Control Engineering Practice 20 (2012) 1183–1194

Contents lists available at SciVerse ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

A modular Petri net to modeling and scenario analysis of a network of road traffic signals

Michel dos Santos Soares [a,*], Jos Vrancken [b]

[a] Universidade Federal de Uberlândia, P.O. Box 593, 38400-902 Uberlândia, Brazil
[b] Delft University of Technology, P.O. Box 5015, NL 2600 GA, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 22 September 2011
Accepted 16 June 2012
Available online 11 July 2012

Keywords:
Petri nets
Traffic signals control
Linear logic
Formal analysis
Real time systems

ABSTRACT

The dynamic behavior of a group of traffic signals controlling a network of intersections is a complex discrete event system that can be modeled by Petri nets. The approach used in this paper proposes a components-based design, which increases modularity, reduces complexity and is a good practice according to modern Systems Engineering. The main system elements are specified based on the proposed Petri net component with time intervals associated to places. The specified models are simulated through the common token player algorithm, and formal analysis using invariants and theorem proving are applied to verify models' soundness and to reason on specific scenarios.

1. Introduction

When designing distributed real-time systems for critical infrastructures, such as transportation and energy networks, system complexity is increased due to the large number of elements, strict real-time constraints, and reliability factors, as these systems involve human life. Therefore, it is necessary to use methods that are capable of addressing complexity and providing highly reliable solutions. Formal methods fulfill these requirements, by allowing formal proofs of desirable behavior and validation of performance characteristics by simulations. In addition, if a formal method features modular design and abstraction, then this contributes substantially to complexity reduction.

A number of approaches have been used to model distributed real-time systems in general, and traffic control systems in particular. Finite State Machines (FSMs) have been used for modeling distributed real-time systems (Cassandras & Lafortune, 2006), but have the shortcoming of state explosion (Brave, 1993). Statecharts (Harel, 1987) extends state-machines by endowing them with orthogonality, depth, and synchronization. An approach to model urban traffic signal control using Statecharts was proposed in Huang (2006).

Petri nets (Murata, 1989) are used in this paper. The choice is based on their modeling expressivity, their facilities for validation and verification activities, the availability of computer tools, and the range of available extensions. Petri nets are suitable to model distributed real-time systems because they offer representation of important situations commonly found in this kind of systems, including conflict situations, shared resources, synchronous and asynchronous communication, and precedence constraints. As a formal language, Petri nets allow formal checks of desirable properties. The formal analysis of a Petri net model can reveal design flaws (Girault & Valk, 2003). For instance, with reachability analysis it is possible to find out whether an unsafe state that could cause an accident can be reached. The extensions proposed in this paper to the basic Petri net model are useful to address complexity and model real-time constraints. Finally, there are a variety of computer-based tools that support the Petri net theory, by performing verification and simulation of models.

1.1. Literature review

Petri nets are a well-known formalism applied in traffic control research, including air traffic control (Huang & Chung, 2011) and railway traffic control (Cheng & Yang, 2009). The core of this research is the application of Petri nets to road traffic control. With focus on only one intersection, Huang, Chung, and ChenSystems (2005) described the traffic signal control using timed color Petri nets. Soares and Vrancken (2007b) proposed an approach based on the application of Petri nets and theorem proving to formally analyze the properties of traffic signals controlling one road intersection. These works are based on Petri net models without explicitly adding time for each phase duration. Considering networks of roads, one of the first applications of Petri nets for traffic signals control in urban networks was

* Corresponding author. Fax: +55 34 3239 4392.
E-mail addresses: michel@facom.ufu.br, mics.soares@gmail.com (M. dos Santos Soares).

0967-0661/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.conengprac.2012.06.005

shown in DiCesare, Kulp, Gile, and List (1994), in which Petri net models were used to do performance analysis and code generation. List and Cetin (2004) used Petri nets to model the control of signalized intersections, and evaluated the good properties of the system by means of invariant analysis and simulation. A modular framework based on Coloured Petri nets to model the dynamics of signalized traffic network systems was presented in Dotoli and Fanti (2006). In this model, the phase durations are fixed by a deterministic value attached to each transition. Examples using a methodology in which modularity is explored are given in Huang and Chung (2008), in which all the traffic operations are ruled by the control logic of Coloured Petri nets. Other examples of works where time is added to represent phase duration are given in Dotoli, Fanti, Mangini, Stecco, and Ukovich (2010), Febbraro and Giglio (2006), Tolba, Lefebvre, Thomas, and Moudnia (2006) . In these works, the durations are based on fixed, pre-determined intervals for each color phase, represented in timed Petri nets (fixed time associated to transitions or places).

This paper proposes to represent phases such as green time using an interval of minimal and maximal values. This is done by using the p-time Petri nets extension, in which an interval is associated to each place. Some works propose a control logic with performance purposes (Febbraro, Giglio, & Sacco, 2004; Tolba et al., 2006) or validation using simulation (Basile, Chiacchio, & Teta, 2012), which are not the focus of this paper. However, in many of the previously cited works, verification of the control logic is poor or nonexistent. This paper proposes the use of various Petri net analysis methods to verify the soundness of models. There is a need to do formal verification of a proposed control logic before implementation in order to discover design flaws, such as deadlocks or the possibility of reaching unsafe states. This is an important activity that is often neglected, in spite of the Petri nets facilities. Errors in initial modeling phases of system development can affect the development costs and duration. Therefore, special attention should be paid to the correctness of models used early in the development process.

1.2. Approach

A group of traffic signals controlling a road network is a complex system in the sense that it has several elements involved, and is applied to a critical infrastructure. The design must address performance and safety, as human life is directly involved and transportation infrastructure is fundamental to the economy of a country. In this paper, a responsive traffic control strategy based on Petri nets with time intervals associated to places is proposed (Section 2). The approach is not based on a single intersection, but on networks of intersections controlled by networks of traffic signals (Section 3). This increases the problem complexity, which calls for a well-defined Systems Engineering process of requirements specification using UML Use Cases. Analysis of properties and specific scenarios are presented in Section 4. The assumption is that before the implementation phase, it is imperative to perform formal analysis of the controllers' model to provide warranties that the proposed control logic is safe. The paper ends with a conclusion and proposals for future work (Section 5).

2. Petri nets and extensions

The formal definition of a Petri net (Murata, 1989) is given as follows:

Definition 2.1. A *Petri net* is a 5-tuple $N=(P,T,Pre,Post,M_0)$, where:

$P=\{p_1,p_2,\ldots,p_n\}$ is a finite set of places;

$T=\{t_1,t_2,\ldots,t_m\}$ is a finite set of transitions;

$Pre:(P\times T)\to\mathbb{N}$ is an input function that defines directed arcs from places to transitions, and $\mathbb{N}$ is the set of nonnegative integers;

$Post:(P\times T)\to\mathbb{N}$ is an output function that defines directed arcs from transitions to places; and

$M_0:P\to\mathbb{N}$ is the initial marking.

If $Pre(p,t)=0$, then there are no directed arcs connecting p to t. In a similar manner, if $Post(p,t)=0$, there are no directed arcs connecting t to p. If $Pre(p,t)=k$ (the same holds for $Post(p,t)=k$), then there exist k parallel arcs connecting place p to transition t (transition t to place p). Graphically, parallel arcs are represented by a simple arc with a number k as label and representing the presence of k multiple arcs (this is also called the *weight* k of an arc).

The dynamic behavior (execution) is described by changing markings (from M to M') when an event occurs, which is represented by the firing of an enabled transition. A transition t is enabled if each place P that has an input arc to t contains at least the number of tokens equal to the weight of the arcs connecting P to t ($M(p)\geq Pre(p,t)$ for any $p\in P$). The flow of tokens represents state changing. The firing of an enabled transition t removes from each input place p_i the number of tokens equal to the weight of the directed arc connecting p_i to t, and deposits in each output place p_o the number of tokens equal to the weight of the directed arc connecting t to p_o. Therefore, the firing of a transition yields a new marking $M'(p)=M(p)-Pre(p,t)+Post(p,t)$ for any $p\in P$.

2.1. Petri nets extensions

In order to improve Petri nets applicability to systems, extensions to the basic theory were proposed. In this paper, the Petri net formalism is extended by associating time intervals to places and by defining a component whose behavior is based on Petri nets. The extensions are explained in the following subsections.

2.1.1. P-time Petri nets

The p-time Petri net (Khansa, Denat, & Collart-Dutilleul, 1996) model used in this paper adds a time interval to each place, as expressed by the following definition.

Definition 2.2. A *p-time Petri net* is a pair (N,Ip) where N is a Petri net and Ip is the application defined as $Ip:P\to(\mathbb{Q}^+\cup 0)\times(\mathbb{Q}^+\cup\infty)$.

With each place $p_i\in P$ an interval $Ip_i=[\alpha_i,\beta_i]$ is associated, with $0\leq\alpha_i\leq\beta_i$. The static interval Ip_i expresses the time interval during which a token in p_i is available. Before α_i, the token in p_i is in the non-available state. After α_i and before β_i, the token in p_i is in the available state for transition firing. After β_i, the token in p_i is again in the non-available state and cannot enable any transition anymore. This token is called "dead", and can be seen as a time constraint that was not respected. The dynamic evolution of a p-time Petri net model depends on the marking M and on the availability of tokens (non-available, available, or dead). The definitions of a visibility interval and enabling transition for p-time Petri nets are given as follows.

Definition 2.3. A visibility interval $[(\delta_p)min,(\delta_p)max]$ associated with a token of a place p of a p-time Petri net defines:

- *the earliest date* $(\delta_p)min$ when the token in p becomes available for the firing of an output transition of p;
- *the latest date* $(\delta_p)max$ after which the token becomes non-available (dead) and cannot be used for the firing of any transition.

Definition 2.4. If a transition has n input places and if each of these places has several tokens in it, then the *enabling interval* $[(\delta_p)min,(\delta_p)max]$ of this transition is obtained by choosing for each one of these n input places a token, the visibility interval associated to this token, and taking the intersection of all the obtained visibility intervals.

2.1.2. Petri net component

Large systems are composed of smaller parts, named components, that perform more specific functionalities. With the composition of several components to form the complete system, the complexity is increased and new properties emerge that do not belong to any specific component. For instance, performance is related more to a group of components than responsibility of just one component. Therefore, it is necessary to analyze their composition.

The component proposed in this paper is composed of behavior and interface. The behavior is specified by Petri nets. The component receives and sends messages through the interface. The interface is composed of input and output ports, that are specific places of the Petri net. Components share only their interfaces. The behavior is not known for other components. This characteristic of a component improves flexibility to perform changes without the need to alter the entire system. The definitions of components and interfaces used in this paper are given as follows.

Definition 2.5. A *Petri net Component* (*PNC*) is a tuple PNC = (*N*,*IP*,*OP*), where N is a Petri net, P is the set of places, $IP = \{Ip_1, Ip_2, \ldots, Ip_n\} \subset P$ is the set of input ports, and $OP = \{Op_1, Op_2, \ldots, Op_n\} \subset P$ is the set of output ports.

Definition 2.6. An *Interface of a Petri net Component* (*IPNC*) is the union of the sets *IP and OP*: $IPNC = OP \cup IP$.

Graphically, a component is a box with internal behavior specified by a Petri net and with communication places positioned on the border (Fig. 1). A PNC receives data from input places (IP's), performs the specified processing, and then sends processed data to output places (OP's). Interface arcs are a type of arc. They connect the outside world with input places of the PNC, and the PNC with the outside world.

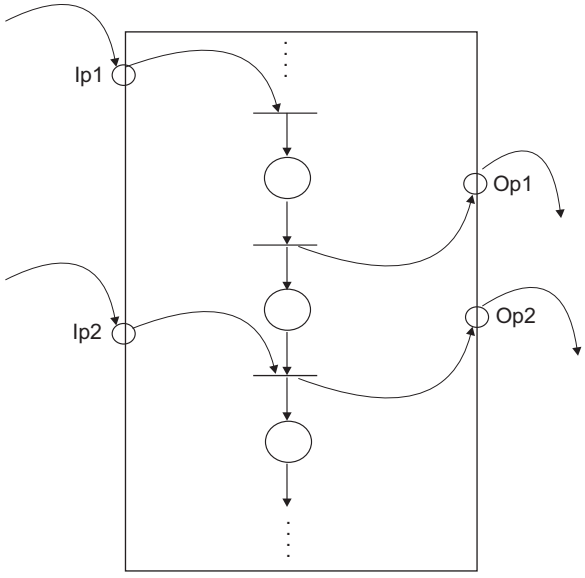

Fig. 1. Petri net component.

2.2. Design strategies using Petri nets

In order to address software-intensive systems' complexity, their design must be performed in a modular way, in which parts of the system, the subsystems, and even parts of the subsystems, can be specified and analyzed separately. Two strategies are most common for the modular design of Petri net models: composition, also called bottom-up, and refinement, also called top-down. The modularity in this paper is relative only to places, both for refinement and composition, as in Vogler (1992, Chapter 4).

2.2.1. Top-down modeling

The top-down modeling strategy decomposes large systems into subsystems. It begins with an aggregate model of the system, which is then refined progressively to introduce more details (Zhou, Dicesare, & Rudolph, 1992). This divide and conquer strategy is well-known in software and systems engineering and is followed in order to address system complexity.

There are several advantages to use refinement and therefore to start with a more abstract model (Choppy, Mayero, & Petrucci, 2008). It gives a better and structured view of the system to be designed by identifying the components within the system. The validation process also becomes easier: the system properties are checked at each refinement step. Thus, abstract models are validated before new details are added. Refinement helps in coping with system complexity since it may preserve some properties or analysis results obtained at an earlier step for a more abstract model.

In a Petri net, one place may in fact represent a subnet with many places and transitions. By refining one place, a set of places can be created, resulting in an expanded Petri net. Fig. 2 depicts the refinement of place PA into places PA1, PA2, and PA3. This refinement is useful for detailing initial design, from a high-level Petri net to a complete model (Vogler, 1992, Chapter 4).

2.2.2. Bottom-up modeling

A bottom-up modeling strategy usually starts by building simple models representing parts of the system, and later combining them into more complex ones until the whole model is built. For instance, places of two different Petri nets are merged (place composition), which creates one single Petri net (Huang & Kirchner, 2009). Composition is applied in order to interconnect Petri net models, based upon the concept of place fusion. The fusion of common places representing the same resource in two different Petri nets is a simple and effective way to model communication between subnets (Girault & Valk, 2003). It also improves the overall design by combining common places in different nets.

In Fig. 3, initially there are two places R, each one representing the same resource that is used by two different processes modeled by two Petri nets. By merging the places R, the two Petri nets are merged into one, in which R is a shared resource.

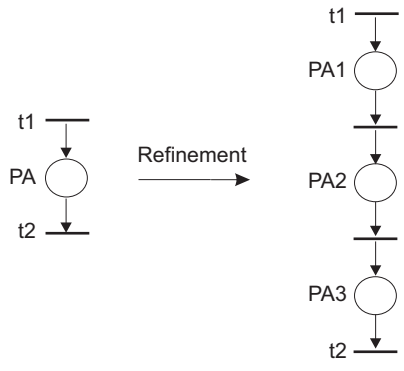

Fig. 2. Example of top-down design.

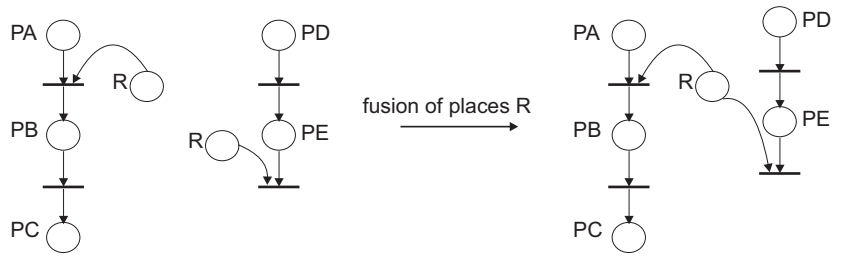

Fig. 3. Example of bottom-up design.

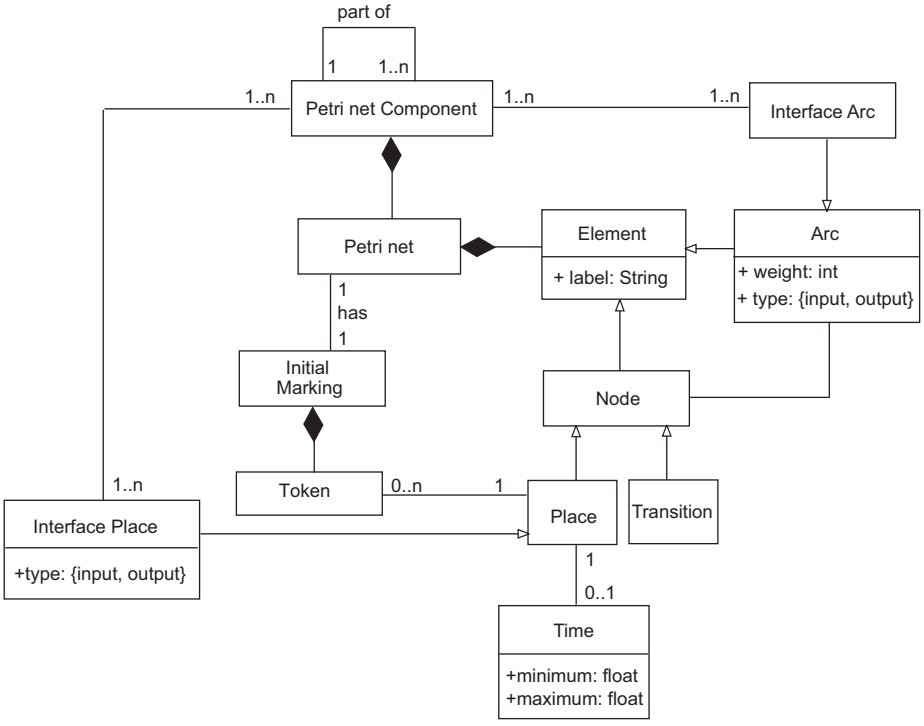

Fig. 4. Petri net metamodel.

2.3. Petri net metamodel

As a result of the extensions, the Petri net metamodel (Soares, 2011) used in this paper is depicted in Fig. 4. The metamodel does not represent the execution of a Petri net. It represents the basic elements of the Petri net model used in this paper at a specific situation, which involves tokens and markings.

3. Design of a network of traffic signals using Petri net components

In order to achieve better results, networks of roads must be evaluated, and problems such as traffic jams solved, not only locally, but in a wide area (Soares & Vrancken, 2007a). Otherwise, the risk is to shift congestion from one place to another in the network. Fig. 5 shows a network of traffic signals controlling a network of road segments. The road network is composed of local roads and arterial roads. Reasoning on networks increases system complexity. The approach used in this paper is based on modeling and analyzing complex systems by hierarchical decomposition into smaller components (divide and conquer) (Rouse, 2003). Thus, the network is divided into subnetworks.

3.1. Basic notions of a network of traffic signals

A road intersection (junction) can be defined as the general area where two or more roads join or cross, including the roadway and roadside facilities for traffic management (Khisty & Lall, 2003). Traffic signals are an important mechanism applied to solve intersections conflict and regulate traffic flow. To correctly control the intersection, traffic signal design must take care of safety and security rules, and when possible, of efficiency and performance. A traffic signal cycle is any complete sequence of signals, from green to yellow and red, and back to green, for all roads that cross at the specific intersection; its duration is called cycle time. When a network of intersections is considered, it is possible to provide green-waves for main roads. The application of a responsive traffic signal strategy in a network increases systems' complexity. The idea is to provide green time to the maximum number of vehicles in a sequence of intersections, such that they can cross without stopping. This is possible based on adjusted offsets (Roess, Prassas, & McShane, 2004) (difference between green initiation times in a network, considering intersections in sequence).

Another classification is to consider that traffic controllers may operate in the pre-timed (fixed-time) (e.g. TRANSYT; Robertson, 1969) or responsive (actuated) mode (e.g. SCOOT; Hunt, Robertson, Bretherton, & Winton, 1981). Within fixed-timed controllers, the phase duration for each road section is determined off-line based on

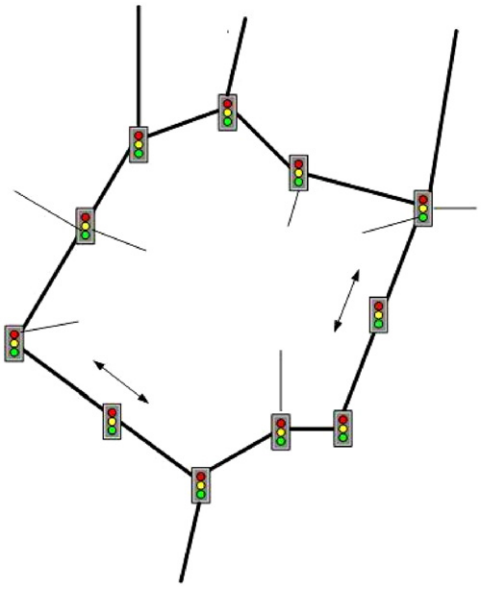

Fig. 5. Network of traffic signals.

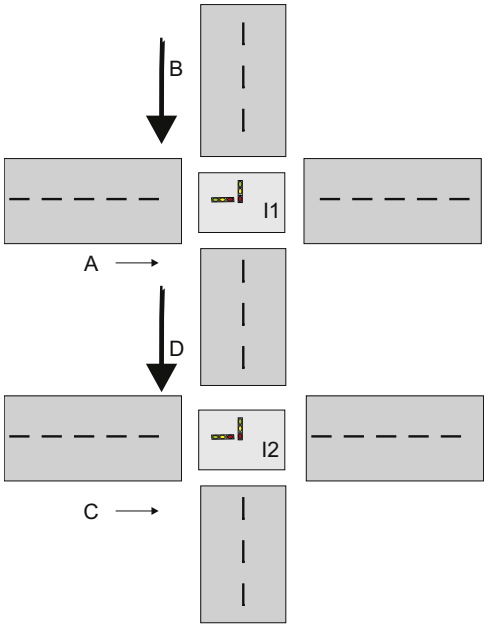

Fig. 6. An example of a subnetwork.

historical data. This simplifies the implementation of the control strategy, but has drawbacks. The demand may vary depending on the time of day and on special events (accidents, sports events, weather condition). The demand may also change in the long term, leading to "aging" of the optimized settings. In big cities with hundreds of intersections controlled by traffic signals, the effort to keep the system up to date is tremendous. Traffic-responsive strategies are more efficient for road traffic control (Lee, Tseng, & Shieh, 2010), as they use real-time measurements from detectors (for instance, inductive loops) that provide data to the controller with real demand information. They can change the length of the green phase (green time) for a road section depending on the demand, from a minimum to a maximum. An example of a traffic-responsive application is to verify if no vehicle passes the detectors during the minimum green specified. In this case, the controller switches to the next phase. Another example is the possibility of extending the green time by a minimum value while vehicles are being detected. This is done until the maximum green is reached.

Traffic-responsive approaches are more costly, as they require the installation, operation and maintenance of a complex real-time control system consisting of software, communication lines and sensors. However, they are cost-effective over long time intervals (5–10 years), due to their ability to adapt to changes in traffic-flow patterns (Klein, 2001).

Every two intersections are considered separately in this paper. They can be considered as building-blocks for larger networks, in the sense that they can be modeled and analyzed in pairs (Soares & Vrancken, 2007c). Fig. 6 shows a subnetwork of roads with two intersections: I1 and I2. The main road has a great flow from B to D, which needs to be improved. A and C are non-priority roads. There are sensors on the main road (from B to D) to detect a platoon of vehicles (groups of closely spaced vehicles) after the green time starts for road B in intersection I1 and allow green time in intersection I2. This coordinated signal control tries to allow that platoons of vehicles can proceed through a continuous series of green lights without stoping. This ideal offset is difficult to achieve when pre-determined offsets are defined. The risk is that if a platoon of vehicles goes slower or faster than expected, the result will inevitably be sub-optimal.

Fig. 7 shows the elements of the system architecture and its relationship with the environment. All elements (controller, traffic signals, detector) are specified using Petri nets Components, which increases modularity. Detectors send information about the traffic state to the controller. The Controller sends signals to the traffic signals respecting the requirements established in Section 3.2, and the traffic signals regulate traffic.

3.2. System requirements

The requirements specified in this section are modeled through a context diagram with Use Cases (Fig. 9).

3.2.1. Requirements for traffic signals control in a network

Some important requirements (non-exhaustive) for the proper functioning of traffic signals are (Gallego, Farges, & Henry, 1996):

- The traffic signal must not allow the green state to two conflicting road sections simultaneously.
- Each traffic signal must follow a defined sequence of active color lights, normally from green to yellow and red, and then back to green.
- The right to use an intersection has to be given to all road sections.

When reasoning on a network of intersections, three additional requirements for traffic signals modeling are important:

- Any user in the intersection should not wait for more than a maximum service delay, otherwise the user may presume that the traffic signal is not functioning, which can lead to non-secure decisions by the users (red light violations), or the formation of big queues.
- The offset is ideally designed in such a way that as the first vehicle just arrives at the next intersection in the network, the signal controlling this intersection turns green.
- The length of green time for each road section can be different depending on the section priority, for instance. This length can be extended to improve traffic flow to a road section depending on real-time demand.

In order to be effective, advanced signal control systems require an accurate current picture of the traffic flow and status on the

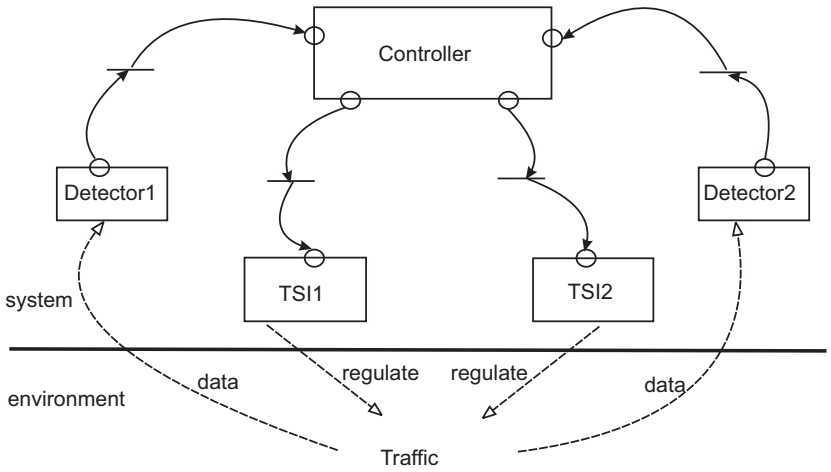

Fig. 7. System architecture.

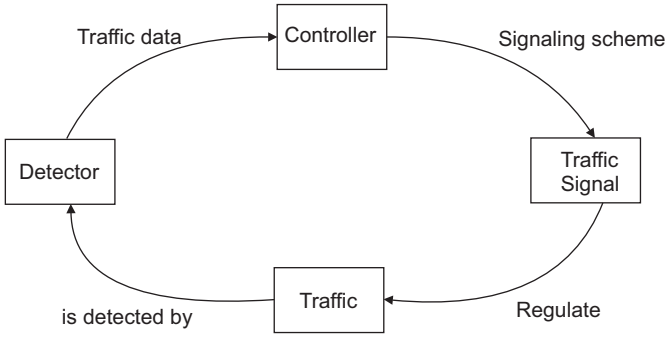

Fig. 8. Traffic scheme.

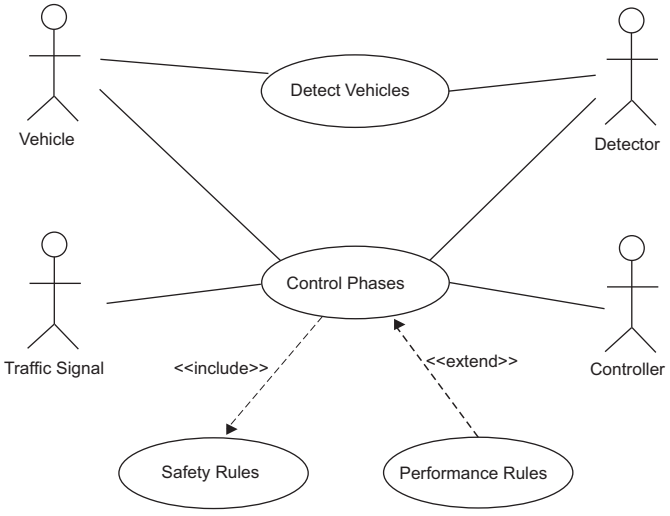

Fig. 9. Context diagram.

roadway network. Actuated control uses information about current demand obtained from sensors within the intersection, to alter one or more aspects of the signal timing (Roess et al., 2004). Responsive controllers may be programmed to accommodate variable green times or phase sequences. This is an advantage of actuated traffic signal control over pre-timed traffic signal control. Responsive controllers are equipped to receive data of traffic flow patterns from various measuring devices (detectors) at preset time intervals. The controller sends the signalling scheme to traffic signals, which influences traffic flow (Fig. 8). The behavior of the sensors, the controller and the traffic signal are specified with Petri net components, as shown in Section 4.

In this paper, a sequence of intersections along a road is controlled as a group. Phase durations are adapted from a minimum to a maximum time, given in seconds. This design is done using the proposed Petri nets with time extension. Thus, the effective green can vary depending on the traffic demand detected by sensors. Another requirement is relative to the offset. Considering two intersections in sequence, each one controlled by a traffic signal (TS), after the green phase is given to a section and the first vehicle is detected, the green phase for the next intersection starts after a calculated delay depending on the first vehicle detected.

3.2.2. Context diagram

Fig. 9 depicts a context diagram for traffic signals controlling a network of intersections and considering the requirements and rules of the previous subsections. The Use Case "Control Phases" depends on actors Vehicle and Detector. Data detected by detectors (presence of vehicles) are sent to the controller. Actor Controller is responsible to select a feasible signalling scheme to be sent to actor Traffic Signal. In order to correctly control the network, safety rules must be applied (*include* relationship). Whenever possible, which means that safety rules will not be violated, performance rules are also considered (*extend* relationship). The dynamic behavior of each Use Case is designed with Petri nets in this paper. In the following sections, the application of Petri nets in the modeling phase is proposed, followed by Petri nets methods for analysis of model properties.

3.3. Basic controller–traffic signal communication

Fig. 10 shows the controller and the traffic signal component. They communicate with each other through the components interface. The Controller sends control signals to the traffic signal, indicated by places such as "A_startGreen", which request the traffic signal to start a green phase for the corresponding road section. After the operation is finished by the traffic signal, a response is sent back by the traffic signal to the Controller ("A_RespG", for instance). After that, the controller can proceed with its processing.

A composed Petri net is generated by place fusion of smaller nets (bottom-up modeling). Combining nets by place fusion is a

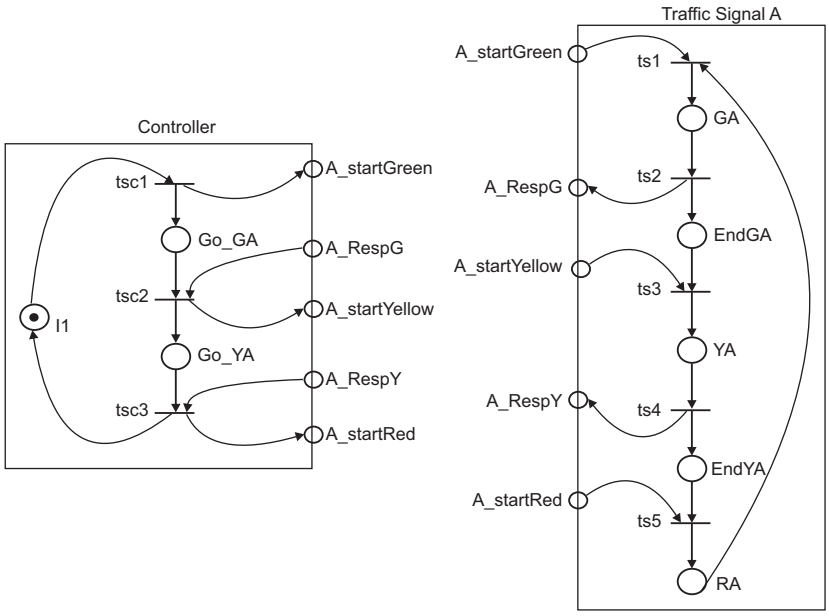

Fig. 10. Basic controller and traffic signal components.

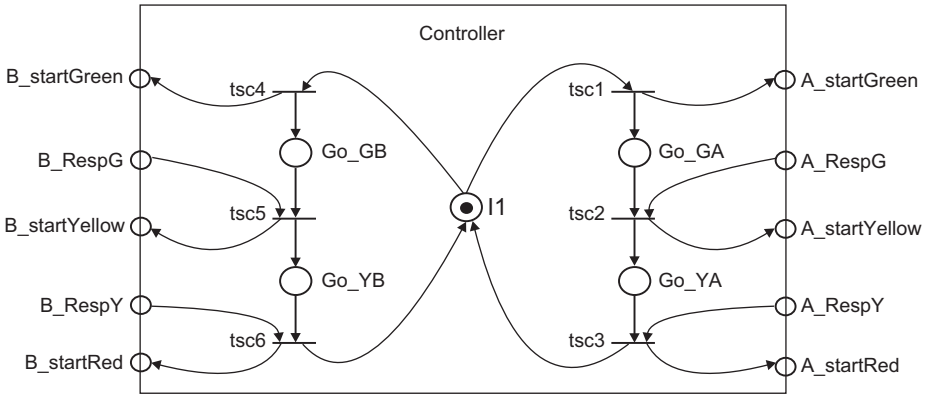

Fig. 11. Controller model for road section I1.

simple and effective way to model communication between Petri net components. Places with the same label, such as "A_startGreen" in both controller and traffic signal components, are fused in order to generate one single component.

3.4. Controller model

Intersections I1 and I2 are shared resources for the road sections, and can be represented in the Petri net model as shared places. Traffic signal phases are represented as places and the transitions separate one phase from another. For instance, GA represents a green for section A, YA a yellow and RA a red. The firing of a transition allows phase changing. A token in a place represents the current phase of the traffic signal.

Fig. 11 depicts the controller model for controlling the traffic signal of intersection I1. In the initial state, both transitions *tsc*1 and *tsc*4 are enabled. The firing of these transitions allows respectively the green phase for road section A and then for road Section B. Supposing that *tsc*1 fires first, it will deposit a token in place "Go_GA", indicating that the controller is commanding the traffic signal of road section A to switch to green. The firing of *tsc*1 also deposits a token in the interface place "A_startGreen", which indicates to the traffic signal of road section A to start the green phase.

3.5. Global net

Fig. 12 represents the global Petri net specification concerning intersection I2, after place fusion of the controller and the traffic signals. A similar design is used for the controller and the traffic signal concerning intersection I1. The detector component is represented as a black box (its internal behavior is abstracted as it is not relevant to the design). Its interface is composed of place VD (vehicle detected), indicating that a vehicle was detected and this information is sent to the controller, which may respond using the place RVD.

To each place of the traffic signals a minimum and a maximum values are associated, which indicates the duration a token must remain before being used to fire the transition. In practice, this is the duration of each phase of the traffic signal.

3.6. Offset mechanism

Coordinated traffic signals are very useful in road networks where traffic signals are located in close proximity. The sequence of green phase is done in such a way that the number of stops at intersections is minimized. However, coordination has some limitations. There are no guarantees that all vehicles in a platoon will be given green in the network at the required time. For long

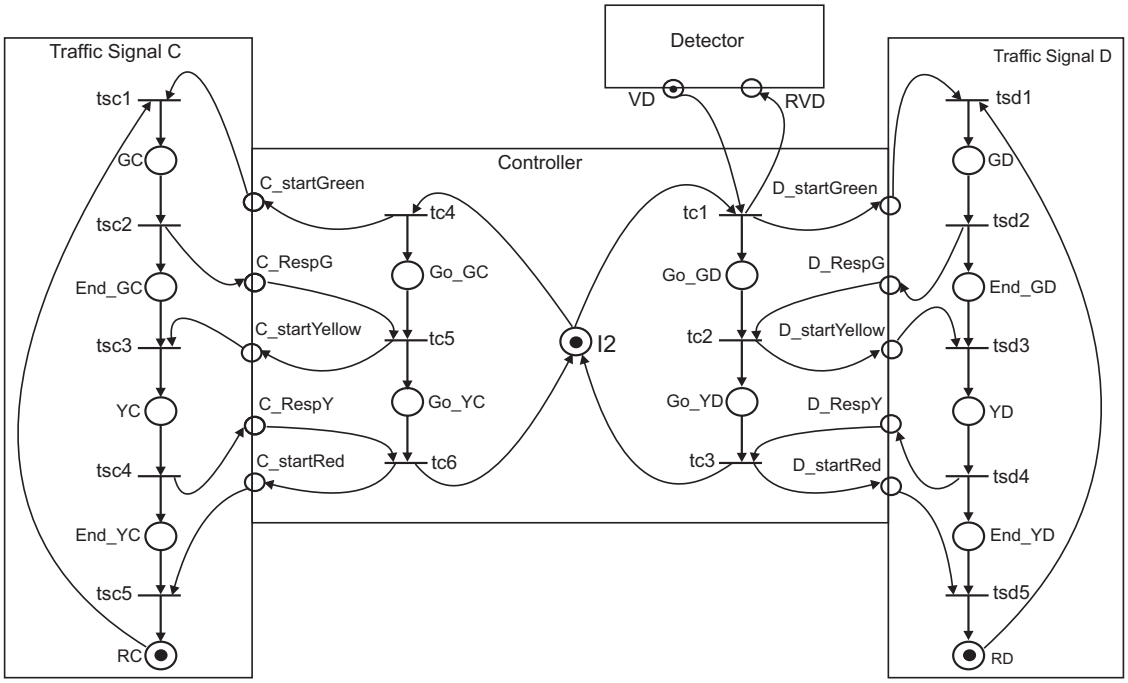

Fig. 12. Combined Petri nets.

sections, the effect may not provide the expected result due to several factors, such as different vehicle speeds, traffic already congested and differences in phases of the subsequent traffic signals. These factors may limit the improvement in traffic flow. Another problem that is often neglected is that the success in coordinating traffic signals in main roads may lead to traffic jams in subsidiary roads (when vehicles wait too long for their green time). This is avoided by limiting the green phase using the *p*-time Petri nets intervals, from a minimum to a maximum.

The design proposed makes possible the occurrence of green waves through the offset mechanism. Ideally, the offset is designed in such a way that as the first vehicle just arrives at the next intersection in the network, the signal controlling this intersection turns green. This ideal offset is difficult to achieve when pre-determined offsets are defined. The aim of the offset mechanism considered in this paper is to facilitate that a platoon of vehicles leaving from section B receives green time when reaching section D, improving traffic flow from intersection I1 to I2.

4. Analysis of models

It is imperative that the proposed control logic is verified before implementation. The Petri net theory allows several different analysis methods to be used in order to verify that desirable properties are present in the models. Detailed explanation about the properties and analysis methods can be found in Murata (1989) and Girault and Valk (2003). The analysis in this research is done using simulation, invariant analysis, reachability graphs, and theorem proving for specific scenarios. The results presented are relative to an extended version of the net of Fig. 12, which includes time intervals attached to places representing traffic signals' phases. Several simulations were performed with different controlling times for the phases. The main purpose was to check inconsistencies through common token player simulation. After the simulations, formal analysis was performed in order to verify the presence of desirable properties (liveness, boundedness and reachability).

4.1. Structural analysis

By examining the linear equations based on the execution rule of Petri nets and the matrices *Pre* and *Post* it is possible to find subsets of places over which the sum of tokens remains unchanged. An integer solution y of the transported homogeneous equation $C \times y = 0$ is called a P-invariant. The non-zero entries in a P-invariant represent weights associated with the corresponding places so that the weighted sum of tokens of these places is constant for all markings reachable from the initial marking.

Definition 4.1.1. For a Petri net N with m places and n transitions, the Incident Matrix $C = [C_{ij}]$ is an $m \times n$ matrix of integers and its typical entry is given by $C = Post - Pre$.

P-invariant analysis is applied to verify safety rules. The Petri net of Fig. 12 is *P*-invariant. There are 36 subsets of places over which the sum of tokens remains unchanged. For instance, the set

{*End_GD*,*End_YD*,*GD*,*RD*,*YD*},

represents that there will be a token in End_GD or End_YD or GD or RD or YD. The sum of tokens in this set is always 1. As a matter of fact, it is not possible to achieve simultaneous green and yellow for road section D. There is an equivalent set for road section *C*.

Another interesting set is {Go_GC, Go_GD, Go_YC, Go_YD, I2}. The interpretation is that the controller will not allow simultaneous green or yellow for conflicting road sections, which fulfills an important safety requirement.

4.2. Reachability analysis

Reasoning about reachable states allows one to decide, for instance, if an unsafe state may occur. In order to find out whether the modeled system can reach a specific state in a Petri net, it is necessary to find a sequence of transition firings that transforms a marking M_i to M_f, where M_f represents a specific state. A marking M_f is said to be reachable from a marking M_i if there is a sequence of transition firings that transforms M_i to M_f. The set of all possible markings reachable from M_i in a Petri net N is denoted by $R(N,M_i)$. The reachability problem for Petri nets is

the problem of finding if a marking is reachable (belongs to the set of possible markings) from a given marking, i.e., if $M_f \in R(N,M_i)$ for a given initial marking M_i.

Reachability analysis was applied to the Petri net of Fig. 12, but without considering the detector interface, which would characterize a specific scenario to be analyzed in the following subsection. The analysis resulted in 32 states, from state $S0=\{$I2 RC RD$\}$ to state $S31=\{$C_startRed End_YC I2 RD$\}$. The result is that the Petri net is alive and bounded.

The reachability graph allows the verification of some desirable properties. In addition, it can demonstrate that some unsafe states are never reached. Nevertheless, the reachability analysis presents some disadvantages. The main one is that its construction is a very hard problem from a computational point of view. This is because the size of the state-space may grow faster than exponentially with respect to the size of the Petri net model. Another problem is that it is not possible to extract the causality links between the different firings, which is fundamental in scenario deriving (Demmou, Khalfaoui, Guilhem, & Valette, 2004). Finally, the reachability graph presents some difficulties in analyzing properties in which concurrency plays a fundamental role (Girault & Valk, 2003).

4.3. Scenario analysis with linear logic

Linear Logic was proposed by Girard (1987) as a refinement of traditional logic in order to deal with resources. In traditional logic, a proposition can be used as many times as one wants. For instance, if a fact A is used to conclude a fact B, the fact A is still available after being used. Within Linear Logic, propositions are resources that can be produced and consumed. It is natural that resources can be counted, but traditional logic has no easy means to do that, due to two structural rules: weakening and contraction. As Petri nets deal correctly with the notion of resource, some results appeared on combining Petri nets and Linear Logic (Brown, 1989; Engberg & Winskel, 1990).

Linear Logic introduces three sets of connectives:

(1) Multiplicatives: $\otimes$ (“times”), $\parr$ (“par”), and $\multimap$ (linear implication).
(2) Additives: & (“with”), $\oplus$ (“plus”).
(3) Exponentials: ! (“of course”) and ? (“why not”).

The negation in Linear Logic, denoted $\perp$, does not express truth/falsehood properties, but rather concepts such as consumption and production of resources. Only the multiplicative connectives $\otimes$ (times) and $\multimap$ (linear implication) are used in this paper. As shown previously (Gehlot & Gunter, 1989), they are sufficient to represent, respectively, Petri nets markings and transition firings. The connective $\otimes$ represents accumulation of resources. Considering that A and B are resources, the presence of both is represented by the equation $A \otimes B$. The connective $\multimap$ represents causal dependency between resources. Considering that A is a resource that is consumed to produce B, this is represented by the equation $A \multimap B$.

In this paper, the translation from Petri nets to equations of Linear Logic is done based on Girault, Pradier-Chezalviel, and Valette (1997) to explicitly deal with markings. A marking M is represented by $M = P_1 \otimes P_2 \otimes \cdots \otimes P_k$ where P_i are place names with the presence of tokens. A transition is an expression of the form $M_1 \multimap M_2$ where M_1 and M_2 are markings representing respectively the consumed and produced resources. In fact, for the Petri net these are the equivalent *Pre* and *Post* functions of the transition. For example, transition tsc2 on the Petri net of Fig. 12 is denoted as $\text{tsc2} = \text{GC} \multimap \text{End\_GC} \otimes \text{C\_RespG}$.

4.3.1. Linear logic proof tree

In Girard (1987), all the rules for the sequent calculus proof are explained. In this paper, the fragment Multiplicative Linear Logic is used. This fragment contains the multiplicative connective $\otimes$, representing accumulative resources, and the linear implication $\multimap$, representing causal dependency. There is no negation and the meta connective “,” is commutative.

The deduction system used in Linear Logic is similar to the one used in classical logic, proposed by Gentzen in 1934 (Girard, 1987). A linear sequent is denoted as $\Gamma \vdash \Delta$, where Γ and Δ are sets of formulas, i.e., $\Gamma = \Gamma_1,\Gamma_2,\ldots\Gamma_n$ and $\Delta = \Delta_1,\Delta_2,\ldots,\Delta_n$. The symbol Γ is the antecedent and the symbol Δ the consequent. Linear Logic is commutative, which means that the order of the formulas in the consequent and the antecedent is indifferent. For instance, an antecedent can be represented as $\Gamma = F,G$ or as $\Gamma = G,F$; in a similar manner, a consequent can be represented as $\Delta = H,I$ or as $\Delta = I,H$.

A linear sequent proof consists of a set of hypotheses over an horizontal line and a conclusion above this line

$$\frac{hypothesis1\ hypothesis2}{conclusion}\ \text{rule}$$

A scenario is a set of events that transform an initial marking into a final one. A linear sequent $M_i,t_s \vdash M_f$ represents a scenario where M_i and M_f are the initial and final markings, respectively, and t_s is a set of non-ordered transitions. The $\vdash$ (turnstyle) and the comma are Linear Logic primitive symbols. The $\vdash$ divides the sequent into the left part (premisses), and the right part (conclusions). The sequent is proved by applying the rules of the sequent calculus. To prove a sequent is to show that it is syntactically correct. The proof tree is constructed bottom up. If the proof tree stops when all the leaves of the tree are identity sequents, such as $P \vdash P$, then the sequent is proved. Otherwise, if there are no further Linear Logic rules that can be applied to transform the sequent into an identity one, then the sequent is not proved.

In this paper, the logical rules $\otimes_L$, $\otimes_R$ and $\multimap_L$ are applied to build the proof tree as follows:

- the rule $\otimes_L$ transforms a marking such as $P1 \otimes P2$ into a list of atoms (P1, P2 for instance). This rule is applied repetitively to separate a marking into atoms, which allows the application of the $\multimap_L$ rule.
- the $\multimap_L$ rule expresses a transition firing. It generates two sequents. The left sequent represents the tokens that were consumed by the transition firing. The right sequent represents the new tokens produced by the transition firing. In case the new sequent is an identity one (ex. $P1 \vdash P1$), a leaf was reached and this branch of the tree was proved. Otherwise, the rule $\otimes_R$ is applied.
- the rule $\otimes_R$ transforms sequents of the form $P1,P2 \vdash P1 \otimes P2$ into two identity ones $P1 \vdash P1$ and $P2 \vdash P2$, which are proved leaves.

The equivalence between Petri nets reachability and the proof of sequents in Linear Logic was proved in Girault et al. (1997). This means that when a sequent is proved by applying the rules of the sequent calculus of Linear Logic, equivalently it can be inferred that the final marking is reached from the initial one by firing a set of transitions. As a direct consequence, if a sequent is not provable, then the marking is not reachable. The result is that Linear Logic can be seen as an analysis tool for Petri nets and can be applied to give some important results about the models. Within Linear Logic, the reachability problem is in fact the problem of sequent proving. It gives a formal and logical framework that assures the coherence of causality between transitions firing. Also, it is possible to extract from the proof tree some information about the order of transition

firing and the evaluation of the scenario (with or without associated time). The other advantage is that it is possible to derive specific scenarios directly from the Petri net without constructing the reachability graph (Demmou et al., 2004).

4.4. Analysis of scenarios

An important issue when designing systems is whether the system can reach a specific state as a result of desirable functional behavior. In a Petri net, this is represented as the existence of specific sequences of transition firings which transform a marking M_i into the required marking M_f. The purpose in this subsection is not to consider all evolutions of the Petri net, but to reason on specific scenarios without generating complete reachability graphs. The scenario considered is the one in which a green phase is given first to road section C (token in I2C) and then to road section D (token in I2D) after a vehicle is detected (Fig. 13, in which place I2 is refined, using the top-down strategy, into I2C and I2D). The same approach can be easily applied to other scenarios.

4.4.1. Reachability of an unsafe state

Considering the proposed scenario, one needs to verify if there is a possibility of reaching an unsafe state, such as the reachability of simultaneous green phases for two conflicting road sections, as for instance, GC and GD. From the Linear Logic point of view, this is equivalent to prove that the state GC ⊗ GD is not reachable from another state. What is to be proved is that from one chosen initial state, the final state GC ⊗ GD is not reached. The equivalent sequent is RC ⊗ RD ⊗ I2 ⊗ VD, tc4, tsc1, tc1, tsd1 ⊢ Go_GC ⊗ GC ⊗ Go_GD ⊗ GD. This sequent states that from red states for both sections, it will be allowed green for sections C and D simultaneously.

Table 1 shows the Linear Logic representation of the transitions. The proof tree is given as follows (places C_startGreen and D_startGreen were respectively renamed to CsG and DsG due to lack of space in the proof tree).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{VD,RD,Go\_GC(I2D \otimes VD \multimap Go\_GD \otimes DsG \otimes RVD),tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}}{RC \vdash RC \;\; CsG \vdash CsG \;\; VD,RD,Go\_GC(I2D \otimes VD \multimap Go\_GD \otimes DsG \otimes RVD),tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}\multimap_L}{I2C \vdash I2C \quad VD,RC,RD,(CsG \otimes RC \multimap GC),tc_1,tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}\multimap_L}{VD,RC,RD,I2C,(I2C \multimap Go\_GC \otimes CsG),tsc_1,tc_1,tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}\multimap_L}{VD,RC,RD,I2C,tc_4,tsc_1,tc_1,tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}\otimes_L}{VD \otimes RC,RD,I2C,tc_4,tsc_1,tc_1,tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}\otimes_L}{VD \otimes RC \otimes RD,I2C,tc_4,tsc_1,tc_1,tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD}\otimes_L$$

$$VD \otimes RC \otimes RD \otimes I2C,tc_4,tsc_1,tc_1,tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD$$

From the generated proof tree, it is clear that the final state is not reached, as there are no further Linear Logic rules that can be applied to transform the branch $VD,RD,Go\_GC(I2D \otimes VD \multimap Go\_GD \otimes DsG \otimes RVD),tsd_1 \vdash Go\_GC \otimes GC \otimes Go\_GD \otimes GD$ of the proof tree into identity sequents. The same reasoning can be done if the green phase is allowed first to road section D. In this case, a similar tree is generated with the same final result.

Other results can be inferred from the proof tree. The unsafe state is not reached because it is not possible to fire transition tc1, which is not enabled. In the proof tree, this is represented by not having the token I2D, as the VD is present. Supposing that this unsafe state is reached due to some error, then it is clear that

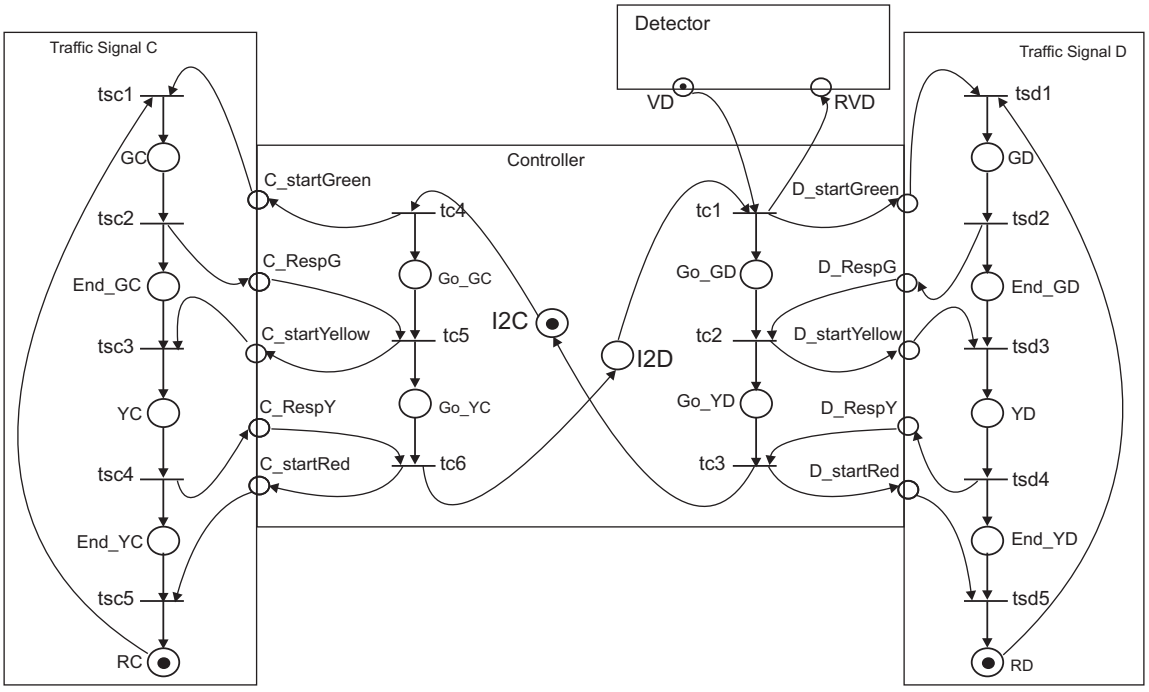

Fig. 13. Petri net for scenario analysis.

Table 1
Linear Logic representation of transitions.

$tc4 = I2C \multimap C\_startGreen \otimes Go\_GC$
$tsc1 = C\_startGreen \otimes RC \multimap GC$
$tc1 = I2D \otimes VD \multimap D\_startGreen \otimes RVD \otimes Go\_GD$
$tsd1 = D\_startGreen \otimes RD \multimap GD$

transition tc1 was fired because it was enabled by a token in I2D, and there is a design problem. In addition, it is clear that the green phase is only reached in this scenario for road section D when there is a token in place VD, which means that a vehicle was detected. Supposing that it is not possible to fire the transition to allow the green phase, one possible reason is that the detector failed or is damaged.

4.4.2. Reversibility

It is fundamental to prove that the controller and the traffic signals are capable of switching from red to green and yellow and returning to red, without possibilities of giving both red and green phases to the same road section. Considering $M_i = M_f =$ RC $\otimes$ RD $\otimes$ I2C as both initial and final markings and t_s={tc4, tsc1, tsc2, tc5, tsc3, tsc4, tc6, tsc5, tc1, tsd1, tsd2, tc2, tsd3, tsd4, tc3, tsd5} the set of transitions to be fired in the scenario, a Linear Logic sequent representing the reversible scenario is given as follows:

$$RC \otimes RD \otimes I2C, t_s \vdash RC \otimes RD \otimes I2C$$

An important remark is that the t_s set of transitions is not ordered. Thus, tsc5 and tc1 can be fired simultaneously, or tsc5 first or tc1 first. This is an advantage of using Linear Logic sequent calculus instead of a reachability graph, in which all the possibilities are evaluated. The same reasoning holds for transitions tsd5 and tc4.

Only part of the proof tree is given as follows (place C_startGreen was renamed to CsG due to lack of space in the proof tree, and only the current enabled transition appears in the proof tree)

$$\dfrac{\dfrac{\dfrac{\dfrac{RC \vdash RC \quad RD \vdash RD \quad I2C \vdash I2C}{RC, RD, I2C \vdash RC \otimes RD \otimes I2C}\otimes_R}{\vdots \quad \vdots \quad \vdots}\otimes_L}{RC, RD, I2C, (I2C \multimap Go\_GC \otimes CsG) \vdash RC \otimes RD \otimes I2C}\multimap_L}{RC, RD, I2C, t_s \vdash RC \otimes RD \otimes I2C}\otimes_L$$
$$\dfrac{RC \otimes RD, I2C, t_s \vdash RC \otimes RD \otimes I2C}{RC \otimes RD \otimes I2C, t_s \vdash RC \otimes RD \otimes I2C}\otimes_L$$

All leaves of the tree are identity ones. This means that the final state is reached from the initial one after firing the set of transitions. As a matter of fact, the Petri net is reversible. The practical implication is that the net is capable of returning to its initial state after each cycle and then repeats the cycle, which is exactly the expected behavior of a traffic signal.

5. Conclusion

In this paper, the application of modular Petri nets and extensions for the design and analysis of a responsive traffic control system is proposed. A divide and conquer strategy is followed in order to address system complexity. Thus, from a network of roads, a subnetwork of two road sections are considered each time, but due to modularity properties of the proposed extension, this is scalable to greater networks. The modularity is based on components whose internal behavior is specified by Petri nets. This modeling strategy is based on the division into components that are later combined by place fusion (bottom-up modeling).

Another Petri net extension used is the association of a time interval to each place, allowing traffic signals phase durations to be specified from a minimum to a maximum value. Model verification is based on several simulations using the common token player algorithm. After that, structural analysis is performed by applying the invariant theory to a Petri net model. The model used is composed of place fusion of Petri nets components for the system elements. Reachability analysis is applied to the generated net in order to give results about some desirable properties. Finally, scenario analysis based on theorem proving using the Linear Logic proof tree is applied to specific scenarios.

One advantage of Petri nets is that the same model can be used as an input for diverse Systems Engineering activities, giving a well-defined methodology from requirements to implementation. As a graphical model, Petri nets can be simulated through the common token player algorithm in a computer based tool. After that, the same model can be formally verified using the many methods provided by the Petri net theory. With the results about model reliability, performance evaluation can be used to infer future system behavior. Finally, when all these activities are successful based on defined requirements and criteria, the models can be used for automatic code generation of the control logic. The activities of modeling, simulation and verification are considered in this paper. Performance evaluation and code generation are activities to be performed in future research.

Acknowledgement

This work was supported by FAPEMIG (www.fapemig.br - CEX-APQ-01589-11).

References

Basile, F., Chiacchio, P., & Teta, D. (2012). A hybrid model for real time simulation of urban traffic. *Control Engineering Practice*, *20*(2), 123–137.
Brave, Y. (1993). Control of discrete event systems modeled as hierarchical state machines. *IEEE Transactions on Automatic Control*, *38*(12), 1803–1819.
Brown, C. (1989). *Relating Petri nets to formulas of linear logic*. Technical Report ECS-LFCS-89-87. University of Edinburgh.
Cassandras, C. G., & Lafortune, S. (2006). *Introduction to discrete event systems*. Secaucus, NJ, USA: Springer-Verlag.
Cheng, Y.-H., & Yang, L.-A. (2009). A fuzzy Petri nets approach for railway traffic control in case of abnormality: Evidence from taiwan railway system. *Expert Systems with Applications*, *36*, 8040–8048.
Choppy, C., Mayero, M., & Petrucci, L. (2008). Experimenting formal proofs of Petri nets refinements. *Electronic Notes in Theoretical Computer Science*, *214*, 231–254.
Demmou, D., Khalfaoui, S., Guilhem, E., & Valette, R. (2004). Critical scenarios derivation methodology for mechatronic systems. *Reliability Engineering & System Safety*, *84*(1), 33–44.
DiCesare, F., Kulp, P. T., Gile, M., & List, G. (1994). The application of Petri nets to the modeling, analysis and control of intelligent urban traffic networks. In *Proceedings of the 15th international conference on application and theory of Petri nets* (pp. 2–15).
Dotoli, M., & Fanti, M. P. (2006). An urban traffic network model via coloured timed Petri nets. *Control Engineering Practice*, *14*(10), 1213–1229.
Dotoli, M., Fanti, M. P., Mangini, A. M., Stecco, G., & Ukovich, W. (2010). The impact of ICT on intermodal transportation systems: A modelling approach by Petri nets. *Control Engineering Practice*, *18*(8), 893–903.
Engberg, U., & Winskel, G. (1990). Petri nets as models of linear logic. In A. Arnold (Ed.), *Proceedings of colloquium on trees in algebra and programming*. Lecture Notes in Computer Science, Vol 389 (pp. 147–161). Copenhagen, Denmark: Springer-Verlag.
Febbraro, A.D., Giglio, D., 2006. Urban traffic control in modular/switching deterministic-timed Petri nets. In *Proceedings of the eleventh IFAC symposium on control in transportation systems*. IFAC.
Febbraro, A. D., Giglio, D., & Sacco, N. (2004). Urban traffic control structure based on hybrid Petri nets. *IEEE Transactions on Intelligent Transportation Systems*, *5*(4), 224–237.
Gallego, J., Farges, J., & Henry, J. (1996). Design by Petri nets of an intersection signal controller. *Transportation Research Part C: Emerging Technologies*, *4*(4), 231–248.

Gehlot, V., & Gunter, C. A. (1989). Nets as tensor theories. In G. D. Michelis (Ed.), *Proceedings of the tenth international conference on application and theory of Petri nets* (pp. 174–191). Bonn, Germany.

Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, *50*(1), 1–102.

Girault, C., & Valk, R. (2003). *Petri nets for systems engineering—A guide to modeling, verification, and applications*. Berlin: Springer-Verlag.

Girault, F., Pradier-Chezalviel, B., & Valette, R. (1997). A logic for Petri nets. *Journal Européen des Systèmes Automatisés*, *31*(3), 525–542.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, *8*(3), 231–274.

Huang, H., & Kirchner, H. (2009). Policy composition based on Petri nets. In *Proceedings of the computer software and applications conference*, Vol 2.

Huang, Y.-S. (2006). Design of traffic light control systems using statecharts. *Computer Journal*, *49*(6), 634–649.

Huang, Y.-S., & Chung, T.-H. (2008). Modeling and analysis of urban traffic lights control systems using timed CP-nets. *Journal of Information Science and Engineering*, *24*(3), 875–890.

Huang, Y.-S., & Chung, T.-H. (2011). Modelling and analysis of air traffic control systems using hierarchical timed coloured Petri nets. *Transactions of the Institute of Measurement and Control*, *33*, 30–49.

Huang, Y.-S., Chung, T.-H., & ChenSystems, C.-T. (2005). Modeling traffic signal control systems using timed colour Petri nets. In *Proceedings of the 2005 IEEE international conference on systems, man and cybernetics*(pp. 1759–1764), Vol 2.

Hunt, P., Robertson, D., Bretherton, R., & Winton, R. (1981). *SCOOT—A traffic responsive method of coordinating signals*. Technical Report. TRRL Laboratory Report 1014. Crowthrone, Berkshire, UK: Transport and Road Research Laboratory.

Khansa, W., Denat, J. P., & Collart-Dutilleul, S. (1996). P-time Petri nets for manufacturing systems. In *Proceedings of the international workshop on discrete event systems* (*WODES'96*) (pp. 94–102).

Khisty, C. J., & Lall, B. K. (2003). *Transportation engineering: An introduction* (3rd ed.). New Jersey, USA: Prentice Hall.

Klein, L. (2001). *Sensor technologies and data requirements for ITS*. Boston, USA: Artech House.

Lee, W.-H., Tseng, S.-S., & Shieh, W.-Y. (2010). Collaborative real-time traffic information generation and sharing framework for the intelligent transportation system. *Information Sciences*, *180*(1), 62–70.

List, G. F., & Cetin, M. (2004). Modeling traffic signal control using Petri nets. *IEEE Transactions on Intelligent Transportation Systems*, *5*(3), 177–187.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, *77*(4), 541–580.

Robertson, D. (1969). *TRANSYT: A traffic network study tool*. Technical Report. LR 253. Crowthrone, Berkshire, UK: Transport and Road Research Laboratory.

Roess, R. P., Prassas, E. S., & McShane, W. R. (2004). *Traffic Engineering* (3rd Edition). Upper Saddle River, New Jersey, USA: Pearson Prentice Hall.

Rouse, W. B. (2003). Engineering complex systems: Implications for research in systems engineering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, *33*(2), 154–156.

Soares, M.S. (2011). Modeling and analysis of discrete event systems using a Petri net component. In *Proceedings of the IEEE international conference on systems, man and cybernetics* (pp. 814–819).

Soares, M.S., & Vrancken, J. (June 2007a). A Multi-agent distributed architecture for road traffic control. In Europe, E.I. (Ed.), *Proceedings of the 6th european congress on intelligent transport systems and services*, Vol. CD-ROM.

Soares, M.S., & Vrancken, J. (2007b). Road traffic signals modeling and analysis with Petri nets and linear logic. In *2007 IEEE international conference on networking, sensing and control* (pp. 169–174).

Soares, M.S., & Vrancken, J. (2007c). Scenario analysis of a network of traffic signals designed with Petri nets. Urban transport XIII. *Urban Transport and the Environment in the 21st Century* (pp. 289–297). UK: Wessex Institute of Technology .

Tolba, C., Lefebvre, D., Thomas, P., & Moudnia, A. E. (2006). Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modelling. *Simulation Modelling Practice and Theory*, *13*(5), 407–436.

Vogler, W. (1992). *Modular construction and partial order semantics of Petri nets*. New York, Secaucus, NJ, USA: Springer-Verlag, Inc.

Zhou, M. C., Dicesare, F., & Rudolph, D. (1992). Design and implementation of a Petri net supervisory for a flexible manufacturing system. *IEEE Transactions on Robotics Automation*, *28*(6), 1199–1208.