INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING
DOI: 10.46300/9106.2022.16.104 Volume 16, 2022

Concurrency and Petri Net Models

Anthony Spiteri Staines
Department of Computer Information Systems University of Malta Msida, MSD 2080 Malta

Received: August 29, 2021. Revised: February 15, 2022. Accepted: March 1, 2022. Published: March 11, 2022.

Abstract: - Concurrency is a fundamental problem and a solution applicable to different areas of computing. Given the complexities and distribution of computer systems and services, concurrency is a modern area requiring proper attention. Petri nets are formalisms based on process representation both from a mathematical view and from a graphical or drawing like view. Petri nets are used to model concurrent processes. This work deals with understanding and representing low level concurrency in Petri nets, when this is not always visible and properly noted from the graphical structure. In this study an algebraic notation has been devised and is used to represent the Petri net structures. This algebraic notation is used as an alternative and simplified way of representation. The notation is explained and several simple examples are given. The notation presented can be used in conjunction with other Petri net analysis and verification methods. Some results and findings are discussed.

Key-Words: - Algebraic representation, Concurrency, Formal methods, Systems Theory, Systems Modelling, Petri nets, Verification

I. INTRODUCTION

Concurrency is a very important topic and concept used in computer science and computer modelling [1]-[4]. Concurrency modelling is visible in the producer consumer design patterns. Concurrency and distributed system modelling gains more importance in modern computer science [18]-[19].

The implications of concurrency require proper understanding and treatment both software and hardware. If a systems does not have proper concurrency control mechanisms activities can behave chaotically and indeterminately. Concurrency mechanisms can create very complex interactions amongst each other and different systems. Verification of concurrency is a non-trivial problem [15]-[19].

Mainstream modern multiuser systems and critical applications can involve high degrees of concurrency. This can imply that several tasks are executed in parallel or sequentially following some temporal ordering. Many experts keep stating that the current reasoning about concurrency is insufficient making verification a common problem. To this end various specialised tools and frameworks are created. Formal languages and formal modelling serve to find suitable ways and expressions to properly represent concurrency [13]-[14].

Petri nets are formalisms that have been widely used to study, represent and execute concurrent system types [1]-[2], [5]-[12]. The study of Petri nets from the view of formal specification languages is an ongoing work. Using Petri nets to model concurrency invites many new interesting problems and scenarios. There are several Petri net languages, some are context free and others not. Several classes of Petri nets exist, ranging from simple restricted classes to higher order Petri net structures along with formal coding languages like ML. Petri net languages can be represented using *concurrent regular expressions*. In this work these will not be considered and instead the focus will be on simple representation directly observed from the net's structure.

The more restricted the Petri net, the easier it becomes to control and represent concurrency. After all the most complex systems in computing have been definitely constructed or derived from *fundamental structures* and *primitive building blocks* [5]-[12].

II. PROBLEM FORMULATION

Parallel, concurrent, sequential and other processing modes are an integral part of modern computer systems. System verification and validation are always gaining greater importance given that modern information systems depend heavily on distributed technologies [15]-[16]. Petri net formalisms in all their forms have been used to contribute to this, it is clear that the structural and behavioral properties of Petri nets need better understanding in this regard. The control mechanisms need proper understanding. Many authors have indirectly tackled concurrency and Petri nets in their works. In various instances, this has been done using higher order nets and structures and are only partially explained. In this work this issue will be tackled at the foundational level to serve as the building blocks.

Petri nets describe, depict pictorially and mathematically the behavior of many system types [20]-[21]. The effectiveness of these notations lie in their simplicity, abstractedness and symbolic representation. Petri nets are a good choice for concurrency and can be easily combined with other formalisms [1]-[12].

When talking about concurrency, what is really meant by this? Concurrency is conventionally defined as the progressing of two activities or programs in parallel. This definition is too generic to really give a clear indication of what concurrency is. Activities that are causally unrelated could happen concurrently just by chance. From a better perspective for true concurrency, the activities must share some type of coordinating relationship. There is a problem of non-determinism in concurrency. I.e. which activity occurs first and in what order do they occur? Concurrency does not necessarily hold completely true to pure parallel behavior.

When modelling concurrency in Petri nets, two elementary types of concurrency have been identified by the author. These can serve to construct more complex types. These are i*) weak concurrency* and *ii) strong concurrency*. Weak concurrency implies that the concurrency situation depends greatly on the temporal situation and might or might not take place as a future event. Whilst strong concurrency will always imply that the activities occur together. For this to be guaranteed, certain enabling conditions must hold true.

Even though concurrency in elementary Petri nets might look quite simple, for the process of concurrency there are several complexity issues and factors that can affect the behaviour. Even a simple Petri net could have several possible non-deterministic firing or execution sequences.

By definition ordinary Petri net structural forms excluding token execution [1]-[3] are classifiable as *directed bi-graphs* or directed *bipartite graphs*. *Structurally Petri nets can be divided into two disjoint and independent sets U,V such that every edge connects to one vertex in U and one vertex in V. There are no odd length cycles in the graph*. *Another concise limited way of defining the Petri net is as a four or five tuple set (P,T,I,O,M) where P,T are non-empty finite sets of places and transitions, I is the input function and O is the output function, M is the initial marking of the net's places. For more detailed definitions it is possible to refer elsewhere.*

Ordinary Petri nets are composed of places and transitions. Places may contain tokens that are timeless and valueless in the context of ordinary nets. The initial token distribution determines the state of the net and the possible firing sequence. Firing requires the consumption of a resource, i.e. a token and possibly outputs a resource or more. Transition firing is an atomic event because it cannot be stopped [1]-[3]. The difference between enabled and activated transitions is fundamental for concurrency. To achieve real parallel processing there have to be at least two or more tokens in the net. Having parallel activity is not necessarily dependent on the graphical connections of the net. The problem of concurrency is a non-deterministic one. It is an axiomatic fact that at the elementary level of the net there is no temporal ordering in which multiple pre-activated transitions have to fire. This situation can give rise to sequentiality, concurrency or both.

Concurrency can be classified into temporal and dependent concurrency. Temporal concurrency refers to concurrency that takes place just because of processes or events taking place at the same time for no specific link between them. It is thus an indeterminate process. On the other hand dependent concurrency refers to concurrency where there is some form of linking between the processes or events. This implies that some form of process synchronisation takes place. These problems are not just limited to Petri nets but exist even when using other forms of modelling like UML activity diagrams.

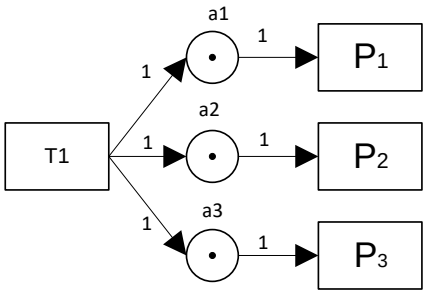


Figure 1: Petri net with three transitions concurrently enabled

As a simple exercise consider a Petri net structure, having three transitions simultaneously enabled. The transitions can be called T_1,T_2,T_3. Even though they are enabled concurrently it does not imply that they will fire together and create outputs together. There is an element of randomness of how they can fire. So the execution could possibly be concurrent, sequential or both. This is just for something so simple, thus the more complex the net the greater the dramatic increase in possibilities. In this simple exercise example, concurrency can have binary relationships with reflexive and symmetrical properties. E.g. the transition T1 could be concurrent with itself (T1) which is a reflexive property and T1 could be concurrent with (T2) which is a symmetrical property.

III. PROPOSED SOLUTION

Different useful notations exist for process representation and abstraction. Unfortunately there is no single representation or notation that captures all the salient details of a Petri net system. Combining formal notations with visual diagrams can give more robust and complete models of concurrency. Works of all sorts currently exist in literature, but many solutions are not simplified and are difficult to reproduce. In principle the idea of combining different forms of representation is really good, but for it to work it has to be applicable to different scenarios and not limited to a particular problem. Combining Petri nets with formal modelling can greatly extend the possible modelling power.

The solution here is to present simple methods how to represent concurrency. For this purpose the solution will be limited to ordinary basic place transition nets. A good solution is one that can be properly understood by different groups of persons and applied with minimum effort. It is clearly stated that the visual part or graphical drawing of Petri nets is fundamental to representing concurrency modelling [1]-[8]. It is insufficient to represent the concurrency in the net using abstract mathematical notations like process algebras or formal languages. *In this paper's solution a two-fold approach is used to model and explain concurrency in the Petri nets: i) visual graphical Petri net mode and ii) a simplified notation created for the net's structural representation.* Sometimes when drawing Petri nets, the drawing model is quite confusing because of the way the edges and nodes are laid out. This makes it very difficult to read the model. These type of models whilst being good for that particular work will make it difficult for a user to understand what is happening at other levels.

An important aspect of creating Petri net models it is the use and application of good diagrammatic principles. These have to be applied to the graph drawings. These can be classified as: i) simplicity by restricting the net to a few elements, ii) aesthetical properties, i.e. the outputs and inputs in the net should not overlap but should be clearly evident. If possible the transition boxes of the net should be of the same size, etc. This should ensure that instead of the user having to decipher the model, embedded deeper layers can be seen at a glance. A well laid out model is easier to remember and can serve to generate various patterns of operation.

IV. ALGEBRAIC NOTATION FOR ORDINARY PETRI NET REPRESENTATION

As was previously discussed, there is no single absolute method or representation for modelling all the important system details in Petri nets. When formal or symbolic notations are combined with visual models, the approach is more complete and detailed. In literature works, this has been accomplished using Haskell, Z, Vienna development method, the ML language, etc. [3], [14], [17]-[19]. This work is all very useful, but then these various solutions are only valid for pre-determined scenarios and are limited to this.

Simple Petri net structures are represented as input and output functions. The simplified definition is given as follows. This has been greatly improved and simplified from the author's previous work in [9], [10].

An input or output operation for the Petri net is defined as X. It is possible to include token values by using X(v) where v is the value of tokens in the input or output place that connects with X. To clarify X(v) would represent input and X<v> would represent output. These are not necessarily included in the representation. The bindings of a place to an input function is given as *place.input function* conversely the bindings of an output function to a place is given as *output function.place*. E.g. $v \cdot X$ represents a place

v binding to a function X. X is the input arc. Similarly the opposite holds for the output. v(q)·X would represent input v and token values q bound to function or arc X. The Δ (delta symbol) will be used to represent transitions. The · (binding) is used for connecting places and inputs or output flows. The operators will be enclosed in {} (syntactic construct). Additional operators are used ∩ (and), ∪ (or), → (ordering and connecting of input and output relations), || (parallel processing).

V. EXAMPLES

A. Standard Algebraic Notations

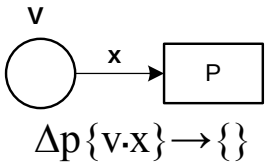


Figure 2: Input process with simplified algebraic process representation

Fig. 2 illustrates the use of the simple algebraic notation that will be used to represent the elementary net models.
It is possible to include tokens as illustrated in fig. 3. This form will not be used for the rest of this work and examples.

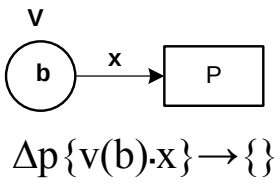


Figure 3: Input process including tokens

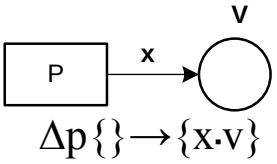


Figure 4: Output process with representation

Fig. 4 gives an example of an output process. Note that the empty {} denote the absence of inputs required for process or transition P. I.e. this implies that there is no input requirement for P to activate. This is commonly depicted in Petri net theory. There is the converse where when a transition fires it is possible that no output is created. This is clearly shown in the equations for fig. 3 and fig. 2.

B. Petri net with exactly one input/output

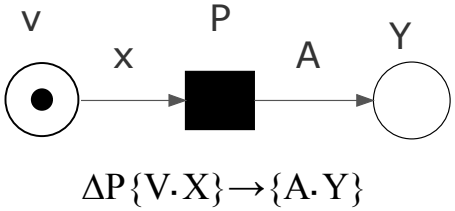


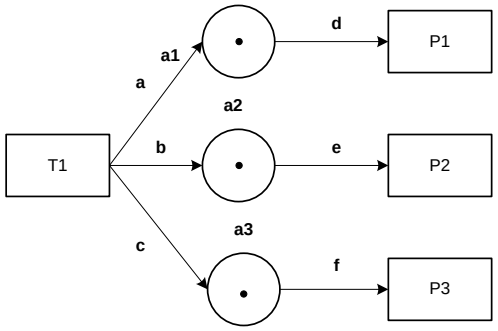
Figure 5: Petri net with exactly one input/ output

Figure 5 is a Petri net with exactly one input and output. The equation representation is quite simple and straightforward. There are no parallel activities or concurrency problems in this model. The equation for this model is simple and elegant.

C. Petri net with three enabled transitions

Considering the Petri net in fig. 1, three transitions are simultaneously enabled or concurrently enabled. The diagram used in fig. 1 is redrawn and properly labelled as per fig. 6. The equation for the net is shown below. Even though P1,P2 and P3 are concurrently enabled. This does not automatically imply that they will fire concurrently or in parallel. The equation that represents this model clearly indicates this. Another interesting fact and finding is that even though at face value the visual model looks to be simple, the mechanisms involved are quite complex. There is possible non-determinism in this model. The non-determinism is better explained in the representation equation than in the net. Actually the equation shows that there are four parallel processes or activities that exist in this model. These are not normally obvious just by glancing at the Petri net structure.
The equation expression is not considering the tokens in the places. So mainly the Petri net is being described from a purely structural perspective for the concurrency representation and issues. However the operational perspective is not considered.
Even though this structure is rather simple, it represents a classic design principle in traditional computer systems. Such a structure is commonly found in UML2 activity diagrams and is known as a fork node. It shows that a single task or process can create several other tasks or processes that possibly could run concurrently to each other. The structure in

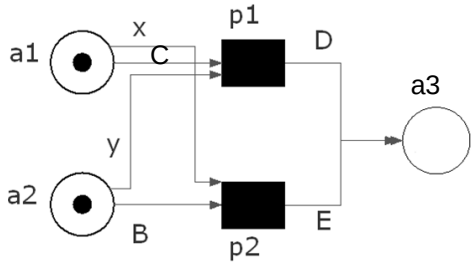
fig. 1 lacks a point of closure or a sink node according to graph theory. Hence it is not possible to really conclude anything about the concurrency of this structure.



$\Delta T1\{\} \rightarrow \{a \cdot a1 \cap b \cdot a2 \cap c.a3\} \| \Delta P1\{a1.d\} \rightarrow \{\} \| \Delta P2\{a2.e\} \rightarrow \{\} \| \Delta P3\{a3.f\} \rightarrow \{\}$

Figure 6: Petri net with three enabled transitions

D. Petri net with two concurrent enabled transitions with restricted firing

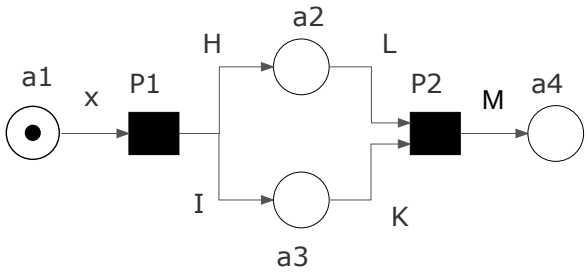


$\Delta p1\{a1 \cdot C \cap a2.Y\} \rightarrow \{D \cdot a3\} \cup \Delta p2\{a1 \cdot X \cap a2.B\} \rightarrow \{E.a3\}$

Fig. 7 Petri net with restricted firing. I.e. choice

In the structure in fig.7 the Petri net shows two transitions that are simultaneously enabled. But this does not automatically imply that firing occurs in parallel. To the contrary if p1 fires p2 is disabled from firing and vice-versa. In Petri net theory this is known as mutual exclusion. The equation just captures the structure of the net. It shows that two processes p1 and p2 are possible. This it is also explained by the equation that these processes are definitely not concurrent. As a short summary there are just p1 or p2. I.e. there is clearly an 'or' condition. It is possible to repeat this *pattern* and include more places and transitions.

E. Petri net with two possible concurrent transitions

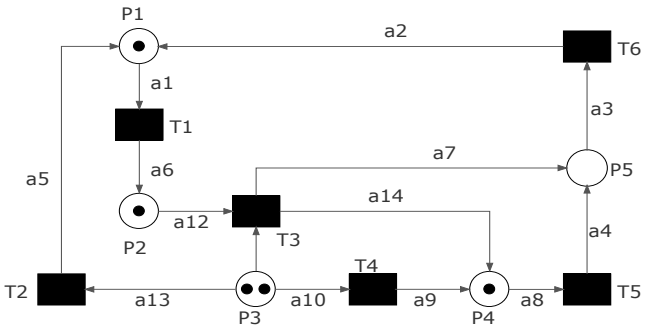


$\Delta P1\{a1 \cdot X\} \rightarrow \{H \cdot a2 \cap I \cdot a3\} \| \Delta P2\{a2 \cdot L \cap a3.K\} \rightarrow \{M.a4\}$

Fig. 8 Petri net suggestive of possible concurrency

Fig. 8 depicts a Petri net structure that suggests the possibility for concurrency. It can be determined that the concurrency is dependent on the enabling conditions of the net. For concurrency to be possible both P1 and P2 must be activated simultaneously. I.e. there must be tokens in a1, a2 and a3. In this case it can be stated that P1 and P2 are enabled. The firing order does not only depend on this. Firing order is undetermined and uncontrolled. This would imply that P1 and P2 could execute simultaneously or sequentially with e.g. P2 occurring before P1 or vice-versa. The given equation specifies precisely that. It does not show the firing order at all but specifically explains what is possible.

F. Comprehensive Petri net example



$\Delta T1\{P1 \cdot a1\} \rightarrow \{a6 \cdot P2\} \|$
$\Delta T2\{P3 \cdot a13\} \rightarrow \{a5 \cdot P1\} \|$
$\Delta T3\{P2 \cdot a12 \cap P3 \cdot a11\} \rightarrow \{a7 \cdot P5 \cap a4 \cdot P4\} \|$
$\Delta T4\{P3 \cdot a10\} \rightarrow \{a9 \cdot P4\} \|$
$\Delta T5\{P4 \cdot a8\} \rightarrow \{a4 \cdot P5\} \|$
$\Delta T6\{P5 \cdot a3\} \rightarrow \{a2 \cdot P1\}$

Fig. 9 A more complex Petri net

Fig. 9 shows a more realistic and complex Petri net model. Such a model has the possibility for six parallel or concurrent processes. These would go unnoticed from just looking at or observing the physical Petri net structure. The equation depicts the actual complexity of this net. It is pointed out that the actual concurrency that takes place depends on the marking states of the net. The equation just gives the possibilities that are available. The equations for these structures show the possibilities of decomposing. I.e. if the part of the equation $\Delta T1\{P1 \blacksquare a1\} \rightarrow \{a6 \blacksquare P2\}$ is considered, this shows that the complex equation for this model is in reality composed of smaller sub equations. The full equation $\Delta T1\{P1 \blacksquare a1\} \rightarrow \{a6 \blacksquare P2\} \| \Delta T2\{P3 \blacksquare a13\} \rightarrow \{a5 \blacksquare P1\} \| \Delta T3\{P2 \blacksquare a12 \cap P3 \blacksquare a11\} \rightarrow \{a7 \blacksquare P5 \cap a4 \blacksquare P4\} \| \Delta T4\{P3 \blacksquare a10\} \rightarrow \{a9 \blacksquare P4\} \| \Delta T5\{P4 \blacksquare a8\} \rightarrow \{a4 \blacksquare P5\} \| \Delta T6\{P5 \blacksquare a3\} \rightarrow \{a2 \blacksquare P1\}$ is just actually a composition of six parallel processes and that's it! There is no special complexity when this is decomposed.

This explains a simple property and a fact. Petri nets can be viewed as compositions of several processes or transitions. The actual bottom level building blocks for the Petri nets are quite simple.

VI. DISCUSSION

Even ordinary place transition Petri nets do have complex issues when representing concurrency. The experiments clearly illustrate that from a static structural perspective it is impossible to determine the proper execution of the net. There is a problem with representation diagrammatic notations as these cannot contain all the details so formalisms are needed to support. The simple algebraic representation approach presented in this work is suitable for giving a simple representation of the concurrency problem in the Petri net.

In real world scenarios completely independent sequential processes do not exist. Normally several concurrent processes are interacting with each other. This is common for modern distributed information systems such as cloud computing and web applications. Even a multi-processing system will cause several processes to strongly interact with one another. The processes of many systems are usually non-sequential as they depend on other events. To use information technology for solving issues in the real world it is imperative to apply concepts used for distributed applications.

The models given in this paper are simple toy examples. They present us with a problem of non-determinism. Even though the structures are simple ones, there is no way how to determine which transitions will fire first or second given that more than one transition is enabled simultaneously. The firing order or sequence depends on the initial marking state of the nets.

The models used in this paper clearly indicate two main types of concurrency: i) non-dependent concurrency or ii) dependant concurrency. With dependent concurrency, there is weak or strong coupling between processes or transitions. With non-dependent concurrency there is no coupling at all between processes or transactions.

These Petri net structures present us with more real problems. E.g. when having concurrency, is it real concurrency or just apparent concurrency? It is possible that many systems give an implicit illusion of parallelism, but then these appearances could be deceptive and the underlying lower level mechanisms at work differ significantly from the workings being depicted at a higher level.

Model checking and verification of different classes of Petri nets is of prime importance. There are several properties of Petri nets like p and t invariants, liveness, fairness, reachability markings, token distribution, etc. that can be used to validate well established properties of the net's structure and workings. Also algorithms have been devised for determining connectivity issues. In our opinion concurrency issues are of a greater importance. In many of these verification techniques the problem of concurrency does not at all figure in these. One of the main reasons for using Petri nets is precisely concurrency.

The algebraic notation or equations used in this work are simple to comprehend. The equations given can be clearly used to represent the structures of the net so they can be combined with other verification methods and techniques currently in use. They can be used to represent fragments or subnets too if it is not required to represent the complete net structure. This approach could be used or included to create better or more modular net structures. More work will have to be considered in this direction. Normally the nets are checked after construction. The algebraic notation presented is possibly useful before and after construction.

The equational forms of representation given here are limited in scope and applicable only to representing the structural forms of modelling. Hence other current techniques are of prime importance too. These other notations and even other forms of formal modelling should be applied when representing systems. From a modelling perspective the rule of more is better applies to diversity of different types of notations that can be all used for representation.

VII. CONCLUSION

This work has presented a simplified but powerful novel algebraic notation that us useful for representing Petri Net structures. Obviously it cannot solve all the representational problems inherent in Petri net modeling. Hence the use of this notation is envisaged in collaboration with other methods and techniques. Petri nets are powerful formalisms for representing concurrency mechanisms at a low-level. There is a lot of support for Petri nets in the computer science community. It is hoped that this work will inspire more research and new developments to continually improve this powerful modeling notation of Petri nets that now has well over four decades of coverage.

REFERENCES:

[1] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proc. Of the IEEE*, Vol 74 issue 4, IEEE, 1989, pp.541-89.

[2] M. Zhou, K. Ventkatesh, *"Modelling Simulation, and Control of Flexible Manufacturing Systems, A Petri Net Approach"*, World Scientific, 1999.

[3] K. van Hee, "*Information Systems: A Formal Approach*", Cambridge Univ. Press, 2009.

[4] A. Knopfel et al., *"Fundamental Modeling Concepts"*, Wiley; 2005.

[5] A. Spiteri Staines, "An Introduction to Bi-Directional Transition Network Modeling", *International Journal of Computers*, IARAS, Vol. 2, 2017.

[6] A. Spiteri Staines, "Matrix Representations for Ordinary Restricted Place Transition Nets", *WSEAS Transactions on Computers*, Vol 16, 2017.

[7] A. Spiteri Staines, Modelling Simple Network Graphs Using the Matrix Vector Transition Net, CSSCC 2016, INASE, Vienna, 2016.

[8] T. Spiteri Staines and F. Neri, "A Matrix Transition Oriented Net for Modeling Distributed Complex Computer and Communication Systems", *WSEAS Transactions on Systems*, Vol. 13, 2014, pp. 12-22.

[9] T. Spiteri Staines, Concurrency Issues in Ordinary Petri Nets, *Communicating Process Architectures*, Vol. 70, 2017, pp. 101-110.

[10] A. Spiteri Staines, Algebraic Representation for Ordinary Place Transition Petri Nets, INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING, Vol. 11, NAUN,2017, pp. 300-305.

[11] A. Spiteri Staines, Implementing a Matrix Vector Transition Net, BJMCS, Vol. 4, Science Domain, 2014, pp. 1921-1940.

[12] A. Spiteri Staines, Graph Drawing Approaches for Petri Net Visualisation and Representation, *WSEAS TRANSACTIONS on INFORMATION SCIENCE and APPLICATIONS, vol 17,* 2020, pp. 110-116.

[13] B. Cliff et. al., *Case Studies in Systematic Software Development*, Pretence Hall, 1990.

[14] D. Lightfoot, "Formal Specification using Z", Palgrave 2001, ISBN 0-333-76327-0, Ch 6, pp. 37 – 97.

[15] R. Davidrajuh, *Petri Nets for Modeling Large Discrete Systems*, Springer, 2021.

[16] I. Flores De La Mota et. al., *Robust Modelling and Simulation*, Springer, 2017.

[17] S.K. Chang, "Principles of Pictorial Information Systems Design", Pretence Hall, 1989.

[18] R. Milner, *A Calculus of Communicating Systems*, Springer- Verlag, 1980.

[19] C.A.R. Hoare, *Communicating Sequential Processes*, Pretence Hall, 1985.

[20] R. Tamassia, G. Di Battista, C. Batini, Automatic Graph Drawing and Readability of Diagrams, *IEEE Transactions on systems, Man and cybernetics*, Vol: 18, no: 1, 1988, pp. 61-79.

[21] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, *Graph Drawing Algorithms for the Visualization of Graphs*, Pretence Hall, 1998.

Contribution of individual authors to the creation of a scientific article (ghostwriting policy)

Author Contributions: All the work in the paper was carried out by Anthony Spiteri Staines. Sections 1-7 and the abstract are the complete work of Anthony Spiteri Staines. This work considerably builds upon and greatly improves previous work by Anthony aka Tony Spiteri Staines presented in Concurrency Issues in Ordinary Petri Nets, *Communicating Process Architectures*, Vol. 70, 2017, pp 101-110 and Algebraic Representation for Ordinary Place Transition Petri Nets, Naun, 2017.

Sources of funding for research presented in a scientific article or scientific article itself

All the funding for this work comes from the University of Malta, Malta. Research Project Funding.