PETRI NETS AND THEIR APPLICATIONS

Petri nets (*PN*) were named after Carl A. Petri, who created a netlike mathematical tool for the study of communication with automata in 1962. Their further development stemmed from the need to specify process synchronization, asynchronous events, concurrent operations, and conflicts or resource sharing for a variety of industrial automated systems at the discrete-event level (1–3). Computer scientists performed most of the early studies. Starting in the late 1970s, researchers with engineering backgrounds, particularly in manufacturing automation, investigated Petri nets' possible usage in human-made systems. These systems have become so complicated that the continuous/discrete-time systems theory has become insufficient to handle them.

In any physical net, we can find two basic elements: nodes and links. Both nodes and links play their own roles. For example, forces could be transferred from one end to another through nodes and links; and different nodes/links may bear different forces. A PN divides nodes into two kinds: places and transitions. The places are used to represent the condition and/or status of a component in a system and are pictured by circles. The transitions represent the events and/or operations and are pictured by empty rectangles or solid bars. Two common events are "start" and "end." Instead of bidirectional links in some physical nets, a PN utilizes directed arcs to connect from places (called input places with respect to a transition) to transitions or from transitions to places (called output places). In other words, the information transfer from a place to a transition or from a transition to a place is one way. Two-way transfer between a place and transition is achieved by designing an arc from a transition to a place and another arc from the transition back to the place.

The places, transitions, and directed arcs make a PN a directed graph, called a Petri net structure. It is used to model a system's structure. A system state is defined by the location of "state markers" in the places of a PN. These state markers are called "tokens" for short. The dynamics are introduced by allowing a place to hold either none or a positive number of tokens pictured by small solid dots. These tokens could represent the number of resources or indicate whether a condition is true or not in a place. When all the input places hold enough tokens, an event embedded in a transition can happen, called transition firing. This firing changes the token distribution in the places, signifying the change of system states.

The introduction of tokens and their flow regulated through transitions allow one to visualize the material, control, and/or information flow clearly. Furthermore, one can perform a formal check of the properties related to the underlying system's behavior (e.g., precedence relations among events, concurrent operations, appropriate synchronization, freedom from deadlock, repetitive activities, and mutual exclusion of shared resources).

PNs belong to state-transition models. The simplest state-transition model is a state machine. Its graphical representation is a state diagram. In a state diagram, a node pictured as a circle represents a state that characterizes the conditions of all the components in a system at a time instant. An arc represents an event that changes one state to another. Note that in a PN, a transition represents an event and an arc represents information, control or material flow. For example, in a robotic assembly system, the initial state is that a robot is ready to pick up a component and a component is ready for pick-up. Then the event "start a pick-up operation" brings the system into the state "the robot holds a component" or "the robot fails to pick it up." At the new state, the next event can occur. State-machine models are suitable when a system has few active components, such as a robot or machine, or needs only a few states to describe. A single robot may need two states, "idle" and "busy," and a dual-robot system requires four states. However, a 20-robot system requires over a million states. Clearly the number of states grows exponentially with the system size. It is difficult to represent systems with an infinite number of states. The states, events, precedence relations, and conflicting situations are explicitly represented, but synchronization concepts, concurrent operations, and mutually exclusive relations are not explicitly represented.

A formalism that can overcome some of the limitations of state-machine modeling is desired to handle complex systems. It should use local states rather than global states, thereby avoiding the state enumeration problems in the modeling/design stage. It can explicitly represent conditions and events together, precedence relations, conflicting situations, synchronization concepts, concurrent and repetitive operations, and mutually exclusive relations. PNs are such formalism.

FORMAL DEFINITION

A *marked Petri net* (PN) $Z = (P, T, I, O, m)$ is a five tuple where

1. $P = \{p_1, p_2, \ldots, p_n\}$, $n > 0$, is a finite set of places pictured by circles;
2. $T = \{t_1, t_2, \ldots, t_s\}$, $s > 0$, is a finite set of transitions pictured by bars, with $P \cup T \neq \emptyset$ and $P \cap T \neq \emptyset$;
3. $I: P \times T \rightarrow N$ is an input function that defines the set of directed arcs from P to T, where $N = \{0, 1, 2, \ldots\}$;
4. $O: P \times T \rightarrow N$ is an output function that defines the set of directed arcs from T to P;
5. $m: P \rightarrow N$ is a marking whose ith component represents the number of tokens in the ith place. An initial marking is denoted by m_0. Tokens are pictured by dots.

The four tuple (P, T, I, O) is called a PN structure that defines a directed graph structure. Introduction of tokens into places and their flow through transitions enable one to describe and study the discrete-event dynamic behavior of the PN, and thereby the modeled system. I and O represent two $n \times s$ nonnegative integer matrices. An incidence matrix $C = O - I$. A PN can alternatively be defined as (P, T, F, W, m), where F is a subset of $\{P \times T\} \cup \{T \times P\}$, representing a set of all arcs, and $W: F \rightarrow N$ defines the multiplicity of arcs.

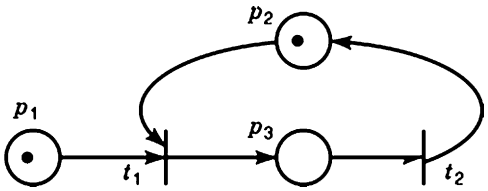


Figure 1. A Petri net example modeling a robot that picks a component and place.

A marked PN shown in Fig. 1 and its formal description are given as follows:

$$P = \{p_1, p_2, p_3\}, T = \{t_1, t_2\}$$
$$I(p_1,t_1) = 1, I(p_1,t_2) = 0;\ O(p_1,t_1) = 0,\ O(p_1,t_2) = 0;$$
$$I(p_2,t_1) = 1, I(p_2,t_2) = 0;\ O(p_2,t_1) = 0,\ O(p_2,t_2) = 1;$$
$$I(p_3,t_1) = 0, I(p_3,t_2) = 1;\ O(p_3,t_1) = 1,\ O(p_3,t_2) = 0;$$
$$m = (110)^{\tau}$$

Input and output functions can be represented as matrices; that is,

$$I = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } O = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The incidence matrix

$$C = O - I = \begin{bmatrix} -1 & 0 \\ -1 & 1 \\ 1 & -1 \end{bmatrix}$$

The execution rules of a PN include enabling and firing rules:

1. A transition $t \in T$ is *enabled* if and only if $m(p) > I(p, t), \forall\, p \in P$.
2. Enabled in a marking m, t *fires* and results in a new marking m' following the rule

$$m'(p) = m(p) - I(p,t) + O(p,t), \forall\, p \in P$$

Marking m' is said to be (immediately) reachable from m. The enabling rule states that if all the input places of transition t have enough tokens, then t is enabled. This means that if the conditions associated with the occurrence of an event are all satisfied, then the event can occur. The firing rule says that an enabled transition t fires and its firing removes $I(p,t)$ tokens from p, and then deposits $O(p,t)$ tokens into p.

In Fig. 2(a), transition t_1 is enabled since $m(p_1) = 1 = I(p_1, t_1)$ and $m(p_2) = 1 = I(p_2, t_1)$; t_2 is not since $m(p_3) = 0 < 1 = I(p_3, t_2)$. Firing t_1 removes one token from p_1 and p_2, respectively, and deposits one token to t_1's only output place p_3. The result is shown in Fig. 2(b). The new marking is $m' = (0\ 0\ 1)^{\tau}$. At m', t_2 is enabled since $m'(p_3) = 1 = I(p_3, t_2)$. Firing t_2 results in $m'' = (0\ 1\ 0)^{\tau}$. At this marking no transition is enabled. Note that t_1 is not since $m''(p_1) = 0 < 1 = I(p_1, t_1)$ although $m''(p_2) = 1 = I(p_2, t_1)$.

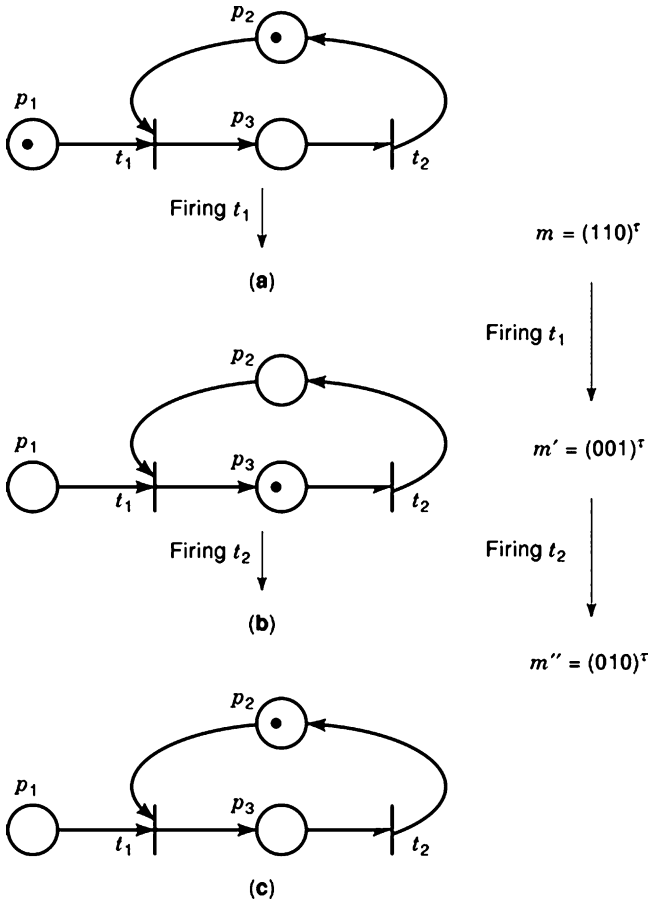


Figure 2. Evolution of markings of a PN: (a) $m = (1\ 1\ 0)^{\tau}$, (b) $m' = (0\ 0\ 1)^{\tau}$, and (c) $m'' = (0\ 1\ 1)^{\tau}$. This illustrates the state change of the modeled system.

PROPERTIES OF PETRI NETS AND THEIR IMPLICATIONS

PNs as a mathematical tool have a number of properties. These properties, when interpreted in the context of the modeled manufacturing system, allow one to identify the presence or absence of the functional properties of the system. Two types of properties can be distinguished: behavioral and structural. The behavioral properties are those that depend on the initial state or marking of a PN. The structural properties, on the other hand, do not depend on the initial marking of a PN. They depend on the Petri net topology or structure.

Reachability

Given a PN $Z = (P, T, I, O, m_0)$, marking m is *reachable* from marking m_0 if there exists a sequence of transitions firings that transforms m_0 to m. Marking m' is said to be immediately reachable from m if firing an enabled transition in m leads to m'. R is used to represent the set of all reachable markings. Reachability checks whether the system can reach a specific state or exhibit particular functional behavior.

Boundedness and Safeness

Given a PN Z and its reachability set R, a place $p \in P$ is *B-bounded* if $m(p) \leq B, \forall\, m \in R$, where B is a positive integer. Z is *B-bounded* if each phase in P bounded. *Safeness* is 1-boundedness. Z is *structurally bounded* if Z is B-bounded for some B, for any finite initial marking m_0.

Places are frequently used to represent storage areas for parts, tools, pallets, and automated guided vehicles in manufacturing systems. Boundedness is used to identify the absence of overflows in the modeled system. When a place models an operation, its safeness guarantees that the controller does not attempt to initiate an ongoing process. The concept of boundedness is often interpreted as stability of a discrete manufacturing system, particularly when it is modeled as a queuing system.

Liveness

A transition t is *live* if at any marking $m \in R$ there is a sequence of transitions whose firing reaches a marking that enables t. A PN is live if every transition in it is live.

A transition t is *dead* if there is $m \in R$ such that there is no sequence of transition firings to enable t starting from m. A PN contains a *deadlock* if there is $m \in R$ at which no transition is enabled. Such a marking is called a dead marking.

Deadlock situations are a result of inappropriate resource allocation policies or exhaustive use of some or all resources. For example, a deadlock may occur when a system is jammed or two or more processes are in a circular chain, each of which waits for resources held by the process next in the chain. Liveness of a PN means that for any marking m reachable from the initial marking m_0, it is ultimately possible to fire any transition in the net by progressing through some firing sequence. Therefore, if a PN is live, there is no deadlock. For discussion of other properties, refer to Refs. .2–4

The net shown in Fig. 1 is safe since each place holds at most one token. It is not live since the net contains a deadlock; that is, at marking $(0\ 1\ 0)^{\tau}$, no transition is enabled. Note that this deadlock results from the exhaustion of the tokens in place p_1. Only markings $(0\ 0\ 1)^{\tau}$ and $(0\ 1\ 0)^{\tau}$ are reachable from the initial marking $(1\ 1\ 0)^{\tau}$. Others are not.

Starting from the initial system condition or state, it is desired to enumerate all the possible states the system can reach, as well as their relationship. The resulting representation is called a reachability tree or graph. The resulting method is called the reachability analysis method. All the behavioral properties can be discovered if the number of states is finite.

TYPES OF PETRI NETS

Marked Graphs

A *marked graph* is a PN such that each place has exactly one input and one output arc. It is also called an event graph and is used to model decision-free concurrent and repetitive systems exhibiting no choice. These systems include robotic manufacturing cells, transportation systems, job shops, and machine centers where the sequences of jobs or car movements are fixed.

Extended PNs

To increase the modeling power of PNs, they can be extended by including inhibitor arcs to test whether a place has no token. An inhibitor arc connects an input place to a transition and is pictorially represented by an arc terminated with a small circle. In the presence of an inhibitor arc, a transition is enabled if $m(p) > I(p, t)$ for each input place connected to t by a normal directed arc and no tokens are present on each input place connected to t by an inhibitor arc. The transition firing rules are the same for normally connected places. The firing, however, does not change the marking in the inhibitor arc–connected places. Another way to enhance the PN's modeling power is to assign priority over the transitions. The resulting nets are called *extended PNs.* They greatly facilitate the PN modeling of complex systems. They can model whatever systems a Turing machine can.

Timed PNs

A *deterministic timed Petri net* (*DTPN*) is defined as a marked graph, and either zero or positive time delays are associated with places, transitions, and/or arcs. The cycle time of a strongly connected deterministic timed PN is determined as follows:

$$\pi = \max_i \{D_i / N_i\}$$

where D_i is the total time delay of loop i, and N_i is the token count of loop i. The ratio D_i/N_i is called the cycle time of loop i. DTPN can be used to analyze a system cycle time and determine a bottleneck machine of a concurrent and repetitive system such as a robotic cell and a job shop.

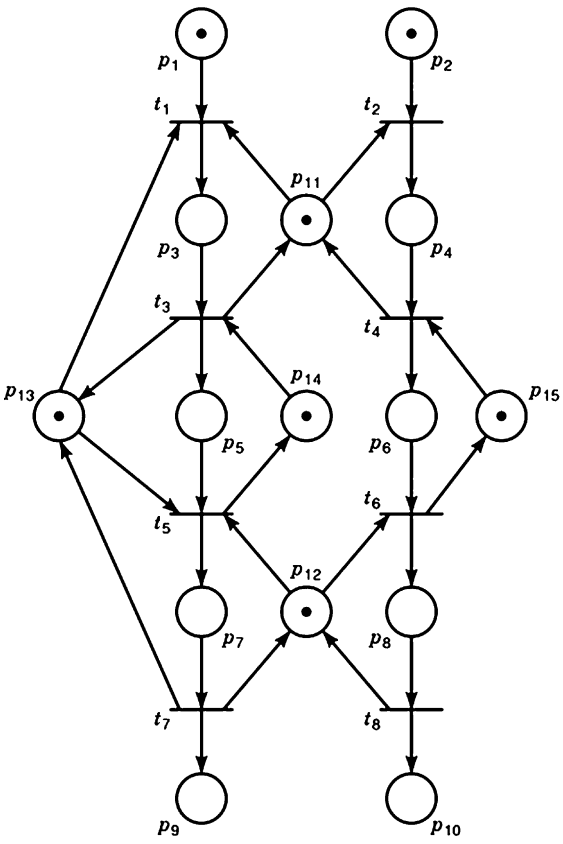
Stochastic PNs

Associating random time delays with exponential distribution yields stochastic PNs (*SPN*). The SPN models that allow for immediate transitions (i.e., with zero time delay) are called generalized SPN (*GSPN*) (5). Both models may include extensions, such as priority transitions and inhibitor arcs, and can be converted into their equivalent Markov processes for analysis. They can be used to model and evaluate flexible manufacturing systems, random polling systems, concurrent programs, and concurrent computer architectures. Their use avoids the initial enumeration of all states, which is needed if Markov processes are used at the beginning. The latter is impossible for a large system.

High-Level PNs

Allowing different tokens, enabling, and executing rules in a PN leads to colored PNs. They can offer compact representation of a system with many similar subsystems. Allowing predicates in transitions leads to predicate-transition nets.

Embedding attributes, procedures, or objects into tokens, places, and/or transitions in a PN leads to object-oriented PNs. All these high-level nets gain their applications in design and development of information systems



Interpretation of places
p_1: Job 1 raw part available
p_2: Job 2 available
p_3: M_1 processing J_1
p_4: M_1 processing J_2
p_5: J_1 part in its buffer
p_6: J_2 part in its buffer
p_7: M_2 processing J_1
p_8: M_2 processing J_2
p_9: J_1 product available
p_{10}: J_2 product available
p_{11}: M_1 available
p_{12}: M_2 available
p_{13}: Robot available
p_{14}: J_1's buffer slot available
p_{15}: J_2's buffer slot available

Figure 3. A Petri net model of a production system: two jobs are to be processed by M_1 and M_2. Job 1 needs Robot for its loading and unloading.

and complicated software systems.

Other special nets include state-machine PN, free-choice PN, asymmetric-choice PN, production-process nets and augmented marked graphs, (dis)assembly PN, augmented timed PNs, and real-time PNs (6, 8).

MODELING, SCHEDULING, AND CONTROL

Modeling is a fundamental step for all the applications of PNs. We illustrate a general modeling method through a production system. The system consists of two machines, M_1 and M_2, and one robot R. It processes two types of jobs, J_1 and J_2. Both have to go through M_1 and M_2 sequentially but require different processing times. J_1 also needs Robot for its part holding. There is one buffer slot assigned to J_1 and J_2 between their two processes, respectively. The number following the resources requirement in Table 1 is the processing time.

Table 1. Job Requirements

Operations/Jobs	J_1	J_2
1	(M_1R, 1)	(M_1, 4)
2	(M_2R, 4)	(M_2, 1)

First, we identify the operations/status as follows:

J_1: M_1 processing J_1, J_1's part in its buffer, and M_2 processing J_1; and

J_2: M_1 processing J_2, J_2's part in its buffer, and M_2 processing J_2.

The resources include M_1, M_2, R, J_1, and J_2's raw parts, buffer slots, and final products.

Next, we identify the relationships among the preceding operations/status. Each job's routing is fixed, and its operations form a precedence relation. Two jobs do not need to follow each other, however.

Third, we design and label the places that represent operations and resources (i.e., p_1–p_{15}, as shown in Fig. 3). Each operation and resource corresponds to a unique place. We arrange the operations in a series for each job since they form a precedence relationship. We designate a transition that starts at J_k and one that ends at J_k, $k = 1$ and 2. We insert transitions between two operation places if they have a precedence relationship. Thus we have t_1–t_8 in Fig. 3. For each transition, we draw an input arc to it from a place if enabling it requires the resource(s) or the completion of the operation(s) represented in the place; we draw an output arc from it to a place if firing it releases resource(s) or signals the initiation of the operation in the place. We take t_1 as an example. We link p_1, p_{11}, and p_{13} to it since availability of J_1's raw part, M_1, and Robot is required to start the operation in p_3. We link it to p_3 since its firing leads to the operation in p_3.

Finally, we determine the initial number of tokens over all places according to the system's initial state. If initially either of J_1 and J_2 has only one raw part, then the initial marking is the one shown in Fig. 3. We associate time delays with places. Hence, p_3, p_4, p_7, and p_8 are associated

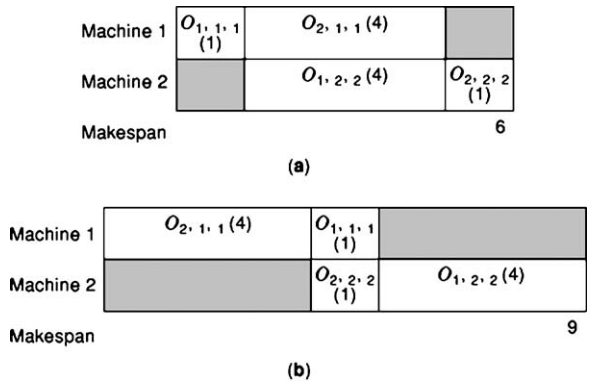


Figure 4. Schedules represented by transition firing sequences (a) $t_1t_3t_2t_5t_4t_7t_6t_8$ and (b) $t_2t_4t_1t_6t_3t_8t_5t_7$. The numbers in parentheses are time units.

with 1, 4, 4, and 1 time unit, respectively, and others with zero.

Once we have its PN model, we can perform analysis, scheduling, and control of the system. The purpose of analysis is to check the properties discussed previously. Scheduling aims to derive a schedule optimizing a certain performance index. Control deals with the coordination and execution of part flow and processing. The controller keeps track of system states, such as the location of all parts and the status of each resource. Based on the current state and production plan, the controller supervises all the individual system components. Sensors and actuators have to be connected to the controller, which is often implemented as a computer or a microprocessor chip.

Consider that both jobs 1 and 2 are in the shop and ready for processing at time 0, with each job having lot size of 1. We seek a production schedule to minimize the time required to complete both jobs. The initial marking is $(1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1)^{\mathrm{T}}$, and the final one $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1)^{\mathrm{T}}$. Both transition firing sequences of $t_1t_3t_2t_5t_4t_7t_6t_8$ and $t_2t_4t_1t_6t_3t_8t_5t_7$ give a path from the initial to final one. The production activities and markings corresponding to the first one are shown in Table 2.

The Gantt charts in Fig. 4 show the two schedules with the makespan of the first one being 6 and the second being 9. $O_{i,j,k}$ represents the jth operation of the ith job being performed at the kth machine. Clearly, the first one should be selected as our schedule.

In automated manufacturing, a deadlock situation may occur due to inappropriate allocation of resources and control, in which any part flow is inhibited. For example, suppose that the lot size of Job 1 in the preceding system is 2. Then the firing of transitions $t_1t_3t_1$ leads the system from initial state $(2\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1)^{\mathrm{T}}$ to deadlock $(0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1)^{\mathrm{T}}$, at which any further part flow is inhibited. The algorithms (8) are available to derive optimal or or near-optimal deadlock-free schedules based on a system's PN model. PN applications to complex manufacturing systems are referred to (2–8, 9).

MULTIDISCIPLINARY ENGINEERING APPLICATIONS

Typically, discrete event systems have characteristics that exhibit synchronization, concurrency, resource sharing/conflicts, time dependency, and repetition. Since such systems are omnipresent in the real world, PNs are modified and extended in several ways and applied as a diversified modeling technique crossing several important disciplines of engineering, such as electrical and computer engineering, manufacturing engineering, industrial engineering, software engineering, biomedical engineering, and systems engineering.

Electrical and Computer Engineering:

PNs are applied for modeling and analyzing communication protocols, validating microprocessors and hardware, and performing process control. Computer-aided software tools are developed using PNs as a formal specification technique for the specification and analysis of computer communication protocols. These tools are used for interactive simulation and exhaustive reachability analysis to determine the liveness (absence of deadlocks) of PN models of communication protocols. By studying the transition sequences in the PN model, events that lead to the undesired behavior of a protocol can be traced. By associating time delays with certain transitions in the PN model, the time required to perform certain operations of a communication protocol can be modeled. Then the performance issues of a protocol, such as total time taken by the protocol to do a job, buffer holding times, and buffer requirements in the communication subsystem, can be investigated.

Generalized stochastic PNs are applied for performance evaluation of interprocessor interrupt mechanisms in a shared but multi-microprocessor system (5). Performance issues, such as each processor's interrupt request origination rate and capacity of message box versus the mean overhead time per interrupt request of each source pro-

Table 2. The production activities for the transition firing sequence $t_1t_3t_2t_5t_4t_7t_6t_8$

Firing Transition	Marking	Production Activities
	$(1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1)^T$	Initial state. Jobs 1 and 2 are ready.
t_1	$(0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1)^T$	Operation 1 of J_1 starts.
t_3	$(0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1)^T$	Operation 1 of J_1 ends.
t_2	$(0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1)^T$	Operation 1 of J_2 starts.
t_5	$(0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1)^T$	Operation 2 of J_1 starts.
t_4	$(0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0)^T$	Operation 1 of J_2 ends.
t_7	$(0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0)^T$	Operation 2 of J_1 ends.
t_6	$(0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1)^T$	Operation 2 of J_2 starts.
t_8	$(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1)^T$	Operation 2 of J_2 ends. Final state.

cessor, can be analyzed. PN models are also used to design distributed communication structures in interprocess communications to recognize and avoid deadlocks.

Programmable logic controllers (*PLC*) are used for the sequential control or process control of manufacturing systems, chemical processing systems, and power plants. Traditionally the PLC programs are developed by ladder logic diagrams. However, to overcome their limitations in dealing with complex systems, PNs have recently been used to develop PLC programs; thus understandable and maintainable control systems can be developed for large automation projects (6, 7).

Manufacturing and Industrial Engineering

PNs serve as a graphical modeling tool for specification of the operations in a computer integrated manufacturing system (*CIMS*). When developed, they are used for analysis, design, scheduling, and simulation of CIMS to study such performance criteria as production rate, resource utilization, work-in-process inventory, number of resources needed, and cost of production. Alternatives can be investigated by changing the parameters, such as operational policies, operation times, number of resources, and mean time between failures of a resource. PNs are also used for supervisory control, sequence control, sequence control, fault detection, fault recovery, and monitoring of CIMS. Some studies integrate such techniques as expert systems and neural networks with PNs for controlling and monitoring of CIMS. PNs are used for rapid prototyping of control software using different programming languages (10). They are also integrated with object-oriented methodologies for the development of control software (6).

Software Engineering

PNs have been used for addressing several issues related to database systems, operating systems, distributed software systems, programming languages, etc. Concurrency control is one of the problems when distributed database systems are implemented. This problem involves modeling synchronization and concurrency among database transactions and how these transactions access data in a database. PNs are successfully used to model concurrency control of distributed databases. Concurrency control algorithms, such as centralized locking algorithm and distributed locking algorithm, are studied for their performance via their PN models. One of the main tasks in developing complex software systems is to design the system to be fault tolerant such that it can detect and eliminate the faults that may arise due to erroneous data, undetected hardware failures, and design flaws in hardware/software. Predicate/transition nets are used as a formal specification tool to describe and model complex software systems. These resulting PN models are then used systematically to integrate fault tolerance properties in the design of these software systems. PNs are combined with concurrent computer programming languages such as Ada and Flat Concurrent Prolog to study certain aspects of computer programs that have concurrency. The complete PN model of a computer program clearly models all the parallel tasks and their sequence dependence in the program. It is then used to detect deadlocks and hence debug the program.

BIBLIOGRAPHY

1. R. David H. Alla *Petri Nets and Grafcet*, Englewood Cliffs, NJ: Prentice Hall, 1992.
2. A. A. Desrochers R. Y. Al-Jaar *Applications of Petri Nets in Manufacturing Systems*, Piscataway, NJ: IEEE Press, 1995.
3. M. C. Zhou F. DiCesare *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Boston: Kluwer Academic, 1993.
4. T. Murata Petri nets: Properties, applications and analysis, *Proc. IEEE*, **77**: 541–580, 1989.
5. M. Ajmone Marsan *et al. Modeling with Generalized Stochastic Petri Nets*, Chichester, England: Wiley, 1995.
6. M. C. Zhou K. Venkatesh *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*, River Edge, NJ: World Scientific, 1998.
7. M. C. Zhou (ed.) *Petri Nets in Flexible and Agile Automation*, Boston: Kluwer Academic, 1995.
8. J.-M. Proth X. Xie *Petri Nets: A Tool for Design and Management of Manufacturing Systems*, New York: Wiley, 1996.
9. M. C. Zhou andM. P. Fanti (Ed.), *Deadlock Resolution in Computer-Integrated Systems*, Marcel Dekker: New York, January 2005.
10. Hruz, B. and M. C. Zhou *Modeling and Control of Discrete Event Dynamic Systems*, Springer, London, UK, 2007.

MENGCHU ZHOU
VENKATESH KURAPATI
HUANXIN HENRY XIONG
New Jersey Institute of Technology, Newark, NJ
American International Group, Inc., New York, NY
Lucent Technologies, Inc., 10 Industrial Way East, Eatontown, NJ