Modeling Elevator System With Coloured Petri Nets

MODELING ELEVATOR SYSTEM WITH COLOURED PETRI NETS

BY

MOHAMMED ASSIRI, B.Ed.

A THESIS

SUBMITTED TO THE DEPARTMENT OF SOFTWARE ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2015) McMaster University
(Software Engineering) Hamilton, Ontario, Canada

TITLE:	Modeling Elevator System With Coloured Petri Nets
AUTHOR:	Mohammed Assiri B.Ed., (Major of Computer) King Khaled University, Abha, Kingdom of Saudi Arabia
SUPERVISOR:	Dr. Ryszard Janicki
NUMBER OF PAGES:	xiii, 94

Dedication

To my inspiring parents,

Saad and Sharifah

To my supportive and beloved wife,

Eman

To my sweet daughters,

Lara and Rema

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. The results prove the compatibility and applicability of this model in various situations and demonstrate the expressive power and convenience of CPN.

Acknowledgements

First and foremost, all the praises and thanks are due to Allah Almighty for giving me the ability and strength to accomplish this thesis. Then, I would like to extend my sincere gratitude and appreciation to my supervisor Prof. Ryszard Janicki for his thoughtful guidance, continuous support, and constructive feedback through my MASc Journey.

I would also like to express my appreciation to my examination committee, Prof. Alan Wassyng and Prof. Emil Sekerinski for their valuable comments. As well, I would like to acknowledge my colleague Mohammed Alqarni for the productive discussions and insightful suggestions that have contributed significantly to the progress of this thesis. In addition, I am grateful to Prince Sattam bin Abdulaziz University for providing the financial support to undertake this work.

Last, but certainly not least, I wish to extend my heartfelt gratitude to my inspiring parents, my dear siblings, my beloved wife, and my sweet daughters for their endless love, prayers, and encouragement that have made all of my achievements possible.

Contents

Abstract **iv**

Acknowledgements **v**

1 Introduction **1**

1.1 The Elevator System 1

1.2 Objectives 2

1.3 Related Works 3

1.4 Contributions 4

1.5 Thesis Outline 5

2 Coloured Petri Nets **6**

2.1 Overview 6

2.2 Informal Introduction to CPN 7

2.2.1 The Structure of CPN-based Models 7

2.2.2 CPN ML Language 8

2.2.3 A Simple Example 10

2.3 Formal Definitions 12

2.4 Computer Tools 14

3 The Abstract Version of CPN-based Model of the Elevator System 16

3.1 Introduction 16

3.2 The Abstract Car-Structure Sub-model 20

3.2.1 Dynamic Initialization of Places 23

3.3 The Abstract Hall-Call Sub-model 24

3.4 The Abstract Car-call Sub-model 31

3.5 The Abstract System-cycle Sub-model 34

4 The Timing Version of CPN-based Model of the Elevator System 39

4.1 Introduction 39

4.2 The Timing Car-Structure Sub-model 40

4.3 The Timing Hall-call Sub-model 44

4.4 The Timing Car-call Sub-model 48

4.5 The Timing System-cycle Sub-model 51

5 The Parking Optimizer Model 55

5.1 Introduction 55

5.2 The Election Sub-model 57

5.3 The Assignment Sub-model 61

5.4 The Position Sub-model 63

6 The Analyses 68

6.1 Introduction 68

6.2 The Reachability Analysis 69

6.3 The Simulation-based Performance Analysis 75

7 Conclusion **87**

7.1 Discussion 87

7.2 Future Work 88

List of Figures

2.1 The basic components of CPN-based models 7

2.2 The process of illuminating buttons (as an example) 11

2.3 An example of a hierarchical CPN-based model 12

2.4 An example of the visualization extension 15

3.1 The sub-models of the elevator system 17

3.1 The development phases of the proposed model 20

3.2 The abstract car-structure sub-model 21

3.3 The abstract hall-call sub-model 25

3.4 Assigning hall calls to cars 26

3.5 Generating arbitrary and identified floors' numbers 29

3.6 Adding directions to the produced floor numbers 31

3.7 The abstract car-call sub-model 32

3.8 Coordinating between cars and their car calls 33

3.9 Producing arbitrary car calls 33

3.10 Producing specified car calls 34

3.11 The abstract system-cycle sub-model 35

3.12 The maintenance stage 36

3.13 The transition stage 37

3.14 The arrival stage 38
4.1 The timing car-structure sub-model 41
4.2 The timing hall-call sub-model 44
4.3 Releasing hall calls 46
4.4 Assigning timing hall calls 48
4.5 The timing car-call sub-model 49
4.6 The timing system-cycle sub-model 51
4.7 The arrival stage of the timing system-cycle sub-model 52
4.8 The maintenance stage of the timing system-cycle sub-model 54
5.1 The parking optimizer model 56
5.2 The election sub-model 57
5.3 Counting the placed hall calls 58
5.4 Sorting the floors of the placed hall calls 59
5.5 Electing the most requested floors 60
5.6 The assignment sub-model 61
5.7 Processing the floors by the assignment sub-model 62
5.8 Altering the cars' parking floors by the assignment sub-model 63
5.9 The position sub-model 64
5.10 Identifying the scope of floors 65
5.11 Completing the information of scopes 66
5.12 Altering the cars' parking floors by the position sub-model 67
6.1 The standard report of the abstract version (1) 71
6.1 The standard report of the abstract version (2) 72
6.2 The standard report of the timing version (1) 73

6.2 The standard report of the timing version (2) 74
6.3 The monitors' statistics of the abstract version 77
6.4 Produced calls from the abstract version 78
6.5 Results from the abstract version 79
6.6 Observing different experiments through the Visualization Extension 80
6.7 The monitors' statistics of timing version 81
6.8 The calls of the timing version 82
6.9 Results from the timing version 83
6.10 Results from the techniques of the parking optimizer model 84
6.11 The comparison between two different algorithms 85
6.12 The comparison between different numbers of cars 86

List of Tables

2.1 The definitions of the simple example 11

3.1 The colour set **Cars** 21

3.2 The colour set **Database** 23

3.3 The essential parameters of the car-structure sub-model 24

3.4 The colour set **Hall Call** 26

3.5 The parameters of the abstract hall-call sub-model 30

3.6 The colour set **Specific Floors** 33

3.7 The parameters of the car-call sub-model 34

3.8 The parameter of the system-cycle sub-model 36

4.1 The colour set **Timing Cars** 41

4.2 The colour sets **Timing Database** 43

4.3 The extra parameter of the timing car-structure sub-model 43

4.4 The colour set **Hall's Buttons** 45

4.5 The parameters of the timing hall-call sub-model 46

4.6 The colour sets **Requested Hall Call** and **Coordinator** 48

4.7 The colour sets **Car's Buttons** and **Calls' Counters** 50

4.8 The parameters of the timing car-call sub-model 50

4.9 The colour sets of the arrival stage 53

5.1 The parameters of the parking optimizer model 56

5.2 The colour sets of the election sub-model 59

5.3 The colour set **Identified Floor** 61

5.4 The colour sets **Scope** and **Scope Statistics** 64

Chapter 1

Introduction

This chapter overviews the elevator system. Section 1.1 introduces the elevator system. Section 1.2 defines the thesis objective. Section 1.3 presents related works. Section 1.4 lists the thesis contributions. Finally, section 1.5 presents the thesis outline.

1.1 The Elevator System

Elevator systems are an integral aspect of buildings from the point at which they are first designed. Transportation between floors, especially with heavy goods, is almost impossible using stairs for ordinary people at least. With high-rise buildings being the typical candidate for elevator systems, 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 Ramadge and Wonham (1989). Moreover, the differences among the types of buildings and their traffic patterns arise also the complexity of the elevator systems. For example, passenger elevators may exist in residential, institutional, or commercial buildings with some or mix of these popular traffic patterns [George R. Strakosch (2010); Barney (2003a)]: 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.

1.2 Objectives

This thesis focuses on proposing a model of the elevator system that fulfilments the constraints of the following definition [Ghezzi *et al.* (2003)]: 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:

1. 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 cancelled when the corresponding floor is visited by the car.

2. 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 cancelled when the assigned car visits the floor.

3. 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.

4. All requests for elevators from floors (i.e. hall calls) must be serviced eventually. The applied algorithm controls the priority of floors.

5. All requests for floors within elevators (i.e. car calls) must be serviced eventually, with floors usually serviced sequentially in the direction of travel.

6. 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.

1.3 Related Works

The elevator system is one of the software engineering benchmarks which are frequently used to test the expressive power, readability, and convenience of various formal specification techniques Ghezzi *et al.* (2003). Petri Nets is one formal specification technique.

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 [Ghezzi *et al.* (2003)]. It has been modeled many times in the past, and that includes Petri Nets.

In [Lin and Fu (1996)] and [Huang and Fu (1998)], dynamic scheduling of the elevator system was modeled by Petri Nets, and hybrid Petri Nets. Timed Petri Nets, Abstract Petri Nets and Elevator Control Petri Nets were used in [Cho *et al.* (1999); Etessami and Hura (1989); Ahmad *et al.* (2014)], respectively. Furthermore, the elevator system was modeled by Coloured Petri Nets in [Fernandes *et al.* (2007)], and Timed Coloured Petri Nets in [Liqian *et al.* (2004)] and [Ye *et al.* (2011)].

Nevertheless, all of these previous models are either static or dependent on a particular number of elevators and floors (often one place was required for each elevator car), the concept of colour as a data type was not fully utilized, or other formalisms such as UML were substantially involved.

1.4 Contributions

Modeling the elevator system by means of Coloured Petri Nets supports the proposed model to not only fulfilment the thesis objective (section 1.2), but also to distinguish by the following attributes:

- The proposed model is independent of the number of floors and cars. (i.e. the number of places and transitions is fixed for any given number of floors and cars). This is not only because of the appropriate choice of the modelling language, but also because of the proper structure of the model.
- The proposed model covers different stages of the elevator system in substantial detail through the division of the model into sub-models to simplify the structure and also to allow easier tracking of errors and faults.

- The proposed model is flexible enough to be adapted to different algorithms and rules that govern real elevator systems, and may eventually evolve into a 'standard' formal model of the elevator system.

1.5 Thesis Outline

The structure of this thesis is as follows:

- **Chapter 2** introduces the principles of Coloured Petri Nets (CPN), which is the modeling language of the proposed model.
- **Chapter 3** demonstrates the abstract version of the elevator system, which concerns the logistics of moving the elevator system.
- **Chapter 4** presents the timing version of the elevator system, which extends the abstract version to include more features.
- **Chapter 5** discusses two techniques of the parking optimizer model, which is expected to reduce the waiting time.
- **Chapter 6** provides the analyses of the proposed model through two techniques.
- **Chapter 7** concludes this thesis and mentions future work.

Chapter 2

Coloured Petri Nets

This chapter presents briefly the concepts of Coloured Petri Nets (CPN). Section 2.1 overviews CPN. Section 2.2 discusses informal introduction to CPN. Section 2.3 defines CPN. Finally, section 2.4 presents the computer tools of CPN.

2.1 Overview

Coloured Petri Nets (CPN) was first proposed in [Jensen (1981)] and later substantially modified and enhanced in [K. Jensen (1994)]. CPN is an extension of Petri Nets (c.f. [Reisig (1991)]), and it is often used to model behaviours of rather complex systems. CPN preserves useful properties of Petri Nets and at the same time extend the initial formalism to allow the distinction between tokens. Coloured Petri Nets (CP-nets or CPNs) is a graphical language for constructing models of concurrent systems and analysing their properties. CPN is a discrete-event modeling language that combines the capabilities of Petri Nets with the capabilities of high-level programming languages. Petri Nets provides the foundation of the graphical notations and the basic primitives for modeling concurrency, communication, and synchronisation.

2.2 Informal Introduction to CPN

This informal introduction demonstrates the main graphical notations of CPN-based models and the basics of CPN ML language with a simple example.

2.2.1 The Structure of CPN-based Models

A graphical structure of a CPN-based model is built using places, transitions, and arcs (see Figure 2.1).

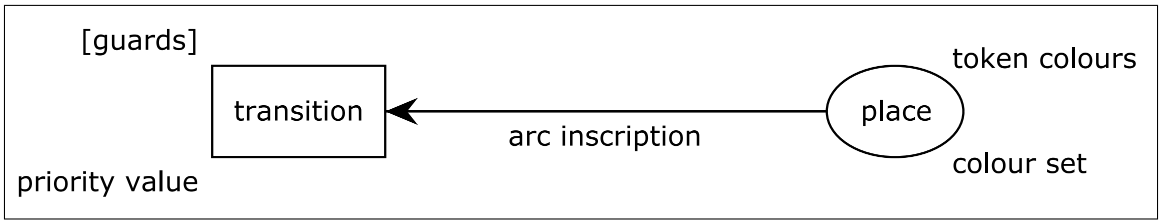


Figure 2.1: The basic components of CPN-based models

Places are drawn as ovals that represent the states of the model. Each place is associated with one colour set and either the multi-set of the token colours or the empty set. The colour set, which is written as an inscription near the place, determines the syntactic format of the data values that are attached to tokens. The possible colour sets are given in section 2.2.2.

Arcs are drawn as arrows that guide the flow of the model. Each arc has an inscription that declares explicitly the quantities and the types of the transferred token colours.

Transitions are drawn as rectangles that represent the events of the model. Each transition has input and output places to which they are connected through arcs (i.e. input places are those connected by arcs going into transitions while output places are those connected by arcs going from transitions). In addition, each transition may be associated with guards and/or a priority value. They, by default, are located at the top-left corner and the bottom-left corner of the transition, respectively. The guards enable or disable the transitions while the priority values administer the sequence of enabled transitions.

Places and transitions may have names, which are written inside them. Even though names are not formal (i.e. do not affect the execution of the model), meaningful names can improve the readability of the model dramatically.

2.2.2 CPN ML Language

CPN ML language is a flexible, expressive, and extensible language that is founded on the functional programming Language: **Standard ML** (SML/NJ implementation). CPN ML provides modeling ability similar to high-level programming languages, where both places and transitions are explicitly described, and data types and hierarchical decomposition are supported. Through the use of CPN ML language, the modeling nets have not only types and inscriptions, but also colour sets, token colours, variable declarations, and functions.

The colour set can be simple or compound. A simple colour set is defined by one of five basic types: *integer (int), real, string, Boolean (bool), or unit.* These five types are inherited from Standard ML into CPN ML (also exist in major programming languages) and therefore are self-explanatory. The construct *"with"* allows defining both a subset of a basic set and a simple enumeration set.

A compound set is a combination of different pre-defined colour sets. It is formed by one of three constructors; *product*, *record*, or *list*. The former two constructors differ syntactically because the *record* constructor labels each combined set while the *product* constructor does not. However, they are similar semantically as they define a fixed content structure. In contrast, the *list* constructor defines a flexible length list of a predefined set.

A non-empty token colour is constructed as "$M`S$" where M denotes the multiplicity of the token colour S. Multiple token colours are structured by a special operator [++], as:

$$M1`S1 + +M2`S2 + +M3`S3$$

An arc inscription is an expression that evaluates to a multi-set. An expression is structured from constants, declared variables, defined functions, or a combination of them. Similar to expressions in programming languages, arc expressions also may contain arithmetic operators. Besides distributing token colours over places, arc expressions may also alter the values of transferred token colours.

A transition guard is a Boolean expression that evaluates to either true or false. It is enclosed by square brackets, and requires at least one of comparison operators such is $=, <> \ (means \neq), <, >, <= \ (means \leq)$, or $>= \ (means \geq)$. Additionally, a compound expressions is formed by logic operators such as *not*, *andalso* (means *and*), *orelse* (means *or*).

Finally, functions are the best candidate for complex calculations or extended expressions. Functions may involve arc inscriptions, transition guards, or initialized token colours. Note that the use of node functions and arc expression functions allows multiple arcs to connect the same pair of nodes with different arc expressions. All these concepts are easily interpreted when any graphical example is seen as in Figure [2.2], for example.

2.2.3 A Simple Example

Figure 2.2 shows a simple model that demonstrates the process of the buttons illumination. In this model, each button a single state either *the button is unilluminated* or *the button is illuminated*. To switch between the two states, we need two transitions, *illuminate* and *unilluminate*. Finally, the arcs connect the places and the transitions.

Furthermore, we use the CPN ML language to define the colour sets of the places and the expressions of the arcs. Firstly, both places have the colour set **Buttons**, which is a compound colour set consists of two simple sets defined in Table 2.1. Consequently, each token has a floor's number and illumination's state. Secondly, the arcs that connect the places to the transitions are called input arcs and expressed by the variable *button*. In contrast, the arcs that connect the transitions to the places are called output arcs and expressed by functions *illuminate* and *unilluminate* (see table 2.1).

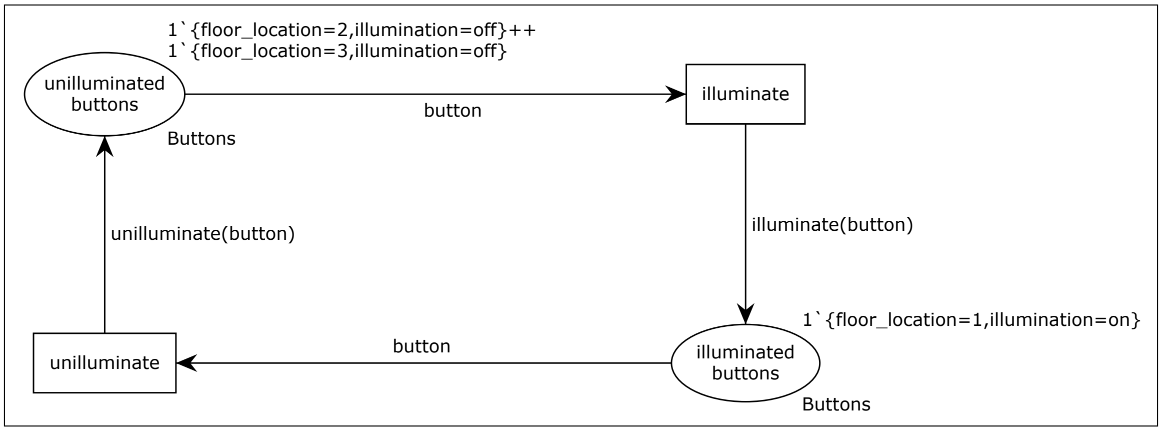


Figure 2.2: The process of illuminating buttons (as an example)

Table 2.1: The definitions of the simple example

```
colset Floor = int; (* a simple colour set *)

var i:Floor; (* variable *)

colset Light = bool with (on,off); (*enumeration colour set *)

colset Buttons = record floor_location:Floor *

                 illumination:Light; (* compound colour set *)

var button:Buttons; (* variable *)

fun illuminate(button:Buttons)= (* function *)

{floor_location=(#floor_location button),illumination=on}

fun unilluminate(button:Buttons)= (* function *)

{floor_location=(#floor_location button),illumination=off}
```

The process of illuminating buttons starts from place *unilluminated buttons* when each token represents an elevator button of the colour set **Buttons**. Illuminating a button requires transition *illuminate* to fire (i.e. be enabled). In this model, all transitions have no guards, and thereby they are always enabled as long as there is a token in the input places. After the firing of transition *illuminate*, a token if removed from the input place *unilluminated buttons* and placed through connected arcs into the output place *illuminated button*. The process of unilluminating buttons is performed similarly.

Finally, CPN supports hierarchical modeling in a fashion similar to programs being constructed from modules. For instance, figure 2.3 has one place that represents the state of *unilluminated buttons*, and a hierarchical model that represents that *Illumination process*. Hence, Figure 2.3 and Figure 2.2 are equivalent.

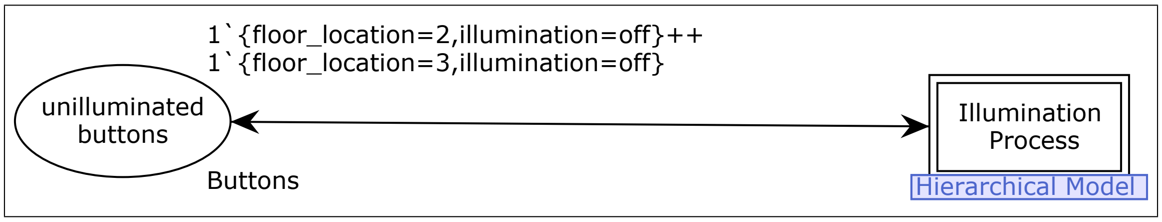

1`{floor_location=2,illumination=off}++
1`{floor_location=3,illumination=off}
unilluminated buttons
Buttons
Illumination Process
Hierarchical Model

Figure 2.3: An example of a hierarchical CPN-based model

2.3 Formal Definitions

A non-hierarchical and a hierarchical Coloured Petri Nets are defined as follows [Jensen and Kristensen (2009)]:

- A Non-hierarchical Coloured Petri Net is a tuple:

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

  - P is a set of places.
  - T is a set of transitions.

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

• A Hierarchical Coloured Petri Nets is a tuple:
$CPN_M = (CPN, T_{sub}, P_{port}, PT), where:$

- $CPN = (P, T, A, \Sigma, C, N, E, G, I)$ is a non-hierarchical Coloured Petri Net.
- $T_{sub} \subseteq T$ is a set of substitution transitions.
- $P_{port} \subseteq P$ is a set of port places.

- $PT : P_{port} \rightarrow IN, OUT, I/O$ is a port type function. It maps a port types into port places.

For more details and theory of CPN, the reader is referred to [Jensen and Kristensen (2009)].

2.4 Computer Tools

There are a variety of tools that can be used for CPN (c.f. [Jensen *et al.* (2006); Westergaard (2013)]). In this thesis the **CPN Tools v.4.0** from [AIS Group (2013)] has been used. The CPN Tools was initially developed by the CPN Group at Aarhus University, Denmark, and since 2010 by the AIS group, Eindhoven University of Technology, The Netherlands. The CPN Tools is composed of a graphical editor for constructing models, a state space tool for verifying properties of models, and a simulator for executing the CPN ML language.
However, even though CPN Tools provides a graphical execution of nets, it is not always convenient to observe the desired properties. Therefore, some tools were developed to improve the visual effects of a CPN model. For example, the PNV tool [Kindler and Páles (2004)] provides 3D visualization of PN-based models. Another tool, the BRITNeY Suite visualization tool [Westergaard (2006)] is compatible with CPN Tools of [AIS Group (2013)]. In [Jørgensen (2008)] the BRITNeY Suite visualization tool was used for a proposed elevator system. Nevertheless, the fourth version of CPN tools [Westergaard (2013)] allows third parties to use standard Java-based applications directly. As a consequence, a visual aid called **Visualization v.0.1** was developed to observe the proposed model of the elevator system (see an example in

Figure 2.4).

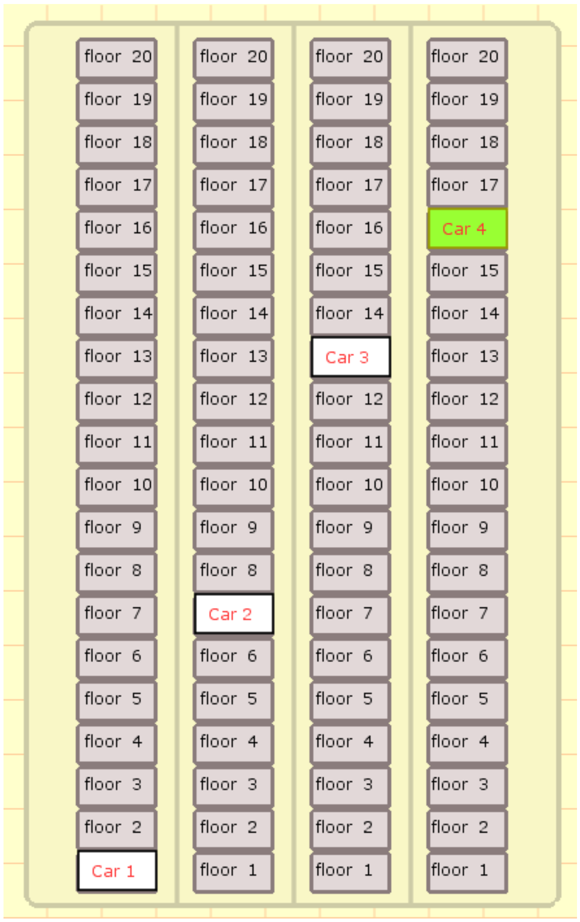


Figure 2.4: An example of the visualization extension

Chapter 3

The Abstract Version of CPN-based Model of the Elevator System

This chapter discusses the abstract version of the proposed model of the elevator system. Section 3.1 introduces the model. Section 3.2 presents the car-structure sub-model of the proposed model. Section 3.3 Studies the hall-call sub-model in detail. Section 3.4 presents the car-call sub-model. Finally, section 3.5 discusses the system-cycle sub-model.

3.1 Introduction

Due to the complexity of the elevator system and the desired flexibility of the structure, the proposed model is composed of four major interconnected but independent sub-models (see Figure 3.1).

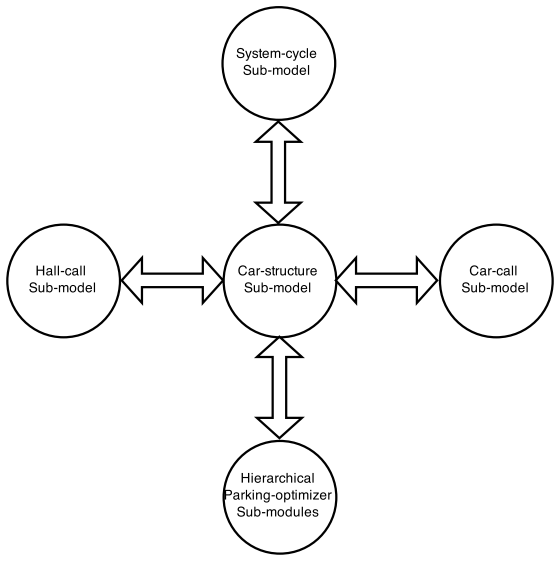
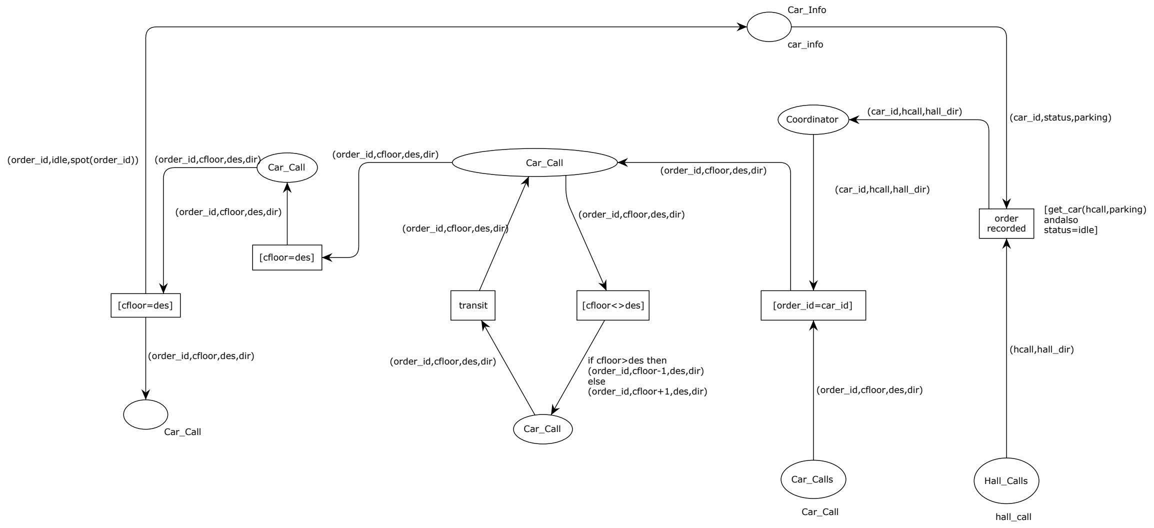


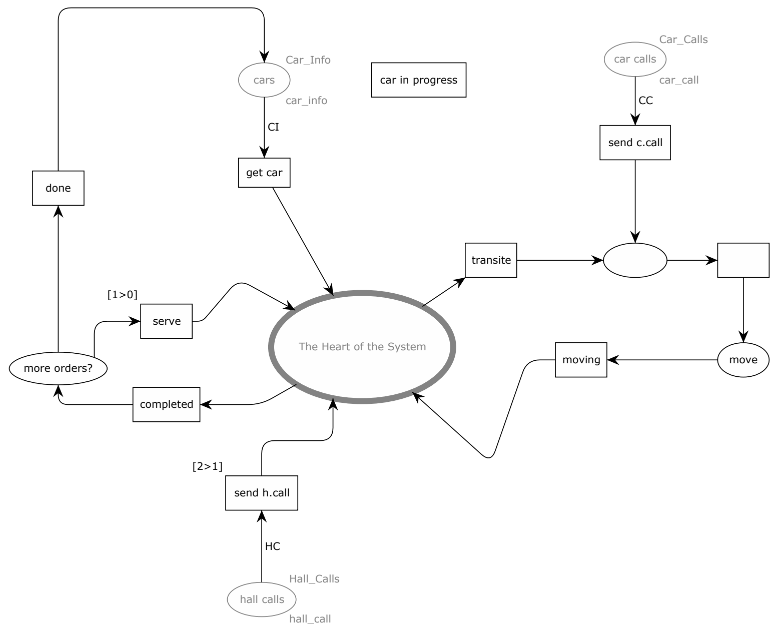
Figure 3.1: The sub-models of the elevator system

These sub-models include the *car-structure sub-model*, the *hall-call sub-model*, the *car-call sub-model*, and the *system-cycle sub-model*. The functions and connections between sub-models are described as follows: The 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 two types of controls: either a hall-call or a car-call. As the names suggest, a hall-call is placed by pressing a button located in the hallway of a given floor while a car-call is place by pressing a button inside the car of the elevator. When a hall-call is placed, by relying on algorithms the hall-call sub-model will assign the hall-call to the appropriate car of the car-structure sub-model. Similarly, the car-call sub-model coordinates the placed car-calls with the cars of the car-structure sub-model. Finally, the system-cycle sub-model operates the cars of the car-structure sub-model to service the requested calls.

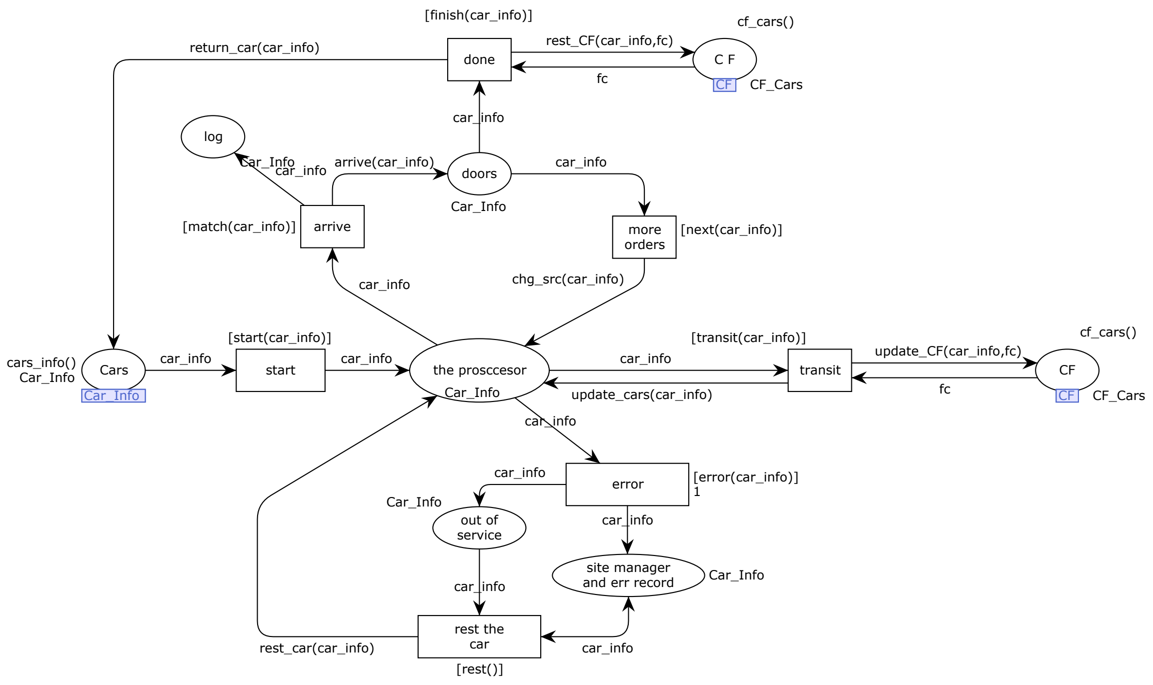
It is worth mentioning here that the design of the proposed model experienced several phases. It was started as one single model shown in Figure 3.1 (a) and developed in Figure 3.1 (b). However, the more the initial model was developed, the more its complexity increased. Therefore, the model then was divided into two sub-models, namely the hall-calls (Figure 3.1 (d)) and the elevator-system (Figure 3.1 (c)). Eventually, the proposed model was divided more into four sub-models, which were described earlier, for the purpose of covering different stages of the elevator system in substantial detail besides the flexibility of tracking errors.


Car_Info
car_info
Coordinator
(car_id,hcall,hall_dir)
(car_id,status,parking)
(order_id,idle,spot(order_id))
(order_id,cfloor,des,dir)
Car_Call
(order_id,cfloor,des,dir)
Car_Call
(order_id,cfloor,des,dir)
(car_id,hcall,hall_dir)
(order_id,cfloor,des,dir)
[cfloor=des]
(order_id,cfloor,des,dir)
(order_id,cfloor,des,dir)
order recorded
[get_car(hcall,parking) andalso status=idle]
[cfloor=des]
transit
[cfloor<>des]
[order_id=car_id]
(order_id,cfloor,des,dir)
(order_id,cfloor,des,dir)
if cfloor>des then (order_id,cfloor-1,des,dir) else (order_id,cfloor+1,des,dir)
(hcall,hall_dir)
(order_id,cfloor,des,dir)
Car_Call
Car_Call
Car_Calls
Car_Call
Hall_Calls
hall_call

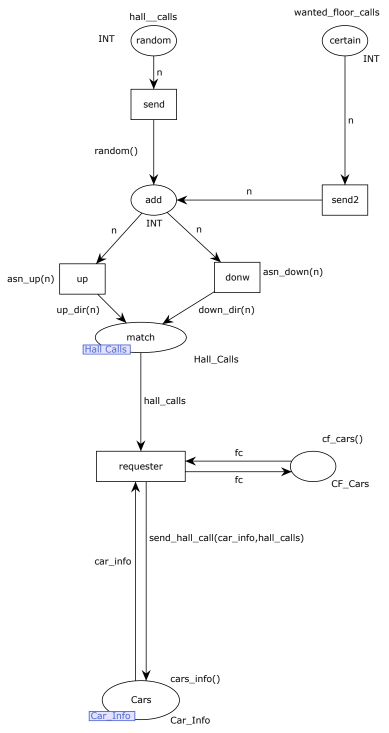
(a) The first phase



(b) The second phase



(c) The third phase (The elevator-system sub-model)



(d) The third phase

Figure 3.1: The development phases of the proposed model

3.2 The Abstract Car-Structure Sub-model

The car-structure model (Figure 3.2) consists of just two places *Cars* and *Database*, which also belong to other sub-models as they are fusion places. Moreover, the car-structure sub-model features essential parameters and dynamic initializations.

The main reason of modeling the car-structure as an independent sub-model is that almost all other sub-models contain the two places of the car-structure sub-model. Therefore, the two places were defined once in an independent sub-model that has a recognizable name.

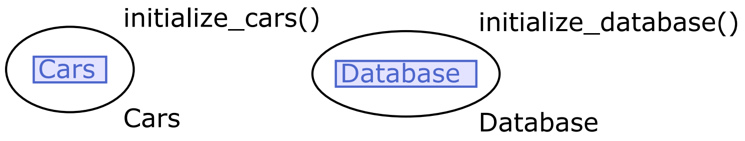


Figure 3.2: The abstract car-structure sub-model

The first place has the colour set **Cars**, which is a record colour set of the Cartesian product of the sets described in Table 3.1.

Table 3.1: The colour set **Cars**

Colour Sets	Definitions
Car ID	$\{i \mid i \in \mathbb{Z}^+ \wedge i \leq total\ number\ of\ cars\}$
Range	$\{r \mid r \in \mathbb{Z} \wedge lowest\ floor \leq r \leq highest\ floor\}$
Status	$\{$*up, down, emergency, idle, out of service*$\}$
Desired Floors	$\{[l] \mid l \in Range\}$
Call Issuer	$\{$*request, system, non, reservation*$\}$
Cars	$\{$*(car id, current floor, status, parking floor, desired floors, call issuer)* $\mid$ *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*$\}$

Consequently, each token, which represents a car, has the following components:

- *Car id*: it represents a unique identification number of the car. In other words, the *ids* are consecutive numbers from one to the total number of cars.
- *Current floor*: it indicates the car's position through its journey between floors.

- *Status*: it represents five possible statuses for the car. First, "*idle*" indicates the car is inactive but standing by for serving calls. Second, "*emergency*" indicates that either a passenger pressed the emergency button located inside the car, or an error occurs in the car's operating; accordingly, the car is suspended as "*out of service*". Finally, "*up*" and "*down*" indicate the direction of the car's journey.

- *Parking floor*: it stores the floor's number on which the car is held when it is idle. The initial value of the parking floor is calculated in general by the following equations:

$$Floors'\ Number = (highest\ floor\ - lowest\ floor\ ) + 1$$
$$Scope = |floors'\ number \div cars'\ number|$$
$$Scope\ Head = ((scope * car\ id) - (scope - 1)) + (lowest\ floor's\ - 1)$$
$$Scope\ Tail = (scope * car\ id) + (lowest\ floor\ - 1)$$
$$\therefore Spot(car\ id)\ = scope's\ head\ (car\ id)$$

  There are other possibilities for the function *Spot*, such as:

  – Assigning each car to the last floor of its scope.

  $$Spot\ (car\ id) = scope's\ tail\ (car\ id)$$

  – Assigning manually each car to a specific floor, especially when the above equations are impractical.

  – Assigning all cars to a constant value. For example:

  $$Spot\ (car\ id) = 1$$

where all cars will be held on the first floor.

Nevertheless, if the parking-optimizer model is enabled, each car will be assigned dynamically and continuously to the most requested floors (see chapter 5).

- *Desired floors*: it is a list that shows the car's destinations. This list accepts no duplication of its elements.
- *Call issuer*: it denotes the motivation of the car's journey where "*Request*" denotes the car is serving placed calls by passenger, "*System*" denotes the car is executing a system-generated call, "*Non*" denotes the car is idle, and "*Reservation*" denotes the car is being reserved and accepting only car calls.

The second place of the car-structure sub-model has the colour set **Database**, defined in Table 3.2. In principle, this is a list of all the necessary information about the states of cars. This list is used by the algorithms of the hall-call sub-model.

Table 3.2: The colour set **Database**

Colour Sets	Definitions
Car Info	$\{$*(current floor, status, destinations, car id)* $\mid$ *current floor* $\in$ *Range, status* $\in$ *Status, destinations* $\in$ *Desired Floors, car id* $\in$ *Car ID*$\}$
Database	$\{[d] \mid d \in Car\ Info\}$

Furthermore, the car-structure sub-model has *essential parameters* that are very convenient for simulation-based analysis. The parameters are defined in Table 3.3:

3.2.1 Dynamic Initialization of Places

As most places of the proposed model, both places; *Cars* and *Database* are initialized dynamically by defined functions. For instance, the function of place *Cars* is defined

Table 3.3: The essential parameters of the car-structure sub-model

Parameter	Legal value
Cars' number	$\{n \mid n \in \mathbb{Z}^+\}$
Lowest floor number	$\{f \mid f \in \mathbb{Z}^+ \wedge f < highest\ floor\}$
Highest floors number	$\{h \mid h \in \mathbb{Z}^+ \wedge lowest\ floor < h\}$

as follows: (Pseudocode)

```
function initialize cars () =
map (function f(i) ={car id=i, current floor=spot(i), status=idle,
                        parking floor=spot(i), desired floors=[],
                        call issuer=non,) l=[car ids];
```

In the above function, *"map"* is a built-in function inherited from the **Standard ML** library. Practically, *"map f l"* means for each element of the list l, applies the function f. Thus, the number of floors and cars has no effect on the models' structure (i.e. the number of places and transitions is fixed for any given number of floors and cars).

3.3 The Abstract Hall-Call Sub-model

This sub-model assigns a placed hall call to the most appropriate car based on the applied given algorithms (which are subject to changes and replacements). Furthermore, this sub-model generates hall calls from arbitrary floors and a selected floor in order to facility efficiently the examination of various rules and algorithms during the simulation-based analysis.

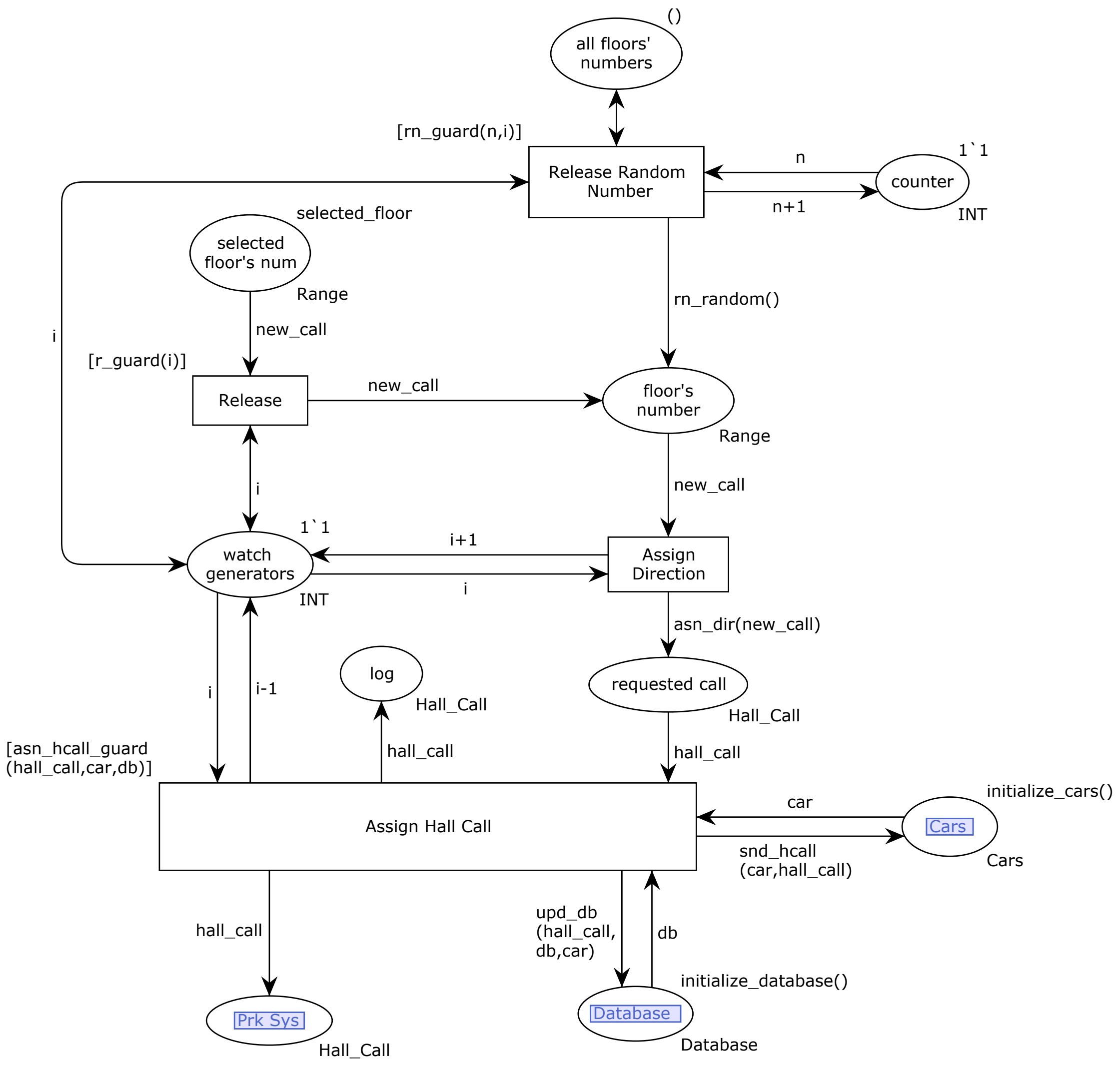


Figure 3.3: The abstract hall-call sub-model

The processing of hall calls (see Figure 3.4) is initialized from place *requested call* where each token represents a placed hall-call of the colour set *Hall Call* defined in Table 3.4. Each token has an appropriate direction and a floor number where the hall call was placed. Assigning a hall call to a car requires the firing of transition *Assign Hall Call* that is enabled if and only if its guards, which represent appropriate rules, are holding. The specific rules that must be satisfied are comprised of the following:

1. The selected car is either idle or travelling toward the direction of the hall call,
2. The selected car is not reserved,
3. The selected car is elected by the applied algorithm.

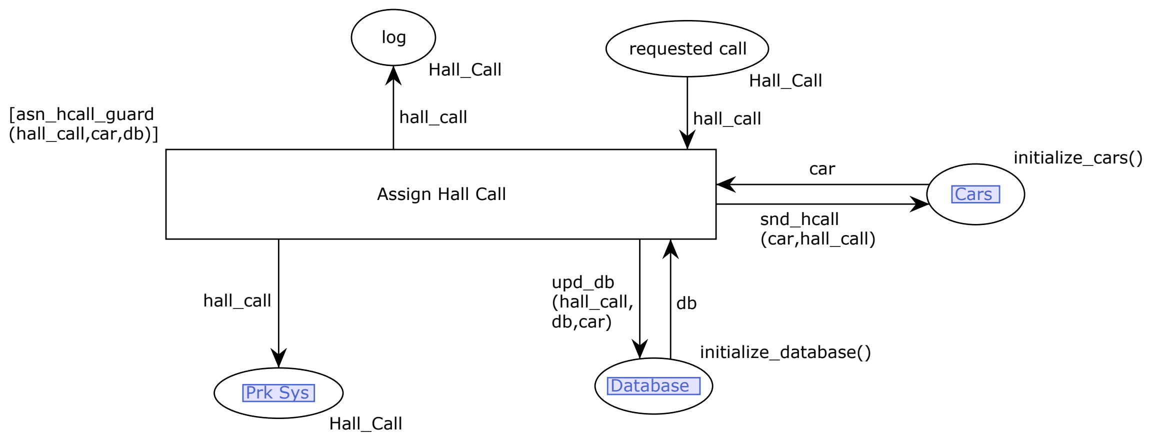


Figure 3.4: Assigning hall calls to cars

Table 3.4: The colour set **Hall Call**

Colour Set	Definition
Hall Call	$\{(\textit{hall call floor, status}) \mid \textit{hall call floor} \in \textit{Range}, \textit{status} \in \textit{Status}\}$

After firing transition *Assign Hall Call*, the token of a placed hall call is removed from place *requested call* and assigned into the desired-floors list of the selected car in place *Cars* with a guided direction, i.e. up or down, if the selected car is idle. Furthermore, this sub-model has been tested successfully with three algorithms that process the assignment of hall calls to cars, namely the *nearest-car* algorithm, the *minimum-waiting* algorithm, and the *scope* algorithm. The adoption of these algorithms and probably many other algorithms that require simultaneous access to all cars' states, can be applied by means of place *Database* (see car-structure sub-model)

The place *Database* facilities the adoption of many different algorithms that require simultaneous access to all cars' states, therefore other algorithms can easily be applied.

The *nearest-car* algorithm [Barney (2003a)] starts by analysing the token of place *Database*. First, the cars with proper status (i.e. cars that are idle or travelling toward the requested hall call) are extracted from the token (each car is represented by a single tuple, so selecting the cars is done by extracting appropriate tuples). Once the proper cars are elected, the distances between hall-call floor and cars' current floors are calculated by the absolute value of the difference between current floors and the hall call floor for each car. Accordingly, the hall call is assigned to the car of the minimum distance to the hall call floor.

The minimum-waiting algorithm improves the nearest-car algorithm by further calculation of stop time consumed by the already placed and being served calls between the floor of the new hall call and each car's current floor. Thus travel times plus stop times are calculated for each car and based on that, the car with the expected minimum waiting-time is assigned to serve the new hall call.

The scope algorithm [Siikonen and Hakonen (2003); Barney (2003b)] is often employed in express elevators and sky-lobby floors, where each car serves a specified range of floors with the allowance of transit floors. The scope algorithm is implemented as extra guards on transition *Assign Hall Call*. For instance, a guard that identifies the range floor of each car, is written as:

$$H \leq A \leq T.$$

where:

H = *the head floor of the car's scope*

A = *the answered hall-call floors*

T = *the tail floor of the car's scope*

It is worth mentioning here that optimizing the elevator systems is continuously developed and plenty of algorithms, other than the mentioned above, have been proposed. Some algorithms require mainly hardware assistance, such as using cameras, sensors, or both [Liu *et al.* (2008); Kim and Moon (2001)], while others are more concerned about energy saving [Liu *et al.* (2010)], or relying on destination registration method [Xu *et al.* (2010)]. There are also some algorithms that are genetic [Bolat and Cortes (2011)] or are based on fuzzy logic approach [Munoz *et al.* (2008)].

In addition, the abstract hall-call sub-model facilities a simulation-based analysis of different algorithms through controllably producing two classes of calls: an identified call of a specific floor that is assumed to be requested repeatedly, and arbitrary calls since the values of hall calls in real elevator systems are usually unpredictable (see Figure 3.5).

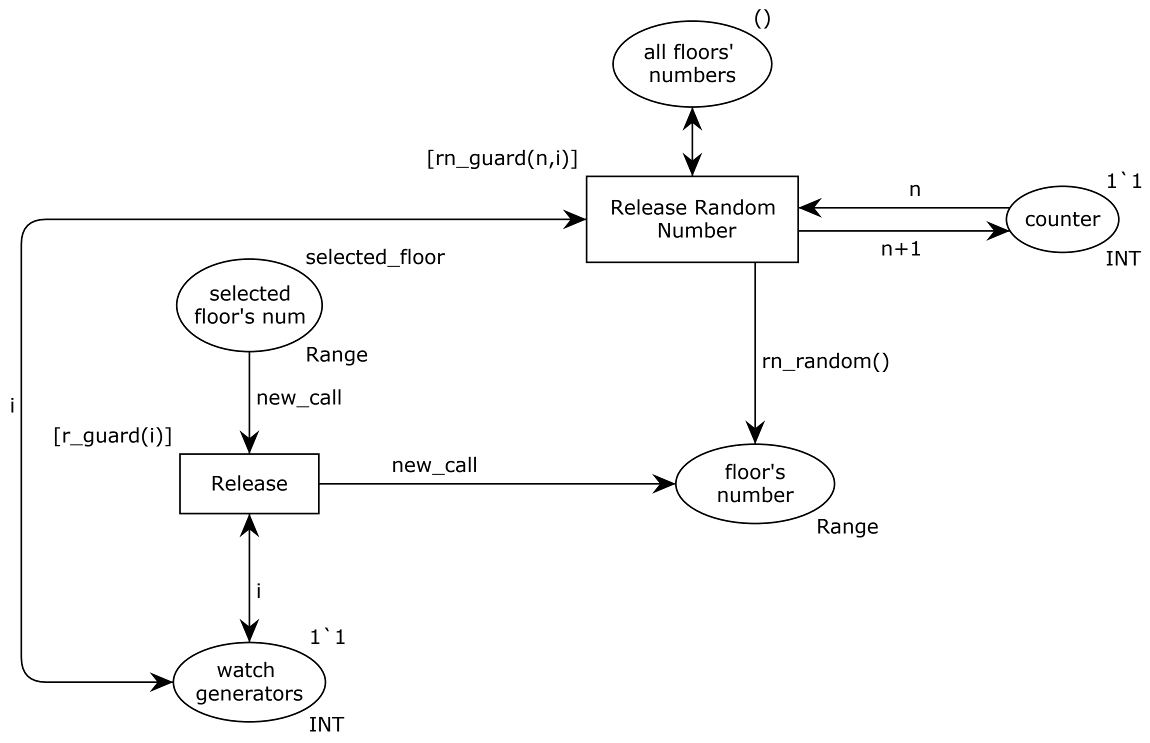


Figure 3.5: Generating arbitrary and identified floors' numbers

An arbitrary call is produced by a token of colour set *unite* defined as a single element, which is represented as "()", in place *all floors' numbers*. When transition *Release Random Number* fires, a copy of the token is converted to an arbitrary number ranged from lowest to highest floor in place *floor's number*. Conversely, an identified call is produced by transferring a token from place *selected floor's num* to place *floor's number* when transition *Release* firs. The value and quantity of tokens in place *selected floor's num* are set by parameters shown in Table 3.5.

Table 3.5: The parameters of the abstract hall-call sub-model

Parameter	Legal value
Producing mode	$\{finite, infinite\}$
Times of finite hall calls	$\{y \mid y \in \mathbb{Z} \wedge 0 < y\}$
The most requested floor	$\{r \mid r \in \mathit{Range}\}$
Duplication of most requested floor	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$
The algorithm of assigning hall calls	$\{\mathit{minimum\ waiting}, nearest, scope\}$
Production's pause number (*)	$\{p \mid p \in \mathbb{Z} \wedge 0 < p\}$

(*) The parameter *production's pause number* disables transition *Release Random Number* in order to balance between the production and the assignment of the hall calls (i.e. it determines the maximum number of tokens in place *requested call*).

In addition, a produced floor number is associated with a direction (Figure 3.6) based on the three rules.

- If the produced number equates the highest floor, then it is associated restrictedly with the down direction,
- Else if the produced number equates the lowest floor, then it is associated restrictedly with up direction.
- Otherwise, the produced is associated non-deterministically (which is modelled as a uniformly distributed random choice) to up direction or down direction.

Thus, a hall call of a valid floor's number and an appropriate direction is produced correctly and completely.

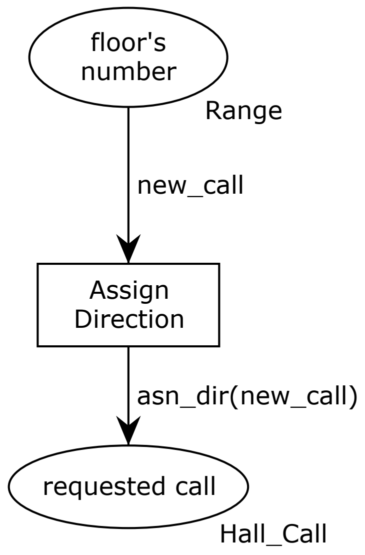


Figure 3.6: Adding directions to the produced floor numbers

3.4 The Abstract Car-call Sub-model

This sub-model (see Figure 3.7) provides the coordination between the cars and their car calls. Additionally and similarly to the abstract hall-call sub-model, this sub-model produces both arbitrary and identified car calls.

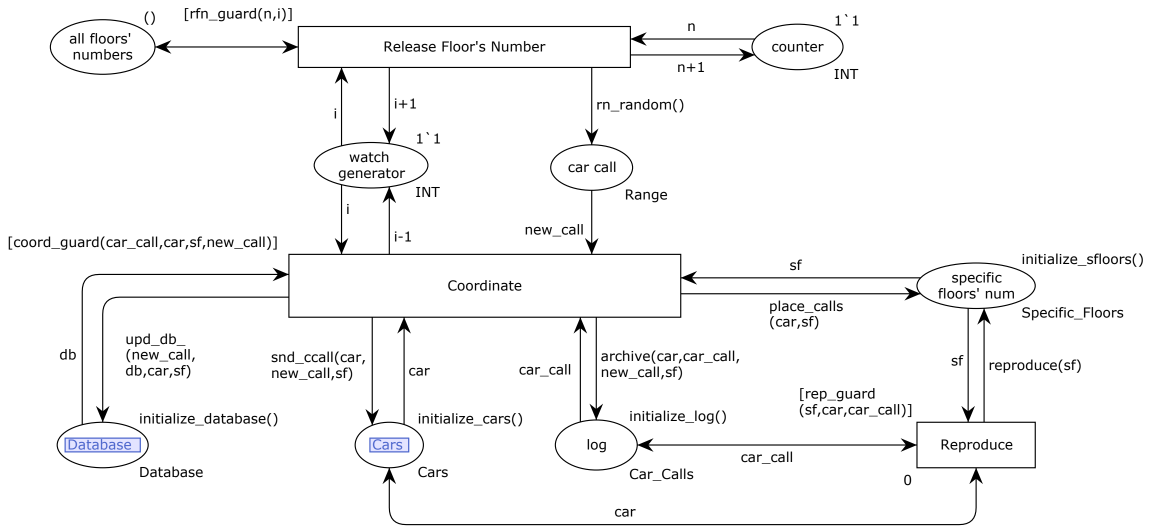


Figure 3.7: The abstract car-call sub-model

The coordination between a car and its car calls (Figure 3.8) starts from place *car call* where each token represents a placed car call of the colour set **Range**, which is a floor number ranged from the lowest to the highest floor. Placing a car call in a car demands firing transition *Coordinate* that is enabled when its guards are satisfied with respect to the producing mode's state, and the applied algorithm on the abstract hall-call sub-model (i.e. in the scope algorithm, the car serves only within the floors of its defined scope). After firing transition *Coordinate*, the placed car call is removed from place *car call* and inserted into the car's desired-floors list with an appropriate direction, if the car is idle. Concurrently, the list of the specified calls, in place *specific floors' num* of the color set **Specific Floors** defined in Table 3.6, is merged into the car's desired-floor list upon the firing of transition *Coordinate.*

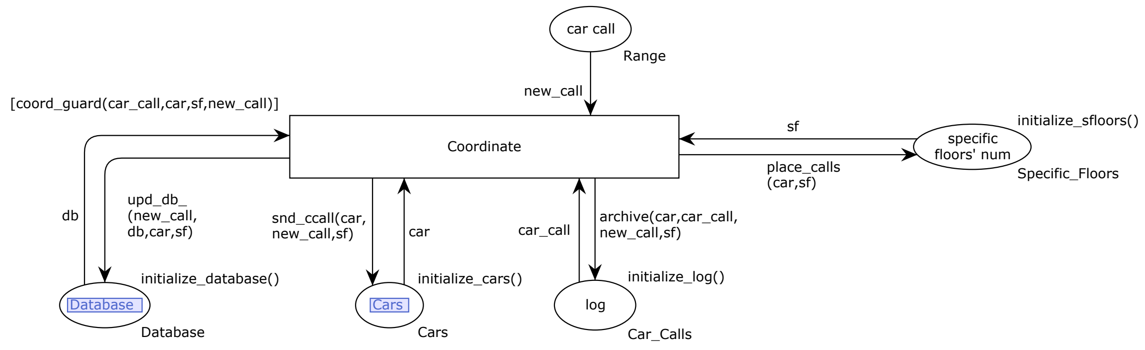


Figure 3.8: Coordinating between cars and their car calls

Table 3.6: The colour set **Specific Floors**

Colour Set	Definition
Specific Floors	$\{$*(car_id, specific calls, repeated times)* $\mid$ *car_id* $\in$ *Car ID, Specific calls* $\in$ *Desired Floors, repeated times* $\in \mathbb{Z}\}$

In addition, the abstract car-call sub-model features two mechanisms of producing car calls. First, arbitrary car calls where a single call is placed into the selected car (Figure 3.9). Second, specified calls where a list of selected floors is placed entirely into the selected car (Figure 3.10). The list of specified calls is re-produced frequently based on the parameters defined in Table 3.7.

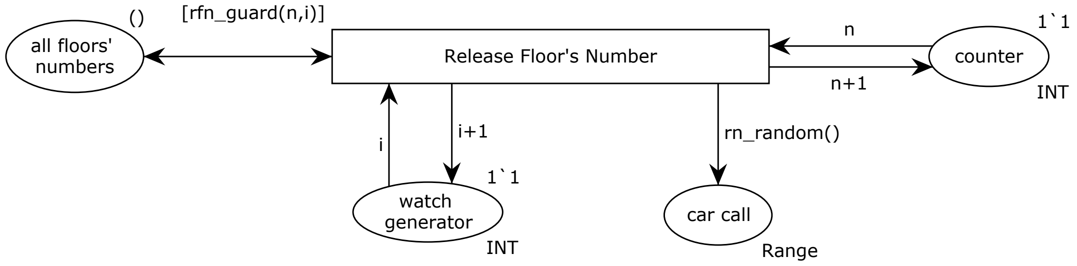


Figure 3.9: Producing arbitrary car calls

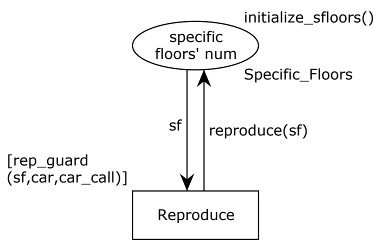


Figure 3.10: Producing specified car calls

Table 3.7: The parameters of the car-call sub-model

Parameters	Legal values
Producing mode	$\{finite, infinite\}$
Times of finite car calls	$\{x \mid x \in \mathbb{Z} \land 0 \leq x\}$
Most desired floors	$\{[f] \mid f \in \mathit{Range}\}$
Frequency of desired floors	$\{d \mid d \in \mathbb{Z} \land 0 \leq d\}$
Production's pause number	$\{p \mid p \in \mathbb{Z} \land 0 < p\}$

3.5 The Abstract System-cycle Sub-model

This sub-model deals with the system cycle of the elevator cars during the operation of the elevator system. Each elevator car experiences three separate stages; the maintenance, the arrival, and the transition (see Figure 3.11).

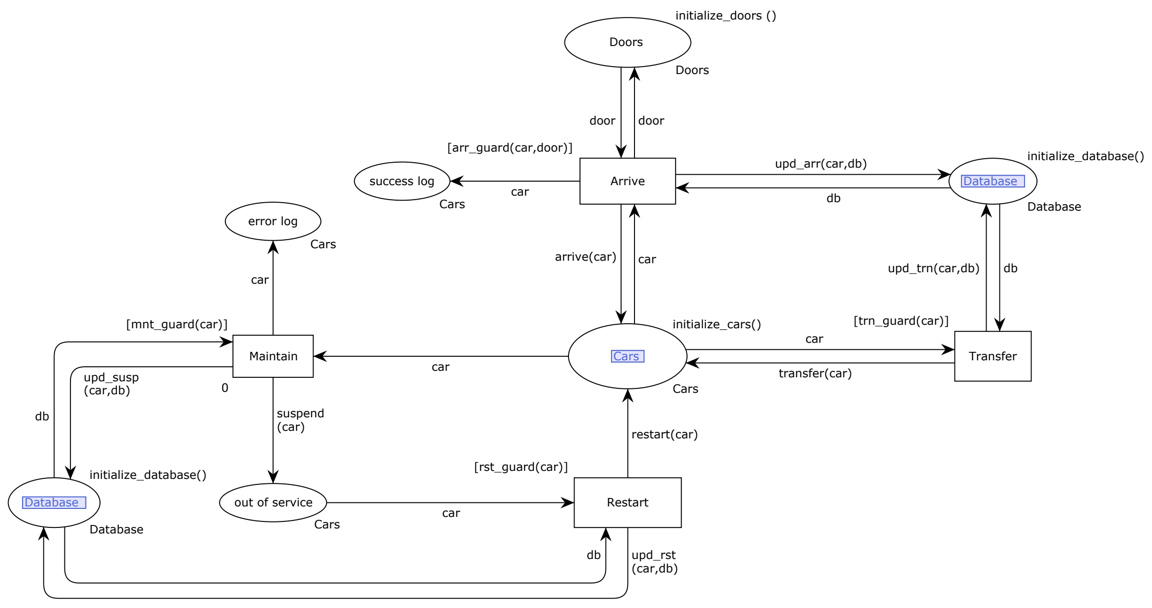


Figure 3.11: The abstract system-cycle sub-model

The maintenance stage models the suspension of a car as presented in Figure 3.12. A car is suspended by either an emergency case when the car's emergency button is pressed, or an operation failure case when an error occurs during the execution of the elevator system. Transition *Maintain* fires if and only if a car is *"out of service"* or in *"emergency"*, and it has the highest priority in the entire sub-model (i.e. when it is enabled, all other transitions are blocked). After the firing of transition *Maintain*, the car's token is transferred temporarily from place *Cars* to place *out of service*, and the token in place *Database* is updated accordingly. Hence, the car is not accessible by any other sub-models that have no access to place *out of service*. However, a pending car can be restarted either automatically or manually based on the value of parameter *restart pending cars automatically* (see Table 3.8). If the parameter is assigned to *"yes"*, then transition *Restart* is enabled immediately. Otherwise, transition *Restart* requires altering manually the status of the pending car to a different status other than *"emergency"* or *"out of service"*. In both cases, the firing of transition *Restart* results in car's token being returned to place *Cars* and place *Database* being updated, and therefore the car is accessible by other sub-models.

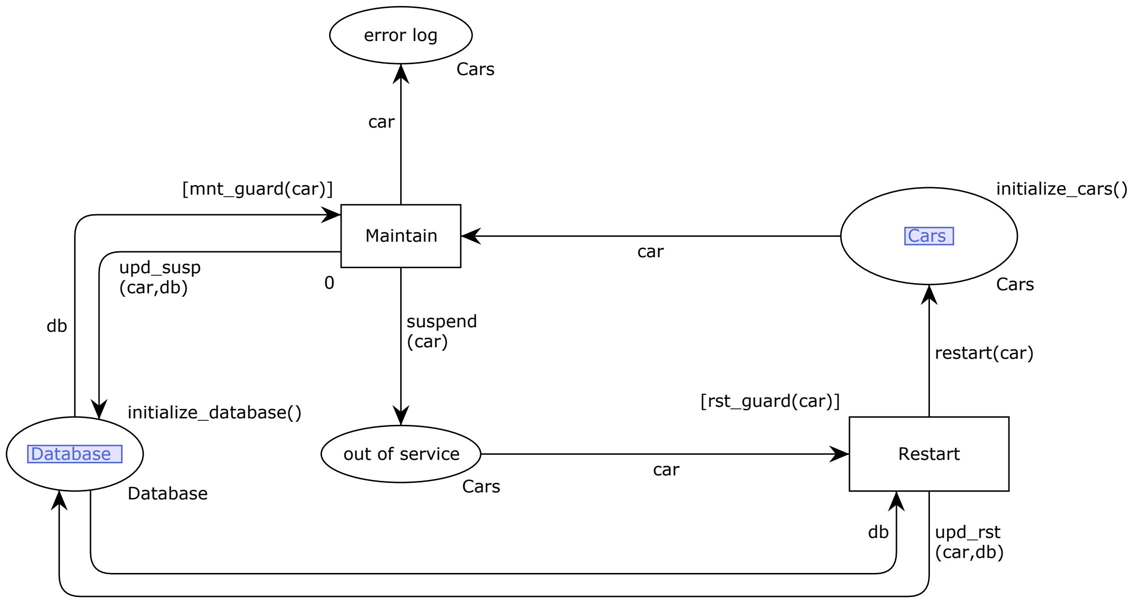


Figure 3.12: The maintenance stage

Table 3.8: The parameter of the system-cycle sub-model

Parameter	Legal value
Restart pending cars automatically	{*"yes"*,*"no"*}

The transition stage describes the process of moving elevator cars between floors as presented in Figure 3.13. Transition *Transfer* is enabled if and only if transition *Maintain* is disabled, the car's desired-floors list is not empty, and the car's current floor matches no calls of the desired-floors list. After firing the transition *Transfer*, the car's token is updated as follows. If the car's desired-floors list has calls beyond the car's current floor,then it remains shifted on the same direction. Otherwise, its direction is reversed. In both cases, the token in place *Database* is updated accordingly.

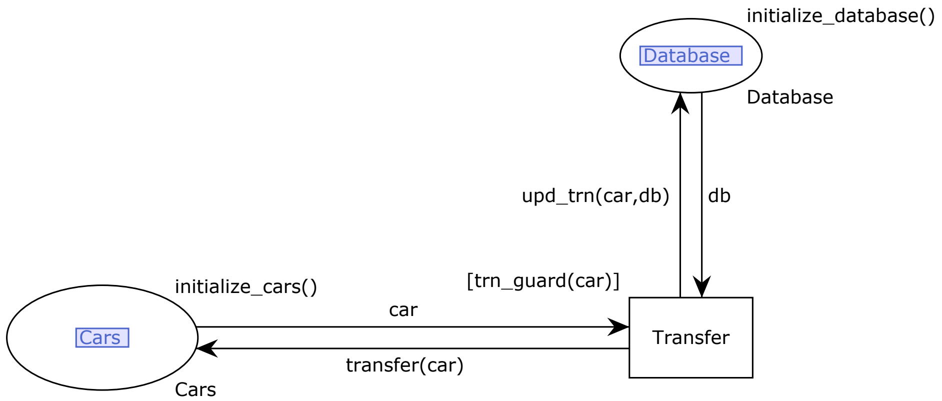


Figure 3.13: The transition stage

Once a car reached its desired destination, it is said to be in the arrival stage as presented in Figure 3.14. At this stage, transition *Arrive* is enabled if and only if transition *Maintain* is disabled, and the car's current floor matches a requested floor (i.e. an element) of the desired-floors list. After firing transition *Arrive*, the car's token is updated by dropping the requested floor from the car's desired-floors list, and then if the car's desired-floors list has more floors, then the car continues serving the requested floors. Otherwise, the car is set to idle if the car's current floor agrees with parking floor, or the car is dispatched to the parking floor with an appropriate direction. Finally, place *Doors* represents the doors' operations. Since such operations are almost trivial, they are included in one place that can be converted into an hierarchical sub-model that shows all the activates of doors.

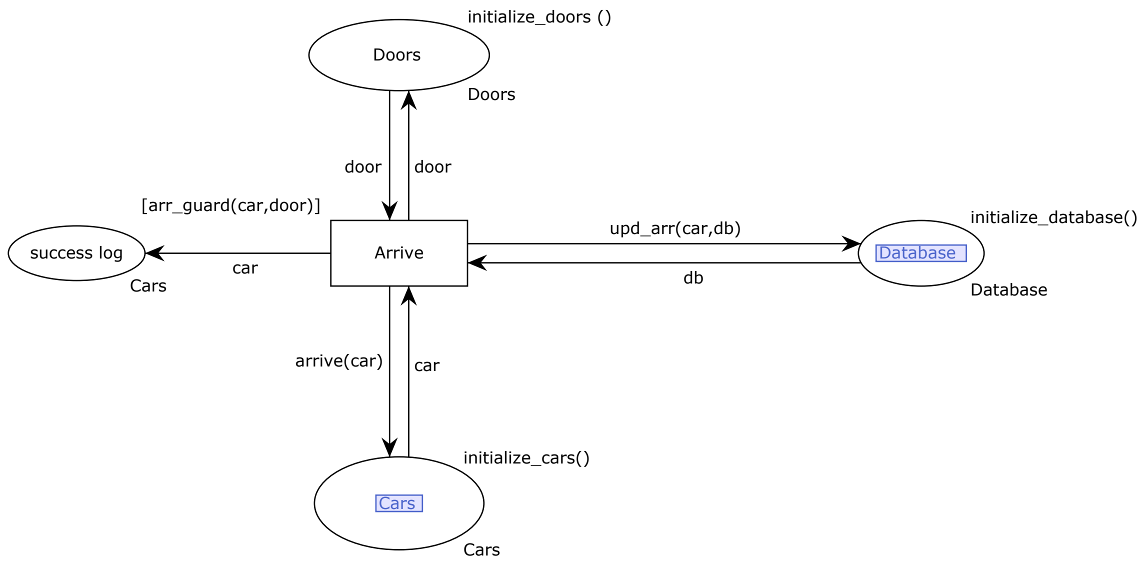


Figure 3.14: The arrival stage

Chapter 4

The Timing Version of CPN-based Model of the Elevator System

This chapter presents the timing version of the elevator system model. Section 4.1 introduces the new features of the timing version. Section 4.2 explains the timing car-structure sub-model. Section 4.3 discusses the timing hall-call sub-model. Section 4.4 presents the timing car-call sub-model. Finally, section 4.5 studies the timing system-cycle sub-model.

4.1 Introduction

Similarly to the abstract version of the elevator system, the timing version is built from four major sub-models that are interconnected but independent. However, the definitions of some colour sets are extended, and the structure of some sub-models is modified partially or completely.

The abstract version, which is discussed in previous chapter, concerns the logical movement of elevator system. In contrast, the timing version, which is presented in this chapter, extends the abstract version to include the modeling of buttons' illumination, the calculation of waiting and serving times, and the predetermining of the number of accepted calls.

Illuminating a hall button of the elevator system starts when the button is pressed as modelled by the timing hall-call sub-model. Later in the timing system-cycle, the illumination is cancelled when the assigned car visits the floor where the button was pressed. Additionally and similarly, a car button is illuminated when it is pressed as modelled by the timing car-call sub-model, and the button is unilluminated when the car, which carries the button, arrives at the corresponding floor of the button.

The waiting time is the period time starts when a hall call is placed and ends when the assigned car arrives at the floor of the placed hall call. On the other hand, the serving time starts with the placing of a car call and finishes when the car arrives at the requested floor of the placed car call.

Finally, the predetermining the number of accepted calls leads to the controlling of traffic congestion during the simulation-based analysis as explained in the timing car-structure sub-model (next section).

4.2 The Timing Car-Structure Sub-model

The timing car-structure sub-model (Figure 4.1) is structurally equivalent to the abstract car-structure sub-model where two places namely Timing Cars and Timing Database are composed this sub-model. However, the colour sets of this sub-model are extended to include the time registration into the database of the elevator system as well as for each car independently.

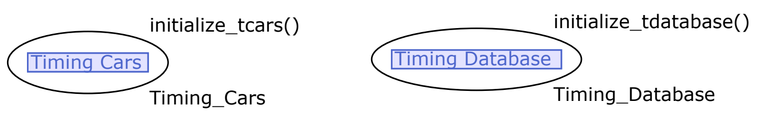


Figure 4.1: The timing car-structure sub-model

Table 4.1: 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	$\{[l] \mid l \in Range\}$
Call Issuer	$\{request,\ system,\ non,\ reservation\}$
INT	$\{n \mid n \in \mathbb{Z}\}$
REAL	$\{r \mid r \in \mathbb{R}\}$
Timing Hall Call	$\{(hall\ call, direction, time) \mid hall\ call \in Range,\ direction \in Status,\ time \in \mathbb{R}\}$
Timing Hall Calls	$\{[h] \mid h \in TimingHallCall\}$
Timing Cars	$\{(car\ id,\ current\ floor,\ status,\ parking\ floor,\ desired\ floors,\ call\ issuer,\ stops\ limitation,\ time\ period,\ served\ hall\ calls) \mid car\ id \in Car\ ID,\ current\ floor \in Range,\ status \in Status,\ parking\ floor \in Range,\ desired\ floors \in Desired\ Floors,\ call\ issuer \in Call\ Issuer,\ stops\ limitation \in INT,\ time\ period \in REAL,\ served\ hall\ calls \in Timing\ Hall\ Calls\}$

The colour set *Timing Cars* is a record colour set of the Cartesian product of the predefined sets in Table 4.1.

As a result, each token is extended with following components:

- *Stops' limitation*: it is intended to predetermine the maximum number of served calls, during the simulation-based analysis in order to control the traffic congestion. Two types of values are allowed: a constant positive integer where small numbers define light traffic and vice versa, or a dynamic value where a function is used to return to each car an arbitrary positive number that is continuously changed after its assigned calls are served. For example, if the function returns two, then after accomplishing two calls, a new value for the stop limitation is generated.
- *Served hall calls*: it is a list that used to temporarily stores and transfers the tokens of assigned hall calls from the timing hall-call sub-model to the timing system-cycle sub-model as one of several steps to calculate the waiting time. The full steps are discussed in the timing hall-call sub-model.
- *Time period*: it is a real number shows the operational time of the car.

The other components have been discussed in the abstract car-structure sub-model in section 3.2.

The colour set **Timing Database** is the list defined in Table 4.2. Hence, not only each car has time registered in the database, but also the times of all cars are accessible through a single token.

Table 4.2: The colour sets **Timing Database**

Colour Sets	Definitions
Timing Car's Data	{*(current floor, status, destinations, car id, time)* | *current floor* ∈ *Range, status* ∈ *Status, destinations* ∈ *Desired Floors, car id* ∈ *Car ID, time* ∈ *REAL*}
Timing Database	$\{[g] \mid g \in TimingCar'sData\}$

Finally, the timing car-structural sub-model has one extra parameter defined in Table 4.3, besides the parameters defined in Table 3.3.

Table 4.3: The extra parameter of the timing car-structure sub-model

Parameter	Legal value
Stops' limitation	$\{s \mid s \in \mathbb{Z}^+ \land s = 0 \lor 1 \leq s\}$

The value of parameter *stops' limitation* is processed by the following function (Pseudocode):

```
function stops' limitation value () =
        if the parameter "stops' limitation" = 0 then
                return random(1, floors' number);
        else
                return the parameter "stops' limitation";
```

Accordingly, an arbitrary number from one to the floors' number is assigned indecently to each car when the value of parameter *stops' limitation* is zero. Otherwise, the value of *stops' limitation* is returned to all cars as a constant value.

4.3 The Timing Hall-call Sub-model

This sub-model is developed from the abstract hall-call sub-model to include the button illumination and waiting time calculation (Figure 4.2).

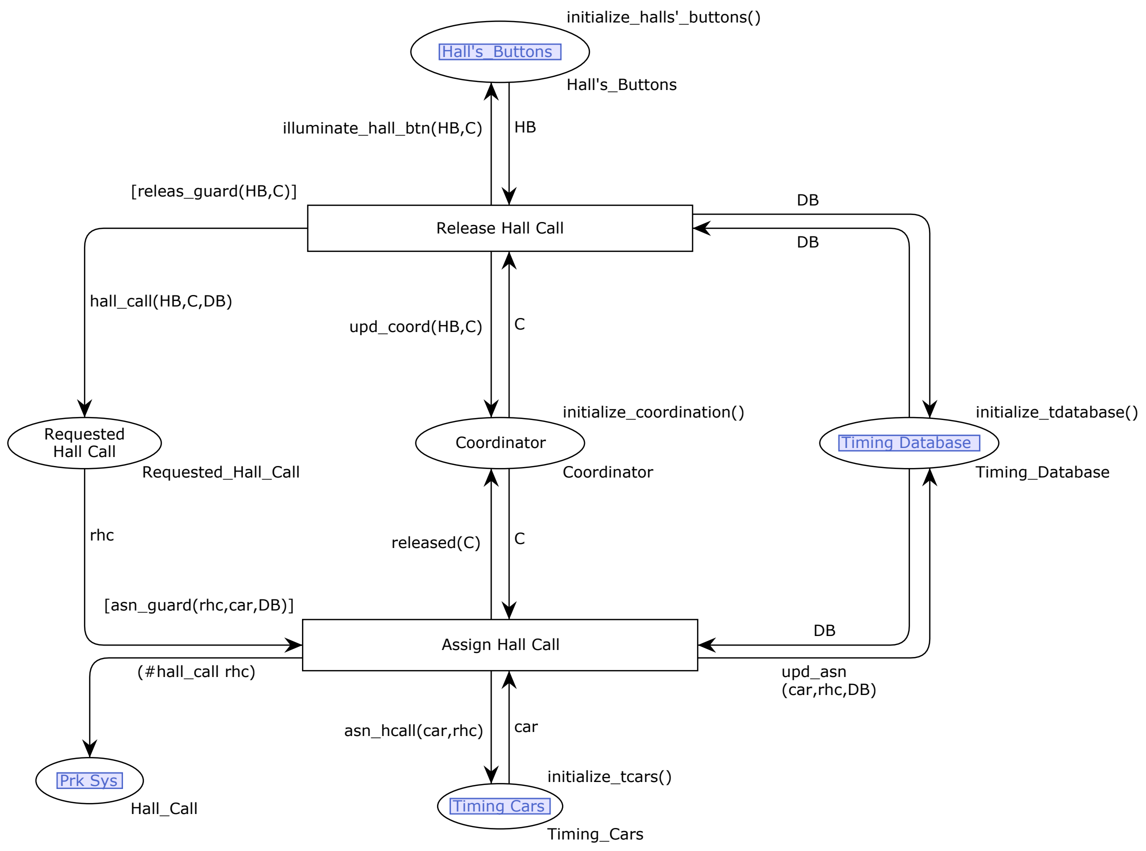


Figure 4.2: The timing hall-call sub-model

The button illumination is associated to the method of producing hall calls that starts from place *Hall's Buttons*, which contains a single token of colours set **Hall's buttons** defined in Table 4.4. In principle, It is a list comprises of three internal lists of hall calls namely illuminated buttons (*IB*), unilluminated-specified buttons (*USB*), and unilluminated-unspecified buttons (*UUB*). All possible hall calls are distributed between these three lists and represented as tuples of colour set *Hall Call.* Producing a hall call requires the firing of transition *Release Hall Call* that is enabled if and only if the following guards are satisfied:

1. The USB list and UUB list are not both empty,
2. If the limit of producing calls is finite, then it has not been already reached,
3. The number of produced calls is less than the value of parameter *pause number* (see Table 4.5) in order to balance between the producing process and the assignment process.

Table 4.4: The colour set **Hall's Buttons**

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

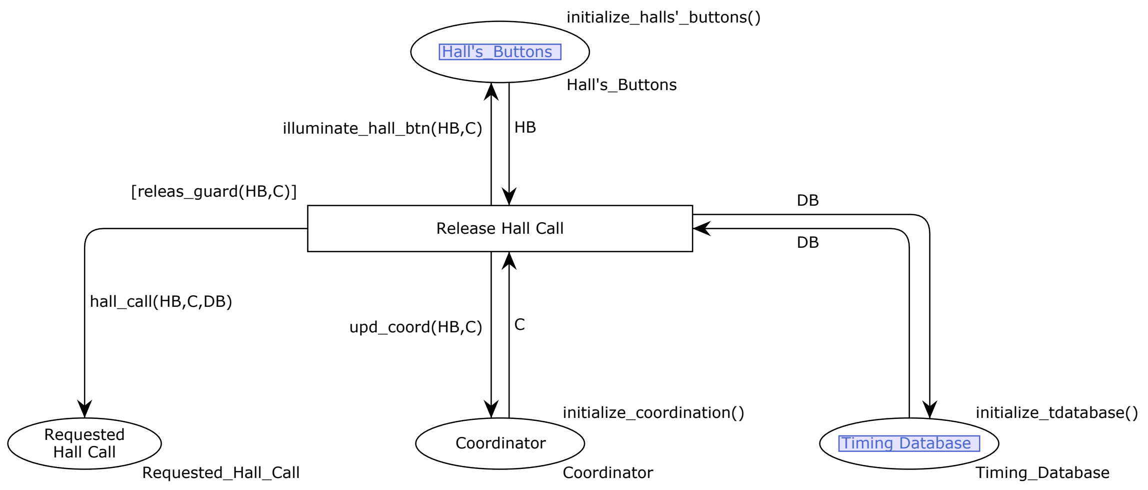


Figure 4.3: Releasing hall calls

Table 4.5: The parameters of the timing hall-call sub-model

Parameters	Legal values
Producing mode	$\{finite, infinite\}$
Times of finite hall calls	$\{y \mid y \in \mathbb{Z} \wedge 0 \leq y\}$
The most requested floors	$\{[hall\ call] \mid hall\ call \in Hall\ Call\}$
Frequency of most requested floors	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$
The algorithm of assigning hall calls	$\{minimum\ waiting, nearest,\ scope\}$
The travel time	$\{t \mid t \in \mathbb{R}^+\}$
The average time of stops	$\{s \mid s \in \mathbb{R}^+\}$
Production's pause number	$\{p \mid p \in \mathbb{Z}^+\}$

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

1. When a list is empty, the other list is always selected.
2. The difference between both lists' length is less or equal to the value of parameter *frequency of most requested floors*.
3. The internal choice between tuples is sequential in USB list and arbitrary in UUB list.

Finally, the waiting time is calculated through three steps. First, when a hall call is released from place *Hall's Buttons* and put in place *Requested Hall Call*, the placed hall call is attached with the current times of all cars (as a token of colour set **Requested Hall Call** defined in Table 4.6). Second, when the placed hall call is assigned to a car, it is removed from place *Requested Hall Call* and inserted into the two lists of the assigned car; the desired-floors list as a floor number of the colour set **Range**, and the served-hall-call list as a tuple of the colour set **Timing Hall Call** defined in Table 4.1. In the served-hall call list, only the time of the assigned car, when the hall call was released, is remained attached to the tuple of the colour set **Timing Hall Call**. Eventually, when the assigned car arrives at the floor or the placed hall call, then the waiting time is calculated as the absolute value of the difference between the current time of the car's arrival and the registered time when the hall call was released (this step is executed by the timing system-cycle sub-model).

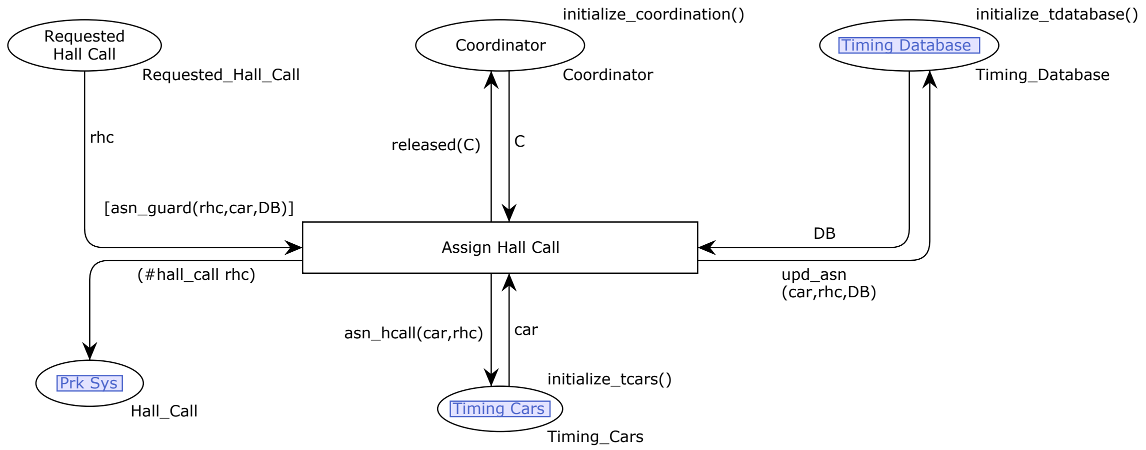


Figure 4.4: Assigning timing hall calls

Table 4.6: The colour sets **Requested Hall Call** and **Coordinator**

Colour Sets	Definitions
Car's Time	$\{$*(car id,time)* $\mid$ *car id* $\in$ *Car ID, time* $\in \mathbb{R}\}$
Cars' Times	$\{[c] \mid c \in Car's\ Time\}$
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* $\in \mathbb{Z}\}$

4.4 The Timing Car-call Sub-model

Similar to the timing hall-call sub-model, this sub-model (Figure 4.5) is developed from the abstract car-call sub-model to cover the modeling of buttons' illumination and the calculation of the serving times.

Place *Car's Buttons* has tokens equate numerically to the cars' number. Each token is formed from of the colour set **Car's Buttons**, which is a list that contains three internal lists, namely the illuminated car buttons (ICB) of the colour set **Served Calls**, the unilluminated-specified car buttons (USCB), and the unilluminated-unspecified

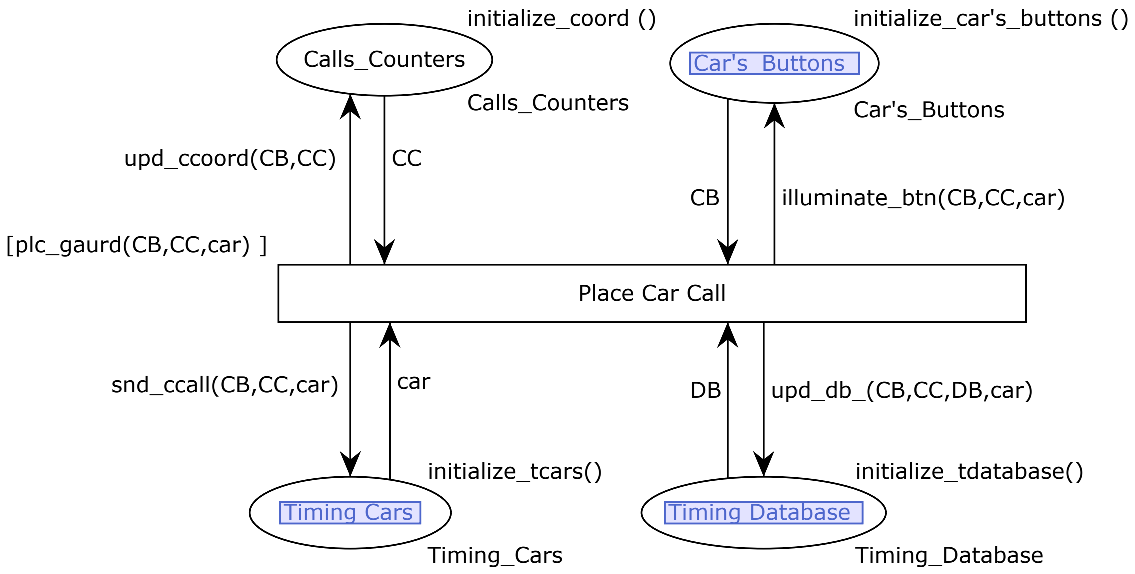


Figure 4.5: The timing car-call sub-model

car buttons (UUCB) both are of the colour set **Floors** defined in Table 4.7. Consequently, the illumination of the car's buttons and also the calculation of the serving times start from place *Car's Buttons* that requires the firing of transition *Place Car Call*. Transition *Place Car Call* is enabled if and only if the following guards are hold:

1. The UUCB list and the USCB list are not both empty,
2. The selected car has not already reached its maximum number of accepted calls,
3. If the limit of producing calls is finite, then it has not been already reached,
4. The applied algorithm on the timing hall-call sub-model is applicable on the car call and the selected car.

After firing transition *Place Car Call*, a floor number from either the UUCB list or the USCB list is transferred into two lists. The two lists are the desired-floors list of a car that is often selected non-deterministically, and the ICB list as a tuple of colour set **Served Calls** (i.e. the floor number is attached with the current time of the selected car). Thus, the button is illuminated and the current time of the selected car is registered as it is used later in timing system-cycle sub-model to calculate the serving time of the placed car call.

Table 4.7: The colour sets **Car's Buttons** and **Calls' Counters**

Colour Sets	Definitions
Floors	$\{[f] \mid f \in Range\}$
Served Calls	$\{[(floor, time)] \mid floor \in Range, time \in \mathbb{R}\}$
Cars Buttons	$\{$*(car id,ICB,USCB,UUCB)* $\mid$ *car id* $\in$ *Car ID, ICB* $\in$ *Served calls, USCB* $\in$ *Floors, UUCB* $\in$ *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 4.8: The parameters of the timing car-call sub-model

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	$\{[f] \mid f \in Range\}$
The frequency of most desired floors	$\{d \mid d \in \mathbb{Z} \wedge 0 \leq d\}$

4.5 The Timing System-cycle Sub-model

This sub-model (see Figure 4.6) is developed from the abstract system-cycle sub-model to extend the arrival stage and the maintenance stage with extra functions besides their original functions discussed in the abstract system-cycle sub-model.

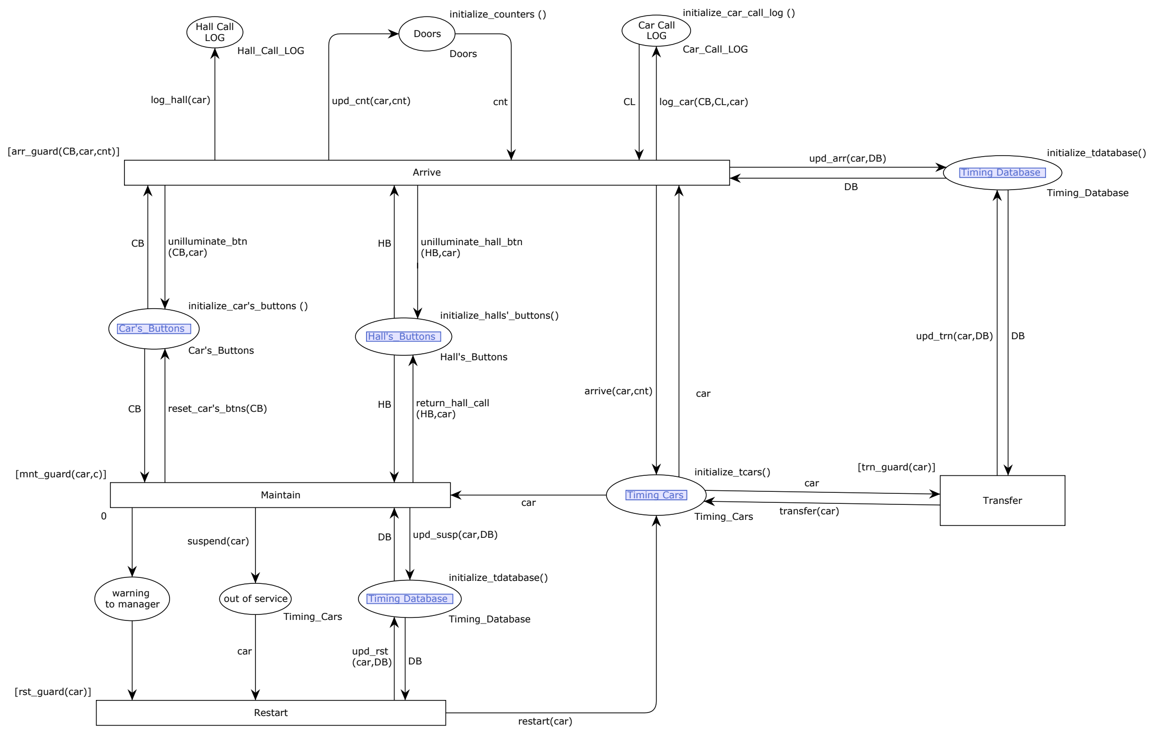


Figure 4.6: The timing system-cycle sub-model

In the arrival stage, when transition *Arrival* fires, 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 (see the timing hall-call sub-model). The calculated result is stored in place *Hall Call LOG* of the colour set **Hall Call LOG** defined in Table 4.9. 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* of the colour set **Car Call LOG** defined in Table 4.9. 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 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. Finally, the colour set **Doors** is re-defined (see Table 4.9) to include counters that track the dynamic values of parameter *stops limitations*.

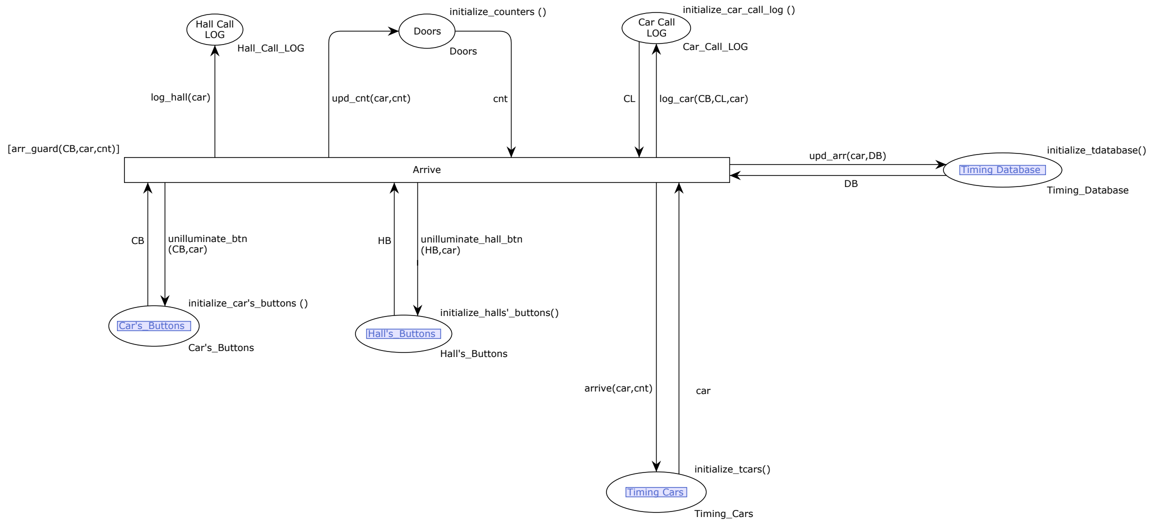


Figure 4.7: The arrival stage of the timing system-cycle sub-model

Table 4.9: The colour sets of the arrival stage

Colour Sets	Definitions
Doors	$\{$*(car id, current delivery number, deliveries' total)* $\mid$ *car id* $\in$ *Car ID, current delivery number* $\in \mathbb{Z}$, *deliveries' total* $\in \mathbb{Z}\}$
Hall Call LOG	$\{$*(hall call, waiting time)* $\mid$ *hall call* $\in$ *Hall Call, waiting time* $\in \mathbb{R}\}$
Serving times	$\{$*[(floor, serving time)]* min *floor* $\in$ *Range, serving time* $\in \mathbb{R}\}$
Car Call LOG	$\{$*(car id, serving times)* $\mid$ *car id* $\in$ *Car ID, serving times* $\in$ *Serving Times*$\}$

In the maintenance stage (Figure 4.8), when transition *Maintain* fires, not only a car is suspended, but also all assigned hall calls and car calls of the pending car are returned to places *Hall's Buttons* and *Car's Buttons*, respectively. In addition, a warning message is sent to the site manager, which is denoted by place *warning to manager*.

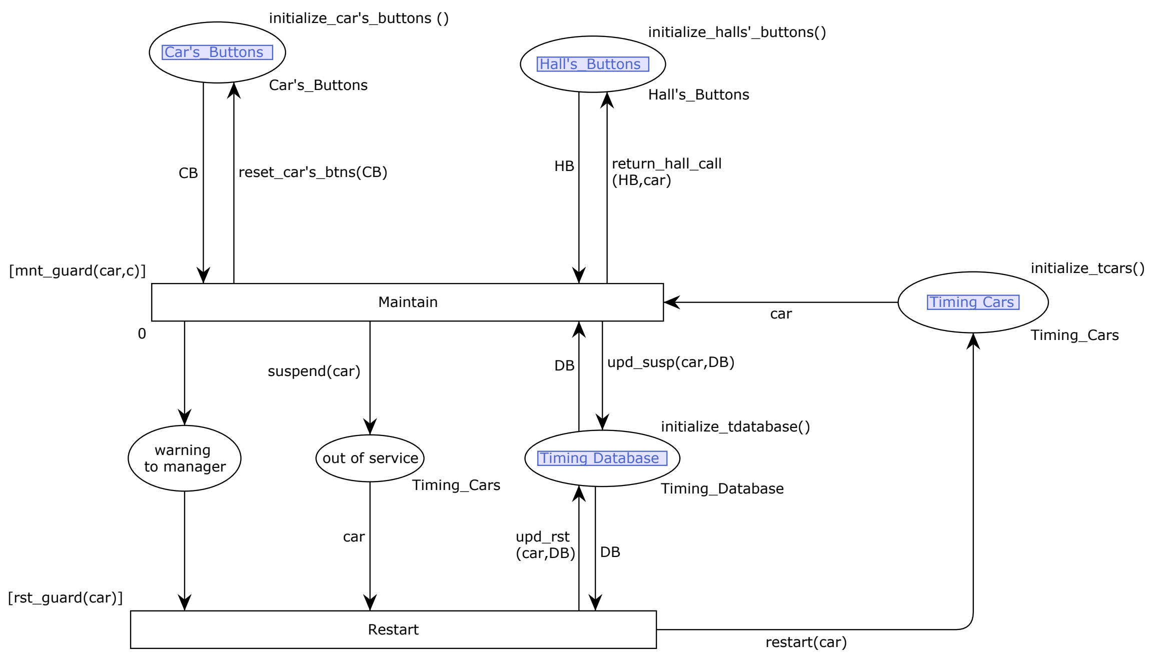


Figure 4.8: The maintenance stage of the timing system-cycle sub-model

Chapter 5

The Parking Optimizer Model

This chapter presents the parking optimizer model. Section 5.1 introduces the model. Section 5.2 studies the election sub-model. Section 5.3 and section 5.4 discuss the assignment technique and the position technique, respectively.

5.1 Introduction

Holding idle cars on selected floors, where hall calls are mostly placed, improves substantially the passenger's satisfaction and the system's energy and efficiency [Brand and Nikovski (2004); Zheng *et al.* (2013)]. Therefore, the cars are initially distributed in fair distances between the floors, and subsequently the hierarchical parking optimizer model continuously analyses the placed hall calls and then, based on different techniques, assigns the elected floors to the cars. The parking optimizer model (Figure 5.1) is composed of three sub-models, namely the election sub-model, the assignment sub-model, and the position sub-model. The parameters that facility the control of the parking optimizer model are defined in Table 5.1.

Table 5.1: The parameters of the parking optimizer model

Parameters	Legal values
Parking system state	$\{"enable", "disable"\}$
Analysing hall calls	$\{ x \mid x \in \mathbb{Z}^+ \}$
Applied parking technique	$\{"new\ assignment", "new\ position"\}$

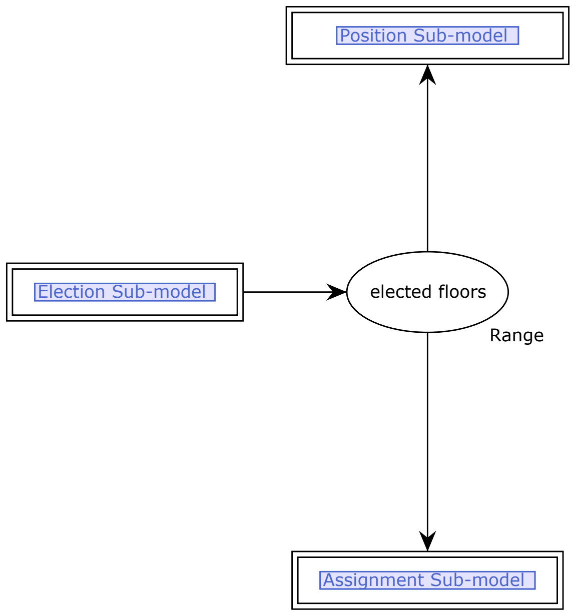


Figure 5.1: The parking optimizer model

Finally, the processes of all sub-models are restrictedly sequential. Therefore, all transitions have priorities (c.f. [van der Aalst *et al.* (2013)]) that are represented by numbers appear in the down-left corner of each transition.

5.2 The Election Sub-model

The election sub-model (Figure 5.2) counts the reputation of all placed hall calls, and then nominates the floors where most of hall calls were repeatedly placed.

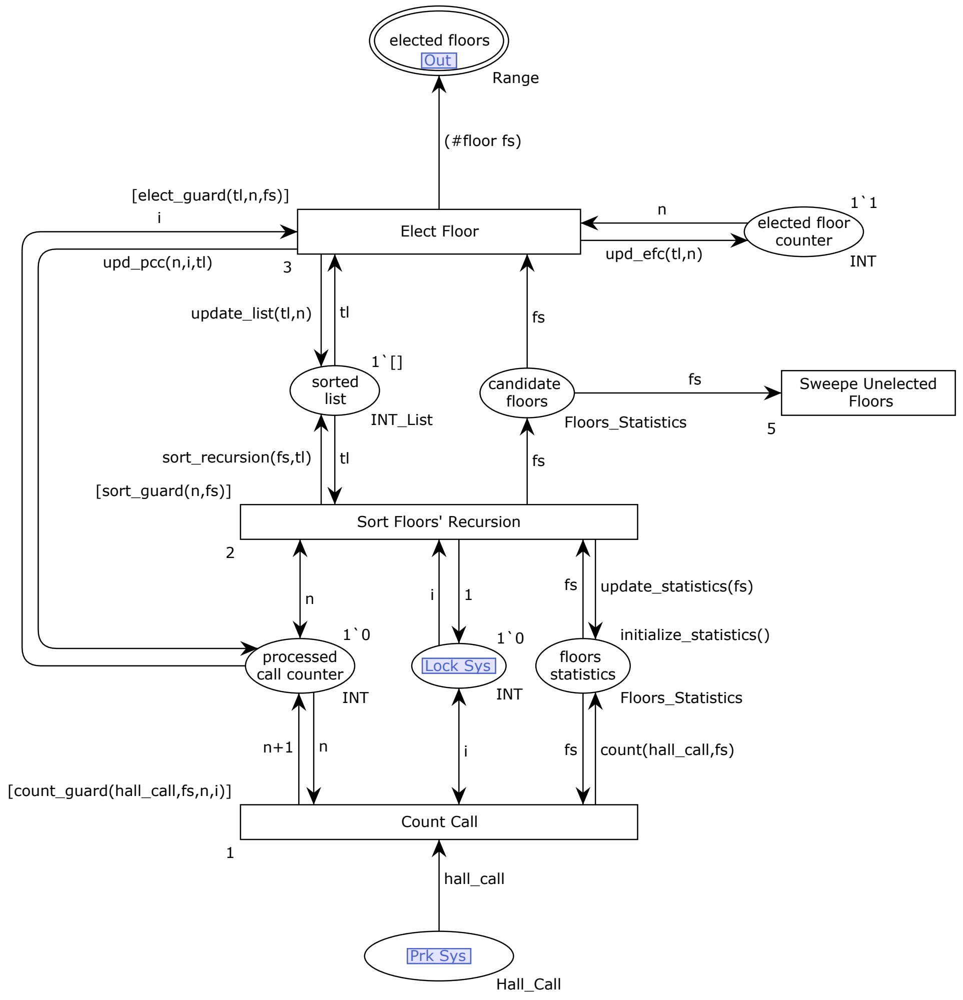


Figure 5.2: The election sub-model

The first step (Figure 5.3) starts from place *Parking System (Prk Sys)*, which transfers a copy of each placed hall call from the hall-call sub-model. The counting of hall calls' reputation demands the firing of transition *Count Call* that is enabled if and only if its guards are satisfied with respect to the parking system state, the number of call that are being counted, and the token's value of place *Lock of the system (Lock Sys)*.

Place *Lock Sys* is functionally similar to an inhibitor arc. If there is a token in a place, an inhibitor arc disables a transition (see [Janicki and Koutny (1995)]). Similarly, if place *Lock Sys* has the value zero "0", then transition *Count Call* is disabled and this locks the system from analysing more hall calls. The important of place *Lock Sys* appears when a car is in the maintenance stage of the system-cycle sub-model and not accessible by the parking optimizer models (i.e. the assignment of all elected floors can not be accomplished successfully).

However, after firing transition *Count Call*, a token is removed from place *Prk Sys*. Accordingly, the token's value of place *processed call counter* is increased, and also the reputation times of the corresponding floor in place *floors statistics* is increased. The colour set **floors statistics** defined in Table 5.2.

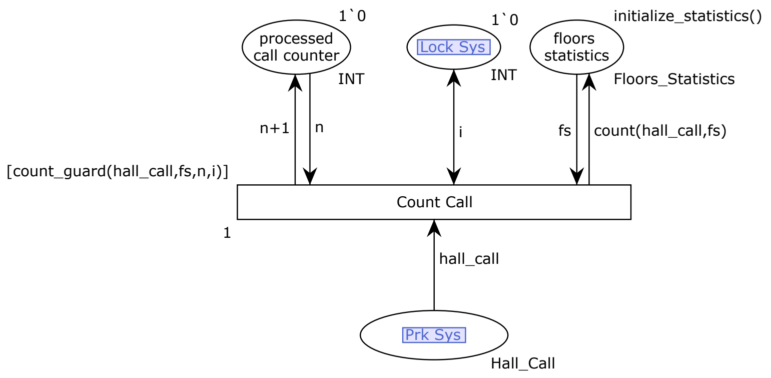


Figure 5.3: Counting the placed hall calls

Table 5.2: The colour sets of the election sub-model

Colour Sets	Definitions
Floors statistics	$\{$*(floor,times)* $\mid$ *floor* $\in$ *Range, reputation* $\in \mathbb{Z}\}$
INT List	$\{ [x] \mid x \in \mathbb{Z} \}$

the second step (Figure 5.4) begins when the number of the counted hall calls in place *processed call counter* reaches the requested number of parameter *analysing hall calls* defined in Table 5.1. As a result, transition *Count Call* is disabled and conversely transition *Sort Floors' Recursion* is enabled. The firing of transition *Sort Floors' Recursion* sorts orderly the reputation times of the placed hall calls in place *sorted list*, and also copies each processed token from place *floors statistics* to place *candidate floors*.

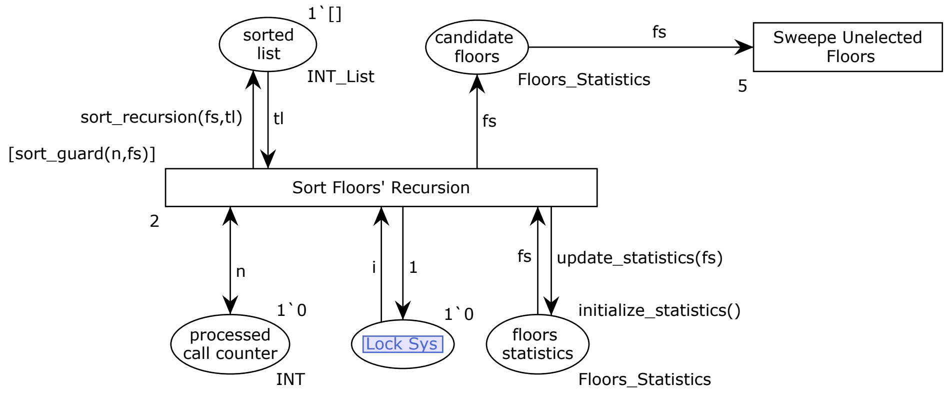


Figure 5.4: Sorting the floors of the placed hall calls

The final step (Figure 5.5) starts when all reputations of all hall calls are sorted, and thereby transition *Sort Floors' Recursion* is disabled (i.e. its guard is dissatisfies when all floors in place *floors statistics* have reputation times of zero). In contact, transition *Elect Floor* is enabled with respect to the number of the candidate floors versus the number of cars.

Therefore, there are three probabilities for the number of the candidate floors. If the candidate floors are numerically less than or equal to the cars, then transition *Elect Floor* fires all candidate floors to place *elected floors*. Otherwise, if the candidate floors are numerically greater than the cars, then transition *Elect Floor* fires a number of candidate floors equates the number of cars if the selected parking technique is *"new assignment"*. On the other hand, if the selected parking technique is *"new position"*, then transition *Elect Floor* fires the candidate floor till the number of the unique reputation times equates the number of the cars (i.e. identical reputation times are counted once) see the position sub-model.

In addition, after transition *Elect Floor* completes the nomination of the floors, it rests some input places to their initial values, and transition *Sweep Unelected Floors* removes by firing the remain tokens in place *candidate floors*. Thus, the election sub-model is ready for a new process.

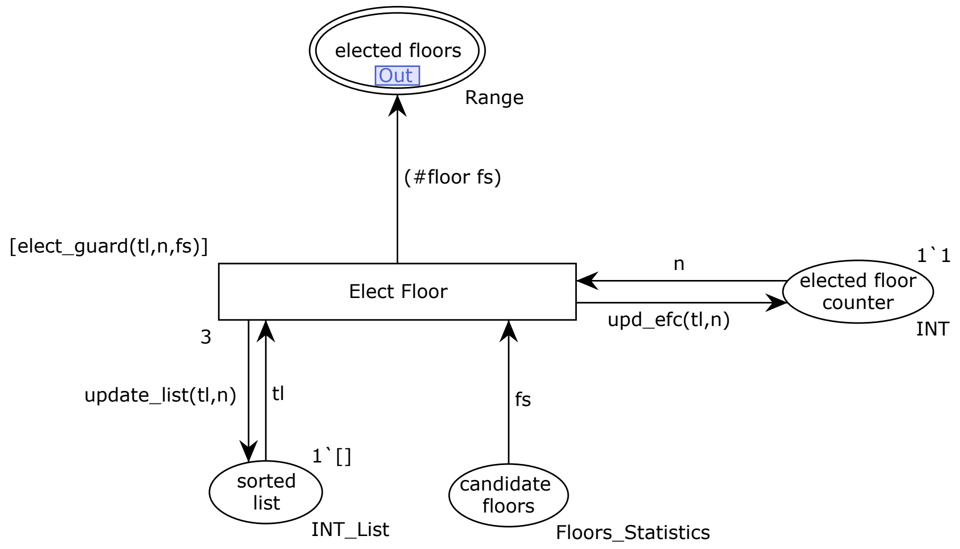


Figure 5.5: Electing the most requested floors

5.3 The Assignment Sub-model

The assignment sub-model (Figure 5.6) performs the technique of assigning the elected floors to the available cars regardless of the cars' scopes.

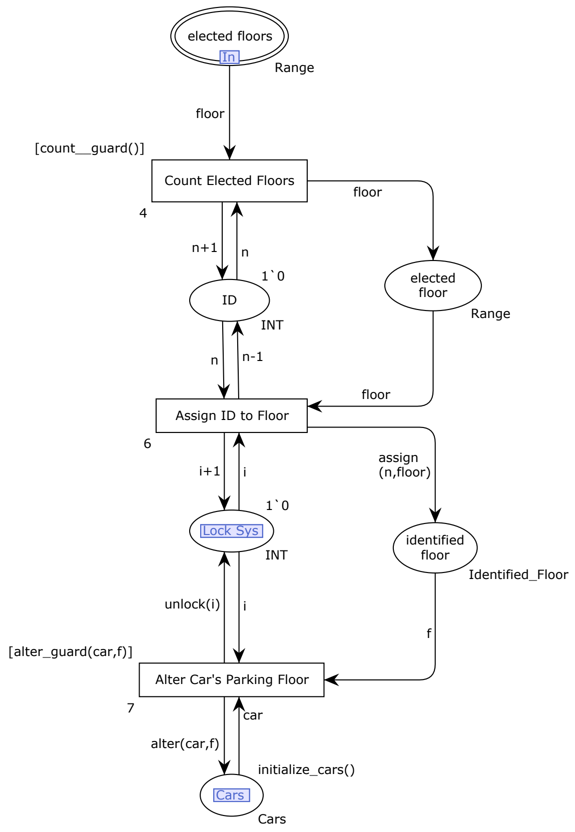


Figure 5.6: The assignment sub-model

Table 5.3: The colour set **Identified Floor**

Colour Set	Definition
Identified Floor	*{(floor id,floor's number) | floor id ∈ Car ID, floor's number ∈ Range}*

The initial step is counting the elected floors for two reasons. First, the elected floors may be less than the cars. Second, each elected floor must be associated with identification number (ID) to deterministically assign the floor to the car that has the same identification number, and thereby each car's parking floor is altered only once.

Counting the elected floors requires the firing of transition *Count Elected Floors* that is enabled if the selected technique is *"new assignment"*. When all elected floors were counted, transition *Count Elected Floors* is disabled since there are no more tokens in place *elected floors*. In contrast, transition *Assign ID to Floor* is enabled. For each firing of transition *Assign ID to Floor*, each elected floor is associated with an identification number and transferred to place *identified floor* of colour set **Identified Floor** defined in Table 5.3.

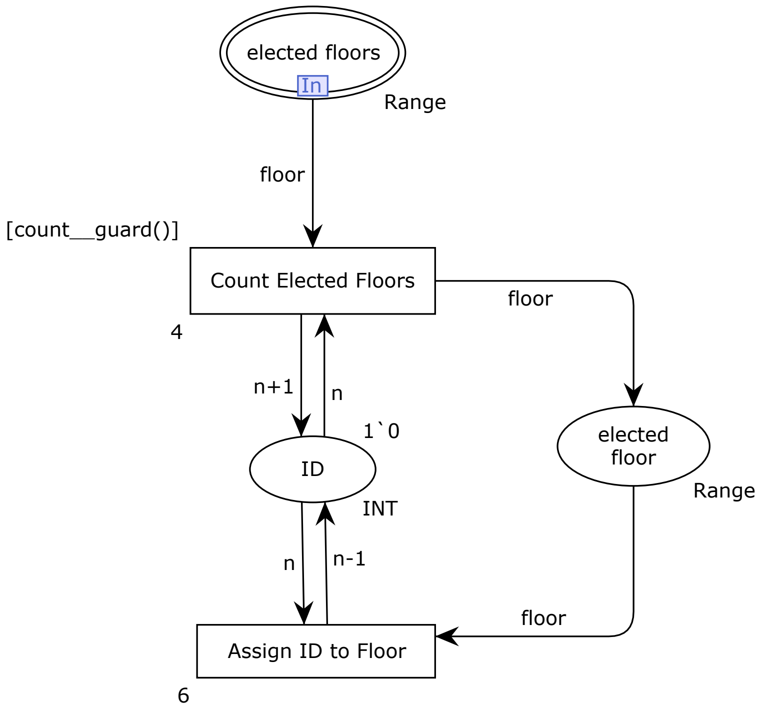


Figure 5.7: Processing the floors by the assignment sub-model

The final step is assigning the elected floors to the cars through the firing of transition *Alter Car's Parking Floor.* Thus, the technique is completed and the parking system is unlocked.

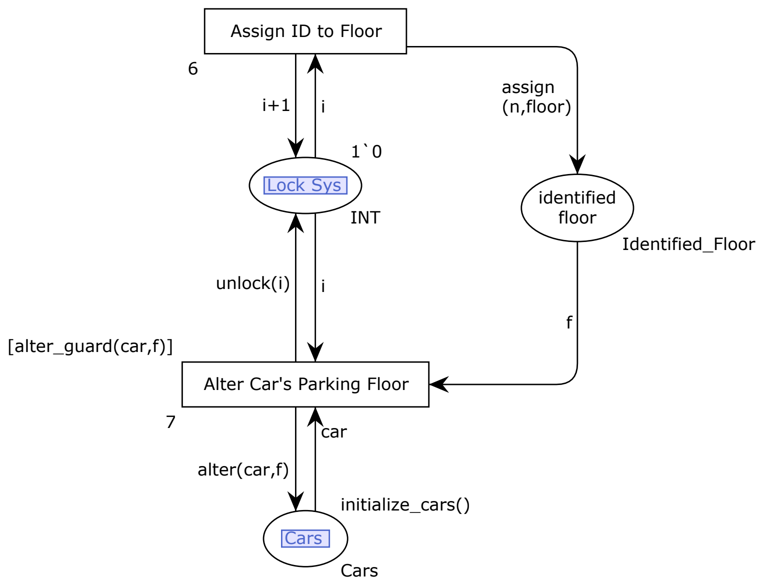


Figure 5.8: Altering the cars' parking floors by the assignment sub-model

5.4 The Position Sub-model

The position sub-model (Figure 5.9) performs the technique of altering the cars' parking floors with respect to their scopes, which perhaps reduces the overall waiting time.

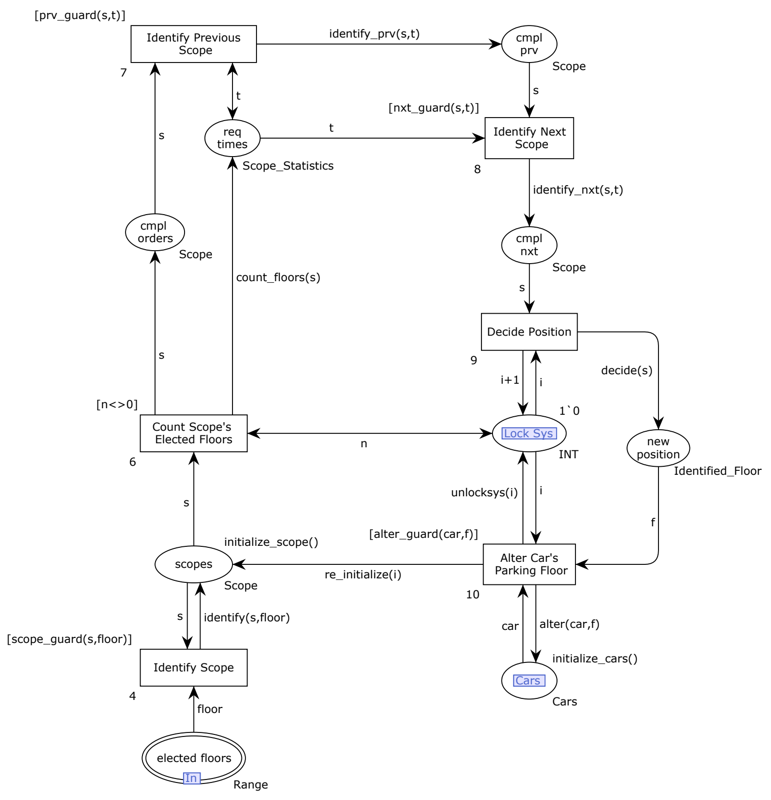


Figure 5.9: The position sub-model

Table 5.4: The colour sets **Scope** and **Scope Statistics**

Colour Sets	Definitions
Scope	$\{$*(scope id,prev,next,elected floors)* $\mid$ *scope id* $\in$ *Car ID, prev* $\in \mathbb{Z}$*, next* $\in \mathbb{Z}$*, elected floors* $\in$ *INT List*$\}$
Scope Statistics	$\{$*(scope id,floors' number)* $\mid$ *scope id* $\in$ *Car ID, floors' number* $\in \mathbb{Z}\}$

The first step is specifying the scopes of the elected floors by the firing of transtion *Identify Scope* that is enabled if the selected technique is *"new position"*. After each firing of transition *Identify Scope*, an elected floor is removed from place *elected floors* and inserted into the list of the corresponding scope in place *scopes* of the colours set **Scope** defined in Table 5.4.

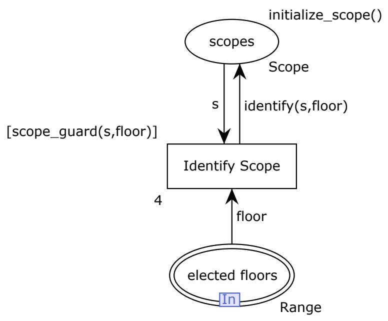


Figure 5.10: Identifying the scope of floors

The second step starts by counting the elected floors in each scope when transition *Count Scope's Elected Floors* is enabled and the parking system is locked (i.e. the value of place Lock Sys is one). After the firing of transition *Count Scope's Elected Floors*, a token (representing a scope) is transferred from place *scopes* to two places; place *complete orders (cmpl orders)* as a copy of the original token, and place *request times (req times)* as a token of the colour set *Scope Statistics* defined in Table 5.4.

After the counting of the elected floors in each scope, each scope is associated with the number of elected floors of the lower and higher scope. Calculating the elected floors of a lower scope requires the firing of transition *Identify previous Scope*. After the firing of transition *Identify previous Scope*, a token of the colour set **Scope (S)** from place *cmpl orders* and another token of the colour set **Scope Statistics (SS)** from place *req times* are removed.

The selection of the tokens is deterministic by the guards of transition *Identify previous Scope* as follows:

if a "scope id (S) of place (cmpl orders) is one, then it is associated with the "scope id (SS)" of place (req times) that is equal to the number of cars
(i.e. if scope id (S) is 1 = the scope id (SS) is cars' number).

Otherwise, each "scope id (S)" of place (cmpl orders) is associated with the "scope id (SS)" of place (req times) that is less by only one
(i.e. scope id (S) = (scope id (SS)) −1).

Similarly, the calculation of the elected floors of a higher scope is acquired through firing transition *Identify Next Scope* that has the following guards:
each "scope id (S)" of place (cmpl prv) is associated with a "scope id (SS)" of place (req times) that is greater by only one
(i.e. scope id (S) = scope id (SS) +1)
unless the "scope id (S)" of place (cmpl prv) equates numerically the cars' number, then the "scope id (SS)" of place (req times) must be one.

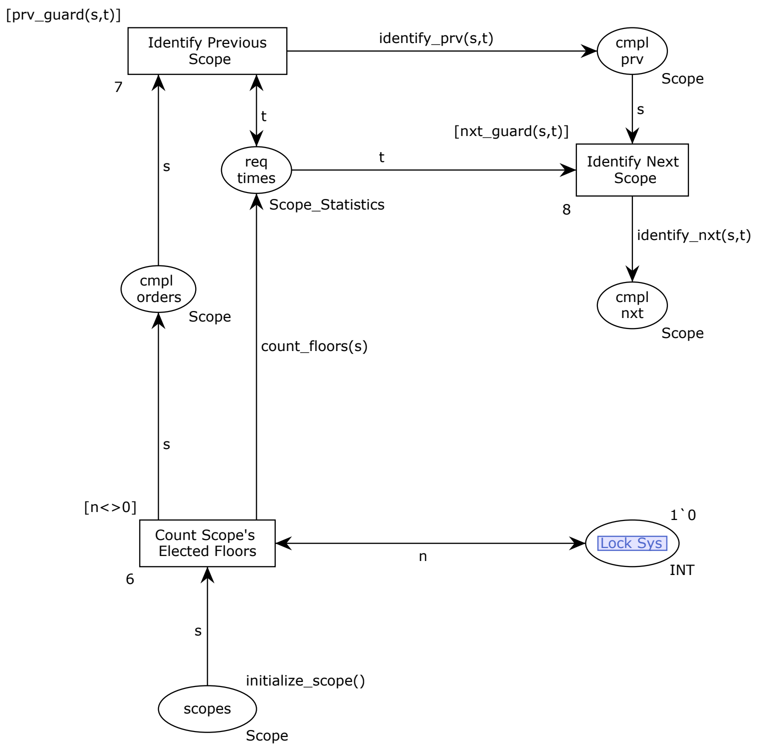


Figure 5.11: Completing the information of scopes

The third step (Figure 5.12) is selecting a new parking floor for each scope through the firing of transition *Decide Position*. After the firing of transition *Decide Position*, a token of place *complete next (cmpl nxt)* is moved to place *new position* as a token of the colour set **Identified Floor** defined in Table 5.2.

The new parking floor of a scope is chosen according to the following rules:

1. If the scope has many elected floors, then the floor of the highest reputation time is selected (the list of the elected floors of the colour set **Scope** is sorted).
2. If the scope has only one elected floor, then this floor is selected.
3. If the scope has no elected floors, then the head floor or the tail floor of the scope is selected in account of the number of the elected floors of the lower and higher scopes.

The final step is assigned each selected floor to a car through the firing of transition *Alter Cars Parking Floor*. Thus, this technique is completed and the parking system is unlocked.

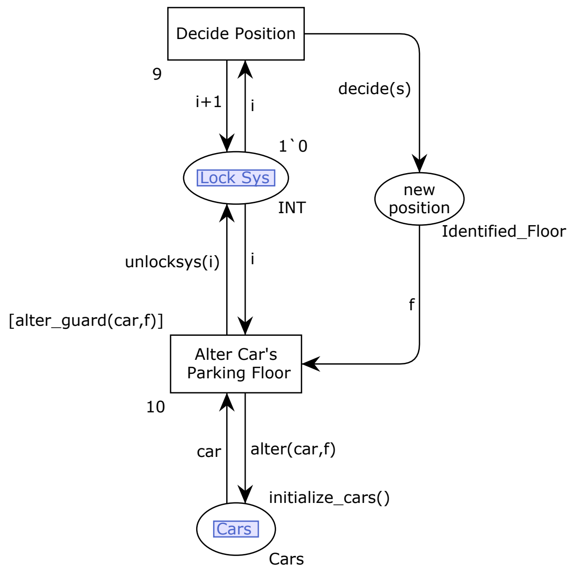


Figure 5.12: Altering the cars' parking floors by the position sub-model

Chapter 6

The Analyses

This chapter provides the analyses of the proposed model of the elevator system (for both the abstract version and the timing version). Section 6.1 introduces the techniques of analysing CPN-based models. Section 6.2 discuses the reachability analysis. Finally, section 6.3 presents the simulation-based performance analysis.

6.1 Introduction

Analysing a CPN model is carried out by various techniques and tools [van der Aalst and Stahl (2011)]. In this thesis, the focus is on two techniques, namely the reachability analysis (section 6.2) and the simulation-based performance analysis (section 6.3).

The reachability analysis is a formal verification technique where typically a tool is executed to construct a graph of nodes that represent reachable markings and arcs that represent enabled transitions of the markings. In this thesis, the State Space tool [Jensen *et al.* (2006)], which has been embedded into the CPN Tools [AIS Group (2013)], was used to analyse the proposed model. The tool reported automatically some properties of the analysed model as shown in (Figure 6.1) and (Figure 6.2).

The simulation-based performance analysis is based on executing the proposed model constantly while its data is being recorded, and then measured and compared. Consequently, this technique is flexible but also is time-consuming. In this thesis, this technique was performed by the simulator tool of the CPN Tools and also was combined with monitors supported by the CPN Tools. The monitors are data-collectors and controller techniques that guide the simulation of the model (Figure 6.3 and Figure 6.7).

6.2 The Reachability Analysis

The full reachability analysis of all sub-models could not be obtained because of the deep concurrency among the sub-models, which results in the state space explosion problem. Therefore, it must be noticeable that the reported results are true for only 58781 nodes of the abstract version and 60251 nodes of the timing version of the proposed model. Some of the reported results of the partial analysis may differ from the reported results of the full analysis of the same model, and thereby the reported results of the partial analysis are not fully reliable. However, few results from the partial analysis are expected to exist or increase in the full analysis of the proposed model. For example, the dead markings that have no enabled transitions as a consequence of *LOG* places, such as place *success log* (in Figure 3.11).

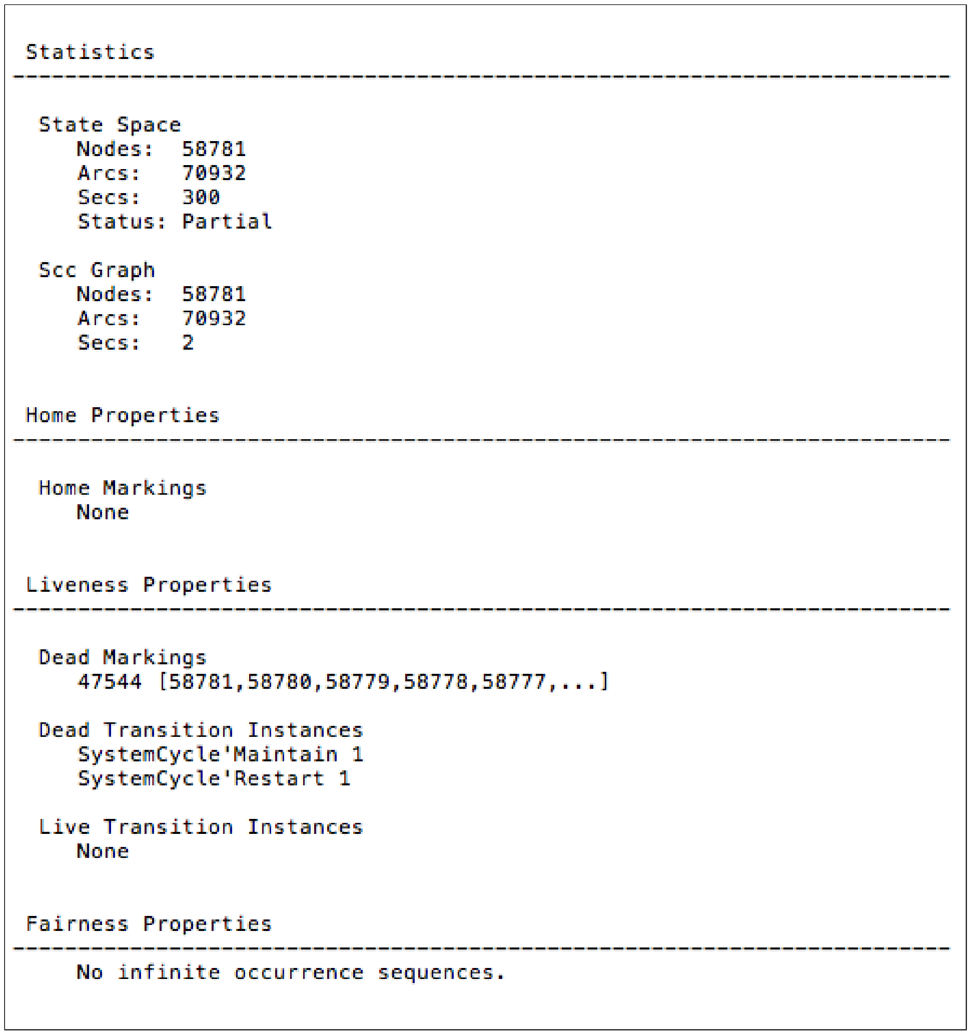
Finally, the reported results of the partial analysis are explained as follows:

- First, the statistics of (Figure 6.1 (a)) and (Figure 6.2 (a)) show the reachable markings and arcs of both the abstract version and the timing version of the CPN-based models of the elevator system, respectively.

- Second, the boundedness properties from (Figure 6.1 (b)) and (Figure 6.2 (b)) account for the maximum and minimum numbers of tokens for each place according to the parameters in (Figure 6.1 (c)) and (Figure 6.2 (c)), respectively.

- Third, both models have no home markings that are reachable from all markings, and consequently no infinite occurrence sequences as presented under home properties and fairness properties in (Figure 6.1 (a)) and (Figure 6.2 (a)), respectively.

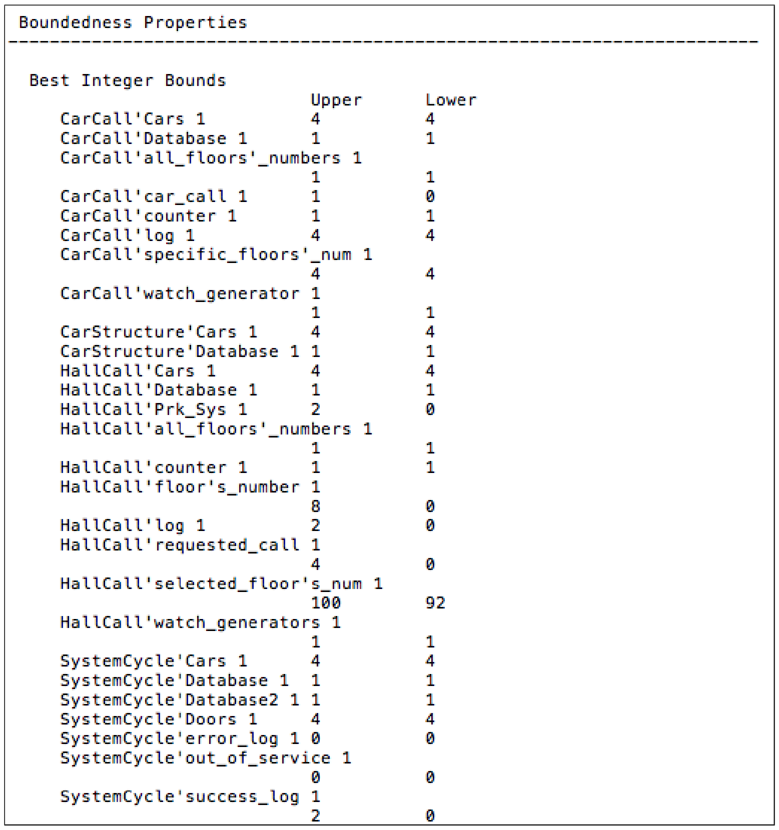
- Finally, the liveness properties specify the dead markings and the dead transitions that are never enabled, such as transition *Maintain* and transition *Restart*, which indicates the maintenance stage of the system-cycle sub-model were never enabled and thereby there was no operation failure in both models (as explained in the maintenance stage of the abstract system-cycle sub-model). Moreover, both versions of the proposed model have no live transitions that always have the possibility of becoming enabled.



```
 Statistics
------------------------------------------------------------------------

  State Space
     Nodes:  58781
     Arcs:   70932
     Secs:   300
     Status: Partial

  Scc Graph
     Nodes:  58781
     Arcs:   70932
     Secs:   2


 Home Properties
------------------------------------------------------------------------

  Home Markings
     None


 Liveness Properties
------------------------------------------------------------------------

  Dead Markings
     47544 [58781,58780,58779,58778,58777,...]

  Dead Transition Instances
     SystemCycle'Maintain 1
     SystemCycle'Restart 1

  Live Transition Instances
     None


 Fairness Properties
------------------------------------------------------------------------
     No infinite occurrence sequences.
```

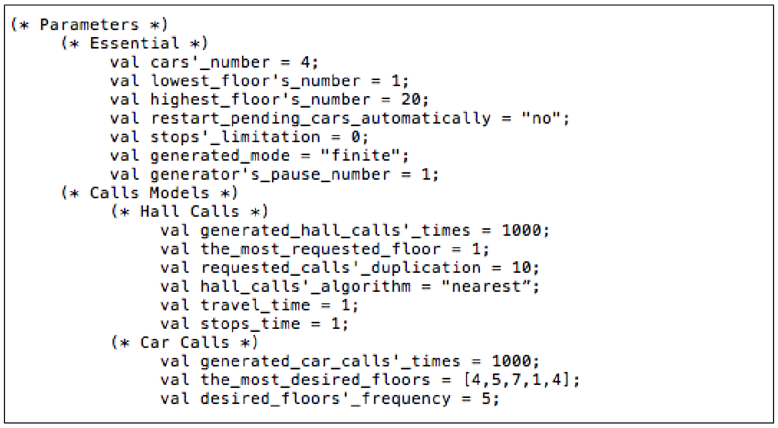
(a) The first part of the standard report

Figure 6.1: The standard report of the abstract version (1)



```
 Boundedness Properties
------------------------------------------------------------------------

  Best Integer Bounds
                            Upper      Lower
     CarCall'Cars 1         4          4
     CarCall'Database 1     1          1
     CarCall'all_floors'_numbers 1
                            1          1
     CarCall'car_call 1     1          0
     CarCall'counter 1      1          1
     CarCall'log 1          4          4
     CarCall'specific_floors'_num 1
                            4          4
     CarCall'watch_generator 1
                            1          1
     CarStructure'Cars 1    4          4
     CarStructure'Database 1 1         1
     HallCall'Cars 1        4          4
     HallCall'Database 1    1          1
     HallCall'Prk_Sys 1     2          0
     HallCall'all_floors'_numbers 1
                            1          1
     HallCall'counter 1     1          1
     HallCall'floor's_number 1
                            8          0
     HallCall'log 1         2          0
     HallCall'requested_call 1
                            4          0
     HallCall'selected_floor's_num 1
                            100        92
     HallCall'watch_generators 1
                            1          1
     SystemCycle'Cars 1     4          4
     SystemCycle'Database 1  1         1
     SystemCycle'Database2 1 1         1
     SystemCycle'Doors 1    4          4
     SystemCycle'error_log 1 0         0
     SystemCycle'out_of_service 1
                            0          0
     SystemCycle'success_log 1
                            2          0
```

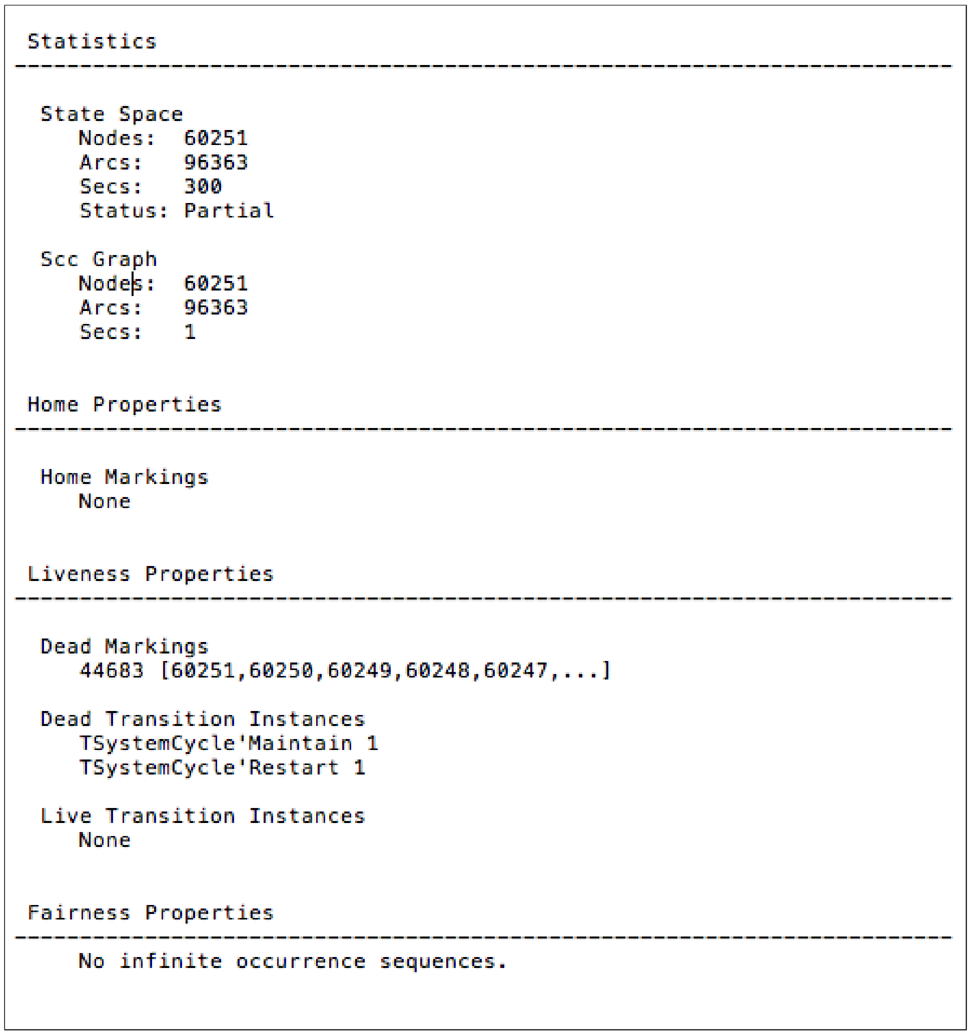
(b) The second part of the standard report



```
(* Parameters *)
     (* Essential *)
          val cars'_number = 4;
          val lowest_floor's_number = 1;
          val highest_floor's_number = 20;
          val restart_pending_cars_automatically = "no";
          val stops'_limitation = 0;
          val generated_mode = "finite";
          val generator's_pause_number = 1;
     (* Calls Models *)
          (* Hall Calls *)
               val generated_hall_calls'_times = 1000;
               val the_most_requested_floor = 1;
               val requested_calls'_duplication = 10;
               val hall_calls'_algorithm = "nearest";
               val travel_time = 1;
               val stops_time = 1;
          (* Car Calls *)
               val generated_car_calls'_times = 1000;
               val the_most_desired_floors = [4,5,7,1,4];
               val desired_floors'_frequency = 5;
```

(c) The adopted parameters

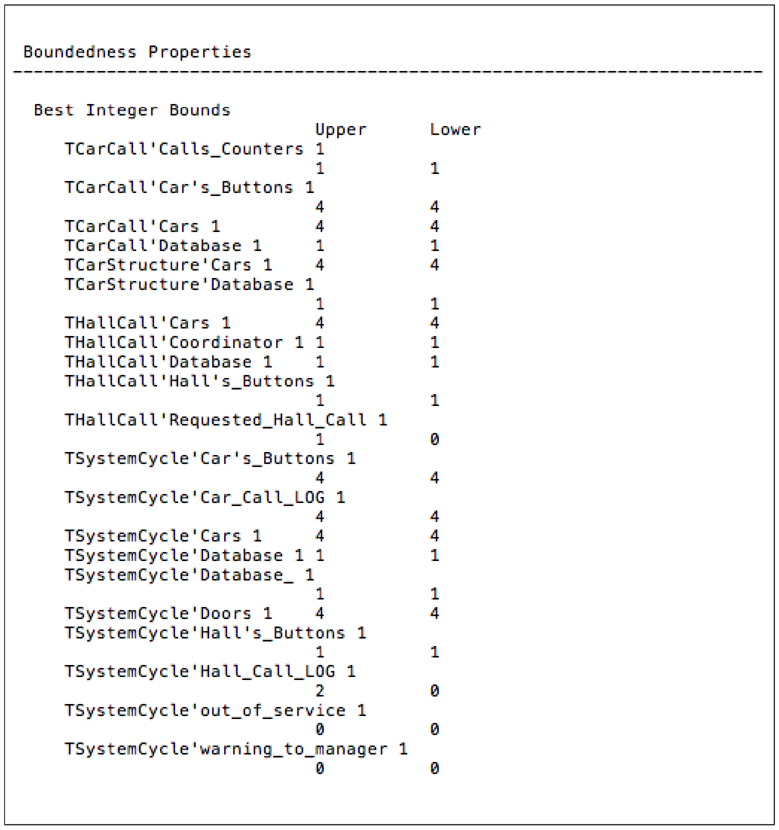
Figure 6.1: The standard report of the abstract version (2)



```
Statistics
------------------------------------------------------------------------

 State Space
    Nodes:  60251
    Arcs:   96363
    Secs:   300
    Status: Partial

 Scc Graph
    Nodes:  60251
    Arcs:   96363
    Secs:   1


Home Properties
------------------------------------------------------------------------

 Home Markings
    None


Liveness Properties
------------------------------------------------------------------------

 Dead Markings
    44683 [60251,60250,60249,60248,60247,...]

 Dead Transition Instances
    TSystemCycle'Maintain 1
    TSystemCycle'Restart 1

 Live Transition Instances
    None


Fairness Properties
------------------------------------------------------------------------
    No infinite occurrence sequences.
```

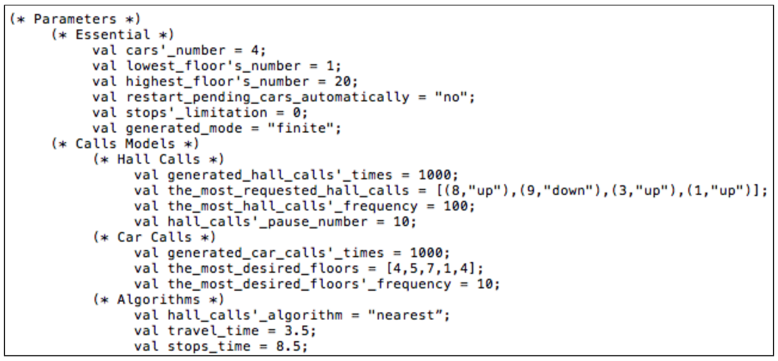
(a) The first part of the standard report

Figure 6.2: The standard report of the timing version (1)



```
 Boundedness Properties
------------------------------------------------------------------------

  Best Integer Bounds
                             Upper      Lower
     TCarCall'Calls_Counters 1
                             1          1
     TCarCall'Car's_Buttons 1
                             4          4
     TCarCall'Cars 1         4          4
     TCarCall'Database 1     1          1
     TCarStructure'Cars 1    4          4
     TCarStructure'Database 1
                             1          1
     THallCall'Cars 1        4          4
     THallCall'Coordinator 1 1          1
     THallCall'Database 1    1          1
     THallCall'Hall's_Buttons 1
                             1          1
     THallCall'Requested_Hall_Call 1
                             1          0
     TSystemCycle'Car's_Buttons 1
                             4          4
     TSystemCycle'Car_Call_LOG 1
                             4          4
     TSystemCycle'Cars 1     4          4
     TSystemCycle'Database 1 1          1
     TSystemCycle'Database_ 1
                             1          1
     TSystemCycle'Doors 1    4          4
     TSystemCycle'Hall's_Buttons 1
                             1          1
     TSystemCycle'Hall_Call_LOG 1
                             2          0
     TSystemCycle'out_of_service 1
                             0          0
     TSystemCycle'warning_to_manager 1
                             0          0
```

(b) The second part of the standard report



```
(* Parameters *)
    (* Essential *)
        val cars'_number = 4;
        val lowest_floor's_number = 1;
        val highest_floor's_number = 20;
        val restart_pending_cars_automatically = "no";
        val stops'_limitation = 0;
        val generated_mode = "finite";
    (* Calls Models *)
        (* Hall Calls *)
            val generated_hall_calls'_times = 1000;
            val the_most_requested_hall_calls = [(8,"up"),(9,"down"),(3,"up"),(1,"up")];
            val the_most_hall_calls'_frequency = 100;
            val hall_calls'_pause_number = 10;
        (* Car Calls *)
            val generated_car_calls'_times = 1000;
            val the_most_desired_floors = [4,5,7,1,4];
            val the_most_desired_floors'_frequency = 10;
        (* Algorithms *)
            val hall_calls'_algorithm = "nearest";
            val travel_time = 3.5;
            val stops_time = 8.5;
```

(c) The adopted parameters

Figure 6.2: The standard report of the timing version (2)

6.3 The Simulation-based Performance Analysis

The proposed model was analysed via the simulation-based performance analysis with different parameters that include the number of cars and floors, the applied algorithm, and the specified floors. After a series of simulating the proposed model, the collected data proves the compatibility and applicability of this model in various situations.

One of the case studies, for example, was four cars serving a twenty-floor building (see the adopted parameter in Figure 6.4 (a)). The collected data shows the following:

1. Car calls were produced by the abstract version and the timing version for each car to each floor. Additionally in the timing version, the car-call buttons were illuminated when released and unilluminated when the cars visited the corresponding floors (Figure 6.4 (b, c) and Figure 6.8 (d, e)).

2. Hall calls were produced by the abstract version and the timing version from all floors to request upward moving cars (with the exception of the highest floor) and downward moving cars (with the exception of the lowest floor). All produced hall calls were assigned and delivered successfully. Additionally in the timing version, the hall-call buttons were illuminated when released and unilluminated when the cars visited the floors of the requested hall calls (Figure 6.4 (d) and Figure 6.8 (b, c)).

3. All cars were held at their parking floors when they received no calls (Figure 6.5 (c) and Figure 6.9 (c)).

4. All requested hall calls were served eventually during the implementation of the adopted algorithms (Figure 6.5 (b) and Figure 6.9 (a)).

5. All requested car calls were served eventually and sequentially in the direction of cars' travel (Figure 6.5 (b) and Figure 6.9 (b)).

6. The maintenance stage of the system-cycle sub-model is designed to deal with the emergency and failure cases. This stage helped in discovering some errors during the programming of the proposed model (CPN ML language was used to code the functions that implement the transitions' guards and arcs' expressions). Some of the discovered and fixed errors were in reversing the cars' direction, synchronizing the database, and controlling the overproduction of calls. Moreover, the maintenance stage provides a controllable mechanism to restart the out-of-service cars, and also the timing version models the sending of a warning signal to the site manager (see section 3.5 and section 4.5).

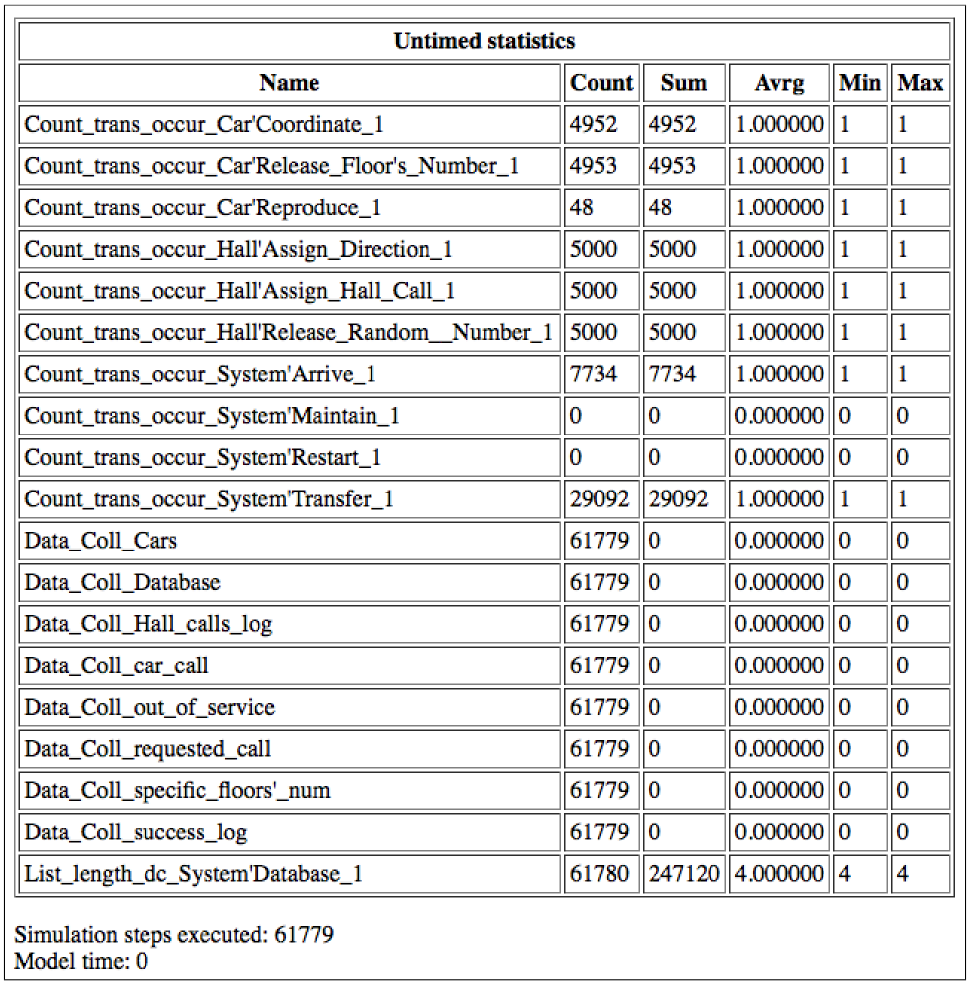
Therefore, the thesis objective was accomplished completely and successfully.

Furthermore, the collected data from different parameters can be measured and compared. Three experiments are presented as follows:

- Figure 6.6 shows the observation of the cars' behaviours through the developed Visualization Extension.

- Figure 6.10 presents results of elected floors by the two techniques of the parking optimizer model.

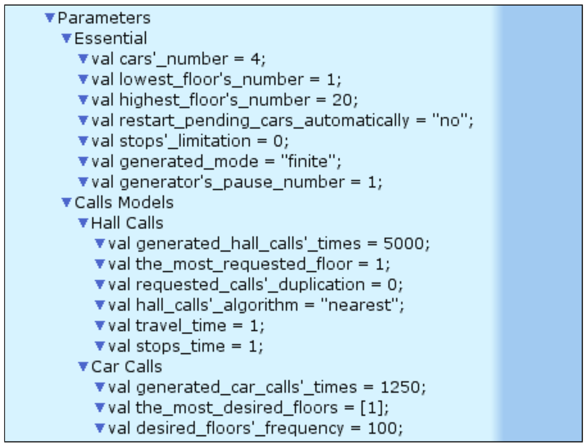
- Figure 6.11 compares the waiting times between the adoption of the nearest-car algorithm and the minimum-waiting algorithm. In contrast, Figure 6.12 compares the waiting times between the adoption of different numbers of cars during the scope algorithm.



Untimed statistics					
Name	Count	Sum	Avrg	Min	Max
Count_trans_occur_Car'Coordinate_1	4952	4952	1.000000	1	1
Count_trans_occur_Car'Release_Floor's_Number_1	4953	4953	1.000000	1	1
Count_trans_occur_Car'Reproduce_1	48	48	1.000000	1	1
Count_trans_occur_Hall'Assign_Direction_1	5000	5000	1.000000	1	1
Count_trans_occur_Hall'Assign_Hall_Call_1	5000	5000	1.000000	1	1
Count_trans_occur_Hall'Release_Random__Number_1	5000	5000	1.000000	1	1
Count_trans_occur_System'Arrive_1	7734	7734	1.000000	1	1
Count_trans_occur_System'Maintain_1	0	0	0.000000	0	0
Count_trans_occur_System'Restart_1	0	0	0.000000	0	0
Count_trans_occur_System'Transfer_1	29092	29092	1.000000	1	1
Data_Coll_Cars	61779	0	0.000000	0	0
Data_Coll_Database	61779	0	0.000000	0	0
Data_Coll_Hall_calls_log	61779	0	0.000000	0	0
Data_Coll_car_call	61779	0	0.000000	0	0
Data_Coll_out_of_service	61779	0	0.000000	0	0
Data_Coll_requested_call	61779	0	0.000000	0	0
Data_Coll_specific_floors'_num	61779	0	0.000000	0	0
Data_Coll_success_log	61779	0	0.000000	0	0
List_length_dc_System'Database_1	61780	247120	4.000000	4	4

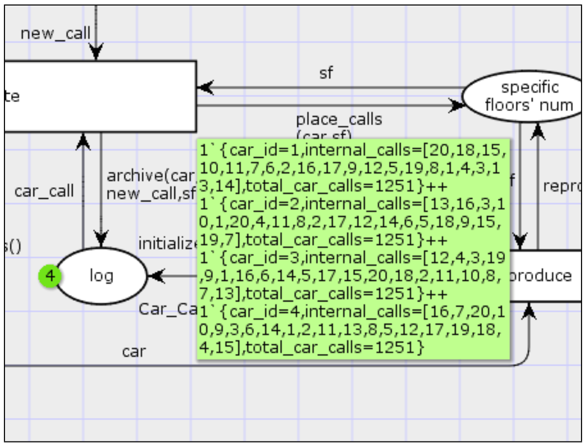
Simulation steps executed: 61779
Model time: 0

Figure 6.3: The monitors' statistics of the abstract version



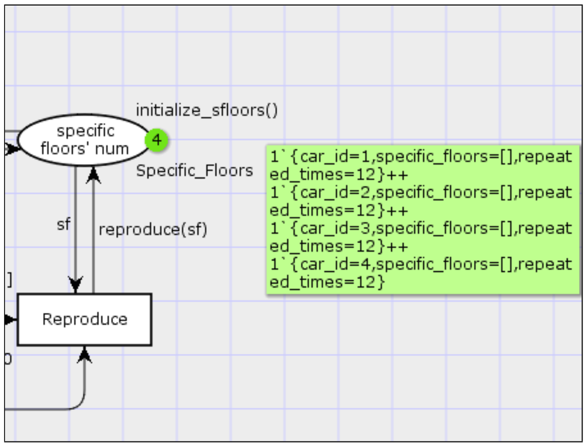
```
▼Parameters
  ▼Essential
    ▼val cars'_number = 4;
    ▼val lowest_floor's_number = 1;
    ▼val highest_floor's_number = 20;
    ▼val restart_pending_cars_automatically = "no";
    ▼val stops'_limitation = 0;
    ▼val generated_mode = "finite";
    ▼val generator's_pause_number = 1;
  ▼Calls Models
    ▼Hall Calls
      ▼val generated_hall_calls'_times = 5000;
      ▼val the_most_requested_floor = 1;
      ▼val requested_calls'_duplication = 0;
      ▼val hall_calls'_algorithm = "nearest";
      ▼val travel_time = 1;
      ▼val stops_time = 1;
    ▼Car Calls
      ▼val generated_car_calls'_times = 1250;
      ▼val the_most_desired_floors = [1];
      ▼val desired_floors'_frequency = 100;
```

(a) The adopted parameters



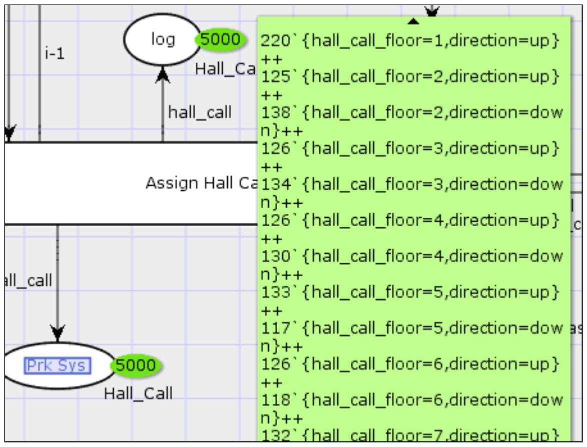
```
1`{car_id=1,internal_calls=[20,18,15,
10,11,7,6,2,16,17,9,12,5,19,8,1,4,3,1
3,14],total_car_calls=1251}++
1`{car_id=2,internal_calls=[13,16,3,1
0,1,20,4,11,8,2,17,12,14,6,5,18,9,15,
19,7],total_car_calls=1251}++
1`{car_id=3,internal_calls=[12,4,3,19
,9,1,16,6,14,5,17,15,20,18,2,11,10,8,
7,13],total_car_calls=1251}++
1`{car_id=4,internal_calls=[16,7,20,1
0,9,3,6,14,1,2,11,13,8,5,12,17,19,18,
4,15],total_car_calls=1251}
```

(b) The log of car calls



```
1`{car_id=1,specific_floors=[],repeat
ed_times=12}++
1`{car_id=2,specific_floors=[],repeat
ed_times=12}++
1`{car_id=3,specific_floors=[],repeat
ed_times=12}++
1`{car_id=4,specific_floors=[],repeat
ed_times=12}
```

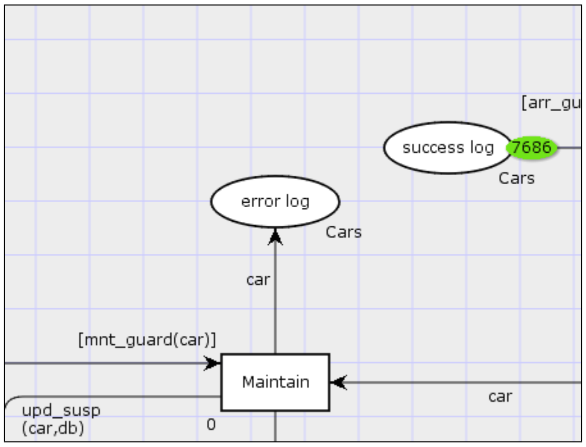
(c) Placed specific floors



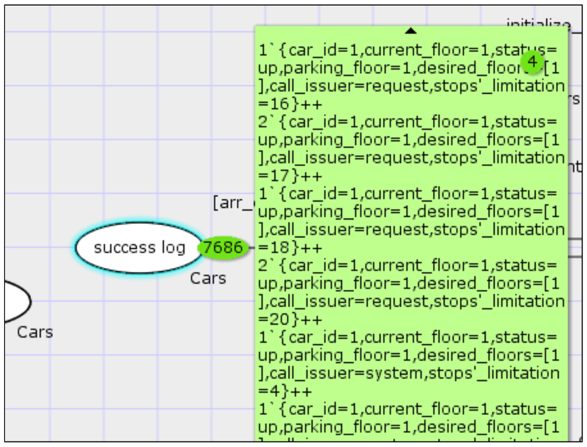
```
220`{hall_call_floor=1,direction=up}
++
125`{hall_call_floor=2,direction=up}
++
138`{hall_call_floor=2,direction=dow
n}++
126`{hall_call_floor=3,direction=up}
++
134`{hall_call_floor=3,direction=dow
n}++
126`{hall_call_floor=4,direction=up}
++
130`{hall_call_floor=4,direction=dow
n}++
133`{hall_call_floor=5,direction=up}
++
117`{hall_call_floor=5,direction=dow
n}++
126`{hall_call_floor=6,direction=up}
++
118`{hall_call_floor=6,direction=dow
n}++
132`{hall_call_floor=7,direction=up}
```

(d) Placed hall calls

Figure 6.4: Produced calls from the abstract version

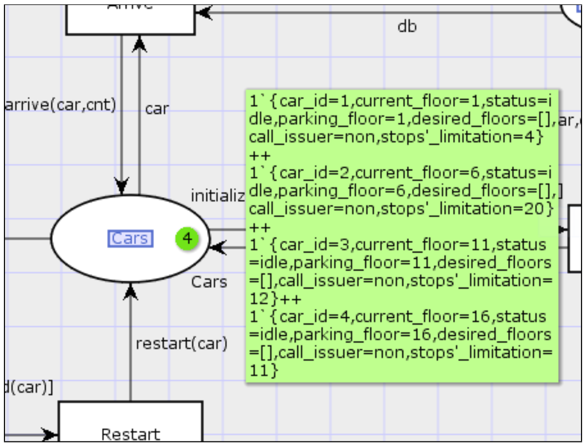


(a) The error and success logs



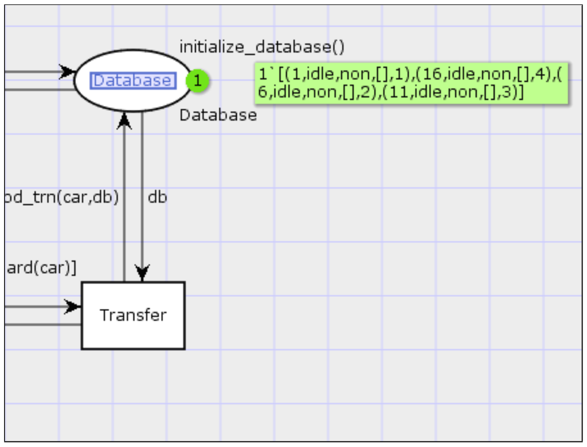
```
1`{car_id=1,current_floor=1,status=
up,parking_floor=1,desired_floors=[1
],call_issuer=request,stops'_limitation
=16}++
2`{car_id=1,current_floor=1,status=
up,parking_floor=1,desired_floors=[1
],call_issuer=request,stops'_limitation
=17}++
1`{car_id=1,current_floor=1,status=
up,parking_floor=1,desired_floors=[1
],call_issuer=request,stops'_limitation
=18}++
2`{car_id=1,current_floor=1,status=
up,parking_floor=1,desired_floors=[1
],call_issuer=request,stops'_limitation
=20}++
1`{car_id=1,current_floor=1,status=
up,parking_floor=1,desired_floors=[1
],call_issuer=system,stops'_limitation
=4}++
1`{car_id=1,current_floor=1,status=
up,parking_floor=1,desired_floors=[1
```

(b) The success log



```
1`{car_id=1,current_floor=1,status=i
dle,parking_floor=1,desired_floors=[],
call_issuer=non,stops'_limitation=4}
++
1`{car_id=2,current_floor=6,status=i
dle,parking_floor=6,desired_floors=[],
call_issuer=non,stops'_limitation=20}
++
1`{car_id=3,current_floor=11,status
=idle,parking_floor=11,desired_floors
=[],call_issuer=non,stops'_limitation=
12}++
1`{car_id=4,current_floor=16,status
=idle,parking_floor=16,desired_floors
=[],call_issuer=non,stops'_limitation=
11}
```

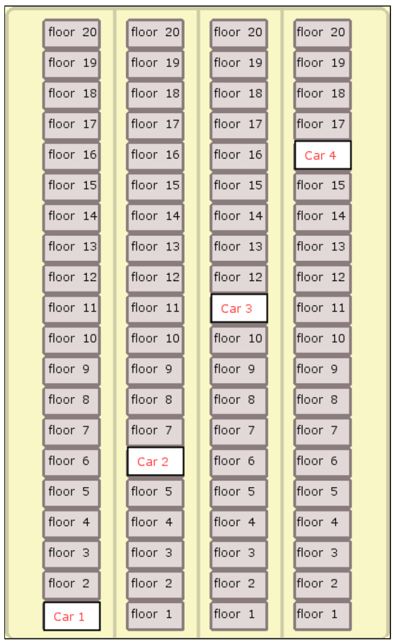
(c) Cars



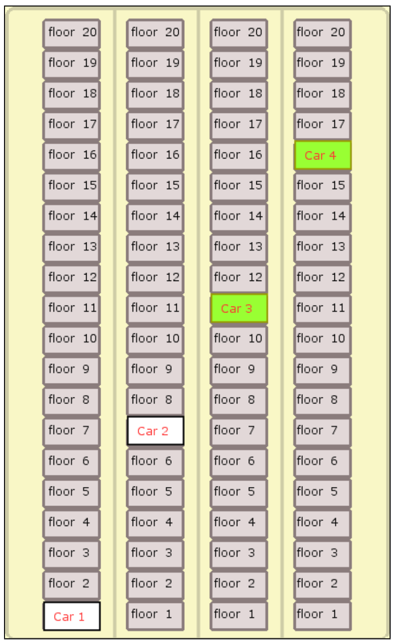
```
1`[(1,idle,non,[],1),(16,idle,non,[],4),(
6,idle,non,[],2),(11,idle,non,[],3)]
```

(d) Database

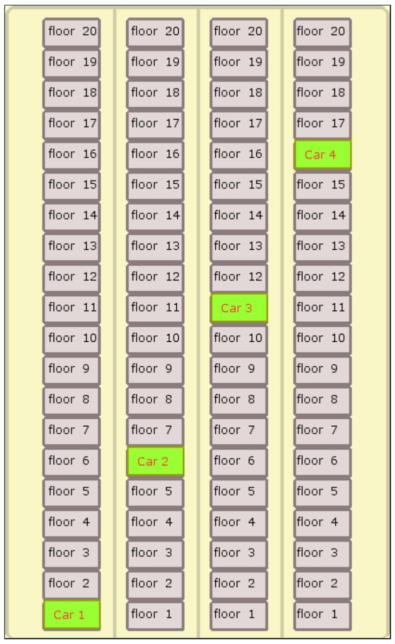
Figure 6.5: Results from the abstract version



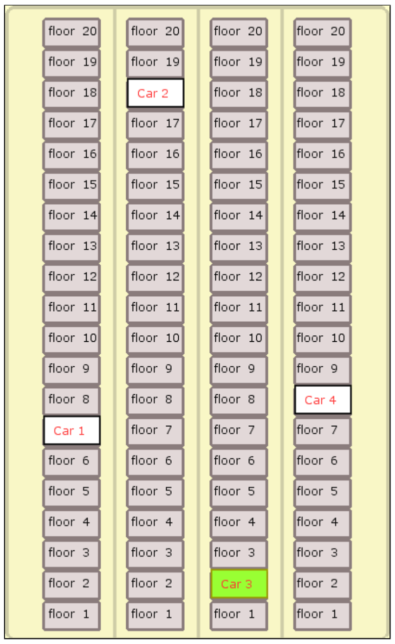
(a)



(b)

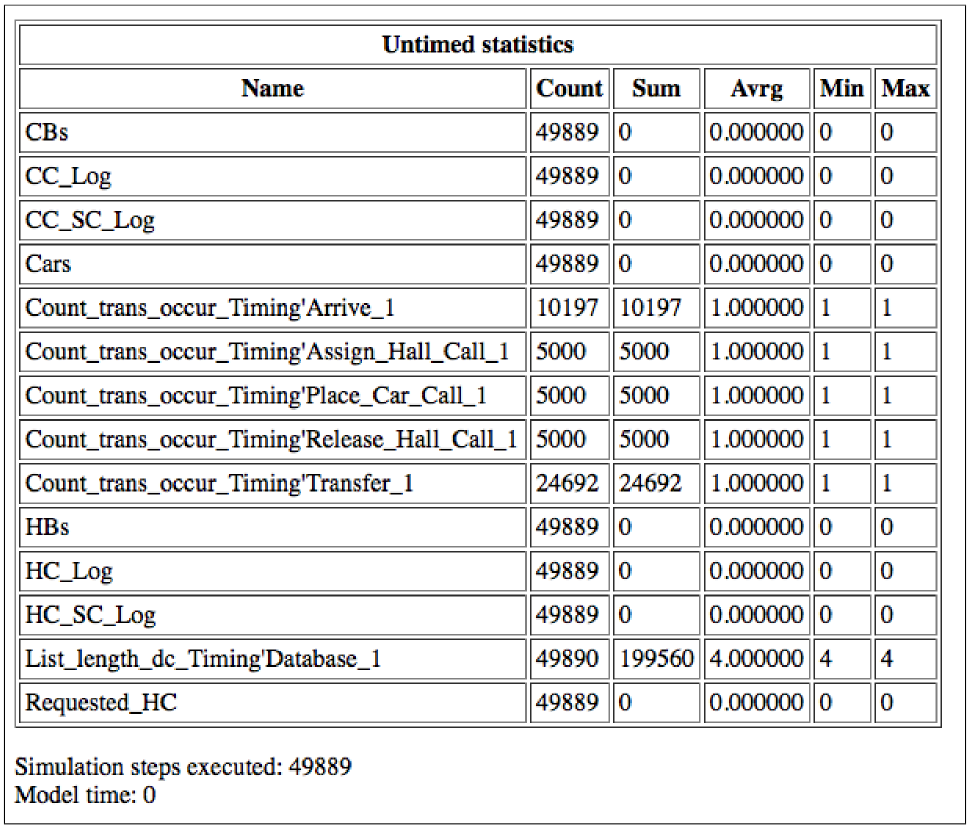


(c)



(d)

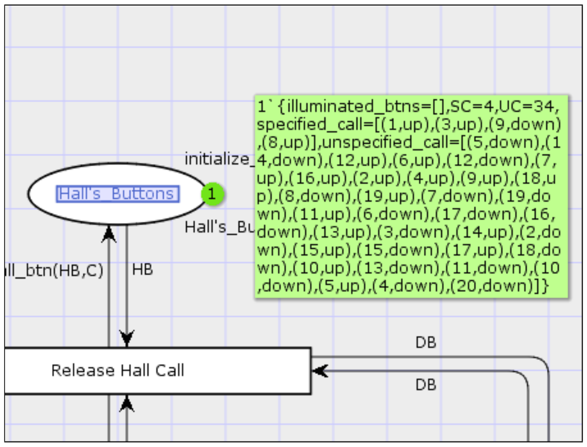
Figure 6.6: Observing different experiments through the Visualization Extension



Untimed statistics					
Name	Count	Sum	Avrg	Min	Max
CBs	49889	0	0.000000	0	0
CC_Log	49889	0	0.000000	0	0
CC_SC_Log	49889	0	0.000000	0	0
Cars	49889	0	0.000000	0	0
Count_trans_occur_Timing'Arrive_1	10197	10197	1.000000	1	1
Count_trans_occur_Timing'Assign_Hall_Call_1	5000	5000	1.000000	1	1
Count_trans_occur_Timing'Place_Car_Call_1	5000	5000	1.000000	1	1
Count_trans_occur_Timing'Release_Hall_Call_1	5000	5000	1.000000	1	1
Count_trans_occur_Timing'Transfer_1	24692	24692	1.000000	1	1
HBs	49889	0	0.000000	0	0
HC_Log	49889	0	0.000000	0	0
HC_SC_Log	49889	0	0.000000	0	0
List_length_dc_Timing'Database_1	49890	199560	4.000000	4	4
Requested_HC	49889	0	0.000000	0	0

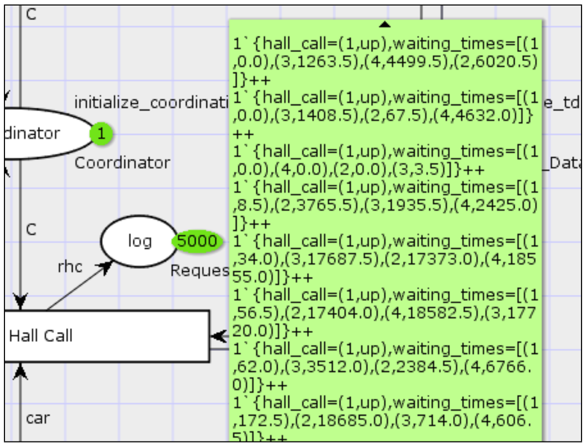
Simulation steps executed: 49889
Model time: 0

Figure 6.7: The monitors' statistics of timing version



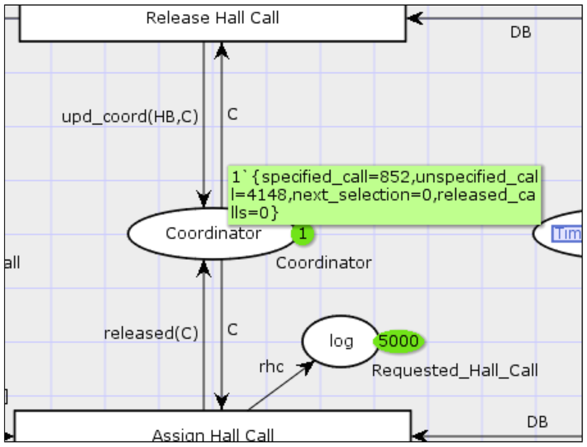
```
1`{illuminated_btns=[],SC=4,UC=34,
specified_call=[(1,up),(3,up),(9,down)
,(8,up)],unspecified_call=[(5,down),(1
4,down),(12,up),(6,up),(12,down),(7,
up),(16,up),(2,up),(4,up),(9,up),(18,u
p),(8,down),(19,up),(7,down),(19,do
wn),(11,up),(6,down),(17,down),(16,
down),(13,up),(3,down),(14,up),(2,do
wn),(15,up),(15,down),(17,up),(18,do
wn),(10,up),(13,down),(11,down),(10
,down),(5,up),(4,down),(20,down)]}
```

(a) The buttons of the hall calls



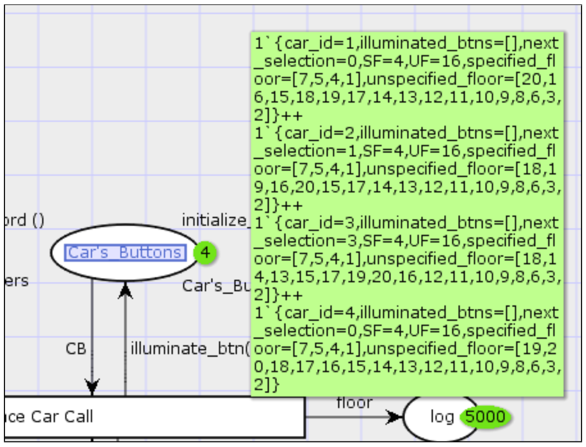
```
1`{hall_call=(1,up),waiting_times=[(1
,0.0),(3,1263.5),(4,4499.5),(2,6020.5)
]}++
1`{hall_call=(1,up),waiting_times=[(1
,0.0),(3,1408.5),(2,67.5),(4,4632.0)]}
++
1`{hall_call=(1,up),waiting_times=[(1
,0.0),(4,0.0),(2,0.0),(3,3.5)]}++
1`{hall_call=(1,up),waiting_times=[(1
,8.5),(2,3765.5),(3,1935.5),(4,2425.0)
]}++
1`{hall_call=(1,up),waiting_times=[(1
,34.0),(3,17687.5),(2,17373.0),(4,185
55.0)]}++
1`{hall_call=(1,up),waiting_times=[(1
,56.5),(2,17404.0),(4,18582.5),(3,177
20.0)]}++
1`{hall_call=(1,up),waiting_times=[(1
,62.0),(3,3512.0),(2,2384.5),(4,6766.
0)]}++
1`{hall_call=(1,up),waiting_times=[(1
,172.5),(2,18685.0),(3,714.0),(4,606.
5)]}++
```

(b) Placed hall calls



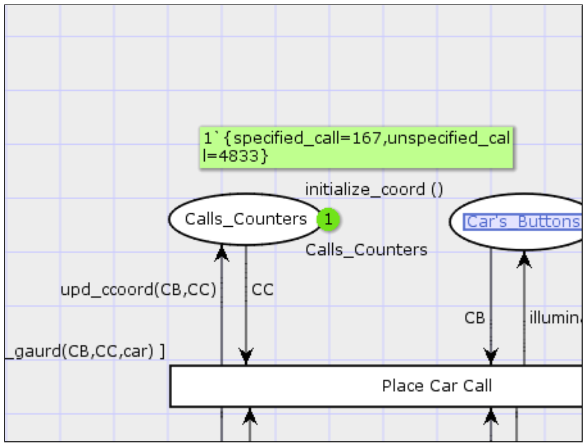
```
1`{specified_call=852,unspecified_cal
l=4148,next_selection=0,released_ca
lls=0}
```

(c) The coordinator of the hall calls



```
1`{car_id=1,illuminated_btns=[],next
_selection=0,SF=4,UF=16,specified_fl
oor=[7,5,4,1],unspecified_floor=[20,1
6,15,18,19,17,14,13,12,11,10,9,8,6,3,
2]}++
1`{car_id=2,illuminated_btns=[],next
_selection=1,SF=4,UF=16,specified_fl
oor=[7,5,4,1],unspecified_floor=[18,1
9,16,20,15,17,14,13,12,11,10,9,8,6,3,
2]}++
1`{car_id=3,illuminated_btns=[],next
_selection=3,SF=4,UF=16,specified_fl
oor=[7,5,4,1],unspecified_floor=[18,1
4,13,15,17,19,20,16,12,11,10,9,8,6,3,
2]}++
1`{car_id=4,illuminated_btns=[],next
_selection=0,SF=4,UF=16,specified_fl
oor=[7,5,4,1],unspecified_floor=[19,2
0,18,17,16,15,14,13,12,11,10,9,8,6,3,
2]}
```

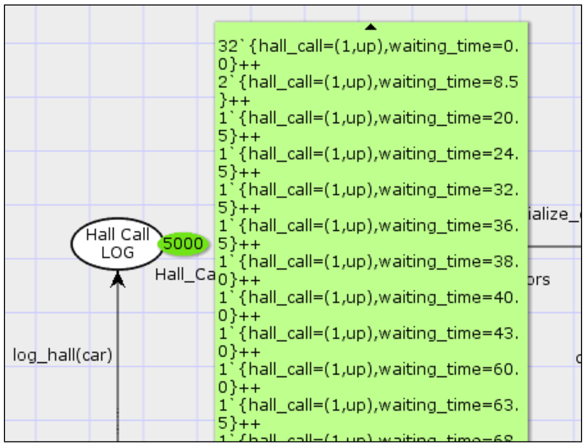
(d) The buttons of the car calls



```
1`{specified_call=167,unspecified_cal
l=4833}
```

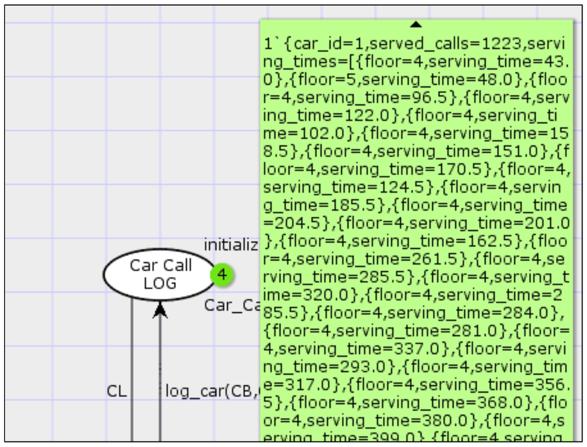
(e) The counters of the car calls

Figure 6.8: The calls of the timing version



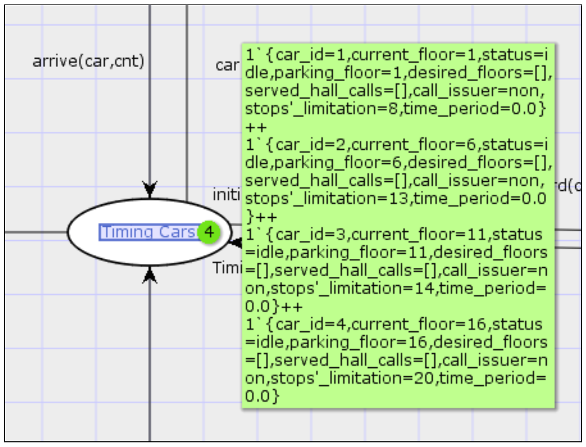
```
32`{hall_call=(1,up),waiting_time=0.
0}++
2`{hall_call=(1,up),waiting_time=8.5
}++
1`{hall_call=(1,up),waiting_time=20.
5}++
1`{hall_call=(1,up),waiting_time=24.
5}++
1`{hall_call=(1,up),waiting_time=32.
5}++
1`{hall_call=(1,up),waiting_time=36.
5}++
1`{hall_call=(1,up),waiting_time=38.
0}++
1`{hall_call=(1,up),waiting_time=40.
0}++
1`{hall_call=(1,up),waiting_time=43.
0}++
1`{hall_call=(1,up),waiting_time=60.
0}++
1`{hall_call=(1,up),waiting_time=63.
5}++
1`{hall_call=(1,up),waiting_time=68
```

(a) Recorded waiting times



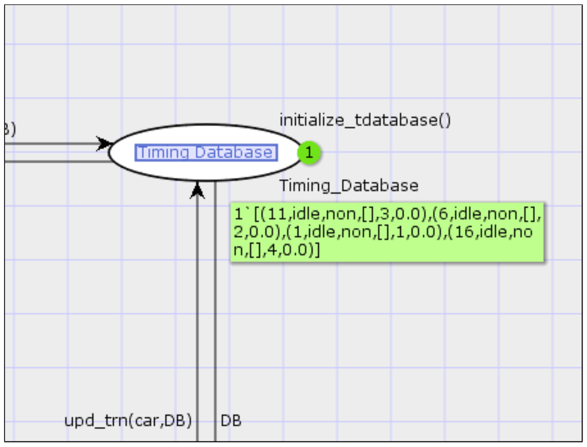
```
1`{car_id=1,served_calls=1223,servi
ng_times=[{floor=4,serving_time=43.
0},{floor=5,serving_time=48.0},{floo
r=4,serving_time=96.5},{floor=4,serv
ing_time=122.0},{floor=4,serving_ti
me=102.0},{floor=4,serving_time=15
8.5},{floor=4,serving_time=151.0},{f
loor=4,serving_time=170.5},{floor=4,
serving_time=124.5},{floor=4,servin
g_time=185.5},{floor=4,serving_time
=204.5},{floor=4,serving_time=201.0
},{floor=4,serving_time=162.5},{floo
r=4,serving_time=261.5},{floor=4,se
rving_time=285.5},{floor=4,serving_t
ime=320.0},{floor=4,serving_time=2
85.5},{floor=4,serving_time=284.0},
{floor=4,serving_time=281.0},{floor=
4,serving_time=337.0},{floor=4,servi
ng_time=293.0},{floor=4,serving_tim
e=317.0},{floor=4,serving_time=356.
5},{floor=4,serving_time=368.0},{flo
or=4,serving_time=380.0},{floor=4,s
erving_time=399.0},{floor=4,serving
```

(b) Recorded serving times



```
1`{car_id=1,current_floor=1,status=i
dle,parking_floor=1,desired_floors=[],
served_hall_calls=[],call_issuer=non,
stops'_limitation=8,time_period=0.0}
++
1`{car_id=2,current_floor=6,status=i
dle,parking_floor=6,desired_floors=[],
served_hall_calls=[],call_issuer=non,
stops'_limitation=13,time_period=0.0
}++
1`{car_id=3,current_floor=11,status
=idle,parking_floor=11,desired_floors
=[],served_hall_calls=[],call_issuer=n
on,stops'_limitation=14,time_period=
0.0}++
1`{car_id=4,current_floor=16,status
=idle,parking_floor=16,desired_floors
=[],served_hall_calls=[],call_issuer=n
on,stops'_limitation=20,time_period=
0.0}
```

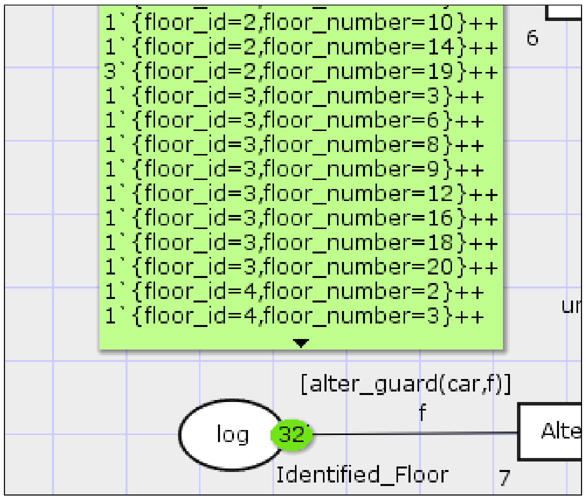
(c) Cars



```
1`[(11,idle,non,[],3,0.0),(6,idle,non,[],
2,0.0),(1,idle,non,[],1,0.0),(16,idle,no
n,[],4,0.0)]
```

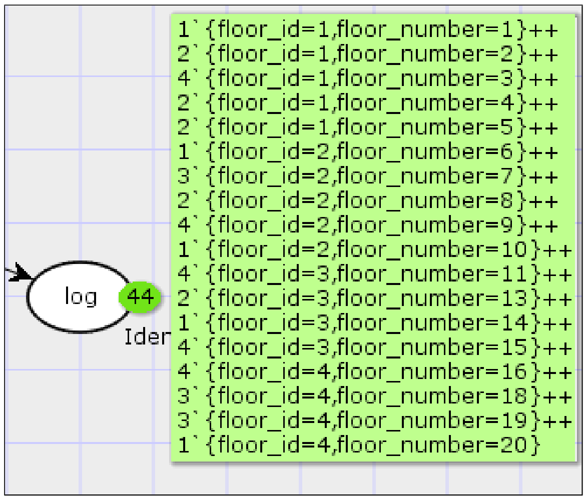
(d) Database

Figure 6.9: Results from the timing version



```
1`{floor_id=2,floor_number=10}++
1`{floor_id=2,floor_number=14}++
3`{floor_id=2,floor_number=19}++
1`{floor_id=3,floor_number=3}++
1`{floor_id=3,floor_number=6}++
1`{floor_id=3,floor_number=8}++
1`{floor_id=3,floor_number=9}++
1`{floor_id=3,floor_number=12}++
1`{floor_id=3,floor_number=16}++
1`{floor_id=3,floor_number=18}++
1`{floor_id=3,floor_number=20}++
1`{floor_id=4,floor_number=2}++
1`{floor_id=4,floor_number=3}++
```

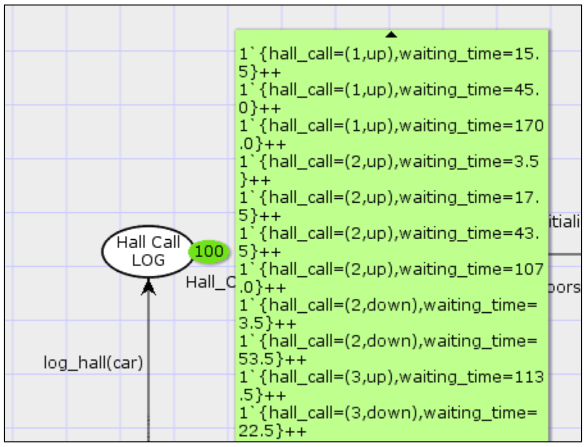
(a) The assignment technique



```
1`{floor_id=1,floor_number=1}++
2`{floor_id=1,floor_number=2}++
4`{floor_id=1,floor_number=3}++
2`{floor_id=1,floor_number=4}++
2`{floor_id=1,floor_number=5}++
1`{floor_id=2,floor_number=6}++
3`{floor_id=2,floor_number=7}++
2`{floor_id=2,floor_number=8}++
4`{floor_id=2,floor_number=9}++
1`{floor_id=2,floor_number=10}++
4`{floor_id=3,floor_number=11}++
2`{floor_id=3,floor_number=13}++
1`{floor_id=3,floor_number=14}++
4`{floor_id=3,floor_number=15}++
4`{floor_id=4,floor_number=16}++
3`{floor_id=4,floor_number=18}++
3`{floor_id=4,floor_number=19}++
1`{floor_id=4,floor_number=20}
```

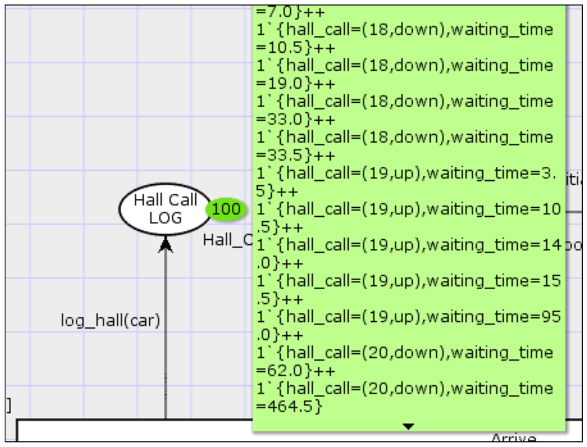
(b) The position technique

Figure 6.10: Results from the techniques of the parking optimizer model



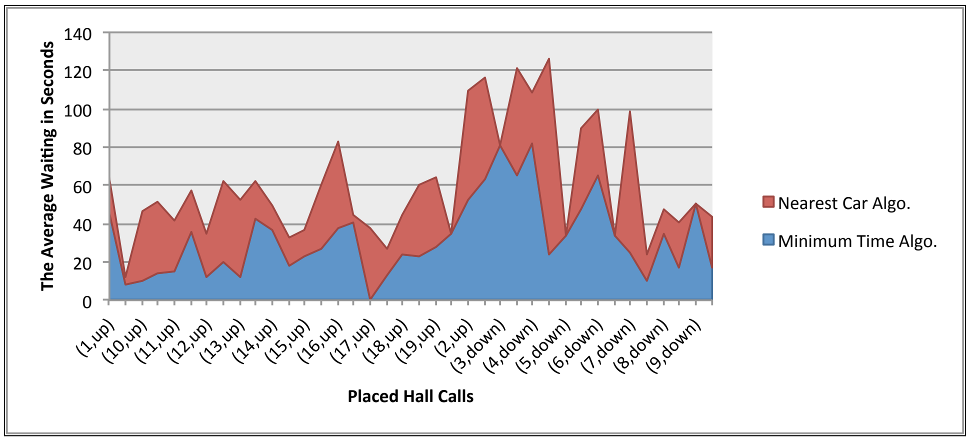
```
1`{hall_call=(1,up),waiting_time=15.
5}++
1`{hall_call=(1,up),waiting_time=45.
0}++
1`{hall_call=(1,up),waiting_time=170
.0}++
1`{hall_call=(2,up),waiting_time=3.5
}++
1`{hall_call=(2,up),waiting_time=17.
5}++
1`{hall_call=(2,up),waiting_time=43.
5}++
1`{hall_call=(2,up),waiting_time=107
.0}++
1`{hall_call=(2,down),waiting_time=
3.5}++
1`{hall_call=(2,down),waiting_time=
53.5}++
1`{hall_call=(3,up),waiting_time=113
.5}++
1`{hall_call=(3,down),waiting_time=
22.5}++
```

(a) Results by the minimum-waiting algorithm



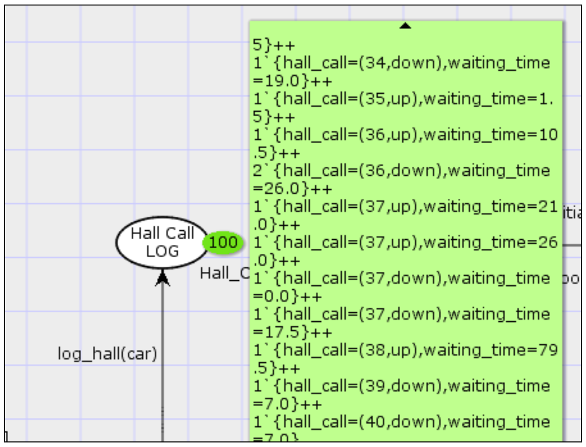
```
=7.0}++
1`{hall_call=(18,down),waiting_time
=10.5}++
1`{hall_call=(18,down),waiting_time
=19.0}++
1`{hall_call=(18,down),waiting_time
=33.0}++
1`{hall_call=(18,down),waiting_time
=33.5}++
1`{hall_call=(19,up),waiting_time=3.
5}++
1`{hall_call=(19,up),waiting_time=10
.5}++
1`{hall_call=(19,up),waiting_time=14
.0}++
1`{hall_call=(19,up),waiting_time=15
.5}++
1`{hall_call=(19,up),waiting_time=95
.0}++
1`{hall_call=(20,down),waiting_time
=62.0}++
1`{hall_call=(20,down),waiting_time
=464.5}
```

(b) Results by the nearest-car algorithm



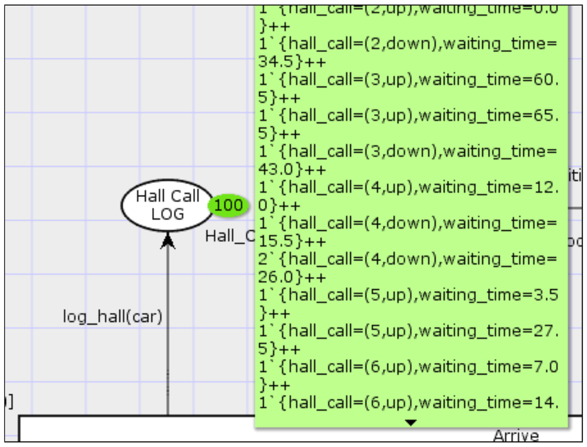
(c) The chart of the comparison

Figure 6.11: The comparison between two different algorithms



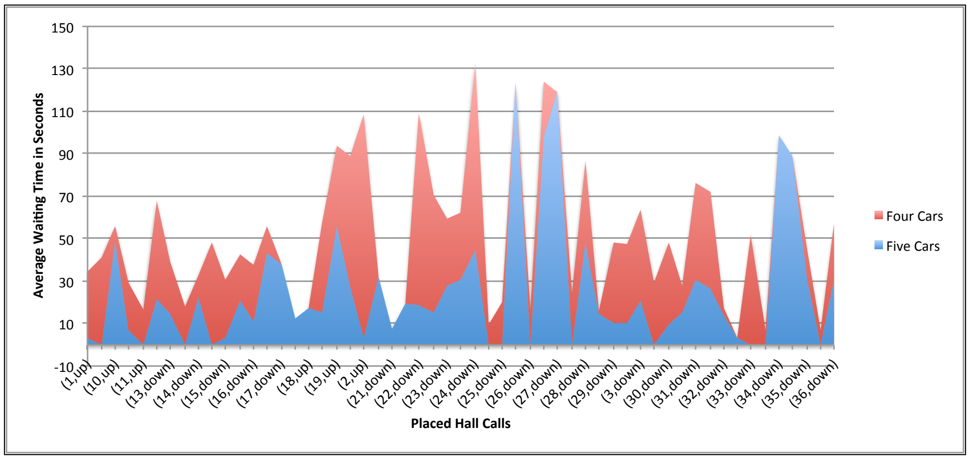
```
5}++
1`{hall_call=(34,down),waiting_time
=19.0}++
1`{hall_call=(35,up),waiting_time=1.
5}++
1`{hall_call=(36,up),waiting_time=10
.5}++
2`{hall_call=(36,down),waiting_time
=26.0}++
1`{hall_call=(37,up),waiting_time=21
.0}++
1`{hall_call=(37,up),waiting_time=26
.0}++
1`{hall_call=(37,down),waiting_time
=0.0}++
1`{hall_call=(37,down),waiting_time
=17.5}++
1`{hall_call=(38,up),waiting_time=79
.5}++
1`{hall_call=(39,down),waiting_time
=7.0}++
1`{hall_call=(40,down),waiting_time
=7.0}
```

(a) Results from adopting four cars



```
1`{hall_call=(2,up),waiting_time=0.0
}++
1`{hall_call=(2,down),waiting_time=
34.5}++
1`{hall_call=(3,up),waiting_time=60.
5}++
1`{hall_call=(3,up),waiting_time=65.
5}++
1`{hall_call=(3,down),waiting_time=
43.0}++
1`{hall_call=(4,up),waiting_time=12.
0}++
1`{hall_call=(4,down),waiting_time=
15.5}++
2`{hall_call=(4,down),waiting_time=
26.0}++
1`{hall_call=(5,up),waiting_time=3.5
}++
1`{hall_call=(5,up),waiting_time=27.
5}++
1`{hall_call=(6,up),waiting_time=7.0
}++
1`{hall_call=(6,up),waiting_time=14.
```

(b) Results from adopting five cars



(c) The chart of the comparison

Figure 6.12: The comparison between different numbers of cars

Chapter 7

Conclusion

This chapter concludes this thesis in section 7.1, and discuses future work in section 7.2.

7.1 Discussion

In this thesis, a fairly general CPN-based model of the elevator system is proposed. The proposed model has an abstract version that concerns substantially the logistics of moving cars between floors, as well as a timing version that supports more features such as the illumination of buttons and the calculation of waiting and serving times. The proposed model successfully fulfilments the objective of this thesis, which considers the constraints of elevator system giving by [Ghezzi *et al.* (2003)]. Additionally, The proposed model shows the expressive power and convenience of Coloured Petri Nets.

The proposed model includes four sub-models that cover different stages and functions of the elevator system in substantial detail. The flexibility of the proposed model allows the adoption of many different algorithms and different rules. In addition, the division of the sub-models allows easier tracking of errors and faults.

The sub-models are the car-stricture sub-model that models the elevator's cars, the hall-call sub-model that processes the production and the assignment of hall calls, the car-call sub-model that also produces and coordinates the car calls, and the system-cycle sub-model that models the operation of elevator's cars between floors.

Furthermore, an optimizer model for the parking system of the elevators is presented. The optimizer model proposes two techniques for selecting the most appropriate floors on which idle cars must be held. The main advantage of this optimizer model is reducing the waiting time of the passengers and consequentially increasing their stratification.

Finally, two analyses techniques were applied to test the proprieties of the proposed model, namely the reachability analysis through the state space tool, and the simulation-based performance analysis by means of the simulator tool included in the CPN Tools. Besides the analyses, a visual aid tool was developed to facility the convenient monitor of the system's behaviours.

7.2 Future Work

In this thesis, the proposed CPN-base model of the elevator system can be improved more through three possible areas.

- First, adopting more complex algorithms such as geriatric algorithms or algorithms based on fuzzy logic.

- Second, improving the analyses by resolving the state space explosion problem to accomplish a full reachability analysis of the model, applying other techniques of analyses, or both.

- Third, developing a Java-based extension that extracts and compares direct data from the proposed model with instance support of charts. This extension can be launched directly from CPN Tools as a third-party extensions.

Bibliography

Ahmad, F., Fakhir, I., Khan, S., and Khan, Y. (2014). Petri net-based modeling and control of the multi-elevator systems. In *Neural Computing and Applications*, volume 24, pages 1601–1612. Springer London.

AIS Group (2013). CPN Tools by The University of Technology, Eindhoven, The Netherlands. *www.cpntools.org*.

Barney, G. (2003a). *Elevator Traffic Handbook: Theory and Practice.* Taylor & Francis.

Barney, G. (2003b). Vertical transportation in tall buildings. *Elevator World*, **51**(5), 66–75.

Bolat, B. and Cortes, P. (2011). Genetic and tabu search approaches for optimizing the hall call-car allocation problem in elevator group systems. *Appl. Soft Comput. (Netherlands)*, **11**(2), 1792–800.

Brand, M. and Nikovski, D. (2004). Optimal parking in group elevator control. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 1002–1008.

Cho, Y. C., Gagov, Z., and Kwon, W.-H. (1999). Timed Petri net based approach for elevator group controls. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1265–1270.

Etessami, E. S. and Hura, G. S. (1989). Abstract Petri net based approach to problem solving in real time applications. *Fourth IEEE Region 10 International Conference*, pages 234–239.

Fernandes, J. M., Baek Jorgensen, J., and Tjell, S. (2007). Requirements Engineering for Reactive Systems: Coloured Petri Nets for an Elevator Controller. In *14th Asia-Pacific Software Engineering Conference*, pages 294–301.

George R. Strakosch, R. S. C. (2010). *Wiley: The Vertical Transportation Handbook*.

Ghezzi, C., Jazayeri, M., and Mandrioli, D., editors (2003). *Fundamentals of Software Engineering*. Pearson Prentice Hall, 2nd edition.

Huang, Y.-H. and Fu, L.-C. (1998). Dynamic scheduling of elevator systems over hybrid Petri net/rule modeling. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1805–1810.

Janicki, R. and Koutny, M. (1995). Semantics of inhibitor nets. *Information and Computation*, **123**(1), 1–16.

Jensen, K. (1981). Coloured Petri Nets and the Invariant Method. *Theoretical Computer Science*, **14**(3), 317–336.

Jensen, K. and Kristensen, L. M. (2009). Coloured Petri Nets Modelling and Validation of Concurrent Systems. Berlin. Springer.

Jensen, K., Christensen, S., and Kristensen, L. M. (2006). CPN Tools State Space Manual. *Department of Computer Science, Univerisity of Aarhus*.

Jørgensen, J. B. K. (2008). Coloured Petri nets and graphical animation: a proposal for a means to address problem frame concerns. *Expert Systems*, **25**(1), 54–73.

K. Jensen (1994). *Coloured Petri Nets*. Springer.

Kim, J.-H. and Moon, B.-R. (2001). Adaptive elevator group control with cameras. In *IEEE Transactions on Industrial Electronics*, volume 48, pages 377–382.

Kindler, E. and Páles, C. (2004). 3D-visualization of Petri net models: Concept and realization. In *Applications and Theory of Petri Nets*, pages 464–473. Springer.

Lin, C.-H. and Fu, L.-C. (1996). Petri net based dynamic scheduling of an elevator system. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 192–199.

Liqian, D., Qun, Z., and Lijian, W. (2004). Modeling and analysis of elevator system based on timed-coloured Petri net. In *Fifth World Congress on Intelligent Control and Automation*, volume 1, pages 226–230.

Liu, H., Qian, Y.-L., Liu, Q., and Li, J.-T. (2008). Count passengers based on Haar-like feature in elevator application. In *2008 International Conference on Machine Learning and Cybernetics*, volume vol.2, page 1202, Piscataway, NJ, USA.

Liu, Y., Hu, Z., Su, Q., and Huo, J. (2010). Energy saving of elevator group control based on optimal zoning strategy with interfloor traffic. In *3rd International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII*, volume 3, pages 328–331, Kunming, China.

Munoz, D. M., Llanos, C. H., Ayala-Rincon, M., and van Els, R. H. (2008). Distributed approach to group control of elevator systems using fuzzy logic and FPGA implementation of dispatching algorithms. *Engineering Applications of Artificial Intelligence*, **21**(8), 1309–1320.

Ramadge, P. J. and Wonham, W. M. (1989). The control of discrete event systems. In *Proceedings of the IEEE 77.1*, pages 81–98.

Reisig, W. (1991). Petri nets, an introduction, 2nd ed. Berlin. Springer Berlin Heidelberg.

Siikonen, M.-L. and Hakonen, H. (2003). Efficient evacuation methods in tall buildings. *Elevator World*, **51**(7), 78–83.

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

van der Aalst, W. M. P., Stahl, C., and Westergaard, M. (2013). Strategies for modeling complex processes using colored petri nets. In *Transactions on Petri Nets and Other Models of Concurrency VII*, pages 6–55. Springer.

Westergaard, M. (2006). The BRITNeY Suite: A Platform for Experiments. In *7th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN 2006)*.

Westergaard, M. (2013). CPN tools 4: multi-formalism and extensibility. In *Application and Theory of Petri Nets and Concurrency*, pages 400–409. Springer.

Xu, Y., Luo, F., and Lin, X. (2010). Hybrid destination registration elevator group control system with artificial immune optimization algorithm. In *2010 8th World*

Congress on Intelligent Control and Automation (WCICA 2010), pages 5067–5071, Piscataway, NJ, USA.

Ye, J., Li, J., Deng, F., and Wang, C. (2011). Simulation of the intelligent control circuit based on Petri net. In *6th International Conference on Computer Science & Education*, pages 66–69.

Zheng, L., Guang, S., and Hui, D. (2013). Research of elevator group scheduling system based on reinforcement learning algorithm. In *International Conference on Measurement, Information and Control (ICMIC)*, volume 1, pages 606–610.