MODELING AND SIMULATION OF PETRI NETS FOR COMPLEX SCHEDULING RULES OF AUTOMATED MANUFACTURING SYSTEMS

Chulhan Kim and Tae-Eog Lee

Department of Industrial and Systems Engineering
KAIST
291 Daehak-ro, Yuseong-gu, Daejeon 305-701, South Korea

E-mail: chulhan.kim@kaist.ac.kr, telee@kaist.ac.kr

ABSTRACT
Discrete event systems such as automated manufacturing systems and engineering systems can be modeled and simulated by Petri nets. Precedence relations between activities or events, concurrent processes, synchronization, resource sharing, mutual exclusion, etc can be well modeled by Petri nets. However, discrete event systems, especially discrete event manufacturing systems, tend to have diverse complex scheduling rules to optimally utilize the resources, meet scheduling requirements and constraints, and optimize the performance measures such as makespan or cycle time. In this paper, we propose ways of modeling such complex scheduling rules by controlling the firing sequences and timing of the associated transitions in the Petri net model. We also present a Petri net model for scheduling a robotized indexer cell for flat panel display manufacturing.

Keywords: Petri net, scheduling, simulation, indexer

1. INTRODUCTION

A Petri net is a graphical and mathematical modeling framework for discrete event systems (Murata 1989). Discrete event systems such as automated manufacturing systems and engineering systems can be modeled and simulated by Petri nets. Transitions, places, arcs, and tokens represent activities or events, conditions or resources, precedence relations between transitions and places, and availability of resources or conditions, respectively. They are graphically represented by rectangles, circles, arrows, and dots, respectively. A more formal definition including transition enabling and firing rules can be found in Murata (1989). Precedence relations between activities or events, concurrent processes, synchronization, resource sharing, mutual exclusion, etc can be well modeled by Petri nets.

Scheduling is to determine the order and timings of processing jobs at a resource. The processing order may be fixed and independent of the system state such as the number of jobs at each resource, or can be dynamically changed depending on the system state. When there are multiple resources that can process a job, a resource should be selected to process the job. Discrete event systems, especially discrete event manufacturing systems, tend to have diverse complex scheduling rules to optimally utilize the resources, meet scheduling requirements and constraints, and optimize the performance measures such as makespan or cycle time.

In a Petri net model, resources and activities that contend for using a resource can be modeled by a place and its output transitions, respectively. The tokens at such a conflict place are properly routed to the output transitions. Therefore, we expect that complex scheduling rules can be modeled by token routing rules, which can depend on the system state information such as the number of tokens and the token sojourn times at the places. However, it is not so straightforward to represent complex scheduling rules by token routing rules. For instance, even a simple state-independent cyclic scheduling rule that processes jobs at a resource in a fixed cyclic order is not simply modeled by routing tokens at a conflict place cyclically. It is because a transition firing for processing a job may be delayed due to delayed arrivals of tokens at the other input places of the transition. Furthermore, there are not enough studies on modeling and simulating scheduling rules in Petri nets. Most Petri net simulators can model decision-free nets that have no scheduling decisions or conflict places, or nets that have probabilistic token routing rules. There are few works on Petri net models or simulators that can model complex token routing rules or scheduling rules effectively.

In this paper, we propose ways of modeling such complex scheduling rules by controlling the firing sequences and timings of the associated transitions in the Petri net model. As an application, we also present a Petri net model for scheduling a robotized indexer cell for flat panel display manufacturing.

2. FIRING POLICIES OF PETRI NETS

We explain policies for controlling firings of transitions in a Petri net. Consider an example of Petri net in Figure 1. Places p1 and p4 are conflict places, each of which has two output transitions. A token routing rule basically determines an output transition to be fired at

each conflict place. A token that entered a conflict place and has stayed there as long as the token holding time of the place, if any, can be released to one of the output transitions of the conflict place. The token released to the output transition can join enabling the transition. The release can be made as soon as the token is ready to be released or delayed. An enabled transition can be fired immediately or after a prescribed firing delay of the transition, if any. It also can be delayed. Therefore, the token routing, delays of token releases and transition firings can be controlled by modeling and simulating scheduling rules. A *token routing rule* indicates such token route and release timing. A token routing rule eventually determines the order and timings in which the output transitions are fired. In the sense, it can be told that the scheduling decisions are made by a *transition firing policy* that determines the order and timings of firing the output transitions of each conflict place. When some intentional delays are made in scheduling decisions, but most practical scheduling rules for manufacturing systems start an activity or job as soon as possible. In other words, the token release and transition firings are not intentionally delayed. Therefore, in this paper, we focus on controlling only the order of transition firings. Of course, job release control rules such as kanban or CONWIP(Constant Work in Progress) intentionally delay jobs or activities. Therefore, such scheduling rules might be modeled by token routing rules or transition firing policies that make intentional delays in token release or transition firing appropriately. However, even such scheduling rules for timing regulation can be incorporated into Petri net model as an appropriate subnet that provides feedback of tokens from some transitions to other transitions (Lee and Park 2005; Mascolo, Frein, Dallery and David 1991). We therefore define four types of transition firing policies that determine only the order of transition firings.

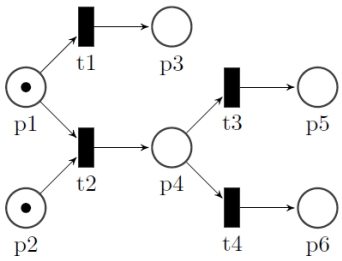

Figure 1: A Petri net which has two conflict places.

2.1. Probabilistic sequence

In a probabilistic sequence, the output transitions of a conflict place are fired in a random order. For example, in Figure 1, output transitions t1 and t2 of the place p1 are fired with the same probability, 0.5. Of course, the probabilities may be unequal. Probabilistic sequence can be used for describing systems of which the order of activities is not important and systems with decision makers that behave randomly.

2.2. Cyclic sequence

A cyclic sequence is a fixed sequence which repeats firing the transitions based on a specified cycle. We use the following expression for a cycle:

$$C(S_1, S_2, \dots, S_n)$$

where S_i is ith firing transition and n is the length of the order list. For example, a cycle $C(t_a, t_b, t_b)$ repeats transition firing by an order of $(t_a \rightarrow t_b \rightarrow t_b \rightarrow t_a \rightarrow t_b \rightarrow t_b \rightarrow t_a \rightarrow t_b \rightarrow t_b \rightarrow \cdots)$.

A cycle is *feasible* if every transition can be fired continuously by the order of it. The Petri net in Figure 1 has $4! = 24$ possible cycles but there is no feasible cycle. On the other hand, the Petri net in Figure 2 has $2! = 2$ cycles and $C(t_1, t_2)$ is feasible.

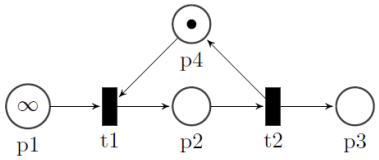

Figure 2: A Petri net which has a feasible cycle, $C(t1, t2)$.

2.2.1. K-cyclic sequence

A k-cyclic sequence is a special case of the cyclic sequence. It selects a firing transition based on a specified cycle which contains every transition of the net. Transitions are fired by the order of cycle and every transition is fired k times in a cycle. For example, 2-cyclic sequence $C(t_a, t_c, t_b, t_b, t_c, t_a)$ means the transitions are fired by an order of $(t_a \rightarrow t_c \rightarrow t_b \rightarrow t_b \rightarrow t_c \rightarrow t_a \rightarrow t_a \rightarrow t_c \rightarrow \cdots)$.

Cyclic sequence is useful to describe systems in which the several events are repeated by a cyclic order. In case of the k-cyclic sequence, since every transition should be fired k times in a cycle and it is independent of the marking of Petri nets, the problem complexity is much more reduced. Hence, much research has been done for finding optimal cycles to maximize/minimize the value/cost using mathematical programming techniques.

2.3. Non-cyclic sequence

Some decision makers such as distribution machines in manufacturing cells may behave without any pattern. A non-cyclic sequence can be used to describe this kind of non-repeating firing orders. A list is described as

$$N(T_1, T_2, \dots, T_m).$$

The only difference from the order list of cyclic sequence is that it does not fire any transition after firing the last one in the list, regardless of the existence of enabled transitions. In most cases it leads to the end of simulation.

2.4. Rule

Probabilistic, cyclic, and non-cyclic sequences select a firing transition based on random numbers or a pre-specified order list which means that they do not care about the marking of Petri nets. In contrast, rule policy is marking-dependent. It means that we can use rule policy when we describe systems with decision makers that depend on the state of the systems.

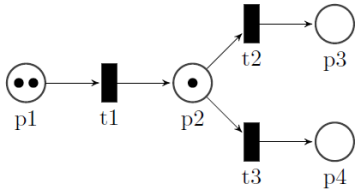

Figure 3: A Petri net.

Table 1: Two rules for a Petri net in Figure 3.

Rule	Statements
#1	If $M(p2) = 0$, then fire t1. Else if $M(p3) \geq 1$, then fire t3. Else, fire t2.
#2	If $M(p3) + M(p4) \leq 1$, then fire t1. Else if $M(p2) \geq 2$, then fire t2. Else, fire t3.

Basically, a *rule* is a series of *if-then* statements. Each statement follows a format of "*if* condition C is satisfied, *then fire* transition T". Every time a transition is fired, the rule is considered to select a next firing transition. Table 1 shows two sample rules for a Petri net in Figure 3 where $M(p)$ is the number of tokens in place p. We do not state the enabling condition of transition in a rule for simplicity. For example, the first statement of Rule #1 in Table 1 actually means "If t1 is enabled and $M(p2) = 0$, fire t1.". The firing order based on Rule #1 and Rule #2 in Table 1 are $(t2 \rightarrow t1 \rightarrow t3 \rightarrow t1 \rightarrow t3)$ and $(t1 \rightarrow t1 \rightarrow t2 \rightarrow t2 \rightarrow t3)$, respectively.

Using probabilistic, cyclic, non-cyclic sequences and rule policies properly, we can model diverse types of discrete event systems and simulate them more effectively.

3. GLOBAL AND LOCAL FIRING POLICY

In section 2, we defined four firing policies to describe various discrete event systems with Petri nets. In this section, we discuss the meaning of global firing policy, local policy, and the difference between adapting a single global firing policy and several local firing policies to a Petri net.

3.1. Global firing policy

A *global firing policy* is a firing policy that deals with every transition of a Petri net. For example, the global cycle policy should include all transitions of a Petri net in its firing order list. We use a single global firing policy for a Petri net to model a system with a single decision maker that controls the whole system.

3.2. Local firing policy

Some systems may have several independent decision makers like a manufacturing cell with two independent transportation robots. They act without considering other decision makers. Describing this kind of system with a global firing policy may not be easy. In this case, we use several *local firing policies* for each *conflict set* in a Petri net.

Definition: *A conflict set of a Petri net is a subnet which includes two nonempty sets P_c and T_c, where*

1. *P_c is the smallest set of places such that $\bigcup_{p_i \in P_c}(p_i \bullet) = T_c$ where $p_i \bullet$ is the set of output transitions of p_i.*
2. *T_c is the smallest set of transitions such that $\bigcup_{t_i \in T}(\bullet t_i) = P_c$ where $\bullet t_i$ is the set of input places of t_i.*

For example, in Figure 1, there are two conflict sets. The first one is a subnet including p1, p2, t1, and t2. Second one consists of p4, t3, and t4. There is no conflict between every pair of conflict sets. It means that firing of a conflict set does not disable transitions in the other conflict sets. That is, every conflict set works *concurrently*. In case of the Petri net in Figure 1, we need to specify two firing policies for each conflict set to describe the system with two independent decision makers.

4. SIMPN

There have been already a lot of free and commercial Petri net tools/simulators available. However, they are not well suited for modeling and simulation of discrete event systems which behave based on complex operating schedules with them because most of them deal with conflict situations randomly or by asking the users to select one of enabled firing transitions. Based on firing policies discussed in section 2 and 3, we developed a java-based p+-time simulator, *SIMPN*. Figure 4 illustrates the main frame of SIMPN.

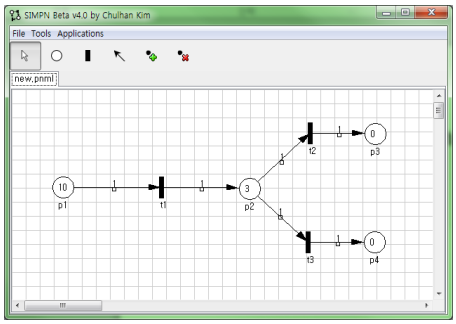

Figure 4: SIMPN

P+-time Petri net is an extension of a timed places Petri net (Wang 1998) whose places have random token holding time and maximum token delay. It is formally defined by Kim and Lee (2008) to be used for modeling cluster tools with non-deterministic processing time and time window constraints. P+-time Petri net is a useful

extension of the Petri net that can describe various types of discrete event systems especially manufacturing systems.

4.1. Simulator

The simulator of SIMPN supports Petri net simulations with a global firing policy and local policies. Basically, a simulation engine works based on the discrete event simulation framework. It controls an event list which prioritizes the events according to the times of event occurrences.

The users should specify firing policies for the Petri net to simulate it. In case of simulation with local firing policies, SIMPN automatically finds all conflict sets of the Petri net.

4.2. Statistical analysis

The users can make the simulator gather data from simulation and analyze the results. The following is a list of possible statistical analysis from SIMPN:

- Average token inter-arrival time of a place,
- Time interval between two firing epochs of transitions, and
- Gantt chart of token holding time of a place.

For example, in a manufacturing line, first option can be used when we want to know the average time interval between two consecutive finished products (cycle time). Second option can be used to measure the whole processing time of products (turnaround time). Lastly, we can make SIMPN draw a Gantt chart for a place to see overall status of processing chamber and to check patterns.

5. APPLICATION: LCD INDEXERS

An LCD indexer is a machine for sorting, loading and unloading glasses in the LCD panel manufacturing. Robot arms are responsible for transportation of glasses between two processing chambers. Therefore, the behavior of the robot arms is closely related to the overall performance or productivity of the LCD indexer.

LCD indexers are quite similar with cluster tools for the semiconductor manufacturing. A cluster tool combines several single-wafer processing modules with wafer handling robots in a closed environment (Lee 2008). There is much research for evaluating the performance of cluster tools (Chan, Yi and Ding 2010; Dwande, Geismar, Sethi and Sriskandarajah 2007) and finding optimal robot arm schedules, especially marking-independent cyclic schedules using timed Petri nets and mathematical programming techniques (Jung 2010).

However, finding optimal robot behavior of an LCD indexer is not simple due to some differences from cluster tools. First, we cannot assume processing time as deterministic parameters because the variance is too large to ignore. Second, some chambers may have time window constraints. That is, if a glass stays too long in a chamber with time window constraint, it becomes defective due to heat or hazardous gases. Similar cases for cluster tools are introduced in (Kim, Lee, Lee and Park 2003; Lee and Park 2005; Kim and Lee 2008). However, they are about analyzing schedulability analysis of timed event graphs which are special cases of timed Petri nets. Third, some modules can process several glasses at once. Batch process tends to make the problem much complicated. Last main difference is that processing order of some abnormal glasses may not be the same as that of the other glasses.

For LCD indexers, we cannot get an optimal operating schedule using mathematical programming techniques because of their stochastic behavior and high complexity. Simulation approach can be used to find good schedules for them. We introduce a case of finding robust and efficient *dispatching rules* of an LCD indexer, annealing oven line.

5.1. Annealing oven line

An annealing oven line is a type of LCD indexer which anneals glasses with heat and cools them. It consists of one or more ovens, coolers and a robot arm. A single-arm robot is responsible for transportation of glasses. Figure 5 illustrates the annealing oven line that we deal with.

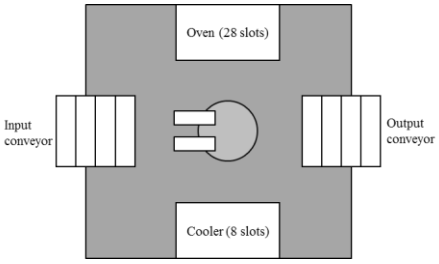

Figure 5: Annealing oven line

Normal glasses are unloaded from the input conveyor, visit oven, cooler and output conveyor, sequentially. On the other hand, abnormal glasses are not processed in the oven and cooler. They just move from the input conveyor to the output conveyor.

We assume no batch process in the system and 1% of total glasses are abnormal. Minimum and maximum processing times of the oven and cooler are 200, 300, 150 and 200 seconds, respectively. Due to hot environment, a glass should be unloaded from the oven within 100 seconds after it is processed. If a finished glass is not unloaded within the specified time window constraint, severe quality problems occur. Times for unloading/loading and transportation are all taken to be 1.

Figure 6 shows a p+-time Petri net model of the system and the meaning of each place and transition is explained in Table 2. In addition, Table 3 shows the time information of each place.

5.2. Deadlock prevention

There are two types of deadlock in this problem. First, deadlock occurs when the robot tries to load a glass into

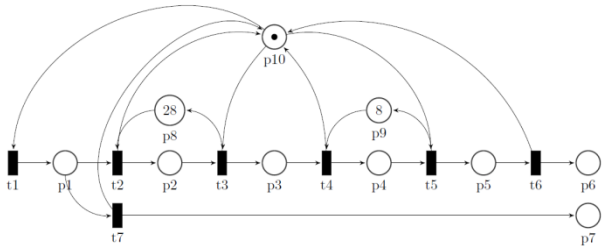

Figure 6: A p+-time Petri net model of annealing oven line in Figure 5.

Table 2: Legend for Figure 6.

Name	Meaning
p1	Robot moves a glass from input conveyor to oven.
p2	Oven is processing.
p3	Robot moves a glass from oven to cooler.
p4	Cooler is processing.
p5	Robot moves a glass from cooler to output conveyor.
p6	Finished normal glasses.
p7	Finished abnormal glasses.
p8	Availability of oven.
p9	Availability of cooler.
p10	Availability of robot arm.
t1	Unloading a glass from the input conveyor.
t2	Loading a normal glass to the oven.
t3	Unloading a normal glass from the oven.
t4	Loading a normal glass to the cooler.
t5	Unloading a normal glass from the cooler
t6	Loading a normal glass to the output conveyor.
t7	Loading an abnormal glass to the output conveyor.

Table 3: Time information of each place in Figure 6.

Place	Holding time	Time window constraint
p1	2	∞
p2	(200, 300)	100
p3	2	∞
p4	(150, 200)	∞
p5	2	∞
p6	0	∞
p7	0	∞
p8	0	∞
p9	0	∞
p10	2	∞

the oven or the cooler without an empty slot. We can prevent this kind of deadlock by defining conditions in firing rules as follows:

1. If $M(p8) \geq 1$, fire t1, and
2. If $M(p9) \geq 1$, fire t3.

Second possible deadlock is due to the time constraint of the oven. If a glass is not unloaded within 100 seconds after processing, the oven stops working. Since we cannot directly control the processing times of the oven and cooler, this type of deadlock is almost impossible to be prevented by defining conditions. A good way to avoid the problem is changing the availabilities of the oven and the cooler on purpose. In this case, there are too many slots in the oven, so the cooler may not be able to handle coming glasses fast enough. Therefore, reducing the number of usable slots in the oven can help prevent deadlock.

5.3. Defining firing rules

Now we define firing rules for two conflict sets which have more than one output transition. For the first conflict set, we have to define firing rules for selecting a firing transition between t2 and t7. Since 1% of glasses are abnormal, rules can be set up as follows:

If RND ≤ 0.01, fire t7;
Otherwise, fire t2,

where RND is a random number between 0 and 1.

The other conflict set has transitions t1, t3, and t5. This conflict set represents the behavior of the robot. Firing policies for the robot are directly related to the performance of this anneal oven line. For a simple experiment, we define two firing rules in Table 4. As the rules in Table 1, we omit the conditions for the enablement of transitions. For example, first statement in Rule #1 means "If t5 is enabled, fire t5.

Table 4: Two rules for a Petri net in Figure 6.

Rule	Statements
#1	Fire t5. Else if $M(p9) \geq 1$, then fire t3. Else if $M(p8) \geq 1$, fire t1.
#2	If $M(p8) \geq 1$, fire t1. Else if $M(p9) \geq 1$, then fire t3. Else, fire t5.

Briefly, a robot with Rule #1 always unloads a glass from the cooler whenever its cooling process is done, while a robot with Rule #2 always loads a glass to the empty oven slot right after it becomes available. The only difference between two rules is the priority of statements.

5.4. Simulation

Simulation experiments are conducted with SIMPN. First, we find the maximum number of available oven slots while avoiding deadlock. It can be simply found by doing simulation reducing the availability of oven one by one until no deadlock occurs during the simulation.

After finding the availability, we obtain the average cycle time of normal glasses. In this case, we can get this result by analyzing the average inter-arrival time of tokens in p6.

Simulation results with simulation time 100,000 are in Table 5. In case of Rule #1, even though all the oven slots are available, the average cycle time of

glasses is much greater than the case of Rule #2 which has 9 available slots. That is because it uses only one oven slot and cooler slot at once. In summary, Rule #1 guarantees no deadlock, but the efficiency is limited. On the other hand, Rule #2 ensures the good average cycle time even though it may have to disable some oven slots on purpose.

Table 5: Simulation results

Rule	Max availability	Average cycle time
#1	28	434.51
#2	9	28.86

6. CONCLUSION

We defined firing policies for Petri nets: probabilistic, cyclic, non-cyclic sequences and rule policy. These firing policies can be used to model complex and dynamic schedules of the system. One global firing policy is defined for a Petri net if there is only one decision maker. Several local firing policies can be also adapted to a Petri net in order to describe systems with multiple decision makers working independently. Petri nets with dynamic firing policies cannot be modeled mathematically in most cases. Simulation approach is valuable for efficient operating schedules for this kind of systems. We developed SIMPN which is a java-based p+-time Petri net simulator with firing policies. We conducted simulation for an annealing oven line for a case study. Experiment results showed the productivity of the system to be highly dependent upon its operating rules.

For further study, firing policies can be more generalized and well-organized for more general schedule. In addition, we have to specify a general framework for finding good firing policies especially marking-dependent firing rules.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0011438).

REFERENCES

Chan, W.K.V., Yi, J. and Ding, S., 2010. Optimal scheduling of multicluster tools with constant robot moving times, Part I: Two-cluster analysis. *IEEE transactions on automation science and engineering*, 8 (1), 5–16.

Dawande, M.W., Geismar, H.N., Sethi, S.P. and Sriskandarajah, C. , 2007. *Throughput optimization in robotic cells*. USA: Springer Science + Business Media, LLC.

Jung, C., 2010. *Cyclic scheduling of timed Petri nets: Behavior, optimization, and application to cluster tools*. Thesis (Ph. D.). KAIST.

Kim, J.H., Lee, T.E, Lee, H.W and Park, D.B, 2003. Scheduling analysis of time-constrained dual-armed cluster tools. *IEEE transactions on semiconductor manufacturing*, 16 (3), 521–534.

Kim, J.H. and Lee, T.E., 2008. Schedulability analysis of time-constrained cluster tools with bounded time variation by an extended Petri net. *IEEE transactions on automation science and engineering*, 5 (3), 490–503.

Lee, T.E. and Park, S.H., 2005. An extended event graph with negative places and tokens for time window constraints. *IEEE transactions on automation science and engineering*, 2 (4), 319–332.

Lee, T.E., 2008. A review of scheduling theory and methods for semiconductor manufacturing cluster tools. *Proceedings of the 2008 winter simulation conference*, 2127-2135. December 7-10, Miami, F.L., USA.

Mascolo, M., Frein, Y., Dallery, Y. and David, R., 1991. A unified modeling of Kanban systems using Petri nets. *International journal of flexible manufacturing systems*, 3 (3-4), 275–307.

Murata, T., 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77 (4), 541–580.

Wang, J., 1998. *Timed Petri nets: Theory and application*. USA: Kluwer academic publishers.

AUTHORS BIOGRAPHIES

CHULHAN KIM received the B.S. degree in Korea Advanced Institute of Science and Technology (KAIST) in 2010. He is in integrated master's and Ph. D. program in the same university. His research interests focus on simulation of Petri nets, and cyclic scheduling with Petri nets using mathematical programming techniques.

TAE-EOG LEE is a Professor with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST). He is also the head of the department. His research interests include cyclic scheduling theory, scheduling and control theory of timed discrete-event dynamic systems, and their application to scheduling and control of automated manufacturing systems.