applied sciences

MDPI

Article

A Novel Method For General Hierarchical System Modeling via Colored Petri Nets Based on Transition Extractions from Real Datasets

Stavros Souravlas [1,2,*,†], Sofia Anastasiadou [2,*,†] and Irene Kostoglou [1,*,†]

[1] Department of Applied Informatics, University of Macedonia, 546 36 Thessaloniki, Greece
[2] Department of Midwifery, School of Health Sciences, University of Western Macedonia, 502 00 Ptolemaida, Greece
* Correspondence: sourstav@uom.edu.gr (S.S.); sanastasiadou@uowm.gr (S.A.); iis20078@uom.edu.gr (I.K.)
† These authors contributed equally to this work.

Abstract: The Colored Petri net (CPN) has evolved into a complete modeling language, which is based on the object-oriented paradigm. Although this language is quite complete and can be used to accurately model a wide variety of systems, it seems that there is a lack of certain modeling rules, which can be used to generate models based on real datasets. This paper is a first attempt to incorporate sequences of real system events and translate them into sequences of Petri net transitions. Then, well-defined modeling rules control the model generation based on these transitions. The specific entities that take part in each of the real system events, as well as their attributes are also considered. The models produced using real data are structured because, in the majority of real systems, each incurring event usually affects a part of the system. Then, this effect is propagated to the entire system. Therefore, it is much easier to divide the entire model into distinct parts when real data are used and then interconnect these parts to build an entire model. This is a very important aspect when modeling very large systems. To test our approach, we used the real data from a courier company to generate its model. Our simulation results have shown that we managed to obtain quite accurate results through the model produced by the actual data.

Keywords: modeling; Colored Petri Nets; simulation

check for updates

Citation: Souravlas, S.; Anastasiadou, S.; Kostoglou, I. A Novel Method For General Hierarchical System Modeling via Colored Petri Nets Based on Transition Extractions from Real Datasets. *Appl. Sci.* **2023**, *13*, 339. https://doi.org/10.3390/app13010339

Academic Editor: Juan-Carlos Cano

Received: 2 November 2022
Revised: 19 December 2022
Accepted: 20 December 2022
Published: 27 December 2022

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The Coloured Petri Net (CPN) is an object-oriented graphical language, which is used to construct system models and to analyze their properties and entities [1–3]. It enhances the basic capabilities of the traditional Petri net formalism by adding high-level programming language features. These features include the definition of data types, objects and classes of objects, methods for describing object behavior, rules for setting the timing of events that incur within a system, and tools for data manipulation. As a result, the CPN formalism can be used for a wide variety of systems and, in fact, it is a general purpose modeling system. Application examples include communication networks, distributed systems, business processes, manufacturing systems, and workflow systems.

The main structuring elements of the basic Petri net (PN) formalism are the places, the tokens, and the transitions. The *places* are represented by circles and they define a state (or condition) at which a system may be. A set of N places defines N different states that may exist within the system. The places are the containers of *tokens*. A token is a dynamic element that can model an object or a set of objects. Tokens are transferred among places. The presence of one or more tokens in one place indicates that the condition which this place defines is true. Apparently, the absence of tokens in one place means that the corresponding condition is false. A *transition* represents an event that, whenever triggered, may change the system's state, that is, the system may transition from one state to another.

Transitions are represented by black rectangles. Generally, we use the verb "fire" to indicate that an event has occurred. In order for a transition to fire, all its conditions must be satisfied. In this case, we say that the transition is "enabled".

The CPN formalism has the same structuring elements as the PN formalism, so it is defined in much the same way. However, the basic structuring elements include some extensions, which are discussed in the beginning of Section 4.

The CPN formalism has been developed to resolve three very important issues of PN, which are described separately below [1,3]:

1. In a typical PN, all the tokens have the same meaning:

The typical PN formalism does not distinguish between the tokens. Instead, all the tokens are the same and they are simply used to represent a system's state. In other words, simple Petri nets use the tokens as black boxes and their existence in a place simply shows if the system has reached a specific state, which is described by the place. When a token reaches a terminal place (a place that can trigger no further events), it simply vanishes or is kept in this place, without holding any important information regarding the events of the system. The only thing that we know is that, at some moment in time, this token enabled the firing of certain transitions. However, in almost every system, its entities have attributes with different values (these values are referred to as *colors* in CPNs). These values are crucial for a researcher to study the entire system. For a researcher who is interested in exploring the attribute values of an entity (for example, the utilization of system resources described by the percentage of time they are used), the typical PN formalism is improper. In the CPN formalism, each token carries a value. Such values may include names, service times, resource types, job ids, job priorities, speed, delay values, and other information which can be useful to study the entire system at any time.

2. The typical PN formalism is not concerned with the actual timing of the events that can change the system's state, but only for their sequence.

The majority of the real-world systems are described by events which occur at a certain time. In a CPN, the transitions are not timeless (as in a PN) but timed. The provision of time gives the modeler the opportunity to describe precisely the temporal details of a system. The temporal approach requires that each token carries one time stamp, which is a timing value used to determine the firing time of the transitions, which are enabled by the places this token is located. The enabled transitions can fire and produce tokens with a delay, which corresponds to the time it takes for an event to occur. To determine the firing time of a transition, we examine all its input places and from each place, we find the token with a minimal time stamp (that is, the next token to leave this place). This corresponds to the earlier time the conditions that enable an event became true. Then, from these time stamps, we pick the one with the larger value. This indicates the latest time by which *all* the necessary conditions became true, to enable a transition.

Generally, when we consider time, there is always a global clock for keeping the time. Time is usually advances in time units, but sometimes, it is useful to use real time metrics (hours, minutes, seconds, etc.).

3. The typical PN formalism does not impose further conditions for firing a transition.

In a PN, an enabled transition will fire regardless of any other conditions that may hold. However, there are many cases where an event requires extra conditions other than the ones determined by the existence of tokens in places. For this reason, the CPN formalism introduces the guardian expressions, which can be used when we need to assign such conditions regarding the time of firings.

In addition, PN and CPN have some common disadvantages:

1. In their basic form, they do not have an inherited capability to transform actual system data and events into PN or CPN transitions and places.

Researchers use PN and CPN to create models, which are then used for simulations, in order to study the behavior of various systems. In other words, researchers first produce

a system model and then generate simulated data to test its accuracy. In this way, the system can be modeled in some level of accuracy; however, it may miss certain important conditions and details which can only be retrieved by the actual events that have incurred within the system. The colors, the time stamps, and guardian expressions introduced in CPNs are not capable of introducing real data into the model. Therefore, a scheme for transforming events into a sequence of transitions is required. Such a scheme should first define certain modeling rules for sequencing transitions. Then, it should read the actual events and try to efficiently transform them into CPN transitions by using the defined rules.

2. Both PN and CPN formalisms are not capable of self-organizing themselves into autonomous smaller subsystems which are easy to study and check for errors.

Dividing the overall model into smaller autonomous systems (which are referred to as *nest* in the rest of this work) is of great importance. The nests release a researcher from the burden of designing an entire model in one time. Based on the actual data, we can focus on a separate nest and then simply work to consider the connections between pairs of nests. The idea of using real data to generate models can be very efficient in producing such nests because, to a wide extent, a system's real events incur within a subsystem and its effects are then propagated to the entire system.

Our work tries to address the two disadvantages that we have described in this introductory section. The specific contributions of this work are listed in the next section. The remaining of this paper is organized as follows: Section 3 presents briefly the use of PN and CPN modeling in various research areas. Section 4 presents our modeling approach. In Section 5, we present a case study: specifically, we show how we have used real data from a courier company to model the company's activities. In Section 6, we show some experimental results from the simulation of our model. These results have shown that the model has quite accurately represented the company's working cycle. Section 7 concludes the paper and offers aspects for improvements of our methodology.

2. Contributions of This Work

To the best of our knowledge, this work is the first effort that uses real datasets to provide system modeling in an object-oriented environment such as colored Petri nets. The main contributions of this work are the following:

1. It defines sets of rules that can be used to model common sequences of events and transform them to CPN transitions.
2. It defines a simple timing mechanism for determining the time that transitions fire.
3. It presents a scheme for transforming real datasets into CPN transitions and places.
4. Based on the actual data, it defines CPN nests, which provide high flexibility to the model design.
5. The approaches discussed can be used to model a variety of different systems.
6. The models become more flexible and re-configurable.

3. Related Work

Petri nets are suitable for system specification, design, and analysis. They have been used as control code modeling and design tools, thus providing clear records of the system. In [4], the authors described this use of Petri nets and examined their application to a variety of production system controllers. They have been characterized as concurrent, asynchronous, distributed, parallel, non-deterministic, and stochastic. They are used as a graphical model and as a mathematical tool, as mentioned in [5]. The PN and CPNet formalism have been widely used in the literature, in numerous fields [6,7]. In our previous work, we implemented a CPN model for resource allocation policies in the cloud, where big data applications are executed [8]. This CPN model combined elements from a series of strategies on cloud systems and resource allocation, which we developed [9–13]. Other fields of PN and CPN applications are mentioned below: parallel processing [14], grid computing applications [15,16], traffic control [17], analysis of safety-

critical interactive Systems [18], manufacturing [19], everyday applications [20], supply chain management [21], medicine [22], industry [23], project management [24], fuzzy systems [25], and communication protocols [26,27].

A process mining approach is presented in [28], where an algorithm for aggregating multiple instance graphs is introduced. Such graphs are visualized using the Aris Process Performance Monitor tool and compiled manually as reported in [29]. It is interesting to note that the EPC modeling language that was used gathers case graphs in a Petri net.

This work does not focus on a specific scientific area of research. Instead, it tries to present some generalized rules that can be applied to generate models from real data and organize them in a hierarchical way. Apparently, this general approach can be employed for implementing any model for a specific area of study.

4. Our Modeling Approach

Our modeling approach is based on extracting transitions through the data received from the real system in the form of log files and on generating sequences of the transitions obtained to form the overall model. In this section, we provide the details. Some background knowledge is needed beforehand.

When modeling a system as a Colored Petri net, the overall system is represented by tokens, places, and transitions. Here, we must add some extensions to the definitions given in the introductory section. A *token* is not just an object as in PN, but it is an instance of an object (objects are also referred to as *entities* in the CPN formalism), which can be distinguished from other instances based on the current values of its characteristics. For example, an entity `employee` may be described as a set of characteristics: `<First Name, Last Name, Education, Years of Experience>` and, at a time, there may be two tokens: <Mary, Jones, High School, 1> and <Paul, Jones, University, 10>. The two tokens are instances of the same entity, yet they differ based on their values.

As in PN, a *place* is represented by a circle and it is a container of tokens. However, in the CPN formalism, a number of different (in terms of their values) tokens inside a place can define a network's state. For example, a state: project in process may contain the tokens <Mary, Jones, High School, 1> and <Paul, Jones, University, 10>. This shows that the system is in a state where the two employees are working on a project. Finally, a transition is a system's change, from one state to another but in a CPN model, it can be enabled by tokens with different values.

The main properties of the CPN formalism are presented below:

1. The CPN formalism allows for distinguishing the tokens, that is, there may be tokens that correspond to the same entity but the attributes (values of the token) differ. In this regard, all the data retrieved about members of an entity will be placed to different tokens instead of one that represents the entire entity.
2. The CPN formalism takes into consideration the actual timing of the events, not just their sequence (as in the basic PN formalism). In this regard, our proposed model will use the timing of the actual events to generate the model.
3. The CPN formalism can employ guard values to define conditions under which an event can (or cannot) occur at a certain time. These conditions can be extracted from records of the actual events that have taken place.

In the following subsections, we present the most important aspects of our CPN modeling approach: sequencing transactions from data, transaction modeling, timing considerations, guardian expressions, and the hierarchical modeling.

4.1. Sequencing Transitions from Data

Our idea is based on getting a sequence of events from real data, coding them, and trying to derive their sequence in order to model them at a later stage, which is described in the next subsection. Assume that we extract an event sequence of a system from real datasets. We use the notation E_i to code these events (transitions) and A_i to define the actors (participants) of these events. In addition, we consider that a complete set of events

constitutes a *case*. Then, we can create a log file that codes these events. This log file is in the following form:

Case	Event	Description	Actors	Other Details (Cost, Time, etc.)
1	E_1	·····	A_1, A_2	100
	E_2	·····	A_3, A_4	200
	E_4	·····	A_3, A_4	200
	E_5	·····	A_4, A_3	500
	E_8	·····	A_1, A_2	400
2	E_1	·····	A_1, A_2	100
	E_4	·····	A_3, A_4	200
	E_3	·····	A_1, A_6	150
	E_5	·····	A_1, A_4	500
	E_7	·····	A_2, A_3	100
3	E_1	·····	A_1, A_2	100
	E_3	·····	A_1, A_4	150
	E_4	·····	A_3, A_4	200
	E_5	·····	A_3, A_4	500
	E_6	·····	A_1, A_2	900
	E_2	·····	A_3, A_6	200
	E_4	·····	A_3, A_4	200
	E_5	·····	A_1, A_4	500
	E_7	·····	A_3, A_2	100
4	E_1	·····	A_1, A_2	100
	E_4	·····	A_3, A_4	200
	E_2	·····	A_3, A_2	200
	E_5	·····	A_3, A_4	500
	E_8	·····	A_1, A_2	400

To be able to model our transitions, we need to generate sequences of them from the log file and record the classes in which the actors belong to. We describe these two procedures in the following paragraphs.

4.2. Event Sequence Table

The Event Sequence Table (EST) is a pictorial representation of all the possible *direct* sequences of events (that is, events that immediately follow other events). We use the following symbols to depict each of the four following cases:

- $\longrightarrow$: An event is followed by another event.
- $\longleftarrow$: An event follows another event.
- $\longleftrightarrow$: An event is followed by another event vice versa.
- $\|$: There is no direct sequence between the two events.

Table 1 shows the EST for our log file example.

Table 1. EST for our log file example.

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
E_1	$\parallel$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
E_2	$\longleftarrow$	$\parallel$	$\parallel$	$\longleftrightarrow$	$\longrightarrow$	$\longleftarrow$	$\parallel$	$\parallel$
E_3	$\longleftarrow$	$\parallel$	$\parallel$	$\longleftrightarrow$	$\longrightarrow$	$\parallel$	$\parallel$	$\parallel$
E_4	$\longleftarrow$	$\longleftrightarrow$	$\longleftrightarrow$	$\parallel$	$\longrightarrow$	$\parallel$	$\parallel$	$\parallel$
E_5	$\parallel$	$\longleftarrow$	$\longleftarrow$	$\longleftarrow$	$\parallel$	$\longrightarrow$	$\longrightarrow$	$\longrightarrow$
E_6	$\parallel$	$\longrightarrow$	$\parallel$	$\parallel$	$\longleftarrow$	$\parallel$	$\parallel$	$\parallel$
E_7	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\longleftarrow$	$\parallel$	$\parallel$	$\parallel$
E_8	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\longleftarrow$	$\parallel$	$\parallel$	$\parallel$

The EST can be used to extract the dependencies between events:

1. *An event is followed by two or more events, which are independent:* An example is E_1, which is followed by E_2, E_3. The Independence of E_2 and E_3 indicates that there is no direct sequence in the form $E_2 \longrightarrow E_3$ vice versa. This means that E_1 will either trigger E_2 or E_3, because if they are both triggered, there is a chance that they can fire sequentially, which contradicts our data.
2. *An event follows two or more other events, which are independent of each other:* In our log file example, E_5 follows E_2 or E_3, which are independent of each other, as we have already described. This means that E_5 may be triggered by either E_2 or E_3 but not by both simultaneously.
3. *An event follows the triggering of two or more combined events:* Let us consider again E_5. Table 1 shows that it also follows E_4. In addition, we note that E_4 follows E_2 or E_3 vice versa. This indicates that, in order for E_5 to occur, the system must have completed (E_2 OR E_3) AND E_4. In other words, E_5 is a consequence of two events, E_4 and either E_2 or E_3.
4. *An event is followed by two or more events, which may be independent or combined at a later stage:* Let us consider the very first event, E_1. This event is directly followed by E_2, E_1, E_2, and E_3. As described, E_2 and E_3 are independent, but both of them can follow or be followed by E_4. In this regard, E_1 is followed by three events, two of which are independent. One of the independent events is combined with a third event at a later stage, to trigger another event, in this case E_5.
5. *An event is simply followed by one event:* This case is trivial, for example, see E_5, which is directly followed by E_6, E_7, E_8.

4.3. Class Recording Table

The Class Recording Table (CRT) is a pictorial representation of the classes of objects which participate in the events. Let us assume that there are two actor classes: `Employees` which include actors A_1 and A_3, and `Resources`, which include actors A_2, A_4, A_4. Table 2 shows the CRT for our log file example.

Table 2. CRT for our log file example.

E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8
							
A_1, A_2	A_3, A_4	A_1, A_6	A_3, A_4	A_3, A_4	A_1, A_2	A_3, A_2	A_1, A_2

where

A_1 = 

A_3 = 

A_2 =

A_4 =

A_6 =

The CRT provides us with the following useful information:

1. Which classes of objects can participate in an event.
2. Which specific instances of a class have participated in an event.
3. In combination with the EST, it provides us possible sequences of interactions between instances (actors), which are very useful in studying the model with details.
4. It can be used to detect any unusual or improper sequences of events, that is, events that rarely occur or actions which were wrong (do not conform to the model).

4.4. Transition Modeling

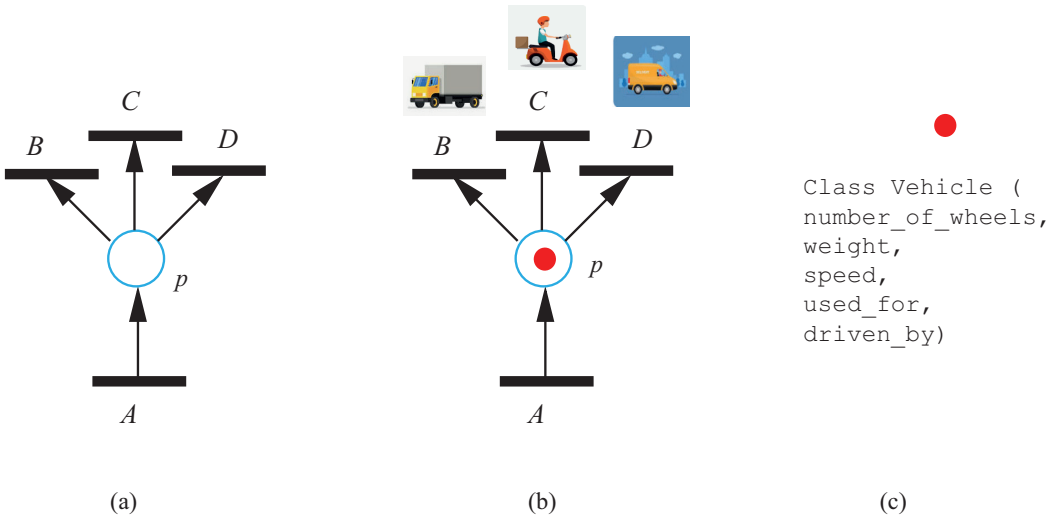
In this paragraph, we show how we take advantage of the Colored Petri Net (CPN) formalism to extract model transitions. We further examine the four basic scenarios (dependencies of events), which are described in Section 4.2.

4.4.1. Scenario 1: An Event Is Followed by Two or More Other Events, Which Are Independent

This scenario is typically described as follows:

$$(A \rightarrow B) \text{ OR } (A \rightarrow C) \text{ OR } \cdots \text{ OR } (A \rightarrow \cdots) \quad (1)$$

The "$\rightarrow$" symbol is translated as "is followed by" and Equation (1) indicates that our data have presented patterns where an event A has been followed by other events $B, C, \ldots$, one at a time. In other words, at different times, we have discovered patterns in the form $A \rightarrow B$, $A \rightarrow C, \ldots$ etc., but not in the form $B \rightarrow C$, $B \rightarrow D, \ldots$, etc. This means that, at a given time, event A may trigger either event B or event $C, \ldots$, etc. We use the term *successors* to define the events that may follow A and the term *predecessor* for A. The successors of A do not trigger each other, that is, they are *independent* of each other. A successor event may be implemented by different instances of the same entity. Figure 1 shows the modeling of this scenario.



(a) (b) (c)

Figure 1. (**a**) Modeling Scenario 1, (**b**) the effects after a successor transition has been chosen, (**c**) pseudo-code for an entity declaration.

In Figure 1a, we note that there are three possible successor events that can be triggered when event A is executed. We assume that event A is the start of a transportation. Transition A will place a single token in place p as can be seen in Figure 1b. This token represents

an entity class named `vehicle`, which describes the means of transportation. This entity class is represented by a red token and its instances may be a lorry, a motorbike, and a car. Thus, one of the transitions B, C, or D will fire and the event that will occur will be a transportation with a certain means. At a different time, event A will trigger a different transition, but, in any case, only one successor event may follow A at a time: B OR C OR D, that is, transportation by lorry, motorbike or car. In addition, note that the notion of "Colored" in CPN formalism indicates rather a class of an entity than a color itself.

4.4.2. Scenario 2: An Event Follows Two or More Other Events, Which Are Independent of Each Other

This scenario is typically described as follows:

$$(A \text{ OR } B \text{ OR } C \text{ OR } \cdots) \rightarrow X \tag{2}$$

Here, event X is the successor of a set of events that can trigger X. The successor X will incur when one of the predecessors has incurred. The predecessors do not follow each other, in the sense that there is no dependence among them. In terms of time, the successor will be able to execute once one predecessor has been executed. Figure 2 shows how we model scenario 2. We note that there are three predecessor events, B, C, D, which have to be combined to trigger the successor X. Once the predecessors fire, they put one entity token to place p. The entity tokens correspond to different entities.

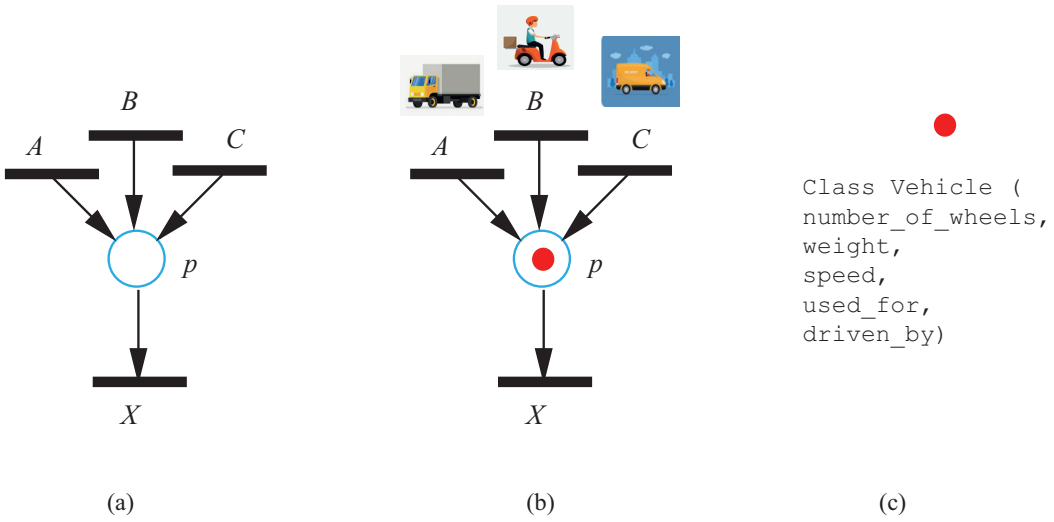


Figure 2. (**a**) Modeling Scenario 2, (**b**) the effects after predecessors fire, (**c**) pseudo-code for an entity declaration.

4.4.3. Scenario 3: An Event Follows the Triggering of Two or More Combined Events

This scenario is typically described as follows:

$$(A \text{ AND } B \text{ AND } C \text{ AND } \cdots) \rightarrow X \tag{3}$$

The modeling of this scenario is shown in Figure 3. To trigger the successor event X, events A and B and C need to be triggered beforehand. The predecessor events are independent of each other. Note that three tokens are needed, one in each of the places $p_1 = p_3$ to trigger X. The predecessors may involve different entities or different instances of the same entities. For example, see Figure 3b: event A is triggered when a lorry is available, while events B and C require the presence of one employee. The class declaration for `Vehicle` and `Employee` are shown in Figure 3c,d. The data pattern extracted reveals that a transportation by lorry requires the presence of two drivers probably with different experience: an experienced and a fresh one. The three tokens from places $p_1 - p_3$ are accumulated in place p_4 as a single token and describe the event of a transportation by means

of a lorry driven by an experienced driver who is accompanied by an inexperienced one. The specific characteristics of each of the drivers (`First_name, Last_name, Birth_date, Years_of_experience, Salary`) are also included in the token, as well as the characteristics of the specific lorry being used (`type, Licence_plate, number_of_wheels, weight, used_for, driven_by`). Thus, an event in p_4 can be fully described as follows: "*The employees John Smith, 12/03/1957, 10 years of experience, with salary EUR 1500 per month and John Roberts, 12/09/1988, 1 year of experience, with salary EUR 1000 per month are on a transportation with a lorry, with licence plate number NIK8259, of weight equal to 1000 Kg, used for long distances and driven by experienced employees*".

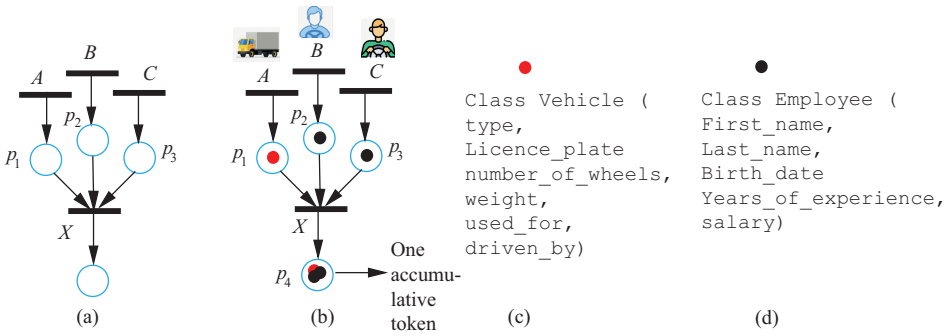


Figure 3. (**a**) Modeling Scenario 3, (**b**) the effects after predecessors fire, (**c**,**d**) pseudo-code for the declaration of two different entities.

4.4.4. Scenario 4: An Event Is Followed by Two or More Events, Which May Be Independent or Combined at a Later Stage

This scenario is typically described as follows:

$$X \rightarrow (A, B \cdots) \tag{4}$$

It is worth noticing that, in cases event X is performed by a combination of entities, then this scenario will separate these entities into different places. For example, after the termination of the event described for p_4 in scenario 3: p_4 can be fully described as follows: "*The employees John Smith, 12/03/1957, 10 years of experience, with salary EUR 1500 per month and John Roberts, 12/09/1988, 1 year of experience, with salary of EUR 1000 per month are on a transportation with a lorry, with licence plate number NIK8259, of weight equal to 1000 Kg, used for long distances and driven by experienced employees*", we will end up with a state where "*The employee John Smith, 12/03/1957, 10 years of experience, with salary EUR 1500 per month*" is now available" (token in p_2), "*The employee John Roberts, 12/09/1988, 1 year of experience, with salary EUR 1000 per month is now available*" (token in p_3) and "*The lorry, with licence plate number NIK8259, of weight equal to 1000 Kg, used for long distances and driven by experienced employees is now available*" (token in p_1). See Figure 4.

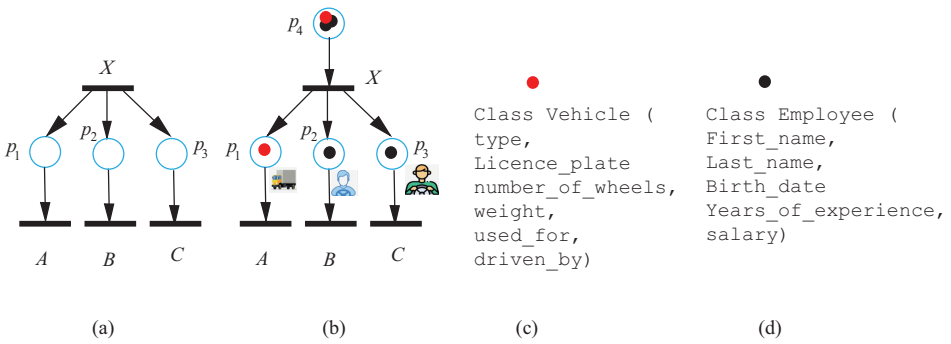


Figure 4. (**a**) Modeling Scenario 4, (**b**) the effects after predecessor X fires, (**c**,**d**) pseudo-code for the declaration of two different entities.

4.5. Timing Considerations

In CPN, every token carries some properties. When considering time, one of these properties has to be the time at which the tokens were generated. Usually, these times follow the Poisson distribution. Let us assume that each token is of a specific class and carries some values (which describe their properties, one ID to distinguish from other tokens of the same class and a timing value:

- *Id* : Token Id.
- *V*: A value or set of values that describe the properties of each token (entity).
- *T*: The time this token has been generated or an updated time following one activity.

Then, we can consider the following cases:

1. *A token resides in a place and it suffices to activate a transition t*: Then, transition t can fire at time T and the result of this firing will be available to the system at time $T + \delta$, where δ is the time required for the corresponding event to occur.
2. *Two or more tokens in different places activate a transition:* In this case, the transition will fire by the time indicated by the last generated token. Clearly, before that time, it is impossible for all the transitions to be active. For example, see Figure 5a. Transition X is active at time 2.5 (the maximum of the two tokens) and its completion is $2.5 + \delta$. The value of δ is randomly chosen between $[0, \ldots, 3]$ (meaning that the duration of X is from 0 to 3 time units). In this example, $\delta = 2$, thus X is completed at time 4.5.
3. *Two or more transitions are active:* To decide the timing sequence, we use the following procedure:
   - We find the time each transaction becomes active based on the input tokens.
   - We choose the one that has been activated earlier.

For example, see Figure 5b. Event X is active at time 2.5 (the first black token has been generated at time 2.0 and the red at time 2.5). Event Y is active at time 4.0 (the first black token has been generated at time 2.0 and the green at time 4.0). Thus, X will fire first and its effects will be available at time $2.5 + 2.0 = 4.5$. Then, Y is still active (by using the second black token generated at time 5.0). Then, it will fire at time 5.0 and its effects will be available at time 5.0 + 2.0 = 7.0 (in both cases, we have assumed that δ has been randomly chosen to be 2.0).

The timing considerations described are generally applicable and can be disregarded in cases our model includes guardian expressions that demand to do so. These expressions are shown in the next subsection.

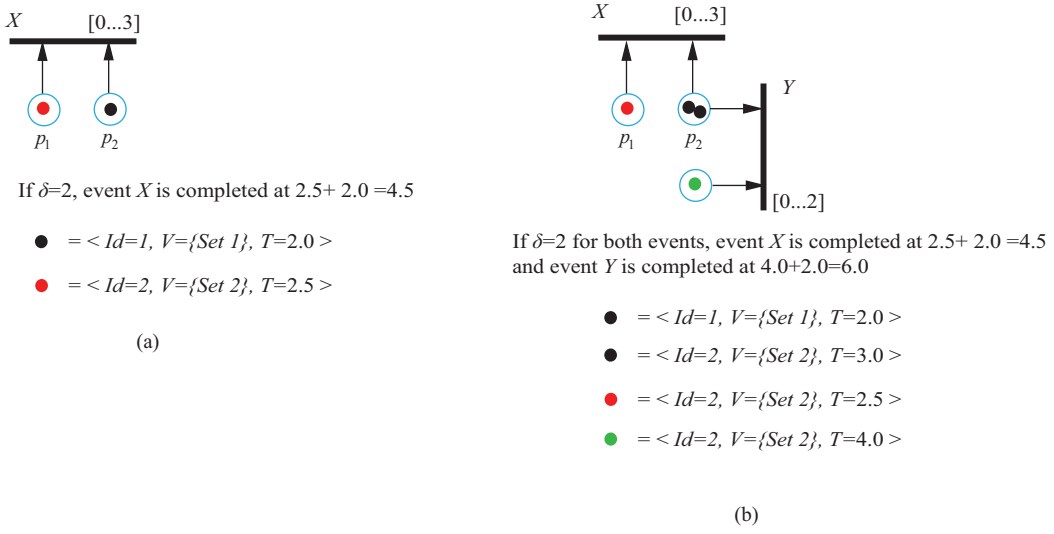


Figure 5. (**a**) Two or more tokens in different places activate a transition, (**b**) two or more transitions are active.

4.6. Guardian Expressions

There are cases where multiple transitions can be enabled at a time. In this case, there must be some type of referee, who decides which transition to fire first. In the CPN, the role of the referee is given to a special type of expressions called guards or guarding expressions. Apart from this role, the guardian expressions can be used to add extra conditions to a transition firing. These expressions are written in parentheses, just above or below the transition. Their general form is as follows:

$$label : \texttt{class.instance} = \text{value} \tag{5}$$

or, in a combined OR/AND expression:

$$label : (\texttt{class.instance} = \text{value}) \quad \text{OR/AND} \quad (\texttt{class.instance} = \text{value}) \ldots \tag{6}$$

The label is usually the name of the transition (a symbol may be added to distinguish between the same events that need to be somehow differentiated, as we will see in our example in Section 4). The guardian expressions require that the tokens that determine a system state and enable a transition should satisfy a certain condition, in order for this transition to fire. In other words, not all the tokens found in the input place of this transition can enable it.

4.7. The Power of Hierarchical Modeling

Instead of producing an entire model, the CPN formalism allows to "break" the system into autonomous units, which are modeled separately and then joined in one model. To implement a hierarchical model, we return to EST and CRT tables of Sections 4.2 and 4.3. The information of Table 1 shows clearly that events $E_6 - E_8$ do follow directly E_5, which, in turn, follows events $E_2 - E_4$. In addition, there is a feedback from E_6 to E_2. In this regard, we can separate the entire set of events into two separate parts, which we name *nests*. The first nest includes events $E_1 - E_4$ and the other nest includes $E_5 - E_8$. The central connecting point between the two nests is E_5 (the central connecting point is an event of one nest, which has the most link-arrows to events of another nest). Figure 6 shows the graphical representation of the connections between the two nests. Notice that we do not show the connections among the event-members of each nest here. Our purpose is to show the "external" links between the two nests. Above these two links (displayed by arrows), we write the events that act as connecting points. The central connecting point is shown in boldface.

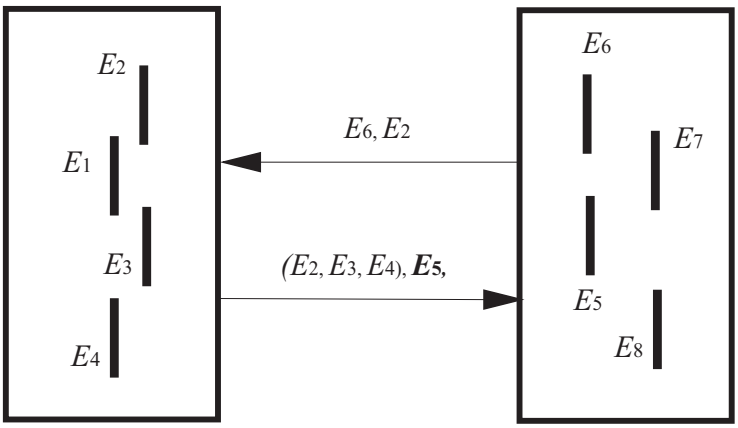


Figure 6. Graphical representation of the hierarchical modeling.

The main advantages of hierarchical modeling are the following:

1. It is extremely helpful in checking our model for errors. Instead of examining the whole model, we can concentrate on the part that seems to be working improperly.

2. We can freely add more places and transitions in the proper nest, without having to consider the entire model.
3. Each nest can be checked independently for possible deadlocks (cases where there is no enabled transition and the system halts). When all the nests are free of deadlocks and properly connected, the entire model is free of deadlocks. This deadlock control is much easier if implemented on a nest basis than implemented for the entire model.
4. The hierarchy does not bother about the classes of objects and their particular instances.

5. Case Study: Modeling and Studying a Courier Company Based on Actual Data

To illustrate how we can apply the ideas presented in Section 3, we show how to develop a model based on real data. For this case study, we have collected data from a branch of a big courier company (we do not provide the name here and the employee names are given by their initials). The data record all the transactions performed for a period of 1 month. The transaction data we received were in the following form (only a small part is obviously shown here).

Case	Activity	Date/Time	Actors	Other Details (Cost, Shipping Address, Weight ...)
1	Reg. Order	01/09, 8:30	JS	50€, Iasonidou 3, Thermi, 40 Kg ...
	Decide "Heavy"	01/09, 8:50	SC	
	Decide "Short"	01/09, 9:02	SC	
	Make shipment plan	01/09, 9:18	DD	
	Check available resources	01/09, 10:00	DD	
	Ship packet	01/09, 12:00,	FG, car	
	Return	01/09. 12:40,	FG, car	
2	Reg. Order	01/09, 8:45	KN	150€, 25 Martiou 3, Veroia, 150 Kg ...
	Decide "Long"	01/09, 9:00	CP	
	Decide "Heavy"	01/09, 9:12	CP	
	Make shipment plan	01/09, 9:52	GI	
	Check available resources	01/09, 10:30	GI	
	Ship packet	01/09, 13:00,	XX & FS, lorry	
	Return	01/09. 15:40,	XX & FS, lorry	
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮
188	Reg. Order	14/09, 8:52	JS	70€, 22 D. Vlachou, Naousa, 1 Kg ...
	Decide "Light"	14/09, 9:19	CP	
	Decide "Long"	14/09, 9:29	SC	
	Make shipment plan	14/09, 9:47	GI	
	Check available resources	14/09, 10:02	GI	
	Ship packet	14/09, 10:08,	AS, motorbike	
	Return	14/09. 10:18,	AS, motorbike	
189	Reg. Order	14/09, 9:21	JS	5€, 11 Solonos , Perea, 4 Kg ...
	Decide "Short"	14/09, 9:32	CP	
	Decide "Light"	14/09, 9:40	SC	
	Make shipment plan	14/09, 11:00	DD	
	Check available resources	14/09, 11:12	DD	
	Ship packet	14/09, 14:00,	PG, car	
	Return	14/09. 16:32,	PG, car	

In addition, we were kindly provided with some general information regarding the branch structure, which includes:

- Three officers, JS, SC, and CP, who are responsible for registering the orders, check the packet weight and shipment address and, based on this information, decide if the packet is "heavy" or "light" and if the distance is "short" or "long".
- Two managers, DD and GI, who decide about the shipment plan and check the available resources (drivers and vehicles).
- Ten experienced drivers, who are entitled to drive cars and lorries. In the part shown here, two instances of "experienced driver" are FG, XX, and PG.
- Fifteen inexperienced drivers, who are entitled to drive cars and motorbikes, but also accompany experienced drivers when packets are transferred over long distances by lorry. In the dataset shown, PS and AS are examples of "inexperienced drivers".

In addition, each driver is entitled to a break after each shipment, which can last from 5 min up to 1 h, depending on the distance they have traveled. Finally, more than one shipments are attached to each shipment plan, but our model is not affected by that fact. Now, we code the activities as follows:

Activity	Encoded as Event	Actors	Other Details
Reg. Order	E_1	JS, SC, CP	...
Decide "heavy"	E_2	JS, SC, CP	...
Decide "light"	E_3	JS, SC, CP	...
Decide "long"	E_4	JS, SC, CP	...
Decide "short"	E_5	JS, SC, CP	...
Make shipment plan	E_6	DD, GI	...
Check available resources	E_7	DD, GI	...
Ship packet	E_8	All driver and vehicle instances	...
Drivers take a break	E_9	All driver instances	...

Based on this encoding, we can create the EST for our log file, as shown in Table 3.

Table 3. EST for the courier company example.

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
E_1	$\parallel$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
E_2	$\leftarrow$	$\parallel$	$\parallel$	$\leftrightarrow$	$\leftrightarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
E_3	$\leftarrow$	$\parallel$	$\parallel$	$\leftrightarrow$	$\leftrightarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
E_4	$\leftarrow$	$\leftrightarrow$	$\leftrightarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
E_5	$\leftarrow$	$\leftrightarrow$	$\leftrightarrow$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
E_6	$\parallel$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\parallel$	$\leftarrow$	$\parallel$	$\parallel$
E_7	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\rightarrow$	$\parallel$	$\rightarrow$	$\parallel$
E_8	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\leftarrow$	$\parallel$	$\rightarrow$
E_9	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\leftarrow$	$\parallel$

The CRT for our courier example is given in Table 4.

Table 4. CRT for our courier example.

E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
JS, SC, CP	JS, SC, CP	JS, SC, CP	JS, SC, CP	JS, SC, CP	DD, GI	DD, GI	All drivers & vehicles	All drivers

Now, let us examine Tables 3 and 4 in detail. Events E_2 and E_3 do not follow each other and they follow strictly E_1. This is a case of *an event that is followed by two or more*

events, which are independent. The same holds for E_4 and E_5. In addition, each of E_2 and E_3 follows and is followed by each of E_4 and E_5. Thus, the two pairs of events should be at the same level of our CPN model, immediately following E_1. In addition, the EST informs us that there is a straightforward sequence after these events, which is composed by E_6 followed by E_7. Events from E_1 and E_6 can form the first net, which we name "shipment plan" nest. The remaining activities are concerned about the actual shipment and they are connected with the "shipment plan" nest via events from E_7 to E_8, as shown in Figure 7. These remaining activities are placed into three different nests, based on the vehicle type and the types of workers who are entitled to drive them.

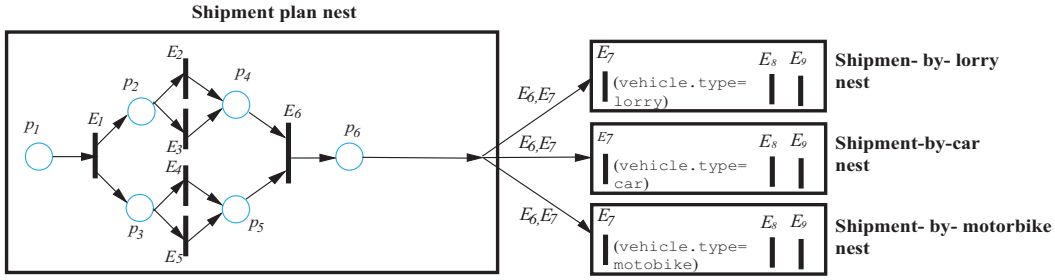


Figure 7. Hierarchical modeling of the courier example.

Figure 8 shows the shipment-by-lorry nest. From Figure 7, notice that once the shipment plan is made and a token is placed in p_6, the flow control is transferred to E_7. This transition is placed in the shipment nests. In the shipment plan nest, p_6 is an input to E_7 along with places $p_7 - p_9$ that show the availability of a lorry and the two types of drivers required: experienced and inexperienced. If E_7 is enabled, that is, the check for available resources finds that there is a lorry, an experienced and an inexperienced driver, then the shipment can start (E_8). This is the third transition modeling scenario, which was described in Section 4.4.3. After the shipment, the two drivers took a rest (places p_{10}, p_{11} have a token), thus E_9 is enabled. Here, there are two events for E_9: E_{9a} refers to the break of the experienced driver and E_{9b} refers to the break of the inexperienced driver. As we explained, they are normally placed above or below the transitions. In addition, letters 'a' and 'b' in the label are used to differentiate the events for experienced and inexperienced drivers. Moreover, note that the resting time for the experienced driver is randomly chosen between 90 and 120 min, while for the inexperienced drivers, this value is between 30–70 min. The guard expressions and the timing values for the transitions have been placed aside for space and clarity reasons.

The shipment-by-car nest is given in Figure 9. This time, we use the transition modeling scenario 1 of Section 4.4.1 to choose either an experienced or an inexperienced driver to make the shipment. This time p_6 is an input to E_{7a} and E_{7b}, which actually checks the available resources. In addition, the car resource inputs both E_{7a} and E_{7b}, which means thaw only one of them will fire: if E_{7a} fires, then an experienced driver will make the shipment, else if E_{7a} fires, then an inexperienced driver will make the shipment. Apparently, if only an experienced or an inexperienced driver is available at a time, he will make the shipment. Finally, note that there is only one input place that inputs transitions E_{9a} and E_{9b}. This means that one of the two events will occur (scenario 1, Section 4.4.1). This is decided by the `driver.type` information found in the token in p_{10}. If it is an experienced driver, then E_{9a} will fire and then the experienced driver will be made available after a break of 30 to 90 min. Similarly, if it is an experienced driver, then E_{9b} will fire and then the inexperienced driver will be made available after a break of 30 to 90 min. (see the guardian expressions aside of Figure 9).

The last nest, shipment-by-motorbike, is shown in Figure 10. Here, things are simpler as only inexperienced drivers are driving the motorbikes. Therefore, no guardian expressions are required and the events are much simpler. Place p_7 that corresponds to the experienced drivers does not exist. In addtion, E_{9a} does not exist. The check is made

for available inexperienced drivers and motorbikes and, if found, the shipment is made. Notice the timing expression near E_{9b}: an inexperienced driver that completes a shipment is entitled to a 10–30 min break. Table 5 gives the meaning of all the places found in the model. In this table, SRN stands for Shipment Plan Nest, SBLN stands for Shipment By Lorry Nest, SBCN stands for Shipment By Car Nest, and SBMN stands for Shipment By Motorbike Nest.

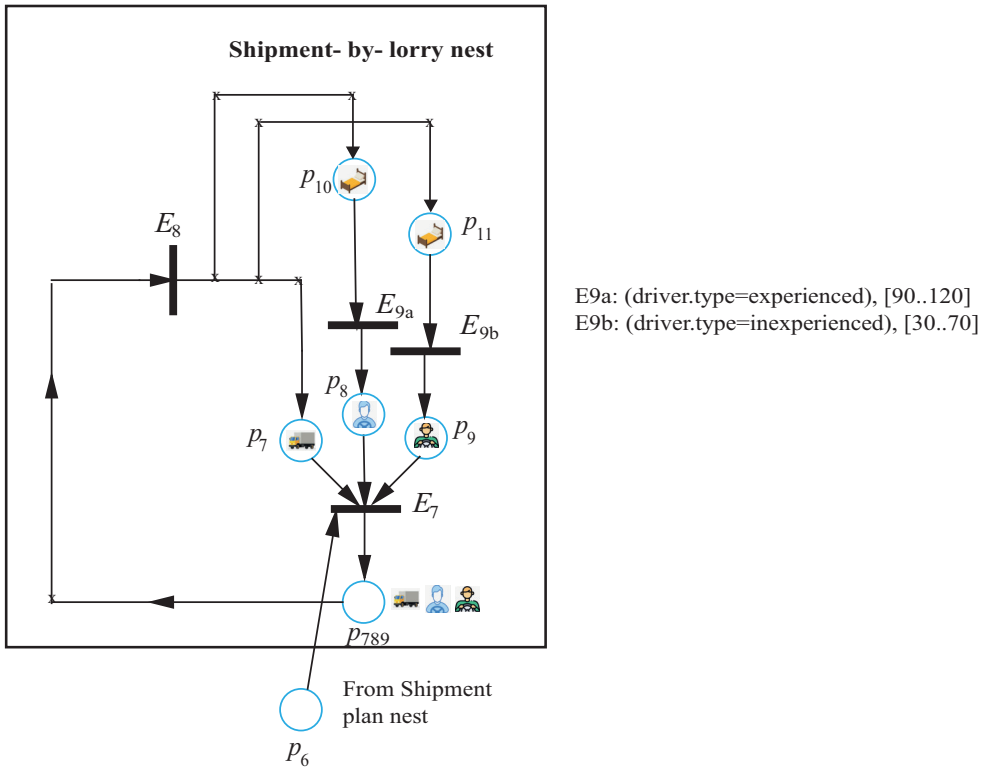


Figure 8. Hierarchical modeling of the courier example: shipment-by-lorry.

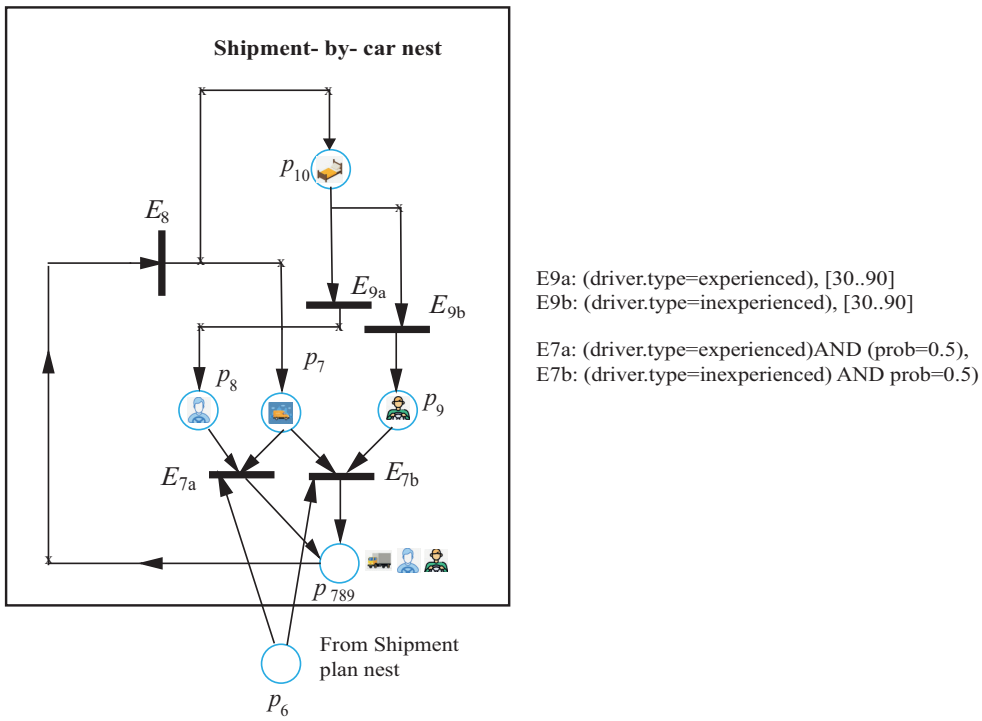


Figure 9. Hierarchical modeling of the courier example: shipment-by-car.

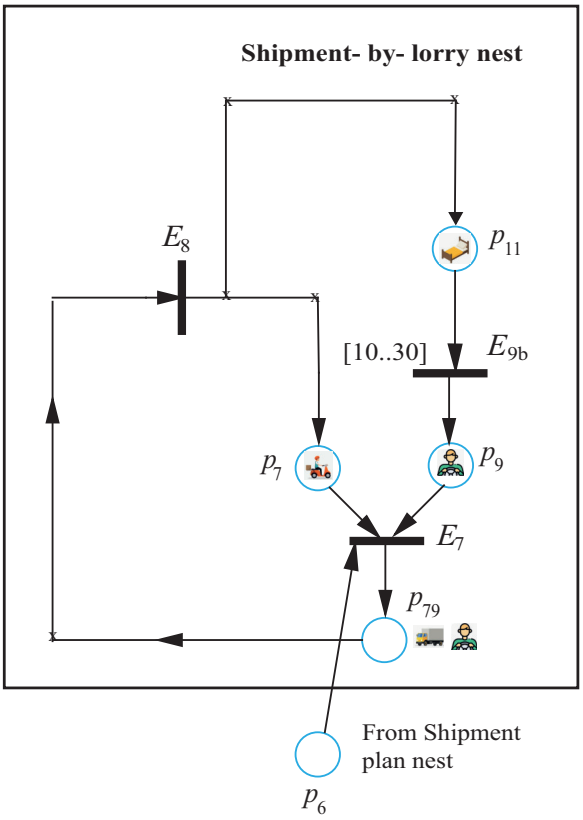


Figure 10. Hierarchical modeling of the courier example: shipment-by-motorbike.

Table 5. Description of places for the courier company model.

Place	System State
SPN.p_1	Request for a shipment
SPN.p_2	Register completed, check weight
SPN.p_3	Register completed, check distance
SPN.p_4	Weight determined for decision
SPN.p_5	Distance determined for decision
SPN.p_6	Shipment decision taken
SBLN.p_7	Available lorry
SBLN.p_8	Available experienced driver
SBLN.p_9	Available inexperienced driver
SBLN.p_{789}	Ready to ship by lorry an experienced and an inexperienced driver
SBLN.p_{10}	Shipment finished, experienced driver break
SBLN.p_{11}	Shipment finished, inexperienced driver break
SBCN.p_7	Available car
SBCN.p_8	Available experienced driver
SBCN.p_9	Available in experienced driver
SBCN.p_{789}	Ready to ship by car and one driver either experienced or inexperienced
SBCN.p_{10}	Shipment finished, experienced or inexperienced driver break
SBMN.p_7	Available motorbike
SBMN.p_9	Available inexperienced driver
SBMN.p_{79}	Ready to ship by motorbike and an inexperienced driver
SBMN.p_{11}	Shipment finished, inexperienced driver break

To conclude, we present the class declarations for the entities involved in this model.

Class vehicle The class `Vehicle` has the following characteristics:

```
Class Vehicle (
number_of_wheels,
plate_ID, weight, speed, used_for,driven_by)
```

There are three instances of class `vehicle`: Lorry, car, and motorbike, each with different characteristics. The `driven_by` characteristic is Experienced for lorries, Experienced and Inexperienced for cars, and Inexperienced for motorbikes.

Class Employee The class `Employee` has the following characteristics:

```
Class Employee (
first_name, last_name,
ID, years_of_experience, salary)
```

The `years_of_experience` characterize each employee as experienced or inexperienced. In our model, all employees with more than 3 years of experience are considered to be experienced, otherwise they are considered to be inexperienced. There are three instances of `Employee`: Officers, Managers, and Drivers.

```
Class Shipment (
shipment_ID, distance,
weight, vehicle_type, employees, start_time, end_time, break_time)
```

This class is used to keep the data about each shipment. These data can be used to examine the efficiency of the drivers and the of the whole company. The `vehicle_type` characteristic records the vehicle used for a shipment. The timing characteristics are the ones used for statistics. The break times that follow each shipment are considered a characteristic of the shipment.

The next section discusses the results we obtained for the human and non-human resource utilization and the shipment completion times by running a number of simulations of our model.

6. Simulation Results and Discussion

As described in the previous section, our model translates real event sequences into CPN transitions using the rules that we have developed. The transitions extracted can be enabled by tokens that can contain different values. Because of the lack of similar modeling approaches, we tested the accuracy of our modeling approach, by comparing our simulation results against the real statistical data. We set up our simulations by assuming that the shipping orders arrive following the Poisson distribution with mean inter-arrival rate from 5 to 25 min. The service times follow exponential distribution with mean service time equal to 5 min for the officers, 5 to 60 min for short distance shipping, and 60 to 240 min for long distance shipping. The break times are also exponential, with mean value of 110 min for experienced drivers that ship products by lorry, 50 min for inexperienced drivers that ship products by lorry, 60 min for all the drivers that take a car, and 15 min for inexperienced drivers that use the motorbike. The figures shown in this section are average values from a set of 20 simulations. To test the validity of our model, we compare the results with the corresponding results obtained from the actual data. These results were averaged for a period of one month. Generally, we found that there is a divergence between the results of our model and the average results. However, this divergence is not so significant and can be explained by the model and the simulation parameters.

6.1. Resource Allocation

Here, we separate the resources into non-human and human. For the vehicles, the actual results indicate that during a working day, the cars and the motorbikes approach

a utilization of almost 100% by about 11.00–12.00. The lorries seem to reach a maximum of about 99% utilization by almost the end of the working day, something that indicates that they are usually used for very long trips and they start later during the working day. Notice that their utilization is much lower (about 88%) by the first working hours, while the motorbikes' utilization approach 97% by early in the morning. Our simulation results indicate a similar behavior, however, the utilization results appear to be 3–4% lower. Probably, this can be explained by two facts: (1) the breaks provided by our model are longer than the actual breaks the employees take, (2) our model considers each shipment as a trip between a starting and a terminal point. Usually, a shipment may include traveling to some intermediate destinations as well, and (3) the actual duration of a trip by either vehicle may differ from the one given by our model, as our model fails to consider factors such as traffic or accidents, etc. Figure 11 shows a comparison between the actual data and our simulation results.

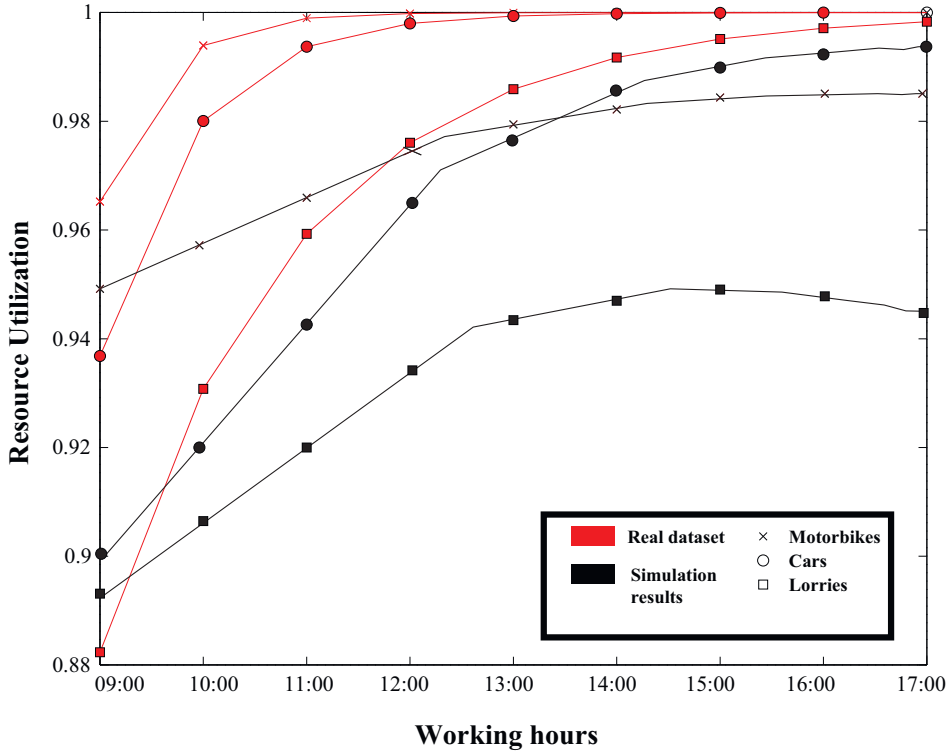


Figure 11. Comparing the actual and simulated non-human resource utilization values.

For the human resources, we averaged the actual data for all the drivers and compared against the average values given by our model. The results are shown in Figure 12. Our simulation result indicate that the employees' utilization is, on the average, about 12% less than the actual one. The peak utilization values were obtained between 12:00 and 14:00 as expected. However, apart from the longer breaks that our model provides (which normally are not taken), our model does not provide equal probability of all drivers of the same class to be used in a shipment, if they are available. However, if one driver has returned from a shipment, even if he has taken a break, his probability of taking over another shipment is reduced compared to a worker that has taken over no shipment or has taken over shorter shipments. This fact somehow reduces the utilization of some drivers, thus reducing the average human resource utilization.

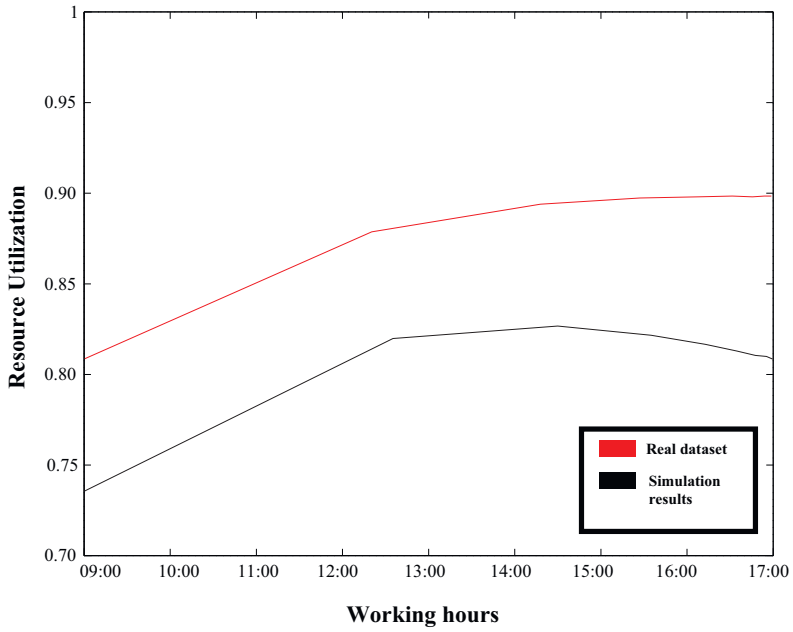


Figure 12. Comparing the actual and simulated human resource (drivers) utilization values.

6.2. Average Number of Jobs (Shipments) Completed

We have also compared the average number of shipments completed during a period of one month, on a weekly basis (5 weeks in total). We note that our model finds that, on average, about 20% are completed compared to the actual data (see Figure 13). Apart from the randomness introduced, our model fails to consider the fact that during a trip (regardless of the vehicle being used), a shipping plan may include more shipping orders than the ones actually scheduled by our model. This is a fact that affects the resource utilization as well.

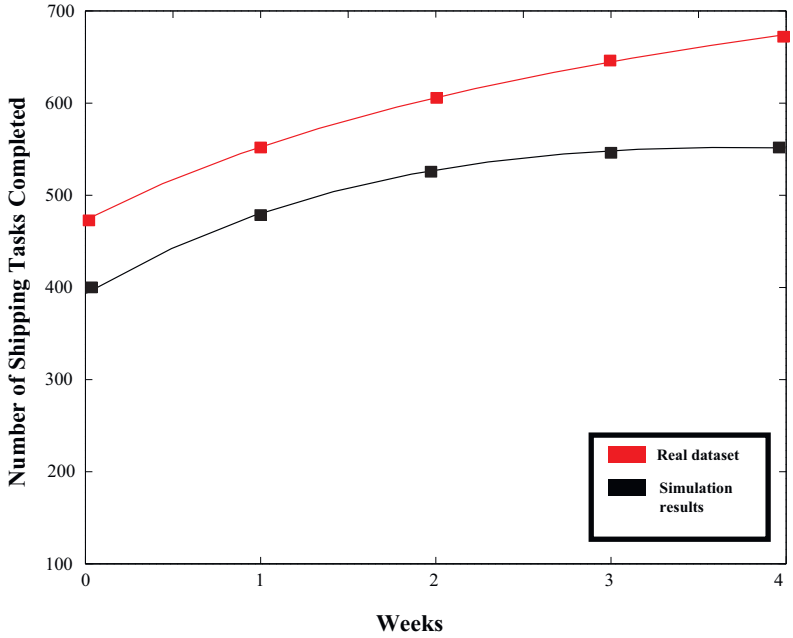


Figure 13. Comparing the actual and simulated average shipment completion values.

From the simulation results we obtained, we made some conclusions regarding our model:

1. The modeling methodology described in Section 3 managed to model a real system based on real data, with quite good accuracy (the divergences were natural, as explained in this section).
2. The hierarchical modeling approach allows the addition of extra details that can help us approach the model with even higher accuracy.
3. Apart from the statistics presented in this section, the model is able to provide statistics for every single entity of the system (worker or vehicle). For example, we could study the utilization of each single driver during a period of time and analyze his efficiency (based on the number of shipping tasks he has completed). This is one of the main strengths offered by our modeling approach: each entity can be separately studied, if necessary.
4. The results can be obtained with numerous different scenarios, for example, reducing or increasing the number of human and non-human resources, without changes to our modeling approach. Thus, our modeling approach can be used as an aid for optimization.
5. The hierarchical modeling approach described in Section 3 can be used to model numerous different real life models.

7. Conclusions

In this work, we presented a novel method for system modeling based on real data using the CPN formalism. To the best of our knowledge, this is the first attempt to impose certain modeling rules based on real data. Based on these rules, the data are transformed into a system model. In addition, we introduced timing characteristics on the generated transitions, which make the model more accurate. For further accuracy, we added guardian expressions, which are used to add extra conditions to the occurrence of an event. Another very important feature is the hierarchical modeling approach, which provides great power to the design of the model, by allowing any change to be implemented locally within a nest, without affecting the rest of the design. We used real data from a courier company to produce a model example using our methodology. The results showed very good accuracy in terms of system utilization and task completion, compared to the actual data.

In the future, we plan to extend our work to include the modeling of more complicated sequences of events. In addition, we plan to introduce a mechanism which examines the frequency of the event sequences in order to provide frequency thresholds. For example, a sequence that has appeared only a few times may be considered as an outlier and excluded from the model. However, such an approach requires careful consideration of real data. Finally, we need to use our approach to model different systems based on their actual data logs. This will help us spot weaknesses of our approach and make proper corrections. One specific area which is of particular interest is community formulation and detection [30,31]. Communities are freely formed everyday in social media and their detection is, to a large extent, relying on the data the users produce. It is a great challenge to try to use the data produced by the users (datasets are freely available, for example, in [32]) to model and study the formulation of social communities.

Author Contributions: Formal analysis, S.S.; Investigation, S.A. and I.K.; Resources, I.K.; Writing—original draft, S.S.; Writing—review & editing, S.A.; Project administration, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Jensen, K.; Kristensen, L.M. *Coloured Petri Nets-Modelling and Validation of Concurrent Systems*; Springer: Berlin/Heidelberg, Germany, 2009.
2. Jensen, K.; Kristensen, L.M.; Wells, L. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.* **2009**, *9*, 213–254. [CrossRef]
3. Heng, A.; Christensen, S.; Mortensen, K.H. Model checking coloured Petri Nets exploiting strongly connected components. In Proceedings of the International Workshop on Discrete Event Systems, Scotland, UK, 19–21 August 1996; pp. 169–177.
4. Garg, K. An approach to performance specification of communication protocols using timed Petri nets. *IEEE Trans. Softw. Eng.* **1985**, *10*, 1216–1225. [CrossRef]
5. Stanton, M.J.; Arnold, W.F.; Buck, A.A. Modelling and control of manufacturing systems using Petri nets. *IFAC Proc. Vol.* **1996**, *29*, 4754-4759. [CrossRef]
6. Kristensen, L.M.; Jørgensen, J.B.; Jensen, K. Application of Coloured Petri Nets in System Development. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 626–685.
7. Peterson, J.L. Petri nets. *ACM Surv.* **1997**, *9*, 223–252. [CrossRef]
8. Souravlas, S.; Katsavounis, S.; Anastasiadou, S. On Modeling and Simulation of Resource Allocation Policies in Cloud Computing Using Colored Petri Nets. *Appl. Sci.* **2020**, *10*, 5644. [CrossRef]
9. Souravlas, S.; Anastasiadou, S. Pipelined Dynamic Scheduling of Big Data Streams. *Appl. Sci.* **2020**, *10*, 4796. [CrossRef]
10. Souravlas, S.; Anastasiadou, S.; Katsavounis, S. More on Pipelined Dynamic Scheduling of Big Data Streams. *Appl. Sci.* **2021**, *11*, 61. [CrossRef]
11. Souravlas, S. ProMo: A Probabilistic Model for Dynamic Load-Balanced Scheduling of Data Flows in Cloud Systems. *Electronics* **2019**, *8*, 990. [CrossRef]
12. Stavros, S.; Katsavounis, S. Scheduling Fair Resource Allocation Policies for Cloud Computing through Flow Control. *Electronics* **2019**, *8*, 1348. [CrossRef]
13. Souravlas, S.; Anastasiadou, S.D.; Tantalaki, N.; Katsavounis, S. A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling. *IEEE Access* **2022**, *10*, 26149–26162. [CrossRef]
14. Souravlas, S.I.; Roumeliotis, M. Petri Net Based Modeling and Simulation of Pipelined Block-Cyclic Broadcasts. In Proceedings of the 15th IASTED International Conference on Applied Simulation and Modelling-ASM, Rhodos, Greece, 26–28 June 2006; pp. 157–162.
15. Shojafar, M.; Pooranian, Z.; Abawajy, J.H.; Meybodi, M.R. Abawajy, and Mohammad Reza Meybodi: An Efficient Scheduling Method for Grid Systems Based on a Hierarchical Stochastic Petri Net. *J. Comput. Sci. Eng.* **2013**, *7*, 44–52. [CrossRef]
16. Shojafar, M.; Pooranian, Z.; Meybodi, M.R.; Singhal, M. ALATO: An Efficient Intelligent Algorithm for Time Optimization in an Economic Grid Based on Adaptive Stochastic Petri Net. *J. Intell. Manuf.* **2013**, *26*, 641–658. [CrossRef]
17. Barzegar, S.; Davoudpour, M.; Meybodi, M.R.; Sadeghian, A.; Tirandazian, M. Traffic Signal Control with Adaptive Fuzzy Coloured Petri Net Based on Learning Automata. In Proceedings of the 2010 Annual Meeting of the North American Fuzzy Information Processing Society, Toronto, ON, Canada, 12–14 July 2010; pp. 1–8.
18. Jaidka, S.; Reeves, S.; Bowen, J. A Coloured Petri Net Approach to Model and Analyze Safety-Critical Interactive Systems. In Proceedings of the 26th Asia-Pacific Software Engineering Conference (APSEC), Putrajaya, Malaysia, 2–5 December 2019; pp. 347–354. [CrossRef]
19. Llopis, J.; Lacasa, A.; Garcia, E.; Montés, N.; Hilario, L.; Vizcaíno, J.; Vilar, C.; Vilar, J.; Sánchez, L.; Latorre, J.C. Manufacturing Maps, a Novel Tool for Smart Factory Management Based on Petri Nets and Big Data Mini-Terms. *Mathematics* **2022**, *10*, 2398. [CrossRef]
20. Adigun, S.O.; Ejidokun, T.; Uche-Ihesiulor, E.C. An Excutable Model for Student Registration System using Timed coloured Petri nets. In Proceedings of the 30th Southern African Universities Power Engineering Conference (SAUPEC), Durban, South Africa, 25–27 January 2022; pp. 1–5. [CrossRef]
21. Esmaeeli, H.; Aleahmad, M. High Level Petri Nets Application for Reliability Visualization on Multi Echelon Supply Chain. In *Logistics and Supply Chain Management*; Molamohamadi, Z., Babaee Tirkolaee, E., Mirzazadeh, A., Weber, G.W., Eds.; LSCM 2020. Communications in Computer and Information Science; Springer: Cham, Switzerland, 2021; Volume 1458. [CrossRef]
22. Ganiyu, R.; Olabiyisi, S.; Badmus, T.A.; Akingbad, O.Y. Development of a Timed Coloured Petri Nets Model for Health Centre Patient Care Flow Processes. *Int. J. Eng. Comput. Sci. IS* **2015**, *4*, 9954–9961.
23. Zurawski, R.; Zhou, M. Petri Nets and Industrial Applications: A tutorial. *IEEE Trans. Ind. Electron.* **1994**, *41*, 567–583. [CrossRef]
24. Bevilacqua, M.; Ciarapica, F.E.; Giovann, M. Timed Coloured Petri Nets for Modelling and Managing Processes and Projects. *Procedia CIRP* **2018**, *67*, 58–62. [CrossRef]
25. Saren, S.; Blaga, F.; Vesselenyi, T. Implementation of fuzzy system using hierarchical Colored Petri Nets to model flexible manufacturing cell. *IOP Conf. Ser. Mater. Sci. Eng.* **2018**, *400*, 42–50. [CrossRef]
26. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 5410–5580. [CrossRef]
27. Johnsonbaugh, R.; Murata, T. Petri nets and marked graphs–mathematical models of concurrent computation. *Am. Math. Mon.* **1982**, *89*, 552–566.
28. van Dongen, B.F.; Van der Aalst, W.M. Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets. In Proceedings of the PNCWB 2005 Workshop, Miami, FL, USA, 20 June 2005; pp. 35–58.

29. Fischer, M. Aris process performance manager. In Proceedings of the 14th GI/ITG Conference-Measurement, Modelling and Evalutation of Computer and Communication Systems, VDE, Dortmund, Germany, 31 March–2 April 2008; pp. 1–3.
30. Souravlas, S.; Anastasiadou, S.D. On Implementing Social Community Clouds Based on Markov Models. *IEEE Trans. Comput. Soc. Syst.* **2022**. [CrossRef]
31. Souravlas, S.; Sifaleras, A.; Katsavounis, S. Hybrid CPU-GPU Community Detection in Weighted Networks. *IEEE Access* **2020**, *8*, 57527–57551. [CrossRef]
32. Leskovec, J.; Krevl, A. *SNAP Datasets: Stanford Large Network Dataset Collection*; Tech. Rep.; University Stanford: Stanford, CA, USA, June 2014. Available online: http://snap.stanford.edu/data (accessed on 13 July 2022).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.