Elevator System. A Case Study of Coloured Petri Nets

Eman Assiri[2], Mohammed Assiri[1] and Ryszard Janicki[1]
[1]Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada L8S 4L8
[2]Mohawk College, Hamilton, Ontario, Canada L9N 1E9

***Abstract*— A fairly general model of the elevator system is presented. Coloured Petri Nets (CPN) and CPN tools are adopted as modeling tools. The model, which is independent of the number of floors and elevators, covers different stages of the elevator system in substantial detail. The model assists simulation-based analysis of different algorithms and rules which govern real elevator systems, including calculating serving time and waiting time. The results prove the compatibility and applicability of this model in various situations and demonstrate the expressive power and convenience of CPN.**

Keywords: Formal Specification, Elevator System, Software Specification Benchmarks, Coloured Petri Nets, Calculating Service Time, Waiting Time

I. Introduction

Coloured Petri Nets (CPN), first proposed in [7] and later substantially modified and enhanced in [8] and [9], are an extension of Petri Nets (c.f. [13]), and are often used to model behaviours of large variety of complex systems. Nevertheless, the question whether or not CPN are an effective technique for modeling real world applications of interconnected communicating systems is still of interest for software developers and modelers. In this paper we discuss a CPN model of the *elevator system*. The elevator system is one of the software engineering benchmarks that are frequently used to test the expressive power, readability and convenience of various formal specification techniques [5].

This paper is an extension and refinement of the results presented in [1], however it can be read independently. The main difference is calculating service time and waiting time.

Various type of Petri nets have been used to model the elevator system before, a detailed list of references can be found in [1], here we mention only [4], [6], [10] and [11].

Nevertheless all the previous models (except [1]) are either static, or the concept of colour as a data type was not fully utilized, or other formalisms as UML were substantially involved.

The model presented in this paper is independent on the number of floors and elevators, it covers in substantial detail different stages of the elevator system. We believe our model is flexible enough to be adapted to different algorithms and rules, and may eventually evolve in a 'standard' formal model of the elevator system.

II. The Elevator System

Elevator systems are an integral aspect of most buildings. Transportation between floors,often with heavy goods, is in many cases almost impossible just by using stairs, especially for high-rise buildings. Quite often multiple elevators are required and such systems are usually very complex. Multiple elevators must be controlled by a centralized control mechanism. The complexity of these elevator systems arises from factors such as scheduling needs, resource allocation, and stochastic control, to name a few. Handling these jobs usually results in systems behaving as discrete event systems [12]. Moreover, the differences among the types of buildings and their traffic patterns also add to the complexity of the elevator systems. For example, elevator passengers in residential, institutional, or commercial buildings might face some mix of popular traffic patterns as: *up-peak traffic* (also called incoming traffic) where the traffic flows mostly from the first floor to other floors, *down-peak traffic* (also called outgoing traffic) where the traffic flows mostly to the first floor from other floors, and *balanced traffic* (also called random traffic) where none of the two previous patterns dominates [2], [14].

The elevator system is usually defined as follows [5]: An *elevator system* is to be installed in a building with m floors and n cars. The elevator and the control mechanisms are supplied by the manufacturer. The internal mechanism of an elevator system is assumed (given). The problem concerns the logistics of moving cars between floors according to the following constraints:

- *Each elevator's car has a set of buttons - one for each floor. These buttons illuminate when pressed and signal the elevator to move to the corresponding floor. The illumination is canceled when the corresponding floor is visited by the car.*
- *On the wall outside the elevator each floor has two buttons (with the exception of the ground and the top floors). One button is pressed to request an upward moving elevator and another button is pressed to request a downward moving elevator. If both buttons are pressed, then each direction is assigned to a different car. These buttons illuminate when pressed. The illumination is canceled when the assigned car visits the floor.*

- *When an elevator has not received any requests for service, it should be held at its parking floor with its doors closed until it receives further requests.*
- *All requests for elevators from floors (i.e. hall calls) must be serviced eventually. The applied algorithm controls the priority of floors.*
- *All requests for floors within elevators (i.e. car calls) must be serviced eventually, with floors usually serviced sequentially in the direction of travel.*
- *Each elevator's car has an emergency button which when pressed causes a warning signal that is sent to the site manager. The car is then deemed "out of service". Each car has a mechanism to cancel its "out of service" status.*

III. Coloured Petri Nets

Coloured Petri Nets are a discrete-event modelling language combining the capabilities of Petri Nets with the capabilities of a high-level programming language. Petri Nets provide the foundation of the graphical notation and the basic primitives for modeling concurrency, communication, and synchronization. Coloured Petri Nets allow tokens to have a data value attached to them. This attached data value is called token colour. Although the colour can be of any arbitrarily complex type, places in CPNs usually contain tokens of one type. This type is referred to as the colour set of the place.

A semi-formal definition can be given as follows:

A *Coloured Petri Net* is a tuple:

$$N = (P, T, A, \Sigma, C, N, E, G, I)$$

where:

- P is a set of places and T is a set of transitions.
- A is a set of arcs and $P \cap T = P \cap A = T \cap A = \emptyset$.
- Σ is a set of colour sets defined within CPN model. This set contains all possible colour, operations, and functions used within CPN.
- C is a colour function which maps places in P into colour in Σ.
- N is a node function which maps A into $(P \times T) \cup (T \times P)$.
- E is an arc expression function which maps each arc $a \in A$ into the expression e. The input and output types of arc expressions correspond to the type of nodes which the arc is connected to.
- G is a guard function which maps each transition $t \in T$ into guard expression g. The output of the guard expression should evaluate to Boolean value true or false.
- I is an initialization function which maps each place p into an initialization expression i. The initialization expression must evaluate to a multiset of tokens with a colour corresponding to the colour of the place $C(p)$.

CPN support hierarchical modeling and are equipped with a modeling language called CPN ML which is based on the standard functional programming language ML. There are a variety of tools that can be used. In this paper the tools from [3] have been used. For more details and theory of CPN, the reader is referred to [9].

Throughout this paper we will use $\mathbb{R}$ ($\mathbb{R}^+$) to denote *Reals* (non-negative *Reals*), $\mathbb{Z}$ ($\mathbb{Z}^+$) to denote *Integers* (non-negative *Integers*), $[x_1, \ldots, x_n]$ to denote the list $x_1, \ldots, x_n$ and $lists(X)$ to denote finite lists built from the elements of X.

IV. CPN-based Modelling of Elevator System

Our model of the elevator system consists of four major interconnected but independent sub-models, namely: *timing car-structure*, *timing hall-call*, *timing car-call*, and *timing system-cycle*.

Their functions and interconnections are described as follows. The timing car-structure sub-model represents the elevator's cars. It is at the centre of all other sub-models that concurrently control the elevator's cars. Typically, an elevator car is requested by either a hall-call or a car-call. A hall-call is placed by pressing a button located in the hallway of a given floor while a car-call is placed by pressing a button inside the car of the elevator. When a hall-call is placed, the timing hall-call sub-model will assign the hall-call to the appropriate car of the timing car-structure sub-model (details depend on the algorithms that are used). Similarly, the timing car-call sub-model coordinates the placed car-calls with the cars of the timing car-structure sub-model. Finally, the timing system-cycle sub-model operates the cars of the timing car-structure sub-model to service the requested calls.

A. Timing Car-Structure

The timing car-structure sub-model (Figure 1) is composed of two places: Timing Cars and Timing Database, that also belong to other sub-models. The Timing Cars place is a Cartesian product defined Table I. The Timing Database place, defined in Table II, is just a list of car states data. Hence, Both places are initialized dynamically by the functions *initialize cars* and *initialize database* respectively (see Figure 1).

Fig. 1: The timing car-structure sub-model

In the colour set *Timing Cars*, the *parking floor* indicates which floor should the car be held on when it is idle, while *stops limitation* predetermines the maximum number of served calls during the simulation-based analysis of traffic congestion. *Stops limitation* is either a positive (small) integer constant, or a function that determines when a new value

TABLE I: The colour set **Timing Cars**

Colour Sets	Definitions
Car ID	$\{i \mid i \in \mathbb{Z} \wedge 1 \leq i \leq total\ number\ of\ cars\}$
Range	$\{x \mid x \in \mathbb{Z} \wedge lowest\ floor \leq x \leq highest\ floor\}$
Status	{*up, down, emergency, idle, out of service*}
Desired Floors	$\{[l_1, \ldots, l_k] \mid l_i \in Range\}$
Call Issuer	{*request, system, non, reservation*}
Timing Hall Call	{(*hall call, direction, time*) | *hall call* $\in$ *Range, direction* $\in$ *Status, time* $\in \mathbb{R}$}
Timing Hall Calls	$\{[h_1, \ldots, l_k] \mid h_i \in Timing\ Hall\ Call\}$
Timing Cars	{(*car id, current floor, status, parking floor, desired floors, call issuer, stops limitation, time period, served hall calls*) | *car id* $\in$ *Car ID, current floor* $\in$ *Range, status* $\in$ *Status, parking floor* $\in$ *Range, desired floors* $\in$ *Desired Floors, call issuer* $\in$ *Call Issuer, stops limitation* $\in \mathbb{Z}$, *time period* $\in \mathbb{R}$ *served hall calls* $\in$ *Timing Hall Calls*}

of stop limitation is produced. For example, if the function returns two, then after accomplishing two calls, a new value for the stop limitation is generated. *Time period* is a real number that shows the operational time of the car, and *served hall calls* is a list if triples $(hall\ \ call, direction, time)$ or *timing hall calls*. *Served hall calls* are used to transfer the tokens of assigned hall calls from the timing hall-call sub-model to the timing system-cycle sub-model. The other elements of the colour set Timing Cars are rather self-explanatory.

TABLE II: The colour sets **Timing Database**

Colour Sets	Definitions
Timing Car's Data	{(*current floor, status, destinations, car id, time*) | *current floor* $\in$ *Range, status* $\in$ *Status, destinations* $\in$ *Desired Floors, car id* $\in$ *Car ID, time* $\in \mathbb{R}$}
Timing Database	$\{[g_1, \ldots, g_k] \mid g_i \in Timing\ Car's\ Data\}$

B. Timing Hall-call

This sub-model executes an algorithm, giving by the user, that assigns a hall-call to the most appropriate car. It also can generate hall-calls from arbitrary floors, selected floors, or both.

Invoking hall calls, which involves button illuminations, starts from place *Hall Buttons*, which contains a single token from the colour set defined in Table III. In principle, it is a list of triples (IB, USB, UUB), where *IB* - illuminated buttons (*IB*), *USB* - unilluminated-specified buttons, and *UUB* - unilluminated-unspecified buttons, are also (internal) lists.

Invocation of a *Hall Call* requires the firing of transition *Release Hall Call*. This transition is enabled if and only if the following conditions are satisfied:

1) At least one of USB or UUB is not empty,
2) If the limit of producing calls is finite, then it has not been already reached,

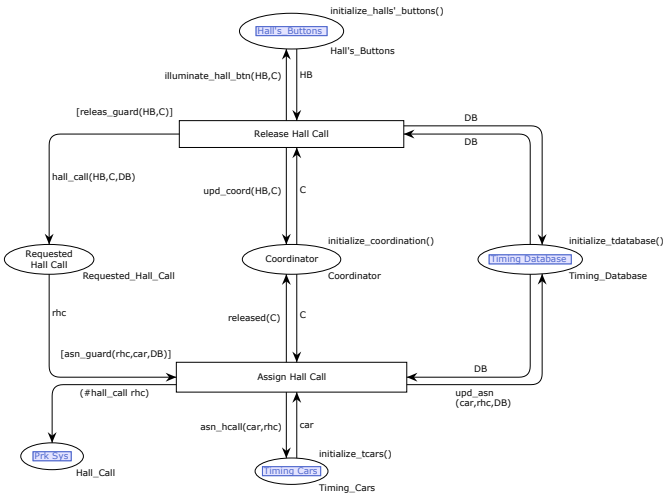

Fig. 2: The timing hall-call sub-model

3) The number of produced calls is less than the value of parameter *pause number* (see Table IV).

The condition (3) guarantees the balance between the producing process and the assignment process.

TABLE III: The colour set **Hall Buttons**

Colour Sets	Definitions
Hall Call	{(*hall call floor, status*) | *hall call floor* $\in$ *Range, status* $\in$ *Status*}
Hall Calls	$\{[h_1, \ldots, h_k] \mid h_i \in HallCall\}$
Hall Buttons	{(*IB, USB, UUB*) | *IB* $\in$ *Hall Calls, USB* $\in$ *Hall Calls, UUB* $\in$ *Hall Calls*}

TABLE IV: The parameters of the timing hall-call sub-model

Parameters	Legal values
Producing mode	$\{finite, infinite\}$
Times of finite hall calls	$\{y \mid y \in \mathbb{Z} \wedge 0 \leq y\}$
Most requested floors	$\{[hc_1, \ldots, hc_k] \mid hc_i \in$ *Hall Call*$\}$
Frequency of most requested floors	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$
Algorithms for assigning hall calls	{*minimum waiting, nearest, scope*}
Travel time	$\{t \mid t \in \mathbb{R}^+\}$
Average stop times	$\{s \mid s \in \mathbb{R}^+\}$
Production pause number	$\{p \mid p \in \mathbb{Z}^+\}$

After firing the transition *Release Hall Call*, an appropriate tuple from either the USB list or the UUB list is removed and placed into both the IB list and place *Requested Hall Call*. The choice between the USB list and UUB list is based on the following rules:

1) When one list is empty, the other list is always selected.
2) The difference between both lengths of lists is less or equal to the value of parameter *frequency of most requested floors*.

3) The internal choice between tuples is sequential for USB list and arbitrary (non-deterministic) for UUB list.

The *waiting time* is calculated in three steps. First, when a hall call is released from place *Hall Buttons* and put in place *Requested Hall Call*, the current times of all cars become attached to this hall call (see Table V). Second, when the placed hall call is assigned to a car, it is removed from place *Requested Hall Call* and the lists: desired-floors and served-hall-calls (see Table I) are appropriately modified.

Third, when the assigned car arrives at the floor of the placed hall call, then the waiting time is calculated as the absolute value of the difference between the time of the car's arrival and the registered time when the hall call was released. For more details and particular algorithms, the reader is refereed to [1].

TABLE V: Colour sets **Requested Hall Call** and **Coordinator**

Colour Sets	Definitions
Car Time	$\{(car\ id, time) \mid car\ id \in Car\ ID,\ time \in \mathbb{R}\}$
Cars Times	$\{[c_1, \ldots, c_k] \mid c_i \in Car\ Time\}$
Requested Hall Call	$\{(hall\ call,\ waiting\ times) \mid hall\ call \in Hall\ Call,$ $waiting\ times \in Cars\ Times\}$
Coordinator	$\{(specified\ call,\ unspecified\ call,\ next\ selection,$ $released\ calls) \mid specified\ call \in \mathbb{Z},$ $unspecified\ call \in \mathbb{Z},\ next\ selection \in \mathbb{Z},$ $released\ calls \in \mathbb{Z}\}$

C. Timing Car-call

This sub-model provides a coordination between a car and its car-calls, and also may generate car-calls.

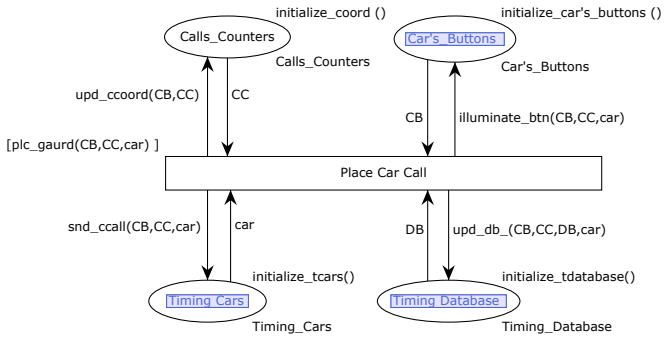

Fig. 3: The timing car-call sub-model

The tokens in place *Car's Buttons* are of the colour *Car Button* that is defined in Table VI. They are quadruples consisting of car identifier and three lists: illuminated car buttons (ICB), unilluminated-specified car buttons (USCB), and unilluminated-unspecified car buttons (UUCB). The process of car buttons illumination and calculation of serving times starts with firing the transition *Place Car Call* (whose input place is *Car's Buttons*.

The transition *Place Car Call* is enabled if and only if the following conditions are satisfied:

1) At least one of the lists UUCB and USCB is not empty,
2) The selected car has not reached its maximum number of accepted calls,
3) If the limit of producing calls is finite, then it has not been reached yet,

Timing algorithm from Timing Hall-call model is used for a given car hall and selected car. Firing the transition *Place Car Call* modifies two lists: *most-desired floors* (see Table VII) and *illuminated car buttons* (ICB), by placing appropriate floor number in both of them respectively.

TABLE VI: Colour sets **Car Buttons** and **Calls Counters**

Colour Sets	Definitions
Floors	$\{[f_1, \ldots, f_k] \mid f_i \in Range\}$
Served Calls	$\{[(floor_1, time_1), \ldots, (floor_k, time_k)] \mid$ $floor_i \in Range, time_i \in \mathbb{R}\}$
Car Buttons	$\{(car\ id, ICB, USCB, UUCB) \mid car\ id \in Car\ ID,$ $ICB \in Served\ Calls, USCB \in lists(Floors),$ $UUCB \in lists(Floors)\}$
Calls Counters	$\{(specified\ call,\ unspecified\ call,\ next\ selection) \mid$ $specified\ call \in \mathbb{Z},\ unspecified\ call \in \mathbb{Z},$ $next\ selection \in \mathbb{Z}\}$

TABLE VII: The parameters of *Timing Car-call*

Parameters	Legal values
Producing mode	$\{finite, infinite\}$
Times of finite car calls	$\{y \mid y \in \mathbb{Z} \wedge 0 \leq y\}$
The most desired floors	$\{[f_1, \ldots, f_k] \mid f_i \in Range\}$
The frequency of most desired floors	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$

D. Timing System-cycle

The Timing System-cycle (Figure 4) models three stages: maintenance, arrival, and transition, of cars operations, and additionally makes a simulation-based analysis possible.

At the arrival stage, when a car reached its desired destination, transition *Arrival* fires and the time of a delivered call is calculated and logged. If the delivered call was a hall call then the waiting time is the absolute value of the difference between the arrival time of the car and the registered time when the hall call was placed. The calculated result is stored in place *Hall Call LOG* (see Table VIII for an appropriate colour sets). Similarly, the serving time of a delivered car call is the absolute value of the difference between the arrival time of the car and the registered time when the car call was placed (see the timing car-call sub-model). The result is stored in place *Car Call LOG* (c.f. Table VIII). The type of a delivered call (i.e. whether it is a hall call, a car call, or both) is determined by checking simultaneously the illuminated-car-buttons list of place *Car's Buttons*, the illuminated-buttons list of place *Hall's Buttons*, and the served-hall-calls

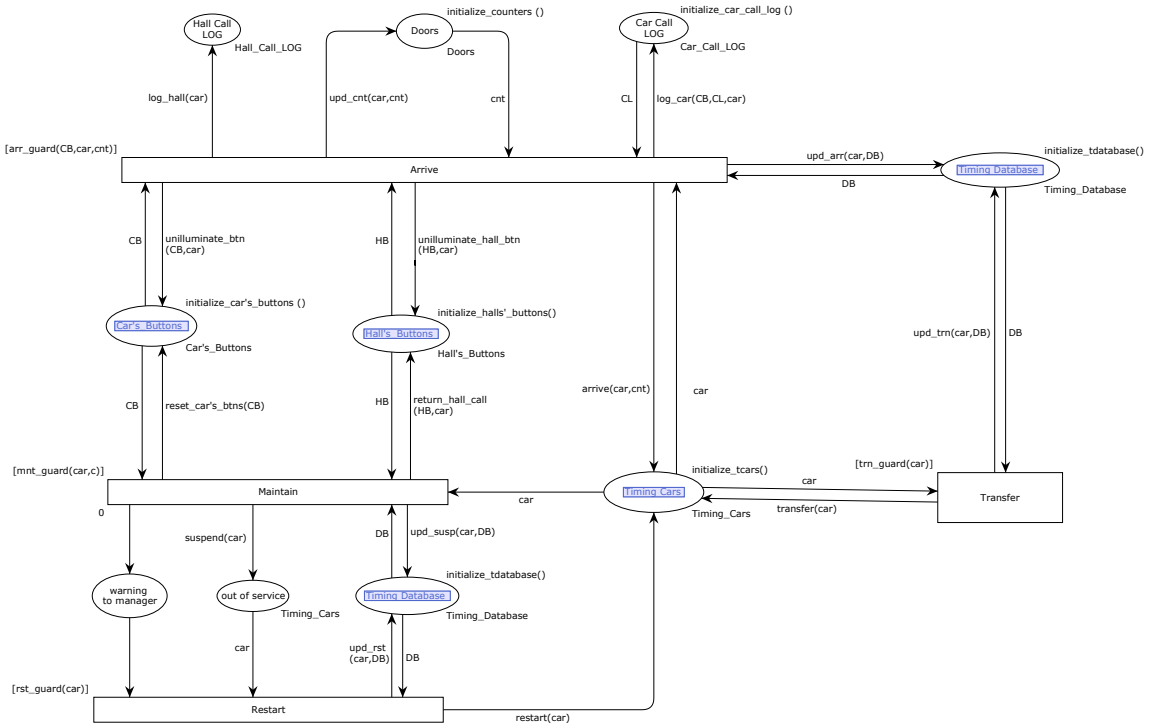

Fig. 4: The timing system-cycle sub-model

of place *Timing Cars*. Moreover, each delivered hall call or car call is returned to its original list after it was removed from the illuminated-buttons list or illuminated-car-buttons list, respectively. The doors operations are represented by the colour set *Doors* (c.f. Table VIII).

TABLE VIII: Colour sets of the arrival stage

Colour Sets	Definitions
Doors	{ (*car id, current delivery number, deliveries' total*) | *car id* $\in$ *Car ID, current delivery number* $\in \mathbb{Z}$, *deliveries' total* $\in \mathbb{Z}$}
Hall Call LOG	{(*hall call, waiting time*) | *hall call* $\in$ *Hall Call, waiting time* $\in \mathbb{R}$}
Serving times	$\{[(f_1, st_1), \ldots, (f_k, st_k)] \mid f_i = floor_i \in Range, st_i = serving\ time_i \in \mathbb{R}\}$
Car Call LOG	{(*car id, serving time*) | *car id* $\in$ *Car ID, serving time* $\in$ *Serving Times*}
Restart cars automatically	$\{yes, no\}$

In the maintenance stage, when a car is suspended either by an emergency case (i.e. emergency button was pressed) or an operation failure case, the transition *Maintain*, which has the highest priority of firing upon enabling in the entire model, is fired. In such case the pending car's token is transferred temporarily to place *out of service*, which is not accessible by any other sub-models, all assigned hall calls and car calls of the pending car are returned to places *Hall's Buttons* and *Car's Buttons*, respectively, and a warning message is sent to the site manager, which is denoted by place *warning to manager*. A suspended car may be restarted, either automatically or manually based on the value of parameter *restart cars automatically* that controls the enabling of transition *Restart*.

Cars movements between floors are modeled in the transition stage. Transitions *Transfer* and *Maintain* are mutually exclusive. Enabling and firing the transition *Transfer* requires that the car desired-floors list is not empty, and the car current floor matches no calls of the desired-floors list. After firing transition *Transfer*, the car token is updated in the following way. If the car desired-floor list has calls beyond the car current floor, it continues moving in the same direction. Otherwise its direction is reversed. In both cases, the token in place Database is updated accordingly.

The transition stage describes the process of moving elevator cars between floors. This is modeled by the firing of transition *Transfer*, which is enabled if transition *Maintain* is not, the car desired-floors list is not empty, and the car's current floor matches no calls of the desired-floors list. After firing transition *Transfer*, the car token is updated as follows. If the car desired-floor list has calls beyond the car current floor, it continues shifting in the same direction. Otherwise its direction is reversed. In both cases, the token in place *Database* is updated accordingly.

Once a car reached its desired destination, it is in the arrival stage. At this stage, transition *Arrive* is enabled

provided that transition *Maintain* is disabled and the car current floor matches a requested call from the desired-floors list. After firing transition *Arrive*, car's token is updated by dropping the requested floor from the car desired-floor list. Additionally, the car state is set to one of three cases. If the car desired-floor list has more calls, then it continues serving the requested calls. Otherwise, the car is set to idle, if the car current floor agrees with its parking floor, or alternatively, the car is dispatched to its parking floor with an appropriate direction.

V. Analysis

CPN models can be analyzed and evaluated by various techniques and tools [15]. In this paper, we used the simulation-based performance analysis using tools from [3], with various parameters including numbers of cars and floors, types of decision algorithms, specified floors, etc. The results of our simulation-based performance analysis has proved validity and applicability of our model in various situations.

Due to page limit, we mention only one case study: *five cars serving a twenty-floor building*. Among others, we obtained the following results.

1) All requested hall calls were eventually served for all algorithms used.
2) All requested car calls were served eventually and sequentially in the direction of car movements.
3) Car calls, hall calls and their illumination buttons were synchronized properly.
4) If no calls, all cars were held at their parking floors.
5) The maintenance stage worked as it supposed to.

The results of two particular experiments are presented in Figure 5.

VI. Conclusion

In this paper, a fairly general CPN-based model of the elevator system, one of specification benchmarks [5], is proposed. The model consists of four separated but interconnected parts and it emphasis the expressive power and convenience of Coloured Petri Nets. Our model is quite flexible and it allows using different algorithms and different rules at ease. Division into four sub-models allows easy tracking of errors and faults. A thorough simulation-based performance analysis by using the simulator provided in [3].

Acknowledgements

The first author was supported by Prince Sattam bin Abdulaziz University, the second author was supported by the Ministry of Education of Saudi Arabia, while the third author acknowledges partial support by NSERC Discovery Grant of Canada.

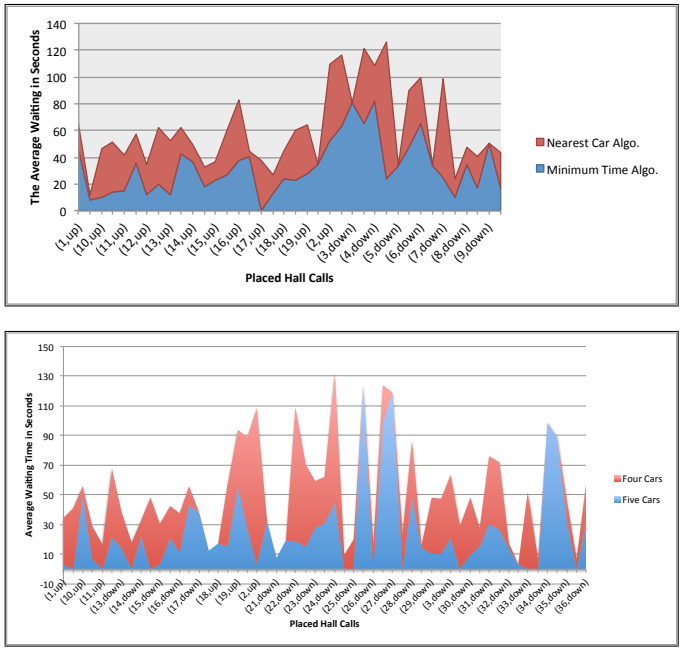

Fig. 5: A comparison of *Nearest Car* and *Minimum Time* algorithms (top) and a comparison of four cars with five cars (bottom).

References

[1] M. Assiri, M. Alqarni and R. Janicki, Modeling Elevator System With Coloured Petri Nets, Proc. of SERP'2015 (Software Engineering Research and Practice), Las Vegas, Nevada, USA, July 27-30, 2015, pp. 183-189, CSREA Press

[2] G. Barney, *Elevator Traffic Handbook: Theory and Practice*. Taylor & Francis, 2003.

[3] CPN Tools AIS Group, The University of Technology, Eindhoven, The Netherlands, http://www.cpntools.org.

[4] E. S. Etessami and G. S. Hura, "Abstract Petri net based approach to problem solving in real time applications," *Fourth IEEE Region 10 International Conference*, pp. 234–239, 1989.

[5] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Eds., *Fundamentals of Software Engineering*, 2nd ed. Pearson Prentice Hall, 2003.

[6] Y.-H. Huang and L.-C. Fu, "Dynamic scheduling of elevator systems over hybrid Petri net/rule modeling," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1998, pp. 1805–1810.

[7] K. Jensen, "Coloured Petri Nets and the Invariant Method," *Theoretical Computer Science*, vol. 14, no. 3, pp. 317–336, 1981.

[8] K. Jensen, *Coloured Petri Nets*. Springer, 1994.

[9] K. Jensen and L. M. Kristensen, "Coloured Petri Nets Modelling and Validation of Concurrent Systems." Berlin: Springer, 2009.

[10] C.-H. Lin and L.-C. Fu, "Petri net based dynamic scheduling of an elevator system," in *IEEE International Conference on Robotics and Automation*, vol. 1, 1996, pp. 192–199.

[11] D. Liqian, Z. Qun, and W. Lijian, "Modeling and analysis of elevator system based on timed-coloured Petri net," in *Fifth World Congress on Intelligent Control and Automation*, vol. 1, 2004, pp. 226–230.

[12] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," in *Proceedings of the IEEE 77.1*, 1989, pp. 81–98.

[13] W. Reisig, "Petri nets, an introduction, 2nd ed." Berlin: Springer Berlin Heidelberg., 1991.

[14] G. R. Strakosch and R. S. Caporale (eds), *The Vertical Transportation Handbook*, Wiley 2010.

[15] W. M. P. van der Aalst and C. Stahl, *Modeling Business Processes: A Petri Net-Oriented Approach*. The MIT Press, 2011.