Petri Nets for Dynamic Event-Driven System Modeling

Jiacun Wang
Department of Software Engineering
Monmouth University
West Long Branch, NJ 07764
jwang@monmouth.edu

Introduction

Petri nets were introduced in 1962 by Dr. Carl Adam Petri (Petri 1962). Petri nets are a powerful modeling formalism in computer science, system engineering and many other disciplines. Petri nets combine a well defined mathematical theory with a graphical representation of the dynamic behavior of systems. The theoretic aspect of Petri nets allow precise modeling and analysis of system behavior, while the graphical representation of Petri nets enable visualization of the modeled system state changes. This combination is the main reason for the great success of Petri nets. Consequently, Petri nets have been used to model various kinds of dynamic event-driven systems like computers networks (Ajmone Marsan, Balbo and Conte 1986), communication systems (Merlin and Farber 1976; Wang 2006), manufacturing plants (Venkatesh, Zhou and Caudill 1994; Zhou and DiCesare 1989; Desrochers and Ai-Jaar 1995), command and control systems (Andreadakis and Levis 1988), real-time computing systems (Mandrioli and Morzenti 1996; Tsai, Yang and Chang 1995), logistic networks (Landeghem and Bobeanu 2002), and workflows (Aalst and Hee 2000; Lin, Tian and Wei 2002) to mention only a few important examples. This wide spectrum of applications is accompanied by wide spectrum different aspects which have been considered in the research on Petri nets.

Petri Net Definition

A Petri net is a particular kind of bipartite directed graphs populated by three types of objects. These objects are *places*, *transitions*, and *directed arcs*. Directed arcs connect places to transitions or transitions to places. In its simplest form, a Petri net can be represented by a transition together with an input place and an output place. This elementary net may be used to represent various aspects of the modeled systems. For example, a transition and its input place and output place can be used to represent a data processing event, its input data and output data, respectively, in a data processing system. In order to study the dynamic behavior of a Petri net modeled system in terms of its states and state changes, each place may potentially hold either none or a positive number of *tokens*. Tokens are a primitive concept for Petri nets in addition to places and transitions. The presence or absence of a token in a place can indicate whether a condition associated with this place is true or false, for instance.

A Petri net is formally defined as a 5-tuple $N = (P, T, I, O, M_0)$, where

(1) $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places;

(2) $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions, $P \cup T \neq \varnothing$, and $P \cap T = \varnothing$;

(3) I: $P \times T \rightarrow N$ is an *input function* that defines directed arcs from places to transitions, where N is a set of nonnegative integers;

(4) O: $T \times P \rightarrow N$ is an *output function* that defines directed arcs from transitions to places; and

(5) M_0: $P \rightarrow N$ is the *initial marking*.

A *marking* in a Petri net is an assignment of tokens to the places of a Petri net. Tokens reside in the places of a Petri net. The number and position of tokens may change during the execution of a Petri net. The tokens are used to define the execution of a Petri net.

Most theoretical work on Petri nets is based on the formal definition of Petri net structures. However, a

graphical representation of a Petri net structure is much more useful for illustrating the concepts of Petri net theory. A Petri net graph is a Petri net structure as a bipartite directed multigraph. Corresponding to the definition of Petri nets, a Petri net graph has two types of nodes. A *circle* represents a place; a *bar* or a *box* represents a transition. Directed arcs (arrows) connect places and transitions, with some arcs directed from places to transitions and other arcs directed from transitions to places. An arc directed from a place p_j to a transition t_i defines p_j to be an *input place* of t_i, denoted by $I(t_i, p_j) = 1$. An arc directed from a transition t_i to a place p_j defines p_j to be an *output place* of t_i, denoted by $O(t_i, p_j) = 1$. If $I(t_i, p_j) = k$ (or $O(t_i, p_j) = k$), then there exist k directed (parallel) arcs connecting place p_j to transition t_i (or connecting transition t_i to place p_j). Usually, in the graphical representation, parallel arcs connecting a place (transition) to a transition (place) are represented by a single directed arc labeled with its multiplicity, or weight k. A circle contains a *dot* represents a place contains a token (Peterson 1981).

***Example* 1:** A simple Petri net.

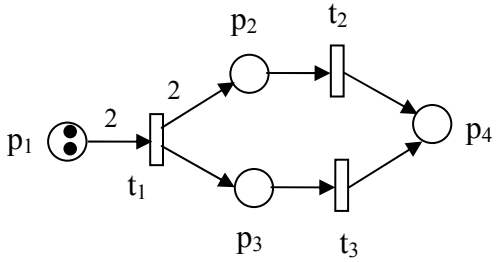

Figure 1 A simple Petri net.

Figure 1 shows a simple Petri net. In this Petri net, we have

$P = \{p_1, p_2, p_3, p_4\}$;

$T = \{t_1, t_2, t_3\}$;

$I(t_1, p_1) = 2$, $I(t_1, p_i) = 0$ for $i = 2, 3, 4$;

$I(t_2, p_2) = 1$, $I(t_2, p_i) = 0$ for $i = 1, 3, 4$;

$I(t_3, p_3) = 1$, $I(t_3, p_i) = 0$ for $i = 1, 2, 4$;

$O(t_1, p_2) = 2$, $O(t_1, p_3) = 1$, $O(t_1, p_i) = 0$ for $i = 1, 4$;

$O(t_2, p_4) = 1$, $O(t_2, p_i) = 0$ for $i = 1, 2, 3$;

$O(t_3, p_4) = 1$, $O(t_3, p_i) = 0$ for $i = 1, 2, 3$;

$M_0 = (2\ 0\ 0\ 0)^T$.

Transition Firing

The execution of a Petri net is controlled by the number and distribution of tokens in the Petri net. By changing distribution of tokens in places, which may reflect the occurrence of events or execution of operations, for instance, one can study the dynamic behavior of the modeled system. A Petri net executes by *firing* transitions. We now introduce the enabling rule and firing rule of a transition, which govern the flow of tokens:

(1) *Enabling Rule*: A transition t is said to be *enabled* if each input place p of t contains at least the number of tokens equal to the weight of the directed arc connecting p to t, i.e., $M(p) \geq I(t, p)$ for any p in P.

(2) *Firing Rule*: Only enabled transition can fire. The firing of an enabled transition t removes from each input place p the number of tokens equal to the weight of the directed arc connecting p to t. It

also deposits in each output place p the number of tokens equal to the weight of the directed arc connecting t to p.

Mathematically, firing t at M yields a new marking

$M'(p) = M(p) - I(t, p) + O(t, p)$ for any p in P.

Notice that since only enabled transitions can fire, the number of tokens in each place always remains non-negative when a transition is fired. Firing transition can never try to remove a token that is not there.

A transition without any input place is called a *source transition*, and one without any output place is called a *sink transition*. Note that a source transition is unconditionally enabled, and that the firing of a sink transition consumes tokens, but doesn't produce tokens.

A pair of a place p and a transition t is called a *self-loop*, if p is both an input place and an output place of t. A Petri net is said to be *pure* if it has no self-loops.

***Example* 2:** Transition firing.

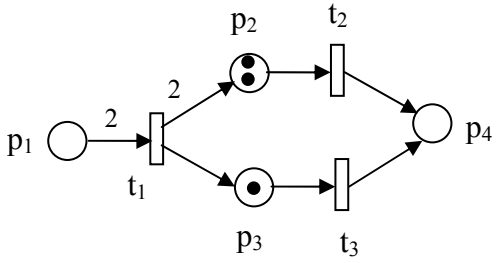

Figure 2 Firing of transition t_1.

Consider the simple Petri net shown in Figure 1. Under the initial marking, $M_0 = (2\ 0\ 0\ 0)$, only t_1 is enabled. Firing of t_1 results in a new marking, say M_1. It follows from the firing rule that

$M_1 = (0\ 2\ 1\ 0)$.

The new token distribution of this Petri net is shown in Figure 2. Again, in marking M_1, both transitions of t_2 and t_3 are enabled. If t_2 fires, the new marking, say M_2, is:

$M_2 = (0\ 1\ 1\ 1)$.

If t_3 fires, the new marking, say M_3, is:

$M_3 = (0\ 2\ 0\ 1)$.

Modeling Power

The typical characteristics exhibited by the activities in a dynamic event-driven system, such as concurrency, decision making, synchronization and priorities, can be modeled effectively by Petri nets.

1. *Sequential Execution*. In Figure 3(a), transition t_2 can fire only after the firing of t_1. This imposes the precedence constraint "t_2 after t_1." Such precedence constraints are typical of the execution of the parts in a dynamic system. Also, this Petri net construct models the causal relationship among activities.

2. *Conflict*. Transitions t_1 and t_2 are in conflict in Figure 3(b). Both are enabled but the firing of any transition leads to the disabling of the other transition. Such a situation will arise, for example, when a machine has to choose among part types or a part has to choose among several machines. The resulting conflict may be resolved in a purely non-deterministic way or in a probabilistic way, by assigning appropriate probabilities to the conflicting transitions.

3. *Concurrency*. In Figure 3(c), the transitions t_1, and t_2 are concurrent. Concurrency is an important attribute of system interactions. Note that a necessary condition for transitions to be concurrent is the existence of a forking transition that deposits a token in two or more output places.

4. *Synchronization*. It is quite normal in a dynamic system that an event requires multiple resources. The resulting synchronization of resources can be captured by transitions of the type shown in Figure 3(d). Here, t_1 is enabled only when each of p_1 and p_2 receives a token. The arrival of a token into each of the two places could be the result a possibly complex sequence of operations elsewhere in the rest of the Petri net model. Essentially, transition t_1 models the joining operation.

5. *Mutually exclusive*. Two processes are mutually exclusive if they cannot be performed at the same time due to constraints on the usage of shared resources. Figure 3(e) shows this structure. For example, a robot may be shared by two machines for loading and unloading. Two such structures are parallel mutual exclusion and sequential mutual exclusion.

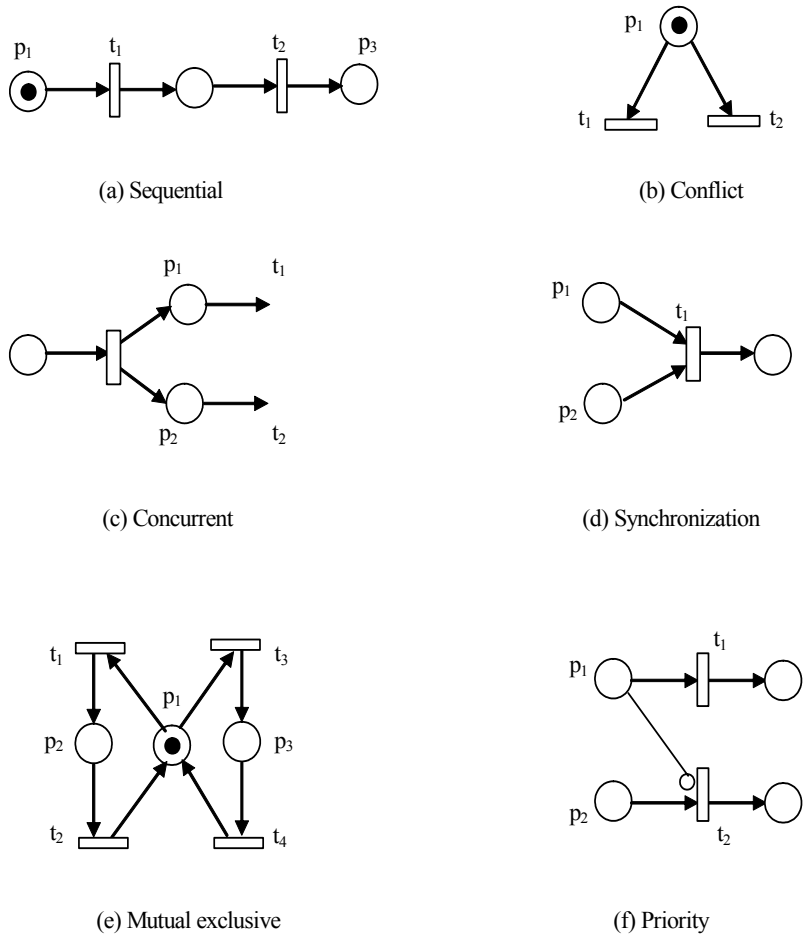

Figure 3 Petri net primitives to represent system features.

6. *Priorities*. The classical Petri nets discussed so far have no mechanism to represent priorities. Such a modeling power can be achieved by introducing an *inhibitor arc*. The inhibitor arc connects an input place to a transition, and is pictorially represented by an arc terminated with a small circle. The presence of an inhibitor arc connecting an input place to a transition changes the transition enabling conditions. In the presence of the inhibitor arc, a transition is regarded as enabled if each input place, connected to the

transition by a normal arc (an arc terminated with an arrow), contains at least the number of tokens equal to the weight of the arc, and no tokens are present on each input place connected to the transition by the inhibitor arc. The transition firing rule is the same for normally connected places. The firing, however, does not change the marking in the inhibitor arc connected places. A Petri net with an inhibitor arc is shown in Figure 3(f). t_1 is enabled if p_1 contains a token, while t_2 is enabled if p_2 contains a token and p_1 has no token. This gives priority to t_1 over t_2.

Petri Net Properties

As a mathematical tool, Petri nets possess a number of properties. These properties, when interpreted in the context of the modeled system, allow the system designer to identify the presence or absence of the application domain specific functional properties of the system under design. Two types of properties can be distinguished, behavioral and structural ones. The behavioral properties are these which depend on the initial state or marking of a Petri net. The structural properties, on the other hand, do not depend on the initial marking of a Petri net. They depend on the topology, or net structure, of a Petri net. Here we provide an overview of some of the most important, from the practical point of view, behavioral properties. They are reachability, safeness, and liveness.

Reachability

An important issue in designing event-driven systems is whether a system can reach a specific state, or exhibit a particular functional behavior. In general, the question is whether the system modeled with a Petri net exhibits all desirable properties as specified in the requirement specification, and no undesirable ones.

In order to find out whether the modeled system can reach a specific state as a result of a required functional behavior, it is necessary to find such a transition firing sequence which would transform a marking M_0 to M_i, where M_i represents the specific state, and the firing sequence represents the required functional behavior. It should be noted that a real system may reach a given state as a result of exhibiting different permissible patterns of functional behavior, which would transform M_0 to the required M_i. The existence in the Petri net model of additional sequences of transition firings which transform M_0 to M_i indicates that the Petri net model may not exactly reflect the structure and dynamics of the underlying system. This may also indicate the presence of unanticipated facets of the functional behavior of the real system, provided that the Petri net model accurately reflects the underlying system requirement specification. A marking M_i is said to be *reachable* from a marking M_0 if there exists a sequence of transitions firings which transforms a marking M_0 to M_i. A marking M_1 is said to be *immediately reachable* from M_0 if firing an enabled transition in M_0 results in M_1.

Safeness

In a Petri net, places are often used to represent information storage areas in communication and computer systems, product and tool storage areas in manufacturing systems, etc. It is important to be able to determine whether proposed control strategies prevent from the overflows of these storage areas. The Petri net property which helps to identify the existence of overflows in the modeled system is the concept of *boundedness*.

A place p is said to be *k-bounded* if the number of tokens in p is always less than or equal to k (k is a nonnegative integer number) for every marking M reachable from the initial marking M_0, i.e., $M \in R(M_0)$. It is *safe* if it is 1-bounded.

A Petri net $N = (P, T, I, O, M_0)$ is k-bounded (safe) if each place in P is k-bounded (safe).

Liveness

The concept of liveness is closely related to the *deadlock* situation, which has been situated extensively in the context of computer operating systems.

A Petri net modeling a deadlock-free system must be *live*. This implies that for any reachable marking M, it is ultimately possible to fire any transition in the net by progressing through some firing sequence. This requirement, however, might be too strict to represent some real systems or scenarios that exhibit deadlock-free behavior. For instance, the initialization of a system can be modeled by a transition (or a set of transitions) which fire a finite number of times. After initialization, the system may exhibit a deadlock-free behavior, although the Petri net representing this system is no longer live as specified above. For this reason, different levels of liveness for transition t and marking M_0 were defined. Refer to (Murata 1989) for details.

Analysis of Petri Nets

We have introduced the modeling power of Petri nets in the previous sections. However, modeling by itself is of little use. It is necessary to *analyze* the modeled system. This analysis will hopefully lead to important insights into the behavior of the modeled system.

There are three common approaches to Petri net analysis: 1) reachability analysis, 2) the matrix-equation approach, and 3) simulation. The first approach involves the enumeration of all reachable markings, but it suffers from the state-space explosion issue. The matrix equations technique is powerful but in many cases it is applicable only to special subclasses of Petri nets or special situations. For complex Petri net models, discrete-event simulation is an option to check the system properties.

Reachability analysis

Reachability analysis is conducted through the construction of reachability tree (also called coverability tree due to the use of ω markings). Given a Perti net N, from its initial marking M_0, we can obtain as many "new" markings as the number of the enabled transitions. From each new marking, we can again reach more markings. Repeating the procedure over and over results in a tree representation of the markings. Nodes represent markings generated from M_0 and its successors, and each arc represents a transition firing, which transforms one marking to another.

The above tree representation, however, will grow infinitely large if the net is unbounded. To keep the tree finite, we introduce a special symbol ω, which can be thought of as "infinity". It has the properties that for each integer n, $\omega > n$, $\omega + n = \omega$ and $\omega \geq \omega$.

The reachability tree for a Petri net N is constructed by the following algorithm.

1. *Label the initial marking M_0 as the root and tag it "new".*
2. *For every new marking M:*
   - 2.1 *If M is identical to a marking already appeared in the tree, then tag M "old" and go to another new marking.*
   - 2.2 *If no transitions are enabled at M, tag M "dead-end" and go to another new marking.*
   - 2.3 *While there exist enabled transitions at M, do the following for each enabled transition t at M*:
     - 2.3.1 *Obtain the marking M′ that results from firing t at M.*
     - 2.3.2 *On the path from the root to M if there exists a marking M″ such that $M'(p) \geq M''(p)$ for each place p and $M' \neq M''$, i.e. M″ is coverable, then replace $M'(p)$ by ω for each p such that $M'(p) > M''(p)$.*
     - 2.3.3 *Introduce M′ as a node, draw an arc with label t from M to M′, and tag M′ "new".*

Merging the same nodes in a reachability results in a *reachability graph*.

***Example* 3:** Reachability analysis.

Consider the Petri net shown in Figure 1. All reachable markings are: $M_0 = (2, 0, 0, 0)^T$, $M_1 = (0, 2, 1, 0)^T$, $M_2 = (0, 1, 1, 1)^T$, $M_3 = (0, 2, 0, 1)^T$, $M_4 = (0, 0, 1, 2)^T$, $M_5 = (0, 1, 0, 2)^T$, and $M_6 = (0, 0, 0, 3)^T$. The reachability tree of this Petri net is shown in Figure 4(a), and the reachability graph is shown in Figure 4(b).

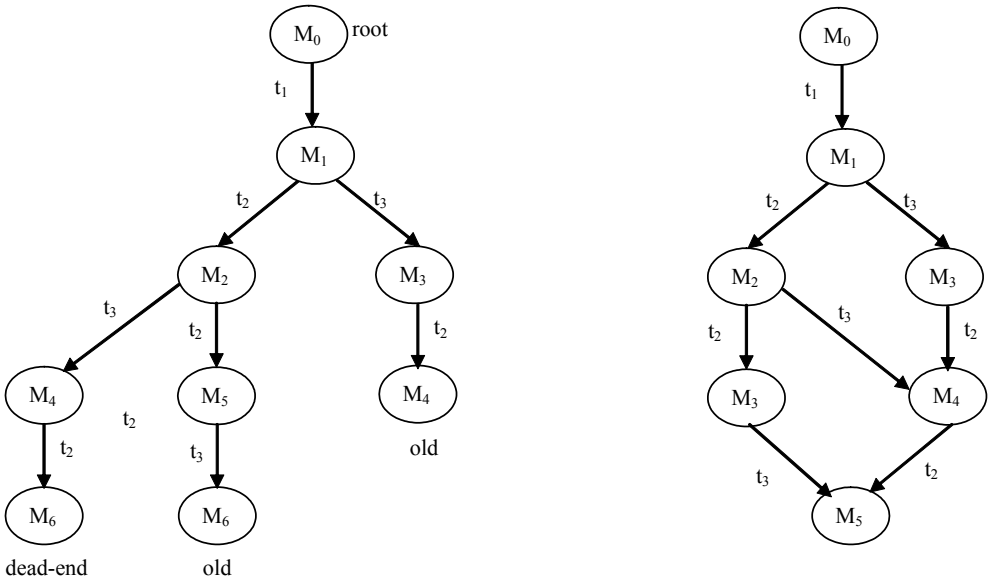

Figure 4 (a) Reachability tree. (b) Reachability graph.

Incidence matrix and state equation

For a Petri net with n transitions and m places, the incidence matrix $A = [a_{ij}]$ is an $n \times m$ matrix of integers and its typical entry is given by

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$$

where $a_{ij}^{+} = O(t_i, p_j)$ is the weight of the arc from transition i to its output place j and $a_{ij}^{-} = I(t_i, p_j)$ is the weight of the arc to transition i from its input place j.

It is easy to see from the transition firing rule that a_{ij}^{-}, a_{ij}^{+}, and a_{ij} represent the number of tokens removed, added, and changed in place p_j when transition t_i fires once, respectively. Transition t_i is enabled at a marking M if and only if

$$a_{ij}^{-} \le M(p_j), j = 1, 2, \cdots, m.$$

In writing matrix equations, we write a marking M_k as an $m \times 1$ column vector. The j-th entry of M_k denotes the number of tokens in place j immediately after the k-th firing in some firing sequence. The k-th firing or *control vector* u_k is an $n \times 1$ column vector of $n-1$ 0's and one nonzero entry, a 1 in the i-th position indicating that transition i fires at the k-th firing. Since the i-th row of the incidence matrix A denotes the change of the marking as the result of firing transition i, we can write the following state equation for a Petri net:

$$M_k = M_{k-1} + A^T u_k, k = 1, 2, \ldots$$

Suppose that a destination marking M_d is reachable from M_0 through a firing sequence $\{u_1, u_2 \ldots, u_d\}$.

Writing the state equation for $k = 1, 2, \ldots, d$ and summing them, we obtain

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k$$

Simulation

For complex Petri net models, simulation is another way to check the system properties. The idea is simple, that is, using the execution algorithm to run the net. Simulation is an expensive and time-consuming technique. It can show the presence of undesirable properties but cannot prove the correctness of the model in general case. Despite this, Petri net simulation is indeed a convenient and straightforward yet effective approach for engineers to validate the desired properties of a discrete event system. The algorithm is given as follows:

(1) Initialization: decide the initial marking and the set of all enabled transitions in the marking;

(2) If the number of preset simulation steps or certain stopping criteria is met, stop. Otherwise, if there is no transition enabled, report a deadlock marking and either stop or go to Step (1).

(3) Randomly pick a transition to fire. Remove the same number of tokens from each of its input places as the number of arcs from that place to the transition and deposit the same number of tokens to each of its output places as the number of arcs from the transition to that place.

(4) Remove enabled transitions which are modified at the new marking by checking the output transitions of the input places used in Step (3). If the output transitions of the output places in Step (3) become enabled, add those enabled ones. Go to Step (2).

The above algorithm can be modified to simulate extended Petri nets such as timed ones. The advantage of the simulation methods is to allow one to derive the temporal performance for a system under very realistic assumptions. A list of Petri net simulation tools along with feature descriptions can be found in the following *Petri Nets World* website:

`http://www.informatik.uni-hamburg.de/TGI/PetriNets/`

High-Level Petri Nets

In a standard Petri net, tokens are indistinguishable. Because of this, Petri nets have the distinct disadvantage of producing very large and unstructured specifications for the systems being modeled. To tackle this issue, high-level Petri nets were developed to allow compact system representation.

Colored Petri nets

Introduced by Kurt Jensen in (Jensen 1981), a *Colored Petri Net* (CPN) has its each token attached with a color, indicating the identity of the token. Moreover, each place and each transition has attached a set of colors. A transition can fire with respect to each of its colors. By firing a transition, tokens are removed from the input places and added to the output places in the same way as that in original Petri nets, except that a functional dependency is specified between the color of the transition firing and the colors of the involved tokens. The color attached to a token may be changed by a transition firing and it often represents a complex data-value. CPNs lead to compact net models by using of the concept of colors. This is illustrated by Example 4.

***Example* 4:** A manufacturing system.

Consider a simple manufacturing system comprising two machines M1 and M2, which process three different types of raw parts. Each type of parts goes through one stage of operation, which can be performed on either M1 or M2. After the completion of processing of a part, the part is unloaded from the system and a fresh part of the same type is loaded into the system. Figure 5 shows the (uncolored) Petri net model of the system. The places and transitions in the model are as follows:

p_1 (p_2): Machine M1 (M2) available

p_3 (p_4, p_5): A raw part of type 1 (type 2, type 3) available

p_6 (p_7, p_8): M1 processing a raw part of type 1 (type 2, type 3)

p_9 (p_{10}, p_{11}): M2 processing a raw part of type 1 (type 2, type 3)

t_1 (t_2, t_3): M1 begins processing a raw part of type 1 (type 2, type 3)

t_4 (t_5, t_6): M2 begins processing a raw part of type 1 (type 2, type 3)

t_7 (t_8, t_9): M1 ends processing a raw part of type 1 (type 2, type 3)

t_{10} (t_{11}, t_{12}): M2 ends processing a raw part of type 1 (type 2, type 3)

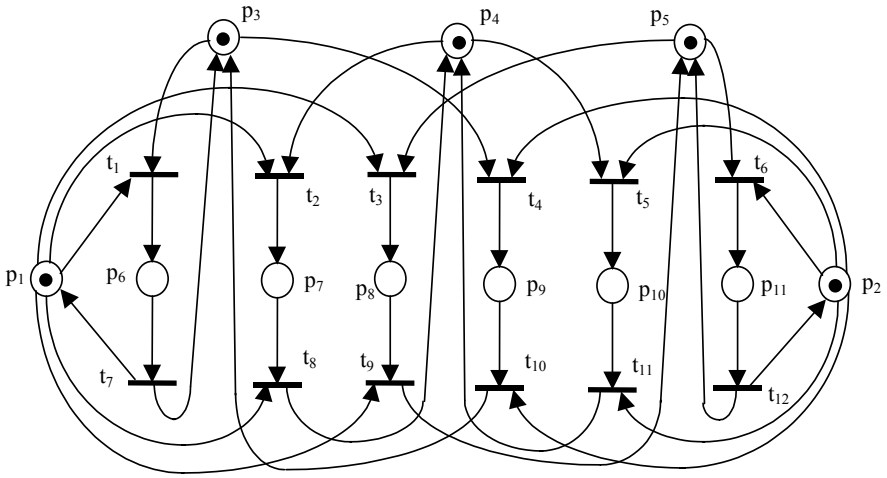

Figure 5 Petri net model of a simple manufacturing system.

Now let us take a look at the CPN model of this manufacturing system, which is shown in Figure 6. As we can see, there are only 3 places and 2 transitions in the CPN model, compared at 11 places and 12 transitions in Fig. 5. In this CPN model, p_1 means machines are available, p_2 means parts available, p_3 means processing in progress, t_1 means processing starts, and t_1 means processing ends. There are three color sets: *SM*, *SP* and $SM \times SP$, where SM = {M1, M2}, SP = {J1, J2, J3}. The color of each node is as follows:

$C(p_1)$ = {M1, M2},

$C(p_2)$ = {J1, J2, J3},

$C(p_3) = SM \times SP$,

$C(t_1) = C(t_2) = SM \times SP$.

CPN models can be analyzed through reachability analysis. As for ordinary Petri nets, the basic idea behind reachability analysis is to construct a reachability graph. Obviously, such a graph may become very large, even for small CPN's. However, it can be constructed and analyzed totally automatically, and there exist techniques which makes it possible to work with condensed occurrence graphs without losing analytic power. These techniques build upon equivalence classes. Another option to the CPN model analysis is

simulation. Readers are referred to (Jensen 1997) for a detailed description of the concepts, analysis methods and practical use of colored Petri nets.

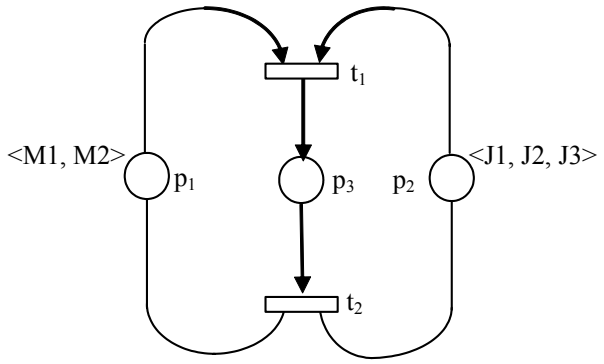

Figure 6 Colored Petri net model of the manufacturing system.

Predicate/transition nets

Predicate/Transition (Pr/T) Nets, also called High-Level Petri Nets in general, have been first introduced in (Genrich and Lautenbach 1981) and were the first class of high-level Petri Nets. Simply put, a Pr/T Net consists of the following constituents:

1. A *directed net* (S, T; F) where S is the set of predicates, T the set of transitions and $F \subseteq S \times T \cup T \times S$ the set of arcs.
2. A *structure* consisting of some sorts of individuals together with some operations and relations.
3. A *labeling of arcs* assigning to all elements of F a formal sum of n-tuples of variables where n is the arity of the predicate connected to the arc.
4. An *inscription on transitions* assigning to some elements of T a logical formula built from the structure where variables occurring free in a transition have to occur at an adjacent arc

Each element of T represents a class of possible changes of markings. Such a change, due to transition firing, consists of removing tokens from a subset of places and adding them to other subsets according to the expressions labeling the arcs. A transition is enabled whenever, given an assignment of individual tokens to the variables which satisfies the predicate associated with the transition, all input places carry enough copies of proper tokens.

***Example* 5:** Five dining philosophers.

Consider the famous synchronization problem consisting of five hungry philosophers who spend some time thinking between copious meals. There are only five forks on a circular table and there is a fork between two philosophers. To eat, each philosopher needs the two adjacent forks. Figure 7 presents the Pr/T net model of the system. In the model, the three places contain two types of tokens, the first type is associated with the philosophers and the second is associated with forks. The arcs are labeled by the token variables. A token has a number of attributes. The tokens residing in both the places "*think*" and "*free forks*" have two attributes: the first attribute being its type and the second attribute being its *identity*, *id.* The tokens residing in the place "*eat*", have four attributes: the first two ones are the same as in place "*think*", and the last two ones are the ids of the forks currently used by the philosopher. The transition "*take forks*" is associated with the predicate which specifies the correct relation between a philosopher and the two forks used by him. The

predicate inscribed on transition "*take forks*" as *i = j* is a concise form of expressing that the second attribute of a <*p*, *i*> token should match the second attribute of the two tokens representing the forks. This means that a philosopher can eat only when the two adjacent forks are free. For example, the forks <*f*, 3> and <*f*, 4> must be free in order to allow the philosopher <*p*, 3> to move to the eating place. Note that a predicate expresses an imperative condition which must be met in order for a transition to fire. A predicate should not be used to express the result associated with transition "*put down forks*", although there is a well-defined relationship between the attributes of the tokens released when "*put down forks*" fires.

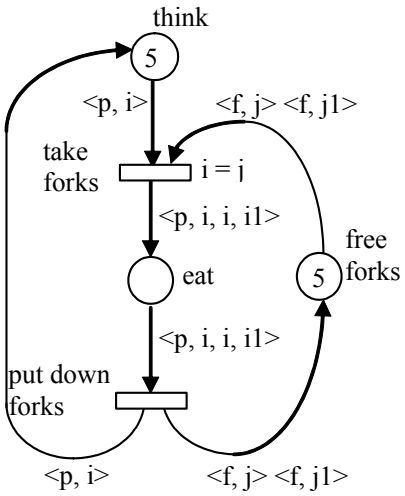

Figure 7 Pr/T net model of the five philosophers system.

Timed Petri Nets

The need for including timing variables in the models of various types of dynamic systems is apparent since these systems are real time in nature. In the real world, almost every event is time related. When a Petri net contains a time variable, it becomes a *Timed Petri Net* (Wang 1998). The definition of a timed Petri net consists of three specifications:

- The topological structure,
- The labeling of the structure, and
- Firing rules.

The topological structure of a timed Petri net generally takes the form that is used in a conventional Petri net. The labeling of a timed Petri net consists of assigning numerical values to one or more of the following things:

- Transitions,
- Places, and
- Arcs connecting the places and transitions.

The firing rules are defined differently depending on the way the Petri net is labeled with time variables. The firing rules defined for a timed Petri net control the process of moving the tokens around.

The above variations lead to several different types of timed Petri nets. Among them, deterministic time Petri nets (Ramchandani 1974) and stochastic timed Petri nets (Molloy 1982; Bause and Kritzinger 2002), in which time variables are associated with transitions, are the two most widely used extended Petri nets.

Deterministic timed Petri nets

The introduction of deterministic time labels into Petri nets was first attempted by Ramchandani (Ramchandani 1974). In his approach, the time labels were placed at each transition, denoting the fact that transitions are often used to represent actions, and actions take time to complete. The obtained extended Petri nets are called *Deterministic Timed Petri Nets* (DTPNs for short). (Ramamoorthy and Ho 1980) used such an extended model to analyze system performance. The method is applicable to a restricted class of systems called *decision-free nets.* This class of nets involves neither decisions nor non-determinism. In structural terms, each place is connected to the input of no more than one transition, and to the output of no more than one transition.

A deterministic timed transitions Petri net (DTPN) is a 6-tuples (P, T, I, O, M_0, τ), where (P, T, I, O, M_0) is a Petri net, $\tau: T \to R^+$ is a function that associates transitions with deterministic time delays.

A transition t_i in a DTPN can fire at time τ if and only if

(1) for any input place p of this transition, there have been the number of tokens equal to the weight of the directed arc connecting p to t_i in the input place continuously for the time interval $[\tau - \tau_i, \tau]$, where τ_i is the associated firing time of transition t_i;

(2) After the transition fires, each of its output places, p, will receive the number of tokens equal to the weight of the directed arc connecting t_i to p at time τ.

An important application of DTPN is to calculate the circle time of a Petri net model. For a decision-free Petri net where every place has exactly one input arc and one output arc, the minimum cycle time (maximum performance) C is given by

$$C = \max\{\frac{T_k}{N_k}: k = 1,2,\cdots,q\}$$

where

$T_k = \sum_{t_i \in L_k} \tau_i =$ sum of the execution times of the transitions in circuit k;

$N_k = \sum_{p_i \in L_k} M(p_i) =$ total number of tokens in the places in circuit k;

q = number of circuits in the net.

***Example* 6:** A communication protocol.

Consider the communication protocol between two processes, one indicated as the sender, and the other as the receiver. The sender sends messages to a buffer, while the receiver picks up messages from the buffer. When it gets a message, the receiver sends an ACK back to the sender. After receiving the ACK from the receiver, the sender begins processing and sending a new message. Suppose that the sender takes 1 time unit to send a message to the buffer, 1 time unit to receiver the ACK, and 3 time units to process a new message. The receiver takes 1 time unit to get the messages from the buffer, 1 time unit to send back an ACK to the buffer, and 4 time units to process a received message. The DTPN model of this protocol is shown in Figure 8. The legends of places and transitions and timing properties are as follows:

p_1: The sender ready

p_2: Message in the buffer

p_3: The sender waiting for ACK

p_4: Message received

p_5: The receiver ready

p_6: ACK sent

p_7: ACK in the buffer

p_8: ACK received

t_1: The sender sends a message to the buffer. Time delay: 1 time unit.

t_2: The receiver gets the messages from the buffer. Time delay: 1 time unit.

t_3: The receiver sends back an ACK to the buffer. Time delay: 1 time unit.

t_4: The receiver processes the message. Time delay: 3 time units.

t_5: The sender receivers the ACK. Time delay: 1 time unit.

t_6: The sender processes a new message. Time delay: 4 time units.

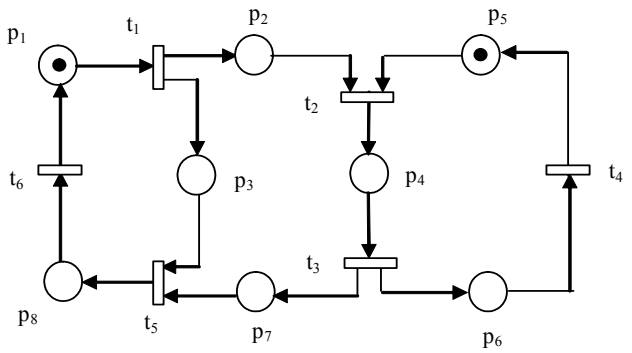

Figure 8 Petri net model of a simple communication protocol.

There are three circuits in the model. The circle time of each circuit is calculated as follows:

circuit $p_1t_1p_3t_5p_8t_6p_1 : C_1 = \dfrac{T_1}{N_1} = \dfrac{1+1+3}{1} = 5$,

circuit $p_1t_1p_2t_2p_4t_3p_7t_5p_8t_6p_1 : C_2 = \dfrac{T_2}{N_2} = \dfrac{1+1+1+1+4}{1} = 8$,

circuit $p_5t_2p_4t_3p_6t_4p_5 : C_3 = \dfrac{T_3}{N_3} = \dfrac{1+1+4}{1} = 6$.

After enumerating all circuits in the net, we know the minimum cycle time of the protocol between the two processes is 8 time units.

Stochastic timed Petri nets

Stochastic timed Petri nets (STPN's) are Petri nets in which stochastic firing times are associated with transitions. An STPN is essentially a high-level model that generates a stochastic process. STPN-based performance evaluation basically comprises modeling the given system by an STPN and automatically generating the stochastic process that governs the system behavior. This stochastic process is then analyzed using known techniques. STPN's are a graphical model and offer great convenience to a modeler in arriving at a credible, high-level model of a system.

The simplest choice for the individual distributions of transition firing times is negative exponential distribution. Because of the memoryless property of this distribution, the stochastic process associated with the STPN is a continuous time homogeneous Markov chain with state space in one to one correspondences to with marking in $R(M_0)$, the set of all reachable markings. The transition rate matrix of the Markov chain can be easily constructed from the reachability graph given the firing rates of the transitions of the STPN. Exponential timed stochastic Petri nets, often called *Stochastic Petri Nets* (SPNs), were independently proposed by Natkin (Natkin 1980) and Molloy (Molloy 1981), and their capabilities in the performance analysis of real systems have been investigated by many authors.

An SPN is a six-tuple (P, T, I, O, M_0, Λ) in which (P, T, I, O, M_0) is a Petri net and Λ: $T \rightarrow R$ is a set of firing rates whose entry λ_k is the rate of the exponential individual firing time distribution associated with transition t_k. Natkin and Molloy have shown that the marking process of an SPN is a continuous time Markov chain. The state space of the Markov chain is the reachable set $R(M_0)$. Suppose there are s markings in $R(M_0)$, and the underlying Markov chain is ergodic, then the steady state probability distribution $\Pi = (\pi_0, \pi_1, \ldots, \pi_s)$ can be obtained by resolving the following linear system:

$$\Pi Q = 0$$

$$\sum_j \pi_j = 1 \text{, where } \pi_j \geq 0, j = 0, 1, 2, \ldots$$

where Q is a transition rate matrix whose elements outside the main diagonal are the rates of the exponential distributions associated with the transitions from state, while the elements on the main diagonal make the sum of the elements of each row equal to zero. Denote by T_{ij} the set of enabled transitions at marking M_i whose firing leads the SPN to marking M_j. Then Q is determined as follows:

$$q_{ij} = \sum_{t_k \in T_{ij}} \lambda_k ,$$

$$q_{ii} = - \sum_{t_k \in E(M_i)} \lambda_k .$$

The probability of marking M_i changing to M_j is the same as the probability that one of the transitions in the set T_{ij} fires before any of the transitions in the set $T \setminus T_{ij}$. Since the firing times in an SPN are mutually independent exponential random variables, it follows that the required probability has the specific value given by

$$\alpha_{ij} = q_{ij} / q_i .$$

In the expression for α_{ij} deduced above, note that the numerator is the sum of the rates of those enabled transitions in M_i, the firing of any of which changes the marking from M_i to M_j; whereas the denominator is the sum of the rates of all the enabled transitions in M_i. Also note that $\alpha_{ij} = 1$ if and only if $T_{ij} = E(M_i)$.

***Example* 7:** A stochastic Petri net.

Figure 9 shows a simple SPN model with its reachable markings and its reachable graph. The linear system of steady state probabilities is

$$\pi_0 + \pi_2 + \pi_2 + \pi_3 + \pi_4 = 1$$

Let $\Lambda = (1\ 1\ 1\ 1\ 1)$, then solution to this system is:

$$\pi_0 = \pi_4 = 2/7,\ \pi_1 = \pi_2 = \pi_3 = 1/7.$$

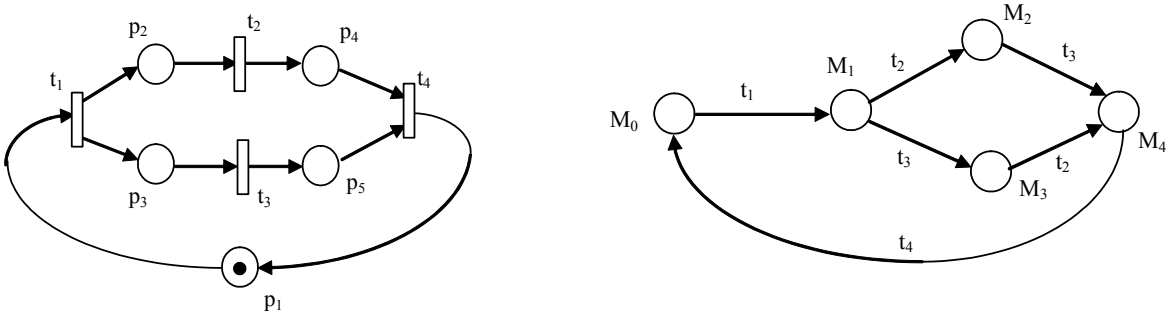

Figure 9 (a) SPN model. (b) Reachability graph.

The analysis of an SPN model is usually aimed at the computation of more aggregate performance indices than the probabilities of individual markings. Several kinds of aggregate results are easily obtained from the steady state distribution over reachable markings. In this section we quote some of the most commonly and easily computed aggregate steady state performance parameters (Ajmone Marsan 1990).

- The probability of an event defined through place markings (e.g., no token in a subset of places, or at least one token in a place while another one is empty, etc.), can be computed by adding the probabilities of all markings in which the condition corresponding to the event definition holds true. Thus, for example, the steady-state probability of the event *A* defined through a condition that holds true for the markings $M_i \in H$ is obtained as:

$$P\{A\} = \sum_{M_i \in H} \pi_i .$$

- The average number of tokens in a place can be obtained by computing the individual probabilities as those of the event "place p_i contains k tokens".

- The frequency of firing a transition, i.e., the average number of times the transition fires in unit time, can be computed as the weighted sum of the transition firing rate:

$$f_i = \sum_{i:t_j \in E(M_i)} \lambda_j(M_i)\pi_i$$

  where f_j is the frequency of firing t_j, $E(M_i)$ is the set of transitions enabled in M_i, and $\lambda_j(M_i)$ is the firing rate of t_j at M_i.

- The average delay of a token in traversing a subnet in steady state conditions can be computed using Little's formula

$$E(T) = \frac{E(N)}{E(\gamma)},$$

  where $E(T)$ is the average delay, $E(N)$ is the average number of tokens in the process of traversing the subnet, and $E(\gamma)$ is the average input (or output) rate of tokens into (or out of) the subnet. This procedure can be applied whenever the interesting tokens can be identified inside the subnet (which can also comprise other tokens defining its internal condition, but these must be distinguishable from those whose delay is studied) so that their average number can be computed, and a relation can be established between input and output tokens (e.g., one output token for each input token).

Concluding Remark

Petri nets have been proven to be a powerful modeling tool for various types of dynamic event-driven systems. Since Petri nets were introduced in 1962, numerous research papers have been published. The research has been conducted on many branches, with each branch exploring a promising application aspect of this formalism. Given the rich research results from the Petri net society, it is hard to cover all of them in such a short book chapter. Therefore, this chapter only aims at briefly introducing the most basic concepts, theory and applications of ordinary Petri nets and a few of most popular extended Petri nets.

References

van der Aalst, Wil, and Kees van Hee. 2000. *Workflow Management: Models, Methods, and Systems*. Massachusetts: MIT Press.

Ajmone Marsan, M. 1990. Stochastic Petri nets: an elementary introduction. *Advances in Petri Nets, LNCS* 424.

Ajmone Marsan, M., M. G. Balbo and G. Conte. 1986. *Performance Models of Multiprocessor Systems*, Massachusetts: The MIT Press.

Andreadakis, S.K., and A.H. Levis. 1988. Synthesis of distributed command and control for the outer air battle. *Proceedings of the 1988 Symposium on C² Research*. SAIC, McLean, VA.

Bause, F., and P. Kritzinger. 2002. *Stochastic Petri Nets -- An Introduction to the Theory*. Germany: Vieweg Verlag.

Desrochers, A., and R. Ai-Jaar. 1995. *Applications of Petri Nets in Manufacturing Systems: Modeling, Control and Performance Analysis*. IEEE Press.

Genrich, J.H., and K. Lautenbach, 1981. System modeling with high-level Petri nets. *Theoretical Computer Science* 13: 109-136.

Haas, P. *Stochastic Petri Nets: Modelling, Stability, Simulation*. 2002. New York: Springer-Verlag.

Jensen, K. 1981. Colored Petri nets and the invariant-method, *Theoretical Computer Science* 14: 317-336.

Jensen, K., 1997. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use (3 volumes). London: Springer-Verlag.

van Landeghem, Rik and Carmen-Veronica Bobeanu. 2002. Formal modeling of supply chain: an incremental approach using Petri nets. *14th European Simulations Symposium and Exhibition* Dresden, Germany.

Lin, Chuang, Liqin Tian and Yaya Wei. 2002. Performance equivalent analysis of workflow systems, *Journal of Software* 13(8): 1472-1480.

Lindemann, C. 1998. *Performance Modelling with Deterministic and Stochastic Petri Nets*. John Wiley and Sons.

Mandrioli, D., A. Morzenti, M. Pezze, P. Pietro S. and S. Silva. 1996. A Petri net and logic approach to the specification and verification of real time systems. In: *Formal Methods for Real Time Computing* (C. Heitmeyer and D. Mandrioli eds), John Wiley & Sons Ltd.

Merlin, P., and D. Farber. 1976. Recoverability of communication protocols - implication of a theoretical study. *IEEE Transactions on Communications* 1036-1043.

Molloy, M. 1981. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. Ph.D. Thesis, UCLA, Los Angeles, CA.

Molloy, M. 1982. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers* 31(9): 913-917.

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

Natkin, S. 1980. *Les Reseaux de Petri Stochastiques et Leur Application a I'evaluation des Systemes Informatiques*. These de Docteur Ingegneur, Cnam, Paris, France.

Peterson, J. L. 1981. *Petri Net Theory and the Modeling of Systems*. N.J.: Prentice-Hall.

Petri, C.A. 1962. *Kommunikation mit Automaten*. English Translation, 1966: *Communication with Automata*, Technical Report RADC-TR-65-377, Rome Air Dev. Center, New York.

Ramchandani, C. 1974. *Analysis of Asynchronous Concurrent Systems by Petri Nets*. Project MAC, TR-120, M.I.T., Cambridge, MA.

Ramamoorthy, C., and G. Ho. 1980. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transaction on Software Engineering* 6(5): 440-449.

Tsai, J., S. Yang, and Y. Chang. 1995. Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications. *IEEE Transactions on Software Engineering* 21(1): 32-49.

Wang, J. 1998. *Timed Petri Nets, Theory and Application.* Boston: Kluwer Academic Publishers.

Wang, J. 2006. Charging information collection modeling and analysis of GPRS networks. *IEEE Transactions on Systems, Man and Cybernetics*, *Part C* 36(6).

Venkatesh, K., M. C. Zhou, and R. Caudill. 1994. Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system. *IEEE Trans. on Industrial Electronics* 41(6): 611-619.

Zhou, M. C., and F. DiCesare. 1989. Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Trans. on Systems, Man, and Cybernetics* 19(5): 963-973.