Chapter 1

Introductory Chapter: Petri Nets in Science and Engineering

Raul Campos-Rodriguez and Mildreth Alcaraz-Mejia

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.79309

1. Introduction

The Petri nets are one of the most widely used methods for the study of the dynamics that falls within the category of Discrete Event Systems (DES) [1]. The DES is a class of systems that are guided by the occurrence of events asynchronous in time, which are becoming more and more relevant nowadays. The Petri nets are graphically represented as a directed graph, with two classes of nodes, called places and transitions. The places allow capturing the state of a system. They also represent the conditions required by the events to occur, or to execute, in the DES.

The transitions represent the events, or actions, executed in a system. The execution of the transitions may require one or more conditions to be activated. Moreover, it is possible that a transition does not include input places, as t_1 in **Figure 2**. This class of transitions allows capturing situations in a DES where an event may be random or stochastic, for example, the arrival of an information package in a communication channel. The explanation of the Petri net in **Figure 2** will be addressed lather in this section, after the introduction of the system that it represents.

Figure 1 depicts a conceptual diagram of a multitasking manufacturing system [6]. The system is supplied with the raw material from two conveyors, C_1 and C_2. A robot arm distributes the raw material to either a mill machine or to a lather machine, depending on the manufacturing recipe. The semi-finished pieces are then moved by transporting bands to the assembly machine.

Figure 2 depicts a Petri net model for this multitasking manufacturing system. The supply of the raw material is represented as two transitions with no inputs. In means the material may arrive at any time that the inventory of raw pieces is able to feed the manufacturing system. The robotic arm moves the raw pieces to the mill machine, by means of t_4, or to the lather machine, by means of t_5. The semi-finished pieces are moved to the assembly station to produce a final product.

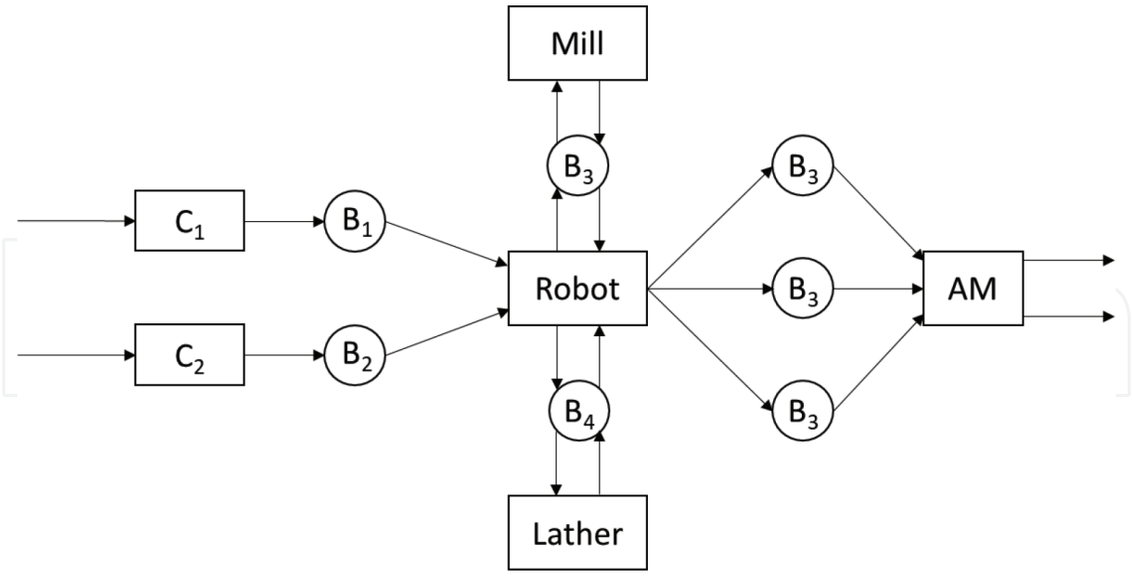

Figure 1. A multitasking manufacturing station. The system is supplied with raw material from two inventories C_1 and C_2. A robotic arm moves the raw pieces to the mill machine or to the lather machine depending on a manufacturing recipe. The robotic arm then moves the semi-finished pieces to the assembly station (AS), where final products are produced.

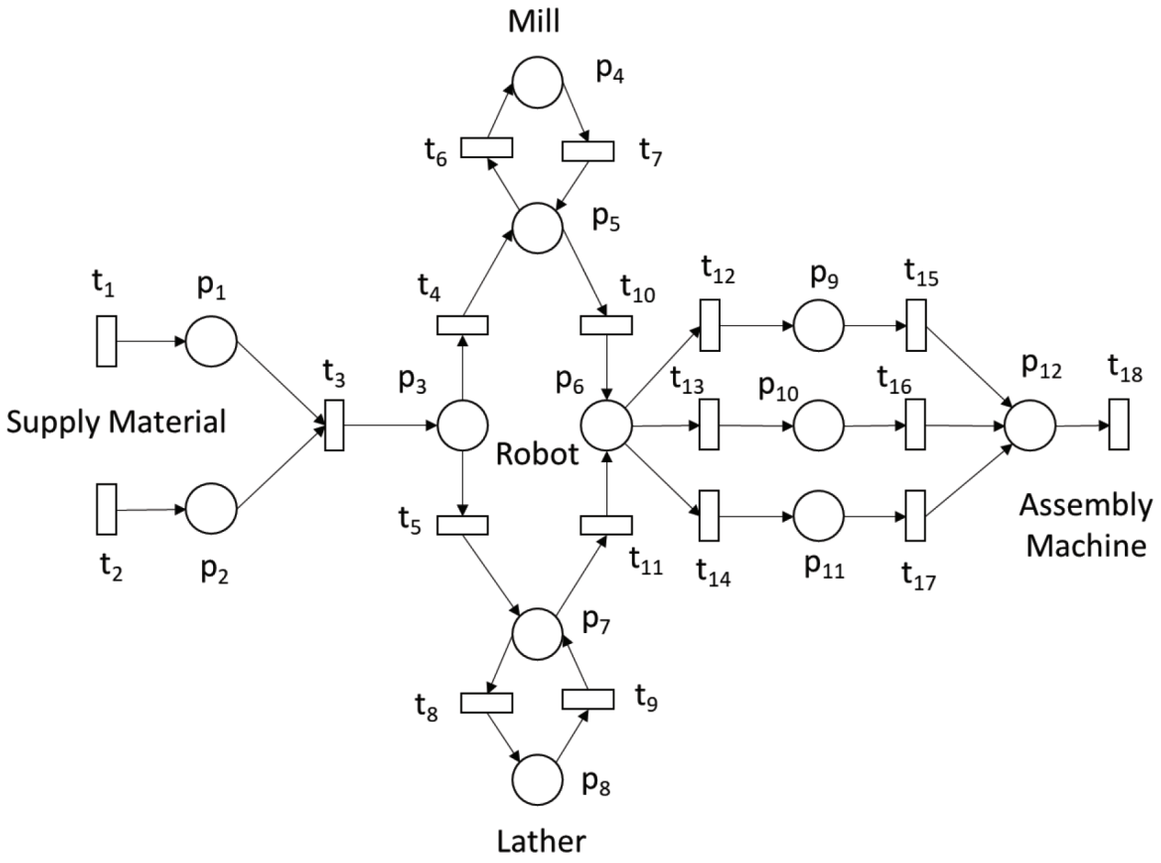

Figure 2. A Petri net model for the multitasking manufacturing system. The model is divided into a supply section, a robot section, lather and mill sections, and assembly section. The supply raw material is handled by a robotic arm that moves it to the lather (t_5) or mill (t_4) machine depending on a recipe. The semi-finished pieces are routed to the assembly machine by three different ways. Once the final product is assembled, t_{18} moves the products to the store section.

http://dx.doi.org/10.5772/intechopen.79309

Depending on what is the interest of study of the system, for example, the design of control strategies or the evaluation of performance of the assembly recipe, the model in **Figure 2** could be refined or extended. Even new sections of the assembly system may be added to the Petri net model.

The manufacturing system and assembly lines, as well as communication protocols, are some of the most popular type of systems that are modeled and studied with Petri net models [8–10]. However, other types of systems such as workflow management or logistic systems are similarly likely to be modeled and studies by means of Petri nets [15–17]. Moreover, the design and implementation of complex software systems is as well plausible to be addressed with Petri net models [18, 19, 26].

The addressing of software design with Petri nets is popular because the construction of models for complex structures and control flow is quite intuitive thanks to its graphical nature. Moreover, the techniques developed around the Petri nets allow the construction of models that are usually more compact than the produced by other methods, such as those developed in graph theory. However, Petri nets and graph theory are not antagonist. On the contrary, the theory developed in one of them is usually extended to the other. Thus, they are usually complementary to each other.

Figure 3 depicts a block diagram of a reader and writer problem in computer sciences. The processes share a region of memory where they can read and write. The diagram depicts the process that can read, process that can write, and process that perform both operations, reading and writing to the shared memory region. This situation arises in several cases in the design of monolithic and distributed system, within the area of software design.

Figure 4 depicts a model for the above problem of readers and writers [2]. The net represents a system with $2k$ readers modeled as p_2. The system allows up to k parallel reads from a shared memory region. It is represented by the marking in p_3. However, the writing operation

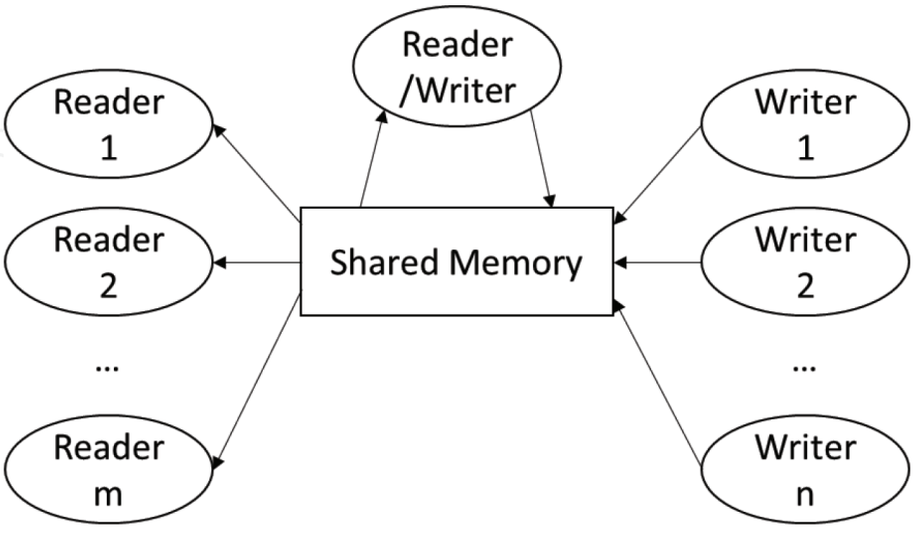

Figure 3. The problem of readers and writers. The problem considers a set of process that can read and write to a shared memory region. The system must allow any number of simultaneous reading operations, while the writing operation requires that no reading operation is in execution. On the other hand, when a writing operation is in execution, no other writing, nor reading, operation is allowed.

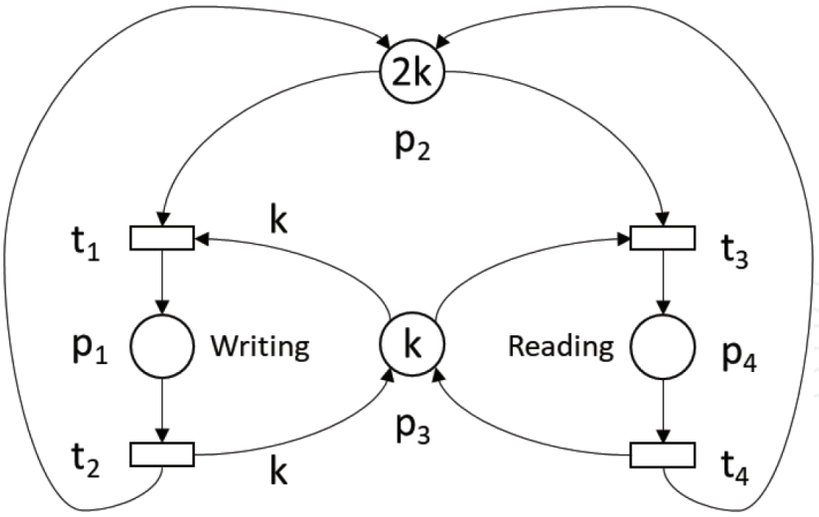

Figure 4. A Petri net model that represents the typical problem of readers and writers. The model allows up to 2*k* processes (p_2) that can read or write to and from shared memory resources (p_3). However, only *k* of them can be concurrently in a reading operation. On the other hand, the writing process requires k tokens to be on the shared place p_3. That is, no reading operation must be executed by any of the 2*k* readers in p_2 to allow the writing operation. Thus, the writing operation must wait until all the reading operations have finished. Similarly, when a writing operation is in execution, no reading operation is allowed, since p_3 is empty.

requires *k* tokens on p_3. That is, it requires that no reading operation is currently in execution. Correspondingly, when a writing operation is in execution, no read operation is allowed. This is represented by *k* weighted arc of t_1. Thus, when writing operation is in execution, by the firing of t_1, the *k* tokens in p_3 are removed. Once the reading operation is done, the firing of t_2 returns *k* tokens to p_3. The Petri net model allows any of the 2*k* processes to read and to write to the shared place p_3 by connecting the place p_2 to the reading or writing sections of the net.

Other attractive attribute of the Petri nets is their solid mathematical basis. The incidence matrix that represents the structure of the net in **Figure 4** is represented by Eq. (1). The incidence matrix is independent of the initial condition of the net. This structure could be analyzed by methods from the matrix theory, linear algebra, or vector spaces, for example.

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -1 & 1 \\ -k & k & -k & k \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{1}$$

The state equation of a Petri net allows a formal definition of its dynamics. The next state of a Petri net can be computed from the current state, and a multiplication of the matrix that represents the structure of the net and a vector that represents the transitions that can fire, as follows:

$$M_{k+1} = M_k + B\vec{u_k} \tag{2}$$

The vector $\vec{u_k}$ represents one or more transitions that are allowed to fire. It is known as the Parikh vector, in a clear relationship to the Parikh's theorem. This theorem relates the strings in a context-free language and the number of the occurrences of the symbols in these strings.

In a similar way, the vector $\vec{u_k}$ represents the number of times each transition is fired at a given stage in the evolution of the net. In this sense, the Parikh vector behaves like a "functor" in the sense of the category theory [3], from the strings over the alphabet of events in a DES to vectors that quantifies the occurrence of events in a DES. That is, the Parikh vector "loses" the execution order of the events in a trajectory of a DES to obtain a pure vector which is simpler to operate by a matrix multiplication.

There are different semantics for the execution of the transitions in a Petri net model. First, in a single firing semantics, only one of the enabled transitions can fire at a time. Second, in the multiple firing semantics, all the enabled transitions are allowed to fire at a time. In all the semantic approaches, the conflicting transitions, that is, the ones whose firing disables the firing of others are resolved by priorities, by a probability distribution, or some other conflict resolution mechanism. Depending on the adopted semantics, the ability of the models to capture dynamics of real systems differs. For example, if the analyzed system is of distributed nature, such as a cluster of computers or a cloud service, then the correct semantics is that of multiple firing. The expressiveness of the different semantic mechanisms is a theoretical question that lies around the computer sciences. The next subsections detail some theoretical aspects of the Petri nets and application in science and theory of systems.

2. Petri nets in science

As mentioned, the Petri nets are a very versatile tool that turns it useful in science, as well as in engineering. In the science field, a wide developed aspect is related to the study of Petri nets as a system and their associated abstract properties.

For example, the study of properties of the Petri net models in terms of vectors and matrices is complemented with the linguistic study in terms of strings and formal languages. The well-developed theory of matrices, linear algebra, and vector spaces are well suited to the analysis of properties in the net models, providing efficient solutions. However, other studies such as the reachability analysis, requires the partial expansion of the state space of the models, which turns the investigation in an inverse direction, from a vector space to string over a language. Though, the advantages of the study of the Petri net properties in terms of vectors, matrices, and linear algebra in general are considerable, and many of the theory developed for Petri nets relies on them.

Indeed, by restricting the marking of a Petri net to be non-negative, the state space entirely lies in the positive cone of Z^+. Thus, some of the theory of positive linear systems could be applied [7]. **Figure 5(a)** depicts the state space, in R^3, of the Petri net model in (b) for two different initial conditions $M_0 = [1\ 0\ 0]$ and $M_0 = [2\quad 0\quad 0]$, the two hyperplanes are orthogonal to the unitary vector $u = [1\quad 1\quad 1]$. Moreover, if the net is conservative (i.e., the number of tokens over all its places remains constant for any evolution trajectory), then it is easy to show that the entire state space of the Petri net lives in one of the hyperplanes orthogonal to a vector in R^n, where n is the number of places of the net.

One of the most active areas of the applications of the Petri nets in science is in the field of the modern control theory. The study of control techniques for discrete event system, including Petri net models, covers the range of applications from design of discrete event controller, design of state observers, analysis of fault tolerant systems, analysis of Lyapunov-like stability, detectability analysis, isolation, and failure recovery techniques, among others [4–6].

In this context, considering a Petri net as an input-output system, as in classical control theory, is useful. **Figure 6** depicts the state equation of a Petri net given by a block diagram. The input vector $\vec{u_k}$ is operated by the matrix B to produce a marking increment that is added to the current marking. The sum of these two quantities becomes the new marking reached by the current evolution of the net. A unit delay block allows the new marking becoming the current marking for the next evolution of the net.

Considering the state equation of a Petri net as a block diagram as in **Figure 6** allows studying the dynamics of the model as in the control theory [4]. Techniques for the construction of feedback controller or state observers could be addressed [6]. Performance analysis is as well, a usual analysis stage in the design of the class of systems that could be modeled by a Petri net [5].

Figure 7 depicts a block diagram of a classical control scheme for a DES modeled as a Petri net. The scheme considers two models, one of the system and the other of a reference. The controller receives the difference of the output of the system and the reference in order to compute the control actions. The objective of the control scheme is to achieve zero error, by the actions that the controller can exert over the system. If the system to be controlled is a software,

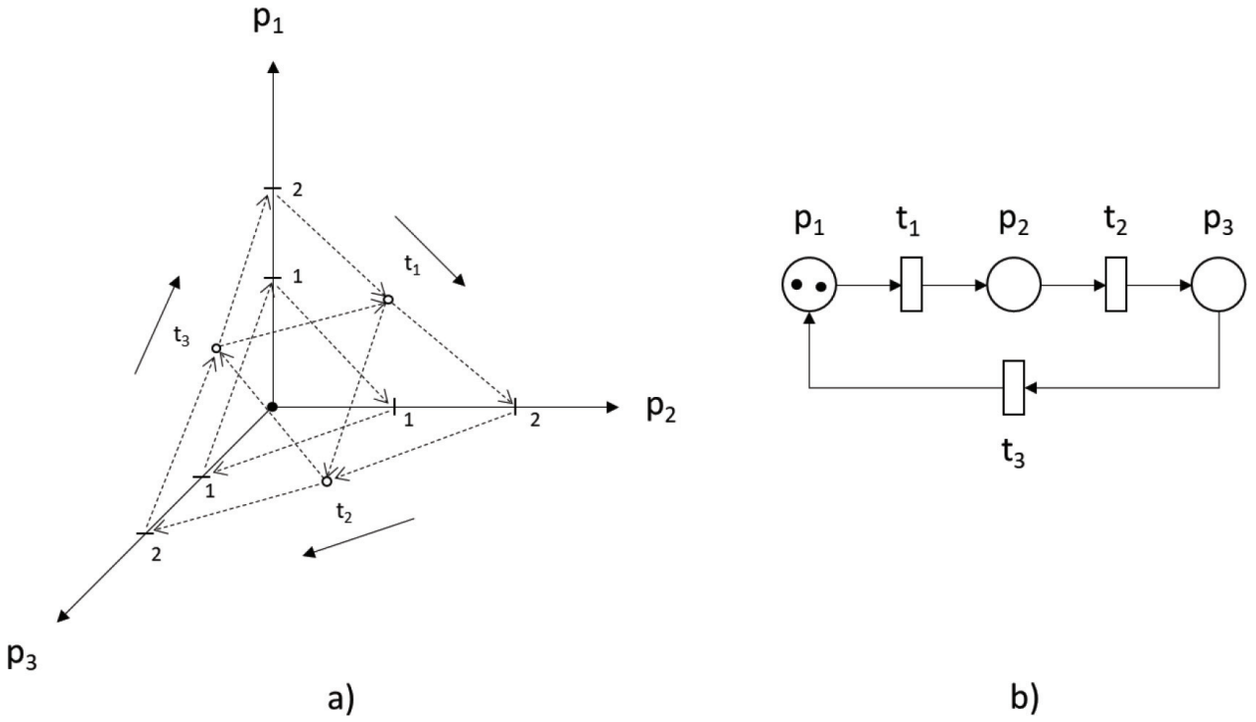

Figure 5. A Petri net model and its respective state space shown as a hyperplane. The solid arrows show the flow of the marking by the firing of the corresponding transition. Dashed arrows represent the marking change by the firing of a single transition. Increasing the number of tokens in the initial marking represents an orthogonal movement of the hyperplane away of the origin. The figure illustrates two hyperplanes. The lower one represents the initial marking with one token at p_1, while the upper one represents the initial marking with two tokens as p_1.

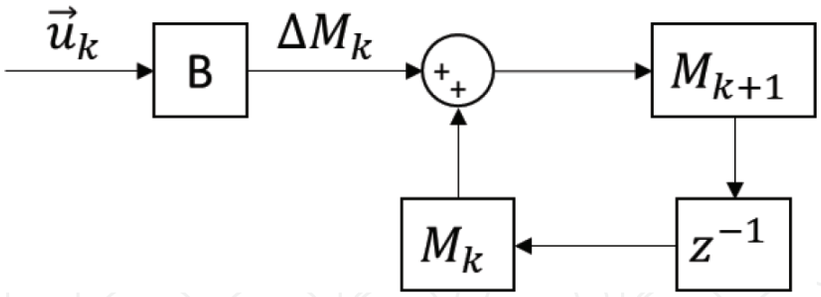

Figure 6. The state equation of a Petri nets as a block diagram. The input vector u_k is multiplied by the matrix B to produce an increment of marking ΔM_k. This increment is added to the current marking M_k to produce the new marking M_{k+1}. This new marking becomes the current marking for the next evolution of the net.

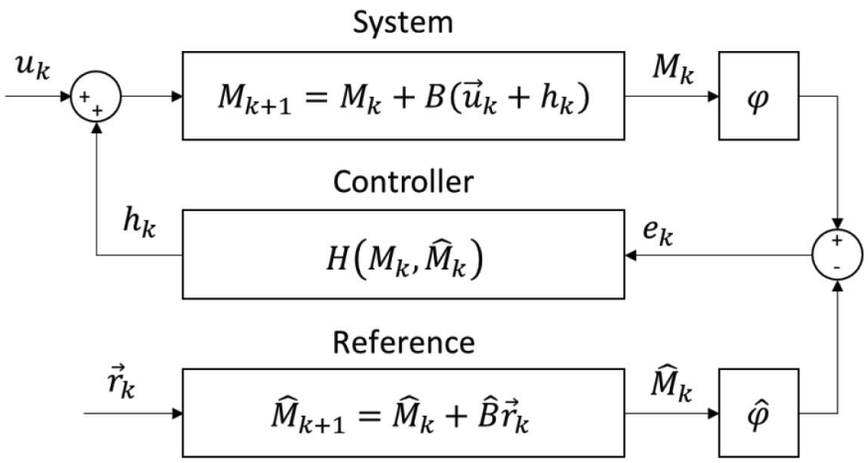

Figure 7. A control scheme for Petri nets. The system is modeled as a Petri net model. The required behavior for the system is modeled as other net model called reference. The objective of the controller is to achieve zero error in the difference of the outputs of the system and reference. If the system is a database software, for example, then the reference is a specification that the database manager requires in the database. The controller is another software that makes the overall scheme autonomous.

for example, either distributed or monolithic, then the reference is a specification or recipe that the software must meet. Then, the controller is another software responsible for computing the required parameters and configurations in order to adapt the main software system to the required behavior in an autonomous fashion. There is a huge trend in cloud computing and artificial intelligence to transform current software systems, such as database clusters, into autonomous intelligent systems that automatically adapts to user requirements and that are even able to predict future workloads and adapt to them [11, 12].

The next section reviews some illustrative use of Petri net models in engineering applications.

3. Petri nets in engineering

The usability of the Petri nets in engineering applications is as well widely accepted. The stages of the design, the implementation, and the validation of systems are suitable addressed with

Petri net models. The covered applications include communication protocols, distributed systems, distributed database, concurrent and parallel programming and systems, industrial control systems, multicore processor platforms, dataflow-computer systems design, workflows and process-driven systems, fault-tolerant systems, and to mention a few. Properties practical interest such as fairness in the execution of tasks, deadlock avoidance, state reachability, process interlocking, among others, are possible to be analyzed within the Petri net framework.

For example, **Figure 8** illustrates a very simple and conceptual communication protocol. The communication act is analyzed from the sending process point of view. A sender process sends a message by the output buffer and blocks its activity while waiting for an acknowledgement by the input buffer. A receiver process is blocked while waiting for an input message. Once a message has arrived, the receiving process reads the message and sends an acknowledgment by the input buffer. After the communication act has finished, the process restarts its logic to be ready for the next communication. This model could be extended to include faulty communication channels, which may lose the messages, acknowledge expiration periods, or other characteristics of practical interest. The analysis and design of communications protocols has been widely addressed with the use of Petri net models [9, 10].

There are some extensions to the Petri nets to handle specific aspects of different engineering problems. Some of the extensions add structure and information to the tokens, transitions, and places of a net. These extensions allow the construction of models that are quite compact

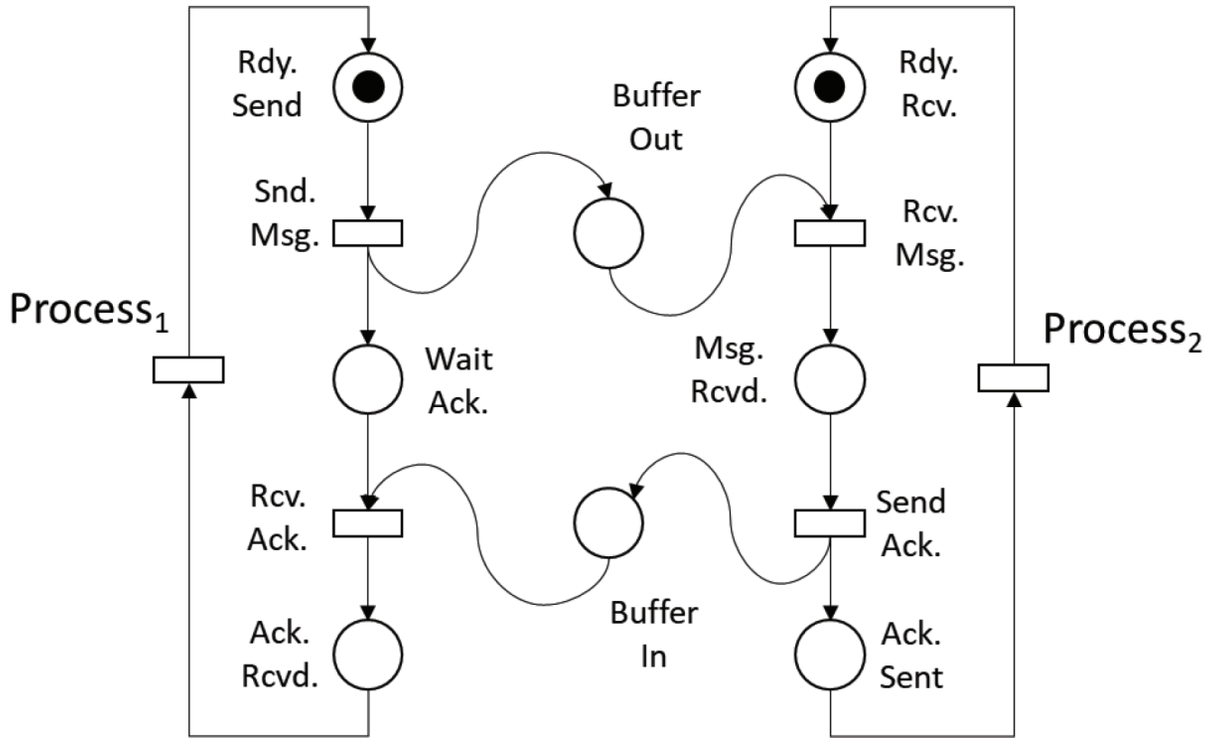

Figure 8. A Petri net representing a communication protocol. Process 1 sends a message by the output buffer and waits for an acknowledgement buy the input buffer. Process 2 reads the message from the output buffer and sends an acknowledgment by the input buffer. After the communication protocol is completed, the both processes restart their logics to get ready for the next communication act.

http://dx.doi.org/10.5772/intechopen.79309

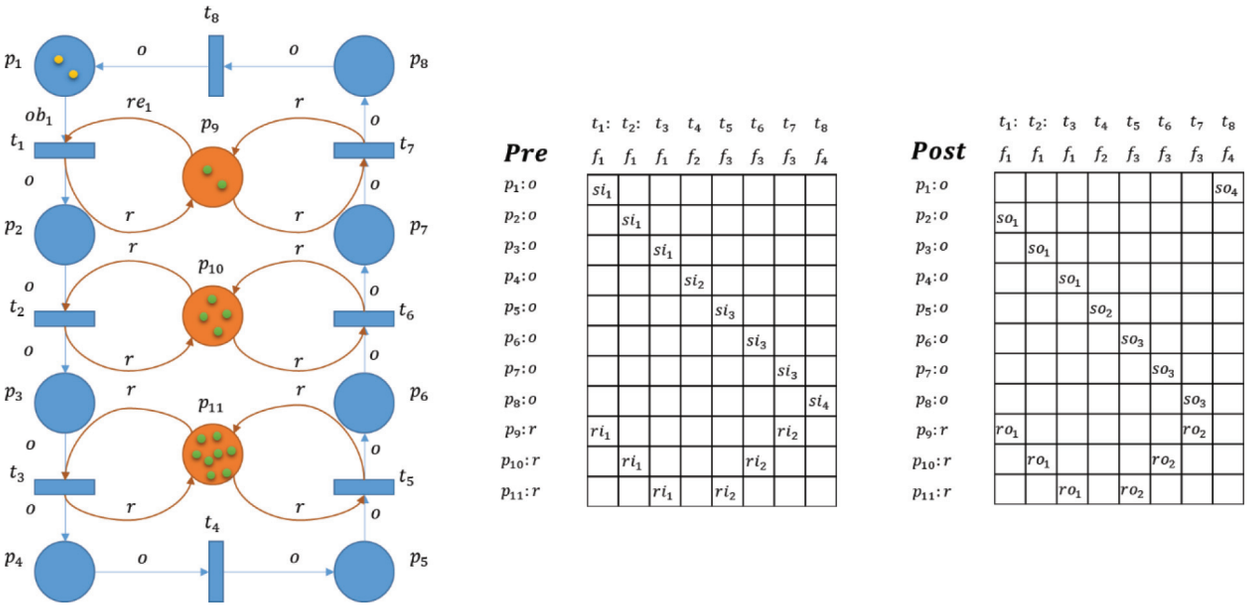

Figure 9. A colored Petri net model of a task scheduling problem. The structure of the net represents the different stages of the processes in a multitasking environment. The working processes compete among them to acquire the jobs that need to be executed. Each token is a composite unit that carries the information about the state of the working process. The simulation of this model allows us to study the performance of different scheduling policies.

compared to the models obtained with the traditional approach. These models are called Colored Petri Net (CPN) [18].

As an illustration, **Figure 9** depicts a CPN for a task scheduling problem. The structure of the net represents the different stages of the working processes in a distributed multitasking environment. The left-hand side of the net structure represents the stages that the processes perform to acquire a job. The right-hand side represents the stages that the working processes perform to release the resources and update the state of the overall scheduling problem. The simulation of the model depicted in the figure allows to study the performance of different scheduling policies over different workload conditions. For example, it is possible to approximate the optimal number of process required by the scheduling problem for a fixed number of tasks. Even more, it is possible to study an optimum rate in the increment of the working processes given a rate in the increment of the tasks over discrete interval of times [27].

Recently, with the increase of the cloud computing and the massive data content in the social networks, the machine learning techniques and the methods related to the data analytics are essential tools in the study and investigation of the big data. There are several proposals to allow the Petri net models learn some kind of fuzzy reasoning and decision making [22–24]. Similar approaches as that of the supervised and unsupervised learning have been addressed [21, 25].

Figure 10 shows a Petri net representing a workflow pattern for a customer reclaim system. The customer may initiate a request at any time. Two activities are launched in parallel once a request is in the system. First, a ticket check process is executed. Second, an examination of the request is performed. At this point, based on the machine learning, data analytics and/or

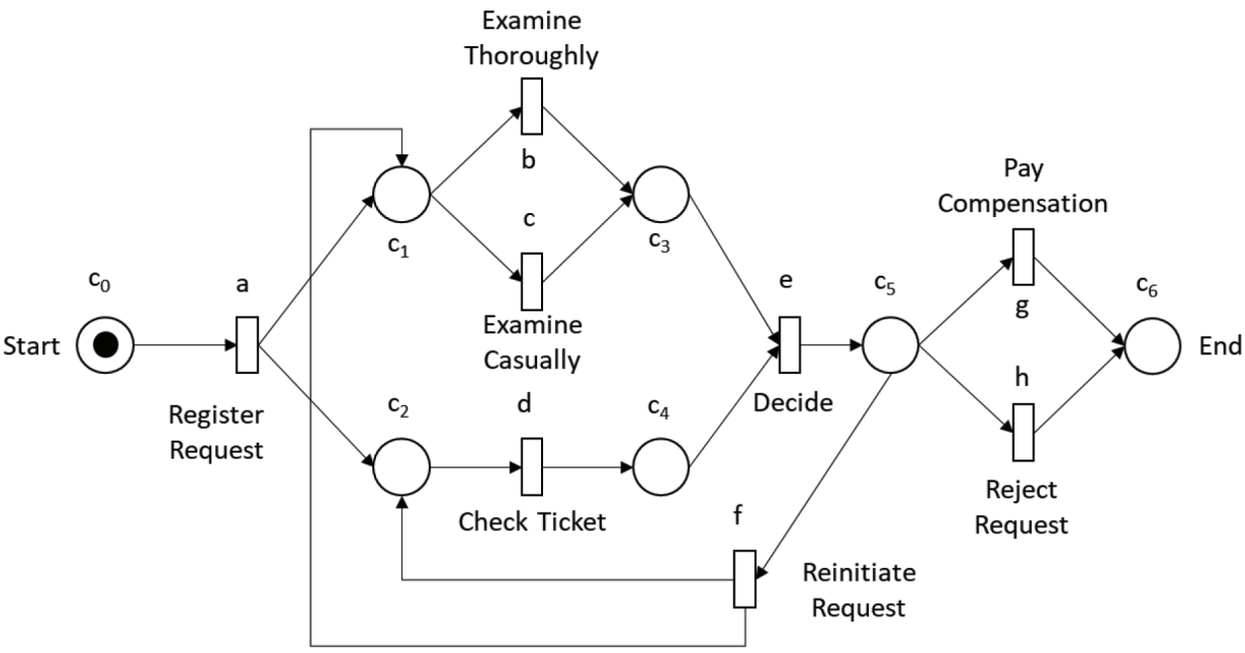

Figure 10. A Petri net model for a workflow pattern of a customer request. The customer may initiate a new request at any time. Two activities are launched in parallel for every request. One activity is to check the ticket. The other is to examine the request. At this point, a decision is made about how deep to examine the request. At this point, a guard based on the machine learning and data analytics is constructed for the transitions b and c. A decision process then comes, where either a compensation payment, a request rejection, or a request reinitiating may apply.

statistical learning mechanisms, guards for the transitions b and c are constructed. The guards allow deciding when it is more convenient to execute an in-deep examination process or a casual examination process. On the one hand, it saves time by executing a casual examination when the guard determines that it is more likely that the characteristics of the request are that of a genuine customer request. On the other hand, it saves money by executing a thoroughly examination process when the guard determines that the current request is more likely to be a fraudulent request.

Other important area of the engineering where the Petri nets have been successfully used is in the automatic code generation. The exponential growth of the cloud computing and proliferation of solutions based on the Internet of Things have made the design of the system software supporting them become more challenging. The set of requirements that this type of systems must address includes the sensing of signals in soft and hard real-time and the traditional support for media-reach services. This mixture of requirements turns the design of a correct and efficient system of this type a whole challenge. Approaches based on model-based design promise useful solutions for these challenges. The complex behavior and set of conditions that this class of software must address can be well represented with Petri net patterns. Synchronization mechanisms, message passing, rise conditions, critical sections, parallel and concurrent process, task activation conditions, and user interactions, to mention a few, could be easily represented with intuitive Petri net blocks. Then, a simulation process may allow the study of the performance of the solution and the adjustment of parameters for a fine-tuning process.

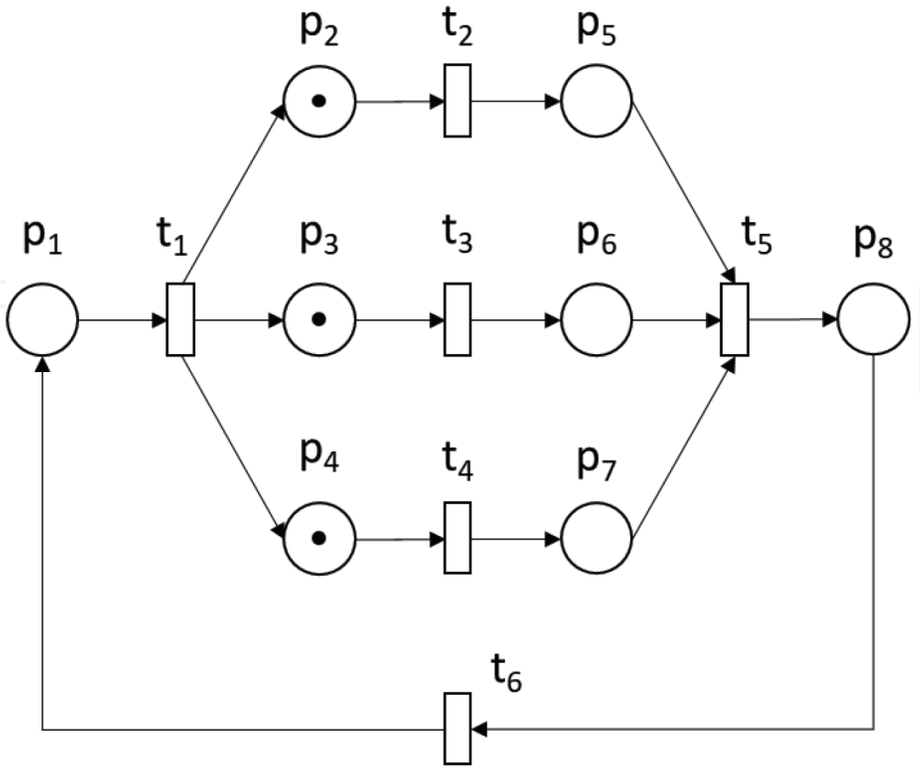

Figure 11. A Petri net model representing three parallel threads. The threads are launched in parallel by the firing of t_1. Each thread runs free to complete its activity. In this design, the end of the threads is synchronized at t_5. That is, if one thread finishes its work before the others, it must wait until the other threads end its activities. Once all the threads have finished, they are reinitialized to repeat the loop.

Figure 11 represents a Petri net model for three parallel processes. The transition t_1 launches the execution of the processes in parallel. Each process runs freely until they end its activities. In this design, the transition t_5 synchronizes the end of the processes. That is, if one process ends its activities, then it must wait the others to end. Once all the processes have finished, the loop repeats infinitely. The transitions t_2, t_3, and t_4 represent the activity load of each process. There are different approaches to add an amount of time to these transitions [13–15]. Within a suitable simulation process, this allows to investigate the performance of the system under different work load conditions, which is a must in the development of real world solutions. Once the parameters of the model have been tuned and its performance evaluated, the next step consists on the synthesis of the code in a target programming language for a specific platform.

For example, **Figure 12** shows a section of code in C/C++ implemented from the model in **Figure 11**. The code implements a set of joinable posix threads. A for loop launches a number of threads defined by the global constant NUM_THREADS. Other for loop waits for the end of the threads. Once all the threads have finished, the loop repeats indefinitely. The automatic code generation from Petri net models has recently been investigated with promissory results [19, 20].

```
...
while(1){
    // Initialize and set threads as joinable
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    for( i = 0; i < NUM_THREADS; i++ ) {
      cout << "main() : creating thread, " << i << endl;
      rc = pthread_create(&threads[i], NULL, wait, (void *)i );
      if (rc) {
         cout << "Error:unable to create thread," << rc << endl;
         exit(-1);
      }
    }
    // free attribute and wait for the other threads
    pthread_attr_destroy(&attr);
    for( i = 0; i < NUM_THREADS; i++ ) {
      rc = pthread_join(threads[i], &status);
      if (rc) {
         cout << "Error:unable to join," << rc << endl;
         exit(-1);
      }
      cout << "Main: completed thread id :" << i ;
    }
    cout << "Re-staring the main infinite loop" endl;
}
...
```

Figure 12. A section of code in C/C++ implemented from the Petri net in the figure above. An infinite loop initializes a set of joinable threads. A for loop launches the number of threads specified by the constant NUM_THREADS. A for loop waits for the end of all the launched threads. This cycle repeats forever.

4. Conclusions

This chapter aims to briefly review the applications of the Petri nets in science and engineering. It not pretends to be a deep review of the applications with complete detail and mathematical foundation. Rather, the objective is to provide an illustrative introduction to Petri nets and its potential applications, as intuitive as possible, avoiding the use of complex mathematical notation and formulation. The focus of this chapter was in the graphical nature of the Petri nets and the intuition about them, and with some emphasis in its mathematical foundation. Also, the intention is that this chapter serves as an introduction to this book entitled Petri Nets in Science and Engineering. The authors hope you find this book illustrative for your different activities in science and engineering.

Sincerely,

R. Campos-Rodriguez, M. Alcaraz-Mejia.

Acknowledgements

The authors want to thank Jose Valerio, from Oracle Guadalajara Development Center, for his valuable comments in the review of this chapter and for his experience and comments in Machine Learning and Data Analytics.

Author details

Raul Campos-Rodriguez[1]* and Mildreth Alcaraz-Mejia[2]

*Address all correspondence to: rr_campos@hotmail.com

1 Monterrey Institute of Technology and Higher Education, Guadalajara Campus, Tlaquepaque, Jalisco, Mexico

2 ITESO University, Tlaquepaque, Jalisco, Mexico

References

[1] Reisig W. Petri Nets: An Introduction. Vol. 4. Luxemburgo: Springer Science & Business Media; 2012

[2] Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE. 1989; **77**(4):541-580

[3] Mac Lane S. Categories for the Working Mathematician. Vol. 5. Berlin: Springer Science & Business Media; 2013

[4] Roxin EO. Control Theory and its Applications. Gordon and Breach; 1997

[5] Cassandras CG. Discrete Event Systems: Modeling and Performance Analysis. Aksen Associates Series in Electrical and Computer Engineering, IFAC Proceedings Volumes. 2000:**33**(13):313-318

[6] Cassandras CG, Lafortune S. Introduction to Discrete Event Systems. Berlin: Springer Science & Business Media; 2009

[7] Farina L, Rinaldi S. Positive Linear Systems: Theory and Applications. New York: John Wiley & Sons; 2011

[8] Viswanadham N, Narahari Y. Performance Modeling of Automated Manufacturing Systems. Englewood Cliffs, NJ: Prentice Hall; 1992. pp. 497-508

[9] Merlin P, Farber D. Recoverability of communication protocols—Implications of a theoretical study. IEEE Transactions on Communications. 1976;**24**(9):1036-1043

[10] Bochmann G, Sunshine C. Formal methods in communication protocol design. IEEE Transactions on Communications. 1980;**28**(4):624-631

[11] Pavlo A, Angulo G, Arulraj J, Lin H, Lin J, Ma L, et al. Self-Driving Database Management Systems. In: CIDR. 2017

[12] Available from: https://www.oracle.com/corporate/pressrelease/oow17-oracle-autonomous-database-100217.html

[13] Wang J. Time Petri nets. In: Timed Petri Nets. Boston, MA: Springer; 1998. pp. 63-123

[14] Popova-Zeugmann L. Time Petri nets. In: Time and Petri Nets. Berlin, Heidelberg: Springer; 2013. pp. 31-137

[15] Ling S, Schmidt H. Time Petri nets for workflow modelling and analysis. In: 2000 IEEE International Conference on Systems, Man, and Cybernetics, Vol. 4. IEEE; 2000. pp. 3039-3044

[16] Yu J, Buyya R. A taxonomy of workflow management systems for grid computing. Journal of Grid Computing. 2005;**3**(3–4):171-200

[17] Van der Aalst WM. The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers. 1998;**8**(1):21-66

[18] Jensen K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1. Berlin: Springer Science & Business Media; 2013

[19] Philippi S. Automatic code generation from high-level Petri-nets for model driven systems engineering. Journal of Systems and Software. 2006;**79**(10):1444-1455

[20] Mortensen KH. Automatic code generation from coloured Petri nets for an access control system. In: Second Workshop on Practical Use of Coloured Petri Nets and Design/CPN; Aarhus, Denmark. October 1999. pp. 41-58

[21] Shen VR, Chang YS, Juang TTY. Supervised and unsupervised learning by using Petri nets. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans. 2010;**40**(2):363-375

[22] Bugarin AJ, Barro S. Fuzzy reasoning supported by Petri nets. IEEE Transactions on Fuzzy Systems. 1994;**2**(2):135-150

[23] Konar A. Machine learning using fuzzy Petri nets. Computational Intelligence: Principles, Techniques and Applications. Berlin Heidelberg: Springer-Verlag, 2005. pp. 521-546

[24] Looney CG. Fuzzy Petri nets for rule-based decision making. IEEE Transactions on Systems, Man, and Cybernetics. 1988;**18**(1):178-183

[25] Bulitko V, Wilkins DC. Machine learning for time interval Petri nets. In: Australasian Joint Conference on Artificial Intelligence. Berlin, Heidelberg: Springer; December 2005. pp. 959-965

[26] Badouel E, Bernardinello L, Darondeau P. Petri Net Synthesis. Heidelberg: Springer; 2015. p. 339

[27] Alcaraz-Mejia M, Campos-Rodriguez R, Caballero-Gutierrez M. Modeling and simulation of task allocation with colored Petri nets. In: Computer Simulation. London: InTech; 2017

IntechOpen