29

Petri Nets and AI Arbitration

Sergio Ocio Barriales

29.1 Introduction

A Petri net is an abstract, formal model of information flow in systems, particularly in those in which events can occur concurrently and where some form of synchronization or ordering is required.

In a video game, there are a variety of situations that require some sort of coordination or arbitration to decide what a group of agents should be doing and make sure their actions do not invalidate their peers'. Deciding who gets to use a special resource (e.g., a mounted gun) or how roles are selected in a combat scenario are examples of problems Petri nets can help resolve. Also, since these nets are represented as graphs and, at a first glance, can look similar to FSMs, they are easy for AI developers who are familiar with that approach to understand.

In this chapter, we will talk about Petri nets and how they can be used for arbitration in multiagent scenarios.

29.2 Petri Net Basics

Petri nets are a graphical and mathematical modeling language used to describe how information flows in a distributed system. They were developed by Carl Adam Petri in 1962 (Petri 1962). A Petri net is a graph built using two different types of nodes: *places* and *transitions*. Places are connected to transitions via directed *arcs* and vice versa, but nodes of the same type can never be connected directly. A *place* represents a condition and a *transition* is a gate between places. There is a fourth element involved, a *token*. Tokens are found inside places, and a single place can hold multiple tokens; when a token is present in a place, it means that the condition associated with that place is met.

In a Petri net, execution is controlled by the position and movement of the tokens. For example, in Figure 29.1a we have a single token in *p*1, which means that this node is the only one that is currently active.

Transitions have a number of preconditions, or input places. A token in an input place is interpreted to mean that the precondition is true. Transitions are fired when they are enabled; a transition is enabled when it has tokens in each of its input places. For example, in Figure 29.1a, *t*1 is enabled and can fire. Figure 29.1b shows the state of the net after the first transition is triggered. When a transition fires, it consumes the tokens in its input places and generates a new token for each of its output places. This new state—also known as a *marking*—of the net enables *t*2; *t*3 is not yet enabled, since it is missing a token in *p*3. The execution continues in Figure 29.1c. Finally, *t*3 is enabled, since both *p*3 and *p*4 have tokens. The net keeps running and *p*5 receives a token, as shown in Figure 29.1d.

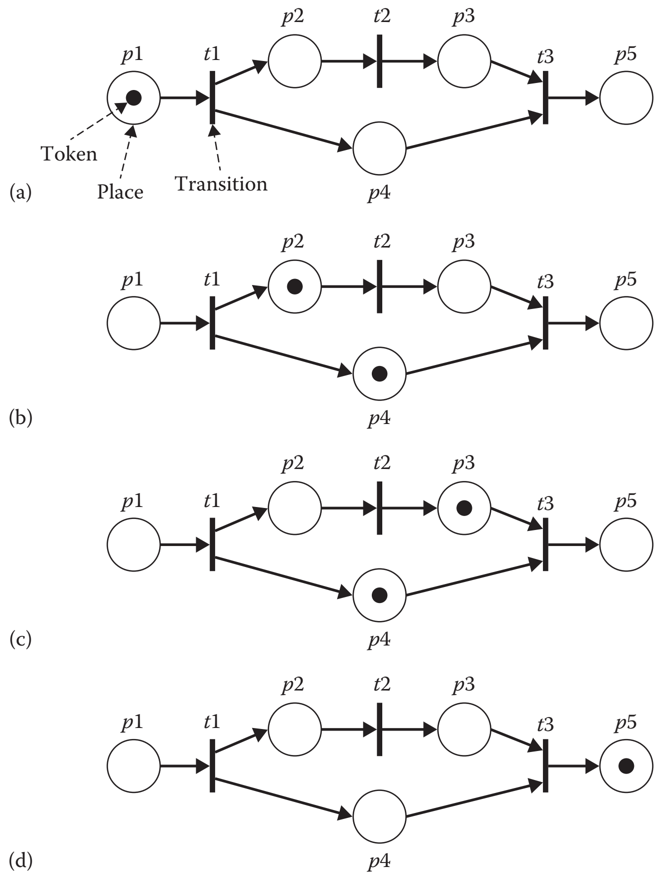

Figure 29.1

An example Petri net with five places and three transitions.

Running a Petri net becomes more complicated if multiple transitions are connected to a single place. In this case, the evolution of the network is no longer deterministic; if we have one token in the place, every transition will be enabled. If we fire any of the transitions, the token will be consumed, invalidating the remaining, previously enabled transitions. In this case, we say the transitions are *in conflict*. Depending on which one we choose to trigger, we will have different resulting markings, as shown in Figure 29.2.

Arcs can be labeled with the number of tokens a transition requires in a particular input place before it can be enabled. After the transition fires, it will consume that number of tokens from the input place. Likewise, a transition can generate multiple tokens if the arc to the output place shows the label. This is depicted in Figure 29.3.

We can also have transitions without any input place—*source* transitions—that are unconditionally enabled and whose sole purpose is generating tokens, and transitions with no output place—*sink* transitions—that only consume tokens.

This section has only presented a few examples and key concepts, but a vast amount of work has been done with Petri nets over the past 50 years. Many resources are available on them; the articles by Peterson (Peterson 1977) and by Murata (Murata 1989) are good places to start.

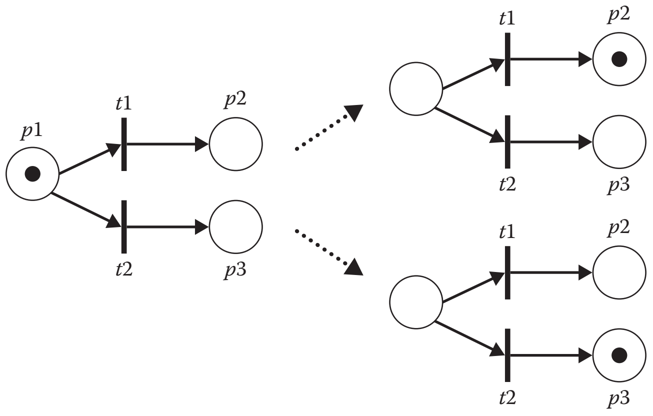

Figure 29.2

Both *t*1 and *t*2 are enabled in this markings; if either fires, the other will be invalidated.

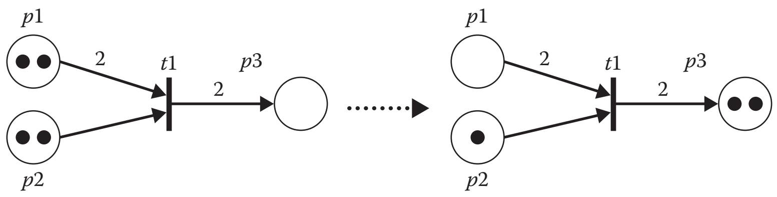

Figure 29.3

Transitions can consume and generate multiple tokens as required.

29.3 An Example Arbitration Scenario

Now that we know the basics, let us discuss how we can use Petri nets to coordinate multiple characters in a setup where they must utilize a finite number of nonshareable resources (i.e., resources that can only be used by a single agent) to accomplish their goal. To do so, we will choose an example and analyze how we would model it with Petri nets.

Let us depict a scenario where a group of agents that were unaware of enemy presence are suddenly attacked by the player. After an initial reaction, the agents must decide how to deal with the attacker. A quick analysis of the surroundings reveals there is a mounted gun nearby, and the AI decides controlling that weapon is the key, but only one character can use the special weapon at a time. So how do we decide who goes first?

One option is to use a first-in, first-out solution, so the first agent that gets updated by the system selects and locks the gun, whereas the others select other actions. However, this could lead to the AI that is farthest from the weapon being chosen, making the characters look stupid and inefficient. We could also modify this approach and have each AI ask the rest of the group "is there anyone closer than me to the gun?" and skip the assignment until the closest agent is updated and selected. This method generates a better looking result, but the behaviors and decision-making logic of our agents gets polluted by these interagent communication requirements.

Trying to resolve every potential scenario by having individual behaviors take into account every other possible AI and their desires and intentions can be problematic. Having a higher level AI entity—that we will call *arbiter*—help agents resolve these resource management disputes can help simplify the system. It is the arbiter's job to track and manage resources and assign them to the appropriate actors.

Going back to our example, a Petri net controls the arbiter. For simplicity, we will just focus on assigning the mounted gun, and will not model how other points are chosen for the NPCs, so the net will just queue AI agents' requests and put them on hold until the special weapon is available. In a more complete solution, our agents would not really directly know about the mounted gun—they would be running a behavior that would try to use the best possible point to attack the enemy, and this point would be returned by the arbiter. Agents would not be reserving this point directly, but just registering to be assigned the best possible one.

Initially, this net's marking is as shown in Figure 29.4a, a single token in the "gun available" place, which indicates nobody has requested using the gun and thus it is still unused. When the agents in the group start reacting to discovering the enemy, they register with the system. For each registered actor, we get a new in the "ready to assign" place. Once the "n" agents in the group are registered, the "start assignment" transition is enabled and run, getting our "n" tokens transferred to the "ready to use gun" place. This is shown in Figure 29.4b and c.

In order for the "assign gun" transition to run, we need the gun to be available and at least one actor to be ready to use it. When both conditions are met, the transition runs some logic to select the best agent to fill the role—based on factors such as proximity to the gun, archetype and capabilities (e.g., an AI agent with better accuracy would be preferred)—and a token will be transferred to "gun busy," while the "gun available" and one of the tokens "ready to use" are consumed. We show this in Figure 29.4d.

If at any point the agent manning the gun is incapacitated, the "agent incapacitated" transition will trigger, moving the token to "gun available," as shown in Figure 29.4e. As long as we have other agents to reassign, the process will continue, just as depicted in Figure 29.4f.

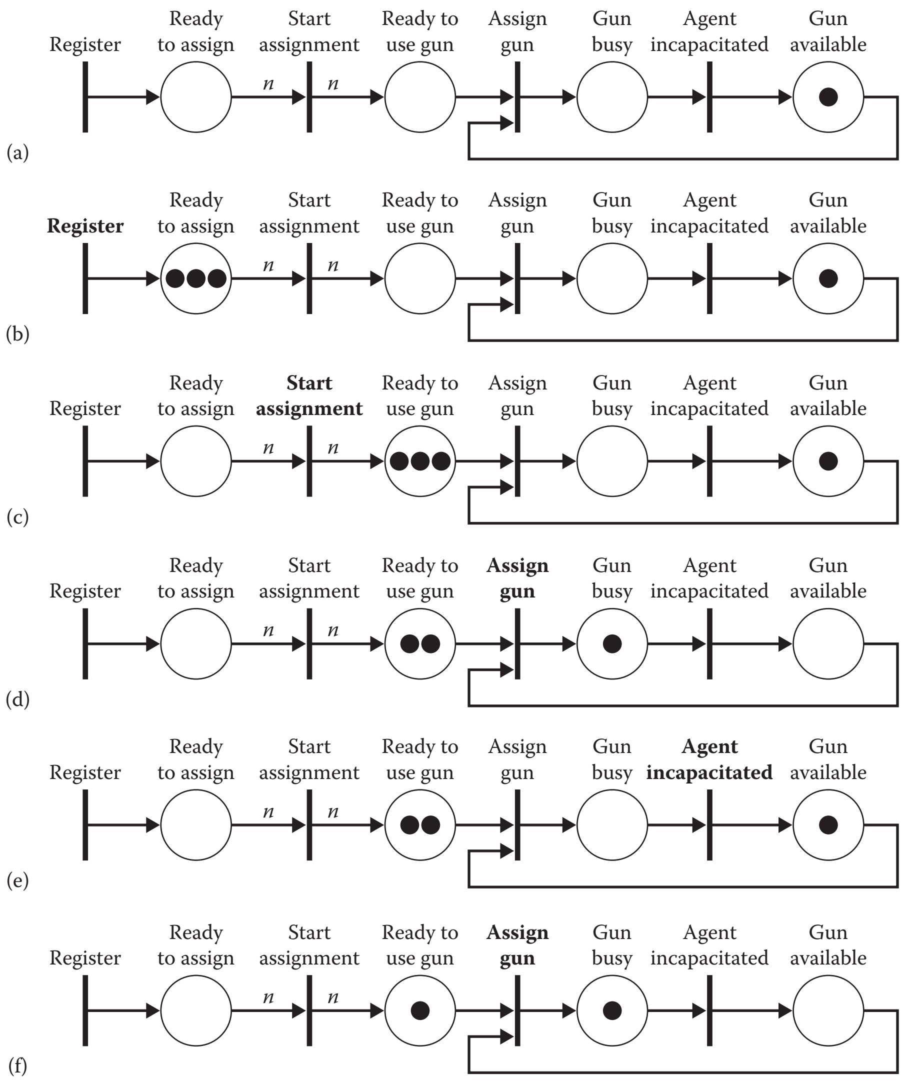

Figure 29.4

The arbiter runs a simple Petri net that controls the status of the mounted gun, allowing other agents to register and get their turn as the weapon becomes available.

29.4 Conclusion

Petri nets are powerful tools that offer a simple way to describe processes where actions need to be synchronized or depend on one another. They have been applied to a broad set of problems, from manufacturing to chemistry and beyond.

In this chapter, we have presented the basics of what the Petri net model offers and how to apply them to model resource arbitration for multiagent scenarios. By using a Petri net, we can separate group coordination to use shared resources from individual behaviors, leaving the high-level decision-making and agent synchronization in the hands of the net. This greatly simplifies the complexity of our single agent behaviors.

References

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

Peterson, J. L. 1977. Petri nets. *ACM Computing Surveys* (*CSUR*), 9(3), 223–252.

Petri, C. A. 1962. *Kommunikation mit Automaten*. Bonn, Germany: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962, Second Edition, New York: Griffiss Air Force Base, Technical Report RADC-TR-65-377, Vol. 1, 1966, Suppl. 1, English translation.