

Simple Petri Nets

This chapter covers the use of classic **stochastic Petri nets** for the modeling and evaluation of stochastic discrete event systems. They represent a graphical and mathematical method for their convenient specification. Petri nets are especially useful for systems with concurrent, synchronized, and conflicting or nondeterministic activities. The graphical representation of Petri nets comprises only a few basic elements. They are, therefore, useful for documentation and a figurative aid for communication between system designers. Complex systems can be described in a modular way, where only local states and state changes need to be considered. The mathematical foundation of Petri nets allows their qualitative analysis based on state equations or reachability graph, and their quantitative evaluation based on the reachability graph or by simulation.

Petri nets come in many different flavors. Common to all of them is that they contain **places** (depicted by circles), **transitions** (depicted by boxes or bars), and directed **arcs** connecting them. A Petri net can thus be mathematically classified as a directed, bipartite graph. Places may hold **tokens**, and a certain assignment of tokens to the places of a model corresponds to its model state (called **marking** in Petri net terms). Transitions model activities (state changes, events). Just like in other discrete event system descriptions, events may be possible in a state – the transition is said to be **enabled** in the marking. If so, they may happen atomically (the transition **fires**) and change the system state. Transition enabling and firing as well as the consequential marking change are defined by the **enabling rule** and **firing rule** of the actual Petri net class. In our understanding of SDES, activities may take some time, thus allowing the description and evaluation of performance-related issues. Basic quantitative measures like the throughput, loss probabilities, utilization, and others can be computed. In the Petri net environment, a **firing delay** is associated to each transition, which may be stochastic (a random variable) and thus described by a probability distribution. It is interpreted as the time that needs to pass between the enabling and subsequent firing of a transition.

This text presents different Petri net model classes. Depending on the type of tokens, we distinguish between standard Petri nets with identical tokens (referred to as **simple Petri nets** here) and **colored Petri nets**. Tokens can not be distinguished in the first model type and are therefore depicted as black dots or just their number in a place. For many complex applications, a more natural and compact description is possible if tokens carry information. This lead to the development of colored Petri nets, of which two variants are covered in Chap. 6.

The remainder of this chapter deals with stochastic Petri nets with identical tokens. A continuous underlying time scale of the stochastic Petri net is adopted. An informal introduction into the class of stochastic Petri nets (SPNs) with a small toy example is given in Sect. 5.1. The dynamic behavior of an SPN is described informally in Sect. 5.2, followed by a formal definition of SPNs in Sect. 5.3. The interpretation of an SPN model in terms of an SDES covers Sect. 5.4. The chapter closes with some historical and bibliographical remarks about Petri nets with identical tokens.

Relevant analysis techniques are presented in Part II. More complex application examples from the fields of manufacturing and communication are presented in Chaps. 13 and 14.

5.1 Introduction to Stochastic Petri Nets

Throughout this section the model class of generalized stochastic Petri nets (GSPN) is introduced informally. This is done along the way of the stepwise construction of a flexible manufacturing system (FMS) modeling example. The model used here is a slightly changed version of a GSPN model presented in p. 208 of [4]. Petri net model construction in a modular way and by refining coarse-grain structures is demonstrated as a byproduct. New model elements are explained when they are necessary. Other variants of stochastic Petri net classes with more transition types than GSPNs are described in the final notes of the chapter.

Tokens in a Petri net model the changing states and locations of objects, while the places they are located in model buffers or attributes of the system. The number of tokens in every place of a model corresponds to the overall system state and is called Petri net **marking**. The natural choice for our example is thus to model parts as tokens, and locations in which they might reside in as places. Transitions model activities that change the system state or object location. When an activity is executed, the transition **fires**. It changes the Petri net marking by removing tokens from input places (the ones that are connected to the transition by a directed arc), and adding tokens to the output places. The number of tokens that are removed and added via one arc is called the **arc cardinality** (or sometimes **multiplicity**). It is written besides the arc in the graphical representation, but the default value of one is omitted. Before it can fire, a transition must be **enabled**, requiring that enough tokens

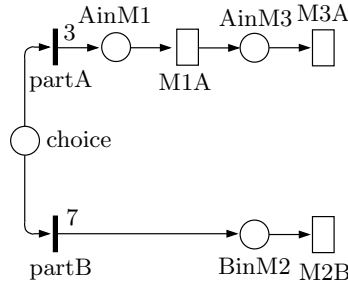


Fig. 5.1. First modeling step for FMS example

exist in its input places. Typical flows of objects like the path of parts along their work plans can be modeled by sequences of transitions and places that correspond to the subsequent production stages.

Figure 5.1 shows a first partial model for the FMS. The flexible manufacturing system has three machines, which are called M1 through M3. It handles two types of products named A and B. Parts of type A go through machine 1 and machine 3, while parts of type B are processed by machine 2. The upper part of the model corresponds to part A in its different stages: a token in **AinM1** means the part is processed by machine 1. The activity of manufacturing part A in machine 1 is modeled by transition **M1A**. Part A in machine 3 and part B in machine 2 are modeled in a similar way by **AinM3**, **M3A**, **BinM2**, and **M2B**.

All manufacturing steps take a certain amount of time. The associated transitions are hence **timed transitions**, which are drawn as rectangles in the figure. The **firing delay** of a transition is the amount of time that needs to elapse between its enabling and firing. In a GSPN, the firing delays of all timed transitions are individual random variables with an exponential probability distribution function (c.f. Sect. 1.4). Each timed transition, therefore, has one parameter that specifies the **rate** of the exponential distribution. Letters λ or μ are normally used to denote these **firing rates**.

New raw material arrives in place **choice**. A decision about what type of part should be produced from one part of raw material needs to be made at that point, before the resulting work plans are started. Activities like this, which typically take no time from the modeling point of view, are modeled by **immediate transitions**. They might take some time in reality, but we decide to abstract from that in the model. We will see later that this distinction between timed and immediate activities can make the evaluation process more efficient.

The firing of transitions **partA** and **partB** decides what kind of part will be produced. Only one outcome of the decision is possible for a token in place **choice**, which corresponds to a **conflict** between transitions **partA** and **partB**. The decision should be made in the model such that a fixed percentage of

parts of each type are manufactured. The conflict resolution can be influenced accordingly by associating a **firing weight** to each of the concerned transitions. In fact, every immediate transition has an associated firing weight, but usually the default value of 1.0 is unchanged. Firing weights of 3 and 7 are set for the two transitions as shown in the figure to achieve the desired behavior. Please note that these values are usually not shown in the graphical model. Later on the detailed semantics of firing weights is defined. Informally, they specify the probability of firing a transition relative to the weights of all other conflicting transitions. Transition **partA** thus fires in $3/(3 + 7) = 30\%$ of all cases.

The first model reflected only a part of the original work plans for the two parts. In fact, the manufacturing steps carried out in machines 2 and 3 can be done on either machine for the two part types. We need to incorporate the variants in the two work plans as well as the decision, on which machine the actual processing is done. Figure 5.2 shows the resulting refined model.

Parts of both types may now be in machines 2 and 3. However, the model needs to distinguish between them. As tokens in simple Petri nets are identical, the distinction must be made by a duplication of the corresponding net portion. The manufacturing place inside machine 2 is now modeled by places **AinM2** and **BinM2** (and similarly for machine 3). This is also necessary to specify the possibly differing properties of both manufacturing steps in one machine using one transition for each operation. For example, processing of part A could take more time than for part B in machine 3. With the duplication it is easy to specify the corresponding firing delays of transitions **M3A** and **M3B**.

Another difference to the previous model is the insertion of four immediate transitions similar to **AsM3**. Its firing models the start of the processing of part A in machine 3. The other three immediate transitions below **AsM3** serve the same purpose for parts A and B in machines 2 and 3. It is often a good idea to separate the start from the ongoing activity in a Petri net model. Without

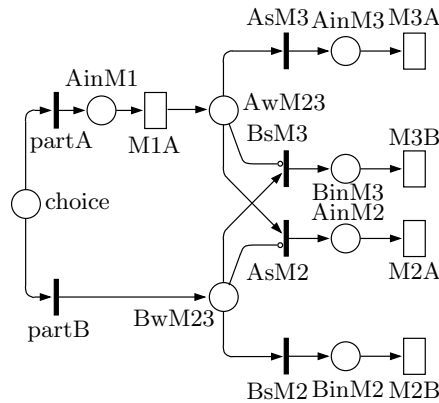


Fig. 5.2. First refinement of the FMS model

immediate transitions, the variants of the work plans would lead to conflicting timed transitions. This would be a modeling error, because then the transition who fires first would disable the other one, resulting in a conflict resolution probability that would depend only on the firing speeds of the transitions. In most models this needs to be treated separately.

An additional advantage of transitions like **AsM3**, which model the beginning of an activity, is that the desired conflict resolution can be specified. This might include a probability of a decision as it has been shown for transitions **partA** and **partB**. In the case of the decision between the two machines in the work plans, the desired behavior is that machine 3 should be used for parts A, and machine 2 for parts B as long as there are any of them waiting. If not, the machine may process parts of the other type. This machine allocation strategy is assumed to be useful in the example because the machines need less time for the mentioned steps. The outcomes of strategies like this one can be evaluated using performance evaluation, to select the best one out of several alternatives (c.f. Part II).

The described machine allocation is achieved in the model using **inhibitor arcs** that connect place **AwM23** with transition **AsM2** and place **BwM23** with transition **BsM3**. Inhibitor arcs always go from a place to a transition, and have a small circle at the transition in the graphical representation. A transition is not enabled in a marking if the number of tokens in the place that is connected by an inhibitor arc is at least as high as the cardinality of the inhibitor arc. In the default case (as in the model), the transition may not fire if there is any token in the connected place.

It should be noted that the same machine allocation behavior could also be reached by associating a higher **priority** to transitions **AsM3** and **BsM2**. Immediate transitions have a priority that specifies which one of them should be fired first in a marking. There is no specific firing priority associated to timed transitions; they all share one priority level, which is less than that of all immediate transitions.

To use firing weights and priorities to specify the later resolution of conflicts, the modeler must be sure to influence the reachability graph generation in the desired way. Both are ways to tell what should be done in cases where different activities are scheduled to be executed at the same time instant. There is thus no timing constraint specifying which one should be executed first. The notion of “first” is a bit strange in that context, because both happen at the same point in time. However, because of the general requirement of atomic action execution, we need to select the one to be executed first. Although the model definition allows for an unambiguous derivation of an underlying semantics, i.e., the stochastic process, the modeler only works at the model level. Firing weights are meaningful only locally inside the sets of transitions (and relative to the other ones) that are in conflict in a marking. Those sets are called **extended conflict sets** (ECS) in the literature. The problem during modeling is now that there is no direct way to specify which transitions should belong to one ECS, except maybe to assign a different priority to

transitions that should go into another one. If the modeler makes a mistake due to an overlooked conflict, the model might not be correct. This is particularly hard to detect manually if a model is structurally constructed in a way that there are transitions which are not in direct conflict, but for which the order of firing leads to different model evolutions. This is mostly due to extended conflict sets in which transitions have different input places, or models with subgraphs of immediate transitions with different priorities. The general problem is known under the term of **confusion** and generally seen as a modeling error. There are approaches that ensure a confusion-free net by analyzing the structure (“at the net level”), or by checking on-the-fly during the reachability graph generation. We will not go into the details of this issue further here; the problem has been extensively studied in the literature [4, 53, 70, 303], to which the interested reader is deferred. This topic is discussed in other settings in this text as well, for instance in Sect. 7.1.

Until now, only the processing of parts has been considered. The model shown in Fig. 5.2 reflects the work plans of the two parts. In addition to that there are transport operations needed in the FMS. Likewise, in communication systems we have packet transfer operations in addition to information processing stages. In our example FMS, an automated guided vehicle system (AGV) and a conveyor move parts between processing stations. Moreover, parts are transported on pallets throughout the system. The according model refinements are included in Fig. 5.3.

Empty pallets that are waiting for raw parts to be mounted are modeled by tokens in place `emptyP` on the left. The loading of a raw part corresponds to transition `loadP`, which takes one pallet token and creates a token in place `loadedP`. Usually such an assembly operation (between pallet and part) would result in two input places of the corresponding transition. However, in our case we assume that no shortage of raw material takes place, thus the always

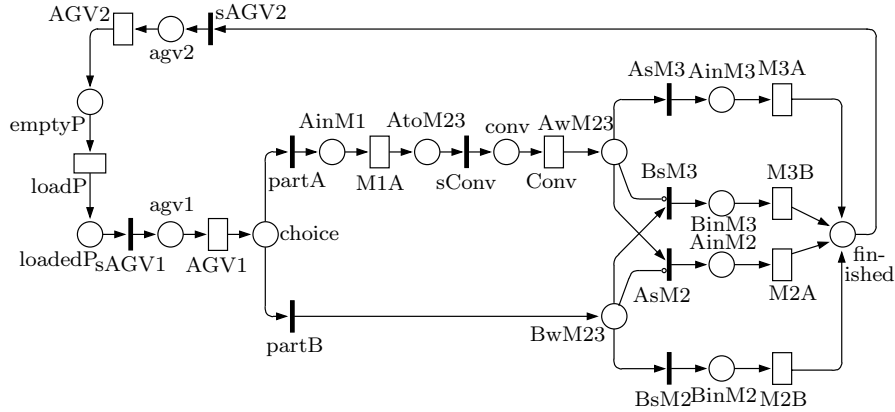


Fig. 5.3. Second refinement of the FMS example

available raw parts do not need to be modeled. This is an implicit input over the model boundaries.

During the first transport operation, a loaded pallet is transported by an AGV from the loading station to the first processing operation of the part. The start of the transport is modeled by **sAGV1**, in-transport state by **agv1**, and the completion and delay by **AGV1**. The second transport moves parts from machine 2 to 3, which is done by a conveyor belt. The corresponding model parts are **sConv**, **conv**, and **Conv**.

Another extension concerns transporting the pallets finally back to the initial buffer. When any one of the production steps carried out by machines 2 and 3 is finished, a part is completed and leaves the scope of the model. The resulting empty pallet in place **finished** is transported back to **emptyP** by the second AGV operation, which is modeled by **sAGV2**, **agv2**, and **AGV2**. Patterns like this circle of moving pallets are often found in Petri net models where objects are transported or change their status, while their total number remains constant.

The model refinement explained so far considers all processing and transport operations. What is still missing are resource constraints. Every operation has been modeled by a starting immediate transition, a place, and a finishing timed transition. We now add one place for every resource type we find in the model, namely each machine, the conveyor, and the AGVs. The start of an operation requires and blocks one resource, while the completion releases it. Therefore, an input arc is added from the required resource of an operation to its start transition, and an output arc from the timed transition back to the resource place. Please note that the model contains only this simple type of resource usage. It is obvious that multiple necessary resources of different types could be modeled in the same way by adding more arcs.

In the case of the two AGV operations, a conflict may arise between the two possible transports, corresponding to a conflict between immediate transitions **sAGV1** and **sAGV2**. The chosen Petri net model pattern assures a mutual exclusion between operations that require exclusive use of a unique resource.

Related to the issue of resource constraints is the specification of the number of resources that are available initially. One token in each of the added places models the availability or idle state of one resource of the corresponding type. The initial marking of the model thus contains as many tokens in every resource place as there are resources available. In our example, every machine as well as the AGV is available only once. The conveyor place models available pallet spaces on the conveyor belt, which is assumed to be three here. The initial marking is depicted by black dots inside the place.

Transition **Conv**, the conveyor transport operation, has a special behavior: the associated operation (and required delay) applies to every pallet that is located on it. This is due to the fact that the transport of one part from the start to the end of the belt needs the same time, independent of how many pallets are otherwise located on it. To simplify the model, we still want to specify the time of one transport as the firing delay of transition **Conv**.

During the model evolution, the conveyor behaves as if there is one individual transport operation for every single part on it evolving in parallel. This type of behavior is often found in modeled systems, and is called **infinite server** (in contrast to the standard **single server**) firing semantics like in queuing models. It is depicted in the figure by **IS** near the transition.

All parts are transported on pallets. The number of available pallets thus limits the overall number of parts in the system. More pallets may lead to a better machine utilization and throughput, but they cost money and increase the amount of work in progress. Place `emptyP` contains empty pallets, for which the initial number is given by `P`. The marking of a Petri net model may be depicted by black dots, a natural number, or an identifier like `P` for which the value needs to be specified for an analysis.

More sophisticated features of GSPNs that have not been used in the example include **marking-dependent** model elements. A **marking-dependent arc cardinality** may be specified, which means that the actual number of tokens to be removed or created in a place is set according to the current model state during the model evolution. The use of marking-dependent arc cardinalities can make a model elegant and readable, but might also lead to a confusing behavior. One should restrict the use to cases in which they are easy to understand, like in the case where all tokens are removed from one place.

Firing delay parameters of timed transitions as well as firing weights of immediate transitions might also be defined depending on the marking. The speed of a transition can, therefore, be changed according to the number of tokens in a place. Timed transitions with infinite server semantics are a special case, because in the simple case they can be treated as a transition where the firing delay parameter is divided by the number of tokens in its input place.

It should be noted that the values of all marking-dependent attributes are computed before the transition fires. This is important because the transition firing might affect the marking from which its own attributes are depending.

Performance Measures

Performance measures need to be added to the structural Petri net model before any meaningful quantitative evaluation. They define what is computed during an analysis. A typical value would be the mean number of tokens in a place. Depending on the model, this measure may correspond to the mean queue length of customers waiting for service or to the expected level of work pieces in a buffer. A set of elementary measures is given below, followed by an explanation of how they are typically used.

For the definition of measures a special grammar is used, which follows the one used in the software tool TimeNET (see Sect. 12.1). A performance measure from the user's perspective is an expression that can contain numbers, marking and delay parameters, algebraic operators, and the following *basic measures*:

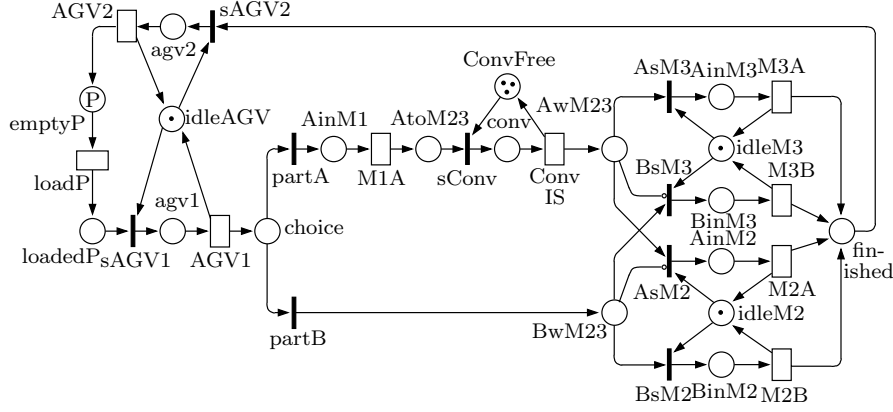


Fig. 5.4. Complete GSPN model of the FMS example

1. $P\{ \langle \text{logic_cond} \rangle \}$ corresponds to the probability of a logic condition, usually containing a comparisons of token numbers in places and numbers.¹ Example: $P\{ \# \text{emptyP} < 3 \}$.
2. $E\{ \# \langle \text{place} \rangle \}$ refers to the expected number of tokens in a place. Example: $E\{ \# \text{conv} \}$
3. $TP\{ \# \langle \text{transition} \rangle \}$ computes the number of transition firings (interpretable as the transition throughput). Example: $TP\{ \# \text{M1A} \}$

Typical ways of using performance measures are now informally explained using the example in Fig. 5.4.

Buffer utilization can be measured with the number of tokens in the place that models the buffer. The number of available empty pallets before loading can be computed using a performance measure $E\{ \# \text{emptyP} \}$. The same type of measures are used for similar questions in different application areas, like communication packets in a buffer, vehicles in a parking lot, and so forth. By adding the values for several places, we can easily calculate the number of objects in a subnet.

Machine utilization corresponds to a resource's probability of being in one of its states, namely to being working. The utilization of machine M2 can thus be computed with a measure $P\{ \# \text{idleM2} \} > 0$, because if there is a token in place `idleM2`, the machine is not working. Similar measures are used if we are asking for resource state probabilities in other applications, like the utilization of a server, a telephone line, or a worker.

¹ Or any marking-dependent boolean function.

Throughput	is also an important issue for quantitative evaluations, thinking of production capacity, communication link load, or traffic intensity. The throughput of machine M1 in our model equals the production of parts of type A (at least in the long run, where it is equal to the added throughputs of transitions M2A and M3A). Both values can thus be computed with a performance measure $TP\{\#M1A\}$.
Processing times	and other delays can be computed indirectly using Little's Law (see p. 70), which states that the number of customers (tokens) in a subnet is equal to the throughput into the subnet times the mean time to traverse it. The overall AGV transportation time for loaded pallets (including waiting for an available AGV and the actual transportation time) can thus be computed as the number of tokens in places loadedP plus agv1 divided by the throughput of transition loadP : $(E\{\#loadedP\} + E\{\#agv1\}) / TP\{\#loadP\}.$

5.2 The Dynamic Behavior of a SPN

A part of the FMS example is used in the following to informally explain the semantics of a generalized stochastic Petri net, which define the dynamic behavior of the model. Figure 5.5 shows the selected part and its first considered marking at the left.

It has already been stressed that places with tokens model states of a system, while transitions stand for operations or actions. The dynamic behavior is determined by the activation and execution of actions, which in turn change the model state. In a Petri net, transition enabling models activation, while transition firing corresponds to execution. The model state is given by the Petri net marking.

Transitions in a Petri net are activated from a structural point of view if there are enough tokens available in their input places. This means that for every input arc directed to that transition, at least as many tokens as the arc cardinality must be contained in the connected place. If there are inhibitor arcs present at the transition, there must be fewer tokens in the connected place than the inhibitor arc cardinality. If all these conditions are true, the transition is said to **have concession**.

A transition with concession is not always allowed to fire in a marking, because there might be other transitions with a higher priority having concession at the same time. Remember that immediate transitions have arbitrary priorities, and are always prioritized over timed transitions. A transition is thus said to be **enabled** in a marking if it has concession and there is no other transition with a higher priority that also has concession. In the left part of Fig. 5.5, only transition **Conv** has concession, and is thus enabled. It is

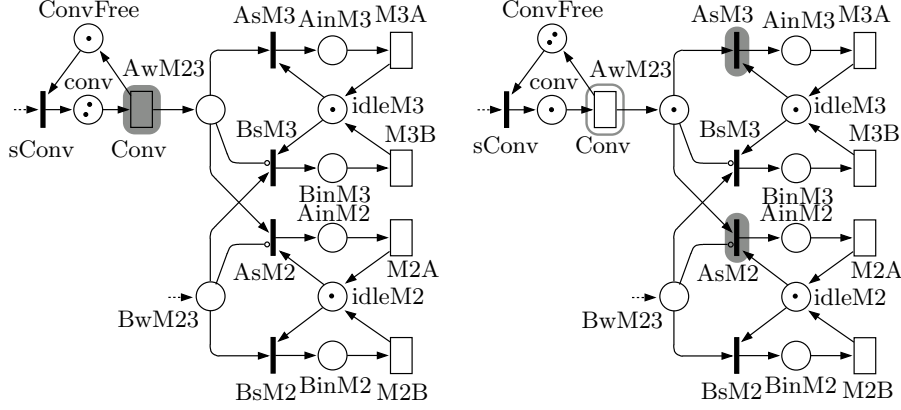


Fig. 5.5. Model behavior: Markings 1 and 2

marked with a grey shadow in the picture to point this out. The transition has no inhibitor arc, and there is a token available in the only input place **conv**.

The execution or completion of an activity corresponds to the **firing** of a transition in a Petri net. A transition fires when it had concession over a period of time given by its delay, provided that there are no other transitions firing first due to their higher priority. During the atomic firing, the number of tokens specified by the input arc cardinalities is taken from each corresponding input place, and the number of tokens given by the output arc cardinalities is added to every output place. Firing transition **Conv** removes one token from **conv**, and adds one token to **ConvFree** and **AwM23**.

The subsequent marking after firing **Conv** is shown on the right of Fig. 5.5. After the firing of a transition, a new marking is reached, and the enabling of transitions must be checked again. In our example, transition **Conv** still has concession, because there is one token left in place **conv**. It is, however, not enabled, because the two immediate transitions **AsM3** and **AsM2** have concession too. The latter are both enabled, because they share the default priority for immediate transitions. Transitions with concession are depicted with a thin grey shadow. Transition **AsM2** is enabled because there is one token in places **idleM2** and **AwM23**, while there is no token in place **BwM23**, which is connected to it by an inhibitor arc. Markings in which at least one immediate transition is enabled are somehow special, because they are left without spending any time in them due to the instantaneous transition firing. Those markings are thus called **vanishing markings** as opposed to tangible markings in which only timed transitions are enabled.

The two enabled transitions both need the token in place of **AwM23** to fire. Therefore, they can not fire both. This situation is called a conflict, and the evolution of the model behavior requires a decision, which one can fire. It has been explained already that conflicting immediate transitions fire with relative probabilities that are derived from their weights. For the two

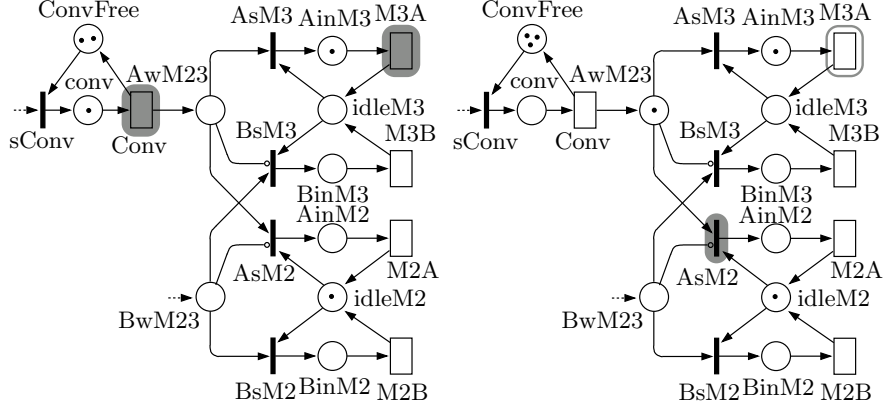


Fig. 5.6. Model behavior: Markings 3 and 4

transitions enabled now, no weight was specified, which means they both have the default weight of one. The probability of firing either one of them is thus 50%.

Let us assume that transition **AsM3** fires, which leads to the marking shown in the left part of Fig. 5.6. Nothing has been changed for transition **Conv**, which therefore still has concession. Because of the token added to place **AinM3**, transition **M3A** gets concession as well. There are no immediate transitions with concession, thus the mentioned two timed transitions are enabled in the shown marking.

For the further model evolution, we need to know the probability of each transition to fire first. Moreover, for the later analysis, it is necessary to derive the time that the model spends in this marking. This is specified by the timing behavior of a timed Petri net. Let us first consider the simplest case in which only one transition is enabled, just like transition **Conv** in the first marking shown in Fig. 5.5. It is clear that the probability of firing this transition first is equal to one, while the time that the model spends in the marking is directly given by the firing time distribution of the transition.

In the general case, a **remaining firing time** (*RFT*) is associated to every timed transition, which counts the time that still has to elapse before the transition fires. In a GSPN this time is always exponentially distributed (c.f. Sect. 1.4), which makes the *RFT* handling much easier because of the memoryless property of the exponential distribution. When a transition becomes newly enabled, this *RFT* is set to a value that is sampled from the firing time distribution of the transition. The *RFT* decreases with model time as long as the transition has concession. When it reaches zero, the transition fires and the *RFT* becomes undefined. If the transition loses concession, the *RFT* is also set to undefined.² If a transition is of type infinite server, one

² If it is disabled temporarily while still having concession, the time does not change.

RFT is maintained as explained for every set of input tokens that lead to an independent transition firing. The resulting **enabling degree** of transition **Conv** in the initial marking was 2, because there are two tokens on its only input place. Hence two independent *RFT* were sampled.

In the subsequent second marking of our example **Conv** still had concession, but was not enabled because of immediate transitions. The enabling degree was one, because of the only token remaining in place **conv**. The *RFT* for the transition was already set in the first marking, and is still running. In the third marking (Fig. 5.6, left) transition **M3A** becomes newly enabled, and a *RFT* is sampled for it therefore. The decision which one of the transitions **Conv** or **M3A** fires first depends on the values of the corresponding *RFT* – the one that reaches zero first (the smaller one) fires. We assume that **Conv** fires in the third marking, leading to the marking shown right in Fig. 5.6. In this final considered marking, transition **M3A** still has concession, but may not fire because of the enabled immediate transition **AsM2**.

5.3 A Formal Definition

A stochastic Petri net can be formally defined as a tuple

$$\text{SPN} = (P, T, \Pi, \text{Pre}, \text{Post}, \text{Inh}, \Lambda, W, \mathbf{m}_0, RV)$$

with the elements described in the following.

P is the set of **places**, which may contain tokens. The **marking** \mathbf{m} of the Petri net associates a (nonnegative integer) number of tokens to each place.

$$\mathbf{m} : P \rightarrow \mathbb{N}$$

The marking can also be viewed as a vector of natural numbers with the size of the number of places.

$$\mathbf{m} \in \mathbb{N}^{|P|} = (\mathbf{m}(p_1), \mathbf{m}(p_2), \dots, \mathbf{m}(p_{|P|}))$$

We denote by M the set of all theoretically possible markings of a Petri net.

$$M = \{\mathbf{m} \mid \forall p \in P : \mathbf{m}(p) \in \mathbb{N}\}$$

T denotes the set of **transitions**, which contains the set of timed T^{tim} and immediate transitions³ T^{im} . It is quite obvious that a node of a Petri net may either be a place or transition, and that a net should not be empty.

$$T \cap P = \emptyset, T \cup P \neq \emptyset$$

³ The partition depends on the firing delay distributions of the transitions and is thus given below.

The priority Π is a function that maps every transition to a natural number.

$$\Pi : T \rightarrow \mathbb{N}$$

Higher numbers mean a higher priority, and only immediate transitions may have a priority greater than zero. The priority thus implicitly defines a mapping to the transition types.

$$T^{tim} = \{t \in T \mid \Pi(t) = 0\} \quad \text{and} \quad T^{im} = \{t \in T \mid \Pi(t) > 0\}$$

In the graphical representation, transitions may be labeled with their priority. This is usually left out for transitions having default priority, namely all timed transitions and immediate transitions with priority equal to one.

Pre describes the multiplicities of the **input arcs** that connect places to transitions. The most general case is a marking-dependent multiplicity of an input arc. Thus **Pre** is defined as a function that maps each place-transition pair together with a marking vector to a natural number (the arc cardinality).

$$\mathbf{Pre} : P \times T \times \mathbb{N}^{|P|} \rightarrow \mathbb{N}$$

For the simple case of a cardinality independent of the marking we write

$$\mathbf{Pre}(p_i, t_j, \cdot) \in \mathbb{N}$$

and if there is no input arc connecting place p_i to transition t_j , $\mathbf{Pre}(p_i, t_j, \cdot) = 0$.

Post denotes the multiplicities of **output arcs** connecting transitions to places. The definition is similar to input arcs.

$$\mathbf{Post} : P \times T \times \mathbb{N}^{|P|} \rightarrow \mathbb{N}$$

Inh specifies the multiplicities of **inhibitor arcs**, similar to the definition of input arcs. A zero value means that there is no arc for a place-transition pair.

$$\mathbf{Inh} : P \times T \times \mathbb{N}^{|P|} \rightarrow \mathbb{N}$$

The **delay** Λ of a transition specifies the time that a transition needs to be enabled before it fires. The delay is defined by a probability distribution function (compare Sect. 1.4) that describes the possibly random delay time.

$$\Lambda : T \rightarrow \mathcal{F}^+$$

W maps each immediate transition to a real number. This value is interpreted as the **firing weight** for immediate transitions.

$$W : T^{im} \rightarrow \mathbb{R}$$

Firing weights for immediate transitions may be written near the transition in the graphical representation, if the value differs from the default 1.

Deg describes the degree of concurrency for each transition.

$$Deg : T \rightarrow \{SS, IS\}$$

SS means **single server** and IS **infinite server**.

\mathbf{m}_0 denotes the **initial marking** of the model. Because \mathbf{m}_0 is a marking, it is of the form

$$\mathbf{m}_0 : P \rightarrow \mathbb{N}$$

RV specifies the set of **reward variables** of the stochastic Petri net model. Three basic types⁴ have been informally introduced in Sect. 5.1. From the mathematical standpoint, there is no big difference between the first two cases, because both are related to rate rewards. If we extend the notion of $\mathbf{E}\{\cdot\}$ from places to any marking-dependent expression, we can express probability-type measures simply by assuming a numerical result of one if the logic condition is true and zero otherwise (as it is done in some programming languages as well, or understood as an indicator variable):

$$\mathbf{P}\{\text{log_cond}\} = \mathbf{E}\left\{\begin{cases} 1 & \text{if log_cond} = \text{True} \\ 0 & \text{otherwise} \end{cases}\right\}$$

Only two types of reward variables are left to be specified after this simplification. This is done by a flag $rtype$ that denotes the type of reward variable, and a parameter $repr$ that either specifies the marking-dependent expression ($\mathbf{E}\{repr\}$) or a transition for which the throughput should be computed in the second case:

$$\begin{aligned} \forall rvar \in RV : rvar &= (rtype, repr) \\ \text{with } rtype &\in \mathbb{B} \\ \text{and } \begin{cases} repr : M \rightarrow \mathbb{N} & \text{if } rtype = \text{True} \text{ (E-case)} \\ repr \in T & \text{if } rtype = \text{False} \text{ (TP-case)} \end{cases} \end{aligned}$$

5.4 An SDES description of SPNs

Remember that a stochastic discrete event system was defined as a tuple

$$\text{SDES} = (SV^*, A^*, S^*, RV^*)$$

The set of state variables SV^* equals the set of Petri net places.

$$SV^* = P$$

⁴ Recall the syntax: **P** for the probability of a condition, **E** for the expected number of tokens in a place, and **TP** for the throughput of a transition.

A marking of the Petri net is equivalent to a state of the SDES. The value of a state variable in a state is given by the number of tokens in the corresponding place in the marking.

The set of SDES actions A^* is given by the set of transitions of the Petri net.

$$A^* = T$$

The sort function of the SDES maps Petri net places to natural numbers.

$$\forall p \in P : S^*(p) = \mathbb{N}$$

The set of all possible states Σ is defined by

$$\Sigma = \mathbb{N}^{|P|}$$

The condition function is always true, because there are no restrictions on the place markings.

$$Cond^*(\cdot, \cdot) = \text{True}$$

The initial value of a state variable is given by the initial marking (number of tokens) of the corresponding place.

$$\forall p \in P : Val_0^*(p) = \mathbf{m}_0(p)$$

Transitions of the Petri net are the actions of the corresponding SDES. The priority Pri^* is the same as the one defined for the Petri net.

$$\forall t \in T : Pri^*(t) = \Pi(t)$$

The enabling degree of transitions is either 1 for infinite server transitions, or ∞ for those of type infinite server. The actual enabling degree $VDeg^*$ in a state returns the actual number of concurrent enablings in one state, and equals thus the maximum number of possible transition firings in a state for infinite server transitions. This number can be computed by dividing the number of tokens in an input place of the transition by the arc cardinality of the connecting arc. In the general case of several input places, the minimum over the computed numbers is used. In any case the result needs to be rounded downwards. In the presence of inhibitor arcs this might be a simplification.

$\forall t \in T, \mathbf{m} \in M :$

$$Deg^*(t) = \begin{cases} 1 & \text{if } Deg(t) = SS \\ \infty & \text{if } Deg(t) = IS \end{cases}$$

$$VDeg^*(t, \cdot, \mathbf{m}) = \begin{cases} 1 & \text{if } Deg(t) = SS \\ \min_{p \in P, \text{Pre}(p, t, \mathbf{m}) > 0} \left\lfloor \frac{\mathbf{m}(p)}{\text{Pre}(p, t, \mathbf{m})} \right\rfloor & \text{if } Deg(t) = IS \end{cases}$$

An action of a SDES for a SPN is completely described by the attributes of a single transition of the Petri net. There are no different variants or modes of transitions, there is no need for action variables.

$$\forall t \in T : Vars^*(t) = \emptyset$$

Thus, there is exactly one action mode for each transition, $|Modes^*(t)| = 1$, and we omit to mention the action mode in the following mappings of Petri net attributes to SDES elements.

A transition of a SPN is enabled if and only if (1) there are enough tokens in the input places of the transition, and (2) the number of tokens in places that are connected to the transition by an inhibitor arc does not exceed the arc multiplicity.

$$\begin{aligned} \forall t \in T, \forall \mathbf{m} \in M : Ena^*(t, \mathbf{m}) = \\ \forall p \in P : \mathbf{Pre}(p, t, \mathbf{m}) \leq \mathbf{m}(p) \quad (\text{enough input tokens}) \\ \wedge \forall p \in P : (\mathbf{Inh}(p, t, \mathbf{m}) > 0) \longrightarrow (\mathbf{Inh}(p, t, \mathbf{m}) > \mathbf{m}(p)) \quad (\text{inhibitor arcs}) \end{aligned}$$

The delay of an action is distributed as specified by the Petri net delay function.

$$\forall t \in T : Delay^*(t) = \Lambda(t)$$

With the delay of a transition defined, we can now formally decide whether it belongs to the set of timed or immediate transitions. The latter type fires immediately without a delay after becoming enabled, the firing time is thus deterministically “distributed” with the result always being equal to zero.

$$T^{im} = \{t \in T \mid \Lambda(t) = \mathcal{F}^{im}\}$$

The set of timed transition is defined as the ones not being immediate. However, we require the firing time distribution of timed transitions to not to have a discrete probability mass at point zero, and to have an expectation greater than zero.

$$T^{tim} = T \setminus T^{im} \quad \text{with} \quad \forall t \in T^{tim} : \Lambda(t)(0) = 0$$

The weight of an action is determined by the transition weight in the case of immediate transitions. For timed transitions, SPN do not define an explicit weight, because the probability of firing two timed transitions at the same time is zero. This is due to the fact that the continuous exponential distributions have a zero probability of firing at a given point in time. However, during a simulation, it is not impossible to have two timed transitions who are randomly scheduled to fire at the same instant of time because of the finite representation of real numbers in computers. We assume an equal weight of 1 for timed transitions to resolve ambiguities in these rare cases.

$$\forall t \in T : Weight^*(t) = \begin{cases} 1 & \text{if } t \in T^{tim} \\ W(t) & \text{if } t \in T^{im} \end{cases}$$

If an enabled action is executed, i.e., an enabled SPN transition t fires, tokens are removed from the input places and added to the output places as determined by **Pre** and **Post**. The change of the marking \mathbf{m} to a subsequent marking \mathbf{m}' is defined as follows.

$$\forall t \in T, \mathbf{m} \in M, p \in P : Exec^*(t, \mathbf{m})(p) = \mathbf{m}(p) - \mathbf{Pre}(t, p, \mathbf{m}) + \mathbf{Post}(t, p, \mathbf{m})$$

The set of reward variables of the Petri net RV is converted into the set of SDES reward variables as follows. The last two elements $rint^*$ and avg^* specify the interval of interest and whether the result should be averaged. Both are not related to the model or the reward variable definition of the Petri net, but correspond to the type of results one is interested in (and the related analysis algorithm). Hence they are set according to the type of analysis and not considered here. There is one SDES reward variable for each Petri net reward variable, such that

$$RV^* = RV$$

and the parameters are set as follows:

$$\forall rvar \in RV^* : \left\{ \begin{array}{l} rrate^* = rexpr, rimp^* = 0 \\ rrate^* = 0, \forall t_i \in T : \\ rimp^*(t_i) = \begin{cases} 1 & \text{if } t_i = t \\ 0 & \text{otherwise} \end{cases} \end{array} \right\} \quad \begin{array}{l} \text{if } rvar = (\text{True}, rexpr) \\ \text{if } rvar = (\text{False}, t) \end{array}$$

This ensures that in the first case the rate rewards are earned according to the value of the marking-dependent expression of the Petri net, while in the second case an impulse reward of one is collected if the measured transition fires.

Notes

The foundation of Petri nets has been laid in Carl Adam Petri's Ph.D. thesis [263] on communication with automata in the early 1960s. One of the main issues was the description of causal relationships between events. Later work showed how the resulting models can be used to describe and analyze concurrent systems with more descriptive power than automata. The earliest results as well as the first books [262, 279] focus on qualitative properties and structural analysis. Petri nets were later used in engineering applications and for general discrete event systems, some selected books include [80, 88, 150, 290]. Petri nets together with their related way of interpreting, modeling, and analyzing systems characterizes them as a *conceptual framework* or *paradigm* [291].

As with SDES model classes in general, there are different subclasses of Petri nets that are easier to analyze in exchange for certain restrictions in their modeling power. Petri nets in which every transition has exactly one input and one output arc are called **state machines**. The dual in which the number of input and output arcs connected to every place equals one is coined a **marked graph**. They only allow either conflicts or synchronization, respectively. **Free-choice nets** allow conflicts only between transitions with the same input places. **Place/Transition nets** are not structurally restricted, but do not contain extensions such like priorities, inhibitor, or flush arcs. The mentioned additions extend the expressive power of Petri nets to the one of Turing machines, but hinder some structural analysis techniques. This is, however, not an issue for a quantitative evaluation as intended in this text.

Time has been added to Petri net models starting in the 1970s [238, 275, 289]. The use of random firing delays and the relation between reachability graphs and Markov chains in the case of a memoryless distribution was subsequently proposed by different authors [246, 248, 251, 300]. A literature overview of timed Petri net extensions is given in [311].

Many classes of stochastic Petri nets (SPNs) with different modeling power were developed since their first proposal. Firing delays of transitions are often exponentially distributed because of their analytical simplicity. The class of generalized stochastic Petri nets (GSPNs, [4, 53]) adds immediate transitions which fire without delay. The underlying stochastic process of a GSPN and other variants of stochastic Petri nets is a continuous-time Markov chain. Those *Markovian SPNs* are well-accepted because of the availability of software tools for their automated evaluation. Chapter 12 lists some of them. However, the assumption of a memoryless firing delay distribution is not realistic in many cases and can lead to significant differences for the computed measures. Transitions with deterministic or more generally distributed firing delays are needed for the modeling of systems with clocking or fixed operation times. Many technical systems of great interest from the fields of communication, manufacturing, and computing belong to this class.

Examples of *Non-Markovian SPNs* are deterministic and stochastic Petri nets (DSPNs, [6, 229]) and extended deterministic and stochastic Petri nets (eDSPNs, [67, 130]), also referred to as *Markov regenerative SPNs* [57]. DSPNs add transitions with fixed firing delay to the class of GSPNs, and eDSPNs increase the modeling power further by allowing transitions with expolynomially distributed firing delay. Expolynomial distributions can be piecewise defined by exponential polynomials (cf. Sect. 1.4).

Stochastic Petri nets with a discrete time scale have been proposed in [58, 247] among others, reference [311] contains a more thorough bibliography on discrete-time Petri nets. This model type was later extended to discrete (time) deterministic and stochastic Petri nets in [337, 338]. The latter are comparable with DSPNs w.r.t. their modeling power: immediate and deterministic transitions are allowed, as well as geometrically distributed delays. The notes on p. 154 briefly comment on the analysis of models with

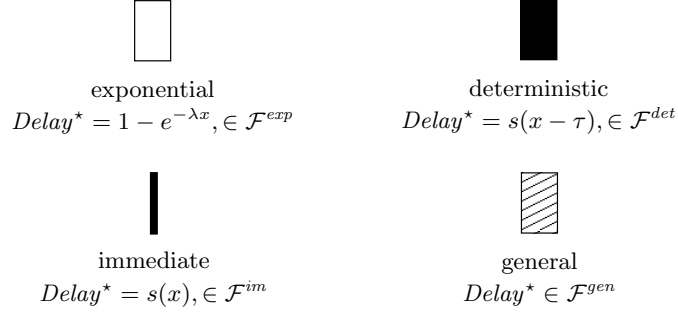


Fig. 5.7. Types of transitions in a stochastic Petri net

an underlying discrete time scale. A good overview of Petri net performance models and analysis methods is contained in [15]. Please refer to e.g., [130] for a more thorough annotated list of relevant references.

It has been mentioned above that the class of a stochastic Petri net (and thus the available analytical methodology) depends on the used types of transitions. Figure 5.7 depicts the most important ones found in the literature and used in this text. It should be noted that despite the various graphical appearances the *structural* behavior of all these transitions is the same; they only differ in the associated firing delay distribution.

Immediate transitions are drawn as thin black rectangles, and their firing delay is zero. The other ones are called **timed transitions**, and contain types where the firing delay is exponentially distributed, deterministic, or general. Please refer to Sect. 1.4 for more details and the set of allowed general delays.

The presentation in this chapter has especially been influenced by the work on modeling and analysis of GSPNs [4] and eDSPN models [130].

Evaluation techniques for stochastic Petri nets are explained in Part II of this text. GSPN models of manufacturing systems are used in Chap. 13 for an automated derivation of heuristically optimal parameter sets. Chapter 14 models and analyzes a train control application example with eDSPNs. Creation and analysis of Petri net models requires a software tool for nontrivial examples. Chapter 12 contains more information on that issue, including the software tool TimeNET [359], which has been used throughout this text.



<http://www.springer.com/978-3-540-74172-5>

Stochastic Discrete Event Systems

Modeling, Evaluation, Applications

Zimmermann, A.

2008, XVI, 392 p. 86 illus., Hardcover

ISBN: 978-3-540-74172-5