the reactive system can be easily obtained by returning to the home state after finishing of a single user command. In detail, the system should be reset to the initial state by setting the marking of some places, see the *Reset* model in Fig. 7, where the input place and output places of transition *reset* are fused with the corresponding places in the former sub models.

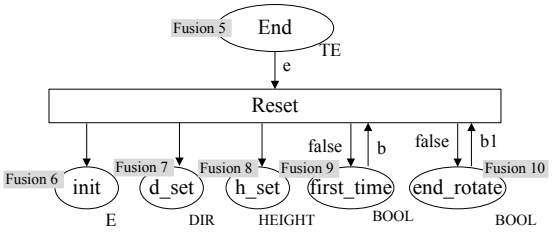

Fig. 7. The TCPN model of the Sensor

G. Modeling in practice

In practice, after system modeling using the proposed method, some original deficient considerations and misunderstandings of the plants can be easily found. For example, for the motor, the request of returning to the initial state after a task execution was not considered in detail at first. When the TCPN model of the motor was constructed, there was no arc from the place *startF* to the transition *stopF* or the arc from the place *startB* to the transition *stopB* in Fig. 5. Through several simulations on the motor model, we find that the motor can self-move in a future time even it stopped with the stop command then. This error is hard to find if the formal model of the plant was not constructed. The correct understanding of no-memory characteristics of the sensor is also obtained from the modeling with the method. In summary, this modeling method can help understand the actions inside the plants as well as the interactions. It can save vast efforts to debug the corresponding errors after PLC programming.

V. Formal Analysis

This section explains how to formally analyze the synchronization problem on the TCPN model and shows the advantage of the proposed analysis method.

A. State Space Analysis

There are three methods which are often used to analyze a Petri-net based model: (1) state space analysis method; (2) simulation method and (3) invariant method [13]. The state space analysis method is taken as our formal analysis method because it can give the complete proof of the dynamic properties of the TCPN model.

The basic idea of the state space analysis method is to construct the occurrence graph which contains a node for each reachable marking and an arc for each occurring binding element. A powerful tool *CPN tools* [17], [18] is adopted to analyze the system modeled by TCPN.

Due to the existence of time evolution, the state space of the TCPN model is infinite. Therefore, the *Reset* sub-model is bypassed to get an acyclic model. In other words, we formally analyze the user command one by one, instead of analyzing the continuously running model directly. Furthermore, to ease the analysis, some other changes will be made on the model. First, since the synchronization is independent of the control panel in this case, the user command is specified as a fixed value: *letting the light down 2 meters*. Thus, the TCPN model of the control panel is modified by removing the initial token from the place *init*, removing the arc from the place *init* to the transition *Start1* and replacing the initial token value of the place d_set by *down*, the place h_set by integer 2. Second, two auxiliary places *countF* and *countB* are added to record the cycles that the motor has clockwise rotated and counter clockwise rotated, respectively. The both places have an initial token with the value of zero. Each time the transition *FRotateOne* fires, the value of the place *countF* increases by 1 and each time the transition *BRotateOne* fires, the value of the place *countB* increases by 1. The modified TCPN model of the Motor is shown in Fig. 8.

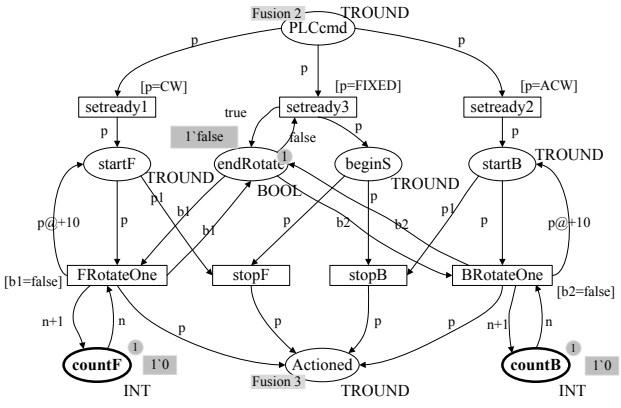

Fig. 8. The TCPN model of the Motor with auxiliary places

After constructing the occurrence graph of the modified TCPN model for the PLC system with *CPN tools*. The analysis result is obtained and shown in Table I.

TABLE I

The general analysis results given by *CPN tools*

State Space		
Nodes number:	145	
Arcs number:	144	
Sections number:	1	
Status:	full	
Best Integer Bounds	Upper	lower
Motor ' Actioned	1	0
Motor ' PLCcmd	1	0
Motor ' beginS	1	0
Motor ' countB	1	1
Motor ' countF	1	1
Motor ' endRotate	1	1
Motor ' startB	1	0
Motor ' startF	0	0
PLC ' End	1	0
PLC ' FirstTime	1	0
PLC ' Getvalue	1	1
PLC ' PLCcmd	1	0
PLC ' counter	1	0
PLC ' goOn	1	0
PLC ' userCmd	1	0
Panel ' d_set	1	0
Panel ' h_set	1	0
Panel ' init	0	0
Panel ' userCmd	1	0
Sensor ' Actioned	1	0
Sensor ' Getvalue	1	1
Sensor ' Sensor	1	1
Dead Markings:	[145]	
Dead Transition Instances:	Motor ' FRotateOne 1 Motor ' StopF 1 Motor ' setready1 1 Panel ' SetDir 1 Panel ' SetH 1	
Live Transition Instances:	None	
Fairness Properties:	No infinite occurrence sequences.	

There are 145 nodes and 144 arcs in the occurrence graph. The full state space was generated for the analysis. From the analysis results of upper bounds and lower bounds of all places in the model, an 1-safe TCPN model is obtained. There is only one dead marking with the node number 145, representing the only deterministic final state. In addition, since the user command is specified on the Control Panel in the modified model, there are five transitions in the model never being enabled. The light control system is non-circular, so there is none live transitions in the model. Finally, the fairness properties are unconcerned since the occurrence graph is finite.

B. Synchronization Analysis

The synchronization problem exists among the actions of the PLC, the Motor and the Sensor. If the system is correctly synchronized, each time the counter inside the PLC decreases by 1, the motor will rotate just one cycle. Then when the system stops in the end, the number of cycles that the motor has rotated should equal to the value computed by formula (1).

some ML functions [17] are coded to check the synchronization of the model. These functions can be checked on the state

TABLE II

ML FUNCTIONS USED TO CHECK SYNCHRONIZATIONS

(1) $ListDeadMarkings()$;
(2) $Mark.Motor'countF\,1\,145$;
(3) $Mark.Motor'countB\,1\,145$;
(4) $PredAllNodes(fn\,n \Rightarrow (ms\_to\_col(Mark.Motor'countB\,1\,n) < 0))$;
(5) $PredAllNodes(fn\,n \Rightarrow (ms\_to\_col(Mark.Motor'countB\,1\,n) > 34))$;
(6) $PredAllNodes(fn\,n \Rightarrow (ms\_to\_col(Mark.Motor'countF\,1\,n) < 0))$;
(7) $PredAllNodes(fn\,n \Rightarrow (ms\_to\_col(Mark.Motor'countF\,1\,n) > 0))$;

space and the results will be returned after the calculations in *CPN tools*. The ML functions and the explanations are given in Table II.

In function (1), *ListDeadMarkings( )* returns a list with all those nodes that are dead, i.e., have no enabled binding elements. For the light control system, this function returns *val it = [145]*. This means that the system is terminable and the only terminal state is the node with the number 145 in the occurrence graph. Then function (2) (3) are applied to check the token values of the places *countF* and *countB* in the state with the number 145, respectively. The result of function (2) is zero and the result of function (3) is 34. This demonstrates that the motor has rotated 34 cycles counter-clockwise and has not any clockwise movement when the system terminated. It implies that the number of the cycles the motor has rotated equals to the value computed by formula (1). In this way, it is verified that the system designed by the TCPN model is correctly synchronized. The conclusion can be further confirmed by Functions (4)-(7). Function (4) is to check all the states on which the place *countB* can have a token with the value less than zero. Functions (5)-(7) are defined in the similar way. Through calculating on the state space of the light control system, all the four functions returns an empty list *val it = [ ]*. This means that for each state during the system running, the number of the cycles the motor has rotated counter clockwise is always not less than zero and not bigger than the maximum value it should rotate. Moreover, the motor never rotates clockwise during this execution.

In the TCPN model, the fact that the system is correctly synchronized is determined by the time information attached to the actions in the system. Now, reduce the time delay that the motor rotates one cycle needs from 10 time units to 2 time units. This might be caused by adopting a new Motor which is four times faster than the original one. The corresponding model is modified by replacing the two appearances of arc inscription *p@+10* with *p@+2* in Fig. 8. Then redo the formal analysis above. The function *ListDeadMarkings( )* returns the value *val it = [217]*. Then the function *Mark.Motor'countB 1 217* returns the value *val it = [35]*. That is to say, when specifying the motor to rotate 34 cycles it rotated 35 cycles instead. Therefore, it shows the wrong synchronization in the new model.

From the two examples of correct synchronization and wrong synchronization above, we can see that taking the TCPN model defined in Section IV, the synchronization problem can be easily analyzed by just several functions. Compared to the other modeling methods on synchronization [9], [10], [11], [12], the method in this paper is more intuitive.

C. Use the analysis result

The advantage of this modeling and analysis method also embodies in that the analysis results can do help in the later equipment choosing and programming phase of the engineering.

For the equipment selection, we assign time delay which are predicted on the TCPN model first, then execute the formal analysis introduced above. If it does not synchronize correctly, this means that, with the given PLC scan cycle and the performance of the equipment, motor for example, the implementation of the control by PLC may get wrong result. Some of the equipments should be replaced to ensure the precise control. we modify the time delays of the plants to be selected, and then analyze the model repeatedly, until the analysis gives the correct result. Therefore, the performance reference values of the equipments are achieved to ensure the correct synchronization and accurate control.

The benefit of our work is the system-level model of the whole PLC system, including the PLC controller and its environment. The system-level model is important for analysis. As to the programming, after the formal analysis and selection of the appropriate plants, the synchronization between PLC and its controlled plants is automatically ensured. Therefore, the PLC implementation can be easily derived reference to the TCPN model of the PLC controller. Take the light control case as example, from the TCPN model in Fig. 3, we can conclude that the PLC program has three inputs: the height, the direction and the sensor value and one output: the control signal to the motor. The calculation of the counter and the judgement can be easily matched to the PLC codes. We'd like to consider the (automatically) code generation work in the future.

In sum, the formal analysis method proposed in this section can give critical information on the function and performance of the designed system. These information can do help to the later phases after the system design.

VI. CONCLUSION

The method of constructing a formal TCPN model for a PLC-based system was presented in this paper. The system properties, such as synchronization, have been formally analyzed. The case studies are given on the light control part in the stage machinery control system developed for the National Grand Theater of China. For the TCPN modeling, we demonstrated how to construct a TCPN model in terms of requirements compositionally. For the properties analysis, we applied the state space analysis method to formally analyze the synchronization problem in the PLC-based system through the time information in the TCPN model. The case studies have shown that the approach is effective in the system modeling and correctness assurance in practical applications.

REFERENCES

[1] R. Lewis, *Programming industrial control systems using IEC 1131-3,volume 50 of Control Engineering Series*. Stevenage, United Kingdom: The Institution of Electrical Engineers, 1998.

[2] P. M. F. Bonfatti and U. Sampieri, *IEC 1131-3 Programming Methodology*. Fontaine, France: CJ International, 1999.
[3] M. Rausch and B. H. Krogh, "Formal verification of PLC programs," in *Proceedings of American Control Conference 1998.*, vol. 1, pp. 234–238, 1998.
[4] T. Mertke and G. Frey, "Formal verification of PLC-programs generated from signal interpreted Petri nets.," in *Proceedings of the SMC 2001, Tucson (AZ) USA*, pp. 2700–2705, Oct. 2001.
[5] G. Canet, S. Couffin, J. J. Lesage, A. Petit, and P. Schnoebelen, "Towards the automatic verification of PLC programs written in instruction list," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, pp. 2449–2454, 2000.
[6] M. Heiner and T. Menzel, "A Petri net semantics for the PLC language instruction list.," in *Proc. IEE Workshop on Discrete Event Systems (WODES98), Cagliari/Italy*, pp. 161–165, Aug. 1998.
[7] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
[8] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.
[9] M. Moalla, J. Pulou, and J. Sifakis, "Synchronized Petri nets: A model for the description of non-autonomous systems.," *Lecture Notes in Computer Science: Mathematical Foundations of Computer Science 1978*, vol. 64, pp. 374–384, Sept. 1978.
[10] R. David and H. Alla, *Petri Nets and Grafcet - Tools for Modeling Discrete Event Systems*. New York, London: Prentice Hall, 1992.
[11] G. Frey, "PLC programming for hybrid systems via signal interpreted Petri nets.," in *Proceedings of the 4th International Conference on Automation of Mixed Processes ADPM, Dortmund, Germany*, pp. 189–194, Sept. 2000.
[12] H. M. Hanisch, J. Thieme, A. Luder, and O. Wienhold, "Modeling of PLC behavior by means of timed net condition/event systems," in *Emerging Technologies and Factory Automation Proceedings, 1997. ETFA '97.*, pp. 391–396, 1997.
[13] K. Jensen, *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Vol. 2: Analysis Methods*. Berlin: EATCS Monographs on Theoretical Computer Science,Springer-Verlag, 1994.
[14] S. Christensen and L. Petrucci, "Towards a modular analysis of coloured Petri nets.," in *Lecture Notes in Computer Science; 13th International Conference on Application and Theory of Petri Nets* (Jensen, K., ed.), vol. 616, pp. 113–133, Springer-Verlag, June 1992.
[15] L. Gomes and P. B. Joao, "Structuring and composability issues in Petri nets modeling," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 2, pp. 112– 123, 2005.
[16] W. Zuberek and I. Bluemke, "Hierarchies of place/transitions refinements in Petri nets," in *Proceedings of the Conference on Emerging on Technologies and Factory Automation*, pp. 355–360, 1996.
[17] A. V. Ratzer, L. Wells, and H. M. Lassen, "CPN tools for editing, simulating, and analysing coloured Petri nets.," in *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN 2003)*, pp. 450–462, Springer-Verlag, June 2003.
[18] "Cpn tools: `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`."