

Design and Animation of Coloured Petri Nets Models for Traffic Signals

Angelo Perkusich[†], Luciana M. de Araújo[‡],
Roberta de S. Coelho[‡], Kyller C. Gorgônio[‡],
Erica de L. G. Ribeiro[‡] and Adriano J. P. Lemos[‡]

[†]Departamento de Engenharia Elétrica

[‡]Laboratório de Redes de Petri

Departamento de Sistemas e Computação

Universidade Federal da Paraíba

Caixa Postal 10105, 58109-970 - Campina Grande - PB - Brazil

perkusic@dee.ufpb.br

Abstract

This paper addresses the design and visualization of animated models for coordinated and decentralized discrete event control systems, e.g. manufacturing systems and traffic control systems. We introduce an approach to the design and animation of such controllers based on Coloured Petri nets models and the Design/CPN tool. The approach is illustrated by a simple traffic control network modeling.

Keywords: discrete event system control, Coloured Petri nets, model animation.

1 Introduction

The design of complex control systems such as Intelligent Transportation Systems [10] requires the coordination of decentralized controllers in different intersections of a traffic network. Also, the need for adaptive control for such systems is increasing in importance so that parameters of the decentralized controllers can be tuned according to the environmental needs, e.g. traffic flow. As a complex system the need for visual

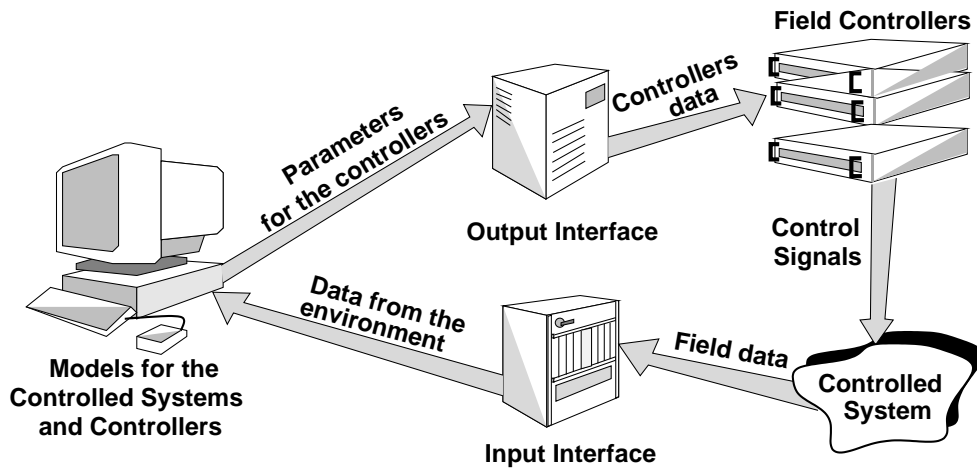


Figure 1: Adaptive control architecture

tools to help during the design phase for a better understanding of the domain problem are very important.

In this paper we introduce an approach to the design and animation of coordinated decentralized controllers for discrete event systems based on Coloured Petri (CPN) nets [3, 4, 2] models created using the Design/CPN tool [5].

The example used in this paper is a traffic signal control system as discussed in [8, 9, 1, 10, 13].

The paper is organized as follows: the architecture and the functionalities of an adaptive control for distributed discrete event systems is presented in Section 2. In Sections 3 and 4 we introduce basic concepts related to the coordinated control of traffic light control and modeling respectively. In Section 5 we present an approach to the animation of a CPN model constructed using the Design/CPN. In Section 6 we detail, based on a simple example the introduced approach. Finally, in Section 7 we present the conclusions of the paper.

2 Decentralized Adaptive Controllers

Discrete event control system are usually organized in a hierarchical structure [12, 11]. In the lowest level of the hierarchy are the local controllers performing local tasks, such as machines controllers in manufacturing systems and traffic light signal controllers. These controllers are usually decentralized and coordinated based on the parameters defined on the upper levels of a control hierarchy. Considering the case of traffic lights, sequences of green, red and yellow are repeated for each traffic lights. In the case of fixed length intersections, the coordination level defines the timing of the green signals

for consecutive intersections along a desired path. Usually this timing considers only the arterial traffic, so that the timing for a green wave of lights is defined. In this case the major problem is the optimization of the timing for the traffic lights in all intersections in the given path.

The block diagram for an adaptive decentralized control architecture is shown in Figure 1. The computer running the models for the controlled system (e.g. a traffic network) and the controllers (e.g. each traffic light controller) is interfaced with the controlled system by means of input and output interfaces. The input interface collects data from the field and sends data to the machine running the model. The parameters of the model that depend on this data are then updated so that new output parameters are defined for each controller in the field. Then, these parameters are sent to the field controllers through an output interface.

As shown in Figure 2 in the context of this paper we use a CPN model built using the Design/CPN tool [5]. Then, the occurrence graph is generated and the behavior specification for the system may be defined, either using functions written in CPN-ML or defined using ASKCTL [7], a temporal logic used for model checking in the Design/CPN tool.

A file with the state information is created, so that the behavior of all field controllers parameters can be extracted from it. The data from the environment is stored in a file and transitions in the CPN model are properly parameterized with the information from this file in a similar way as discussed in [6].

3 Traffic Signal Systems

Nowadays a traffic signal may exist as a single isolated controller or may be part of a multi-signal traffic control system. These traffic control systems are comprised of interacting components such as signals, detectors, and communication infrastructure that are arranged in a manner which effectively and efficiently coordinate traffic flow along a corridor or throughout a network. Traffic engineers are continually refining the control strategies in an effort to allow for the safe and efficient movement of people and goods. Due to the prohibitive costs, the increase in the use of motor vehicle without a proportional increase in the infrastructure lead to many different problems, such that increase in fuel consumption and pollution. The sophistication level of traffic control has advanced beyond the exclusive use of time-of-day programs and local flow detection. Today, many research is devoted to real-time traffic adaptive control systems. These systems not only operate based on local detector demand but also make control decisions based on predicted future demand [13, 10].

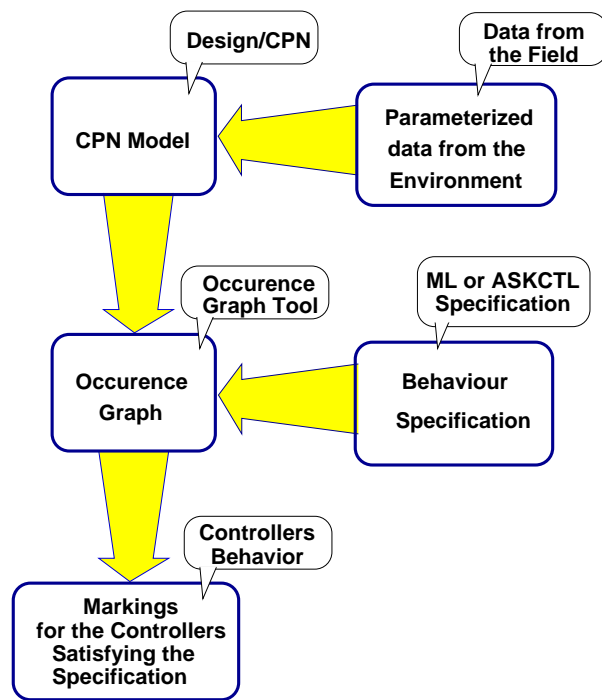


Figure 2: Block diagram illustrating the life cycle for the field controllers behavior specification

In arterial and network systems, traffic is thought of as moving groups, namely platoons. Typically, signals located at the boundary of the system group vehicles during the red phase and then discharge this platoon under green. The traffic control system coordinates the cycles for traffic signals at the intersections in order to maintain the traffic flow. Such a control system consists of coordinated-actuated controllers that are distributed in nature. Also, the control systems must provide actuated control capabilities that are intended to improve the functioning of a traffic control system. This is accomplished by allowing individual intersections to respond to varying traffic demands while also maintaining coordination from one intersection to another.

To improve the current practice of using coordinated-actuated controllers for traffic signal control systems, advanced traffic control systems are being developed. The reader can refer to Figure 1 in what follows. The basic methodology for these adaptive systems is to have a model for the traffic network, an input interface to gather data from the intersection traffic flow, and an output interface to the various intersection controllers. The model is then used together with up to date information from the traffic flow in the network and then finds out a better control policy for a given behavior specification, as for example optimize green wave on throughput.

There are different levels of sophistication in traffic control systems. The most

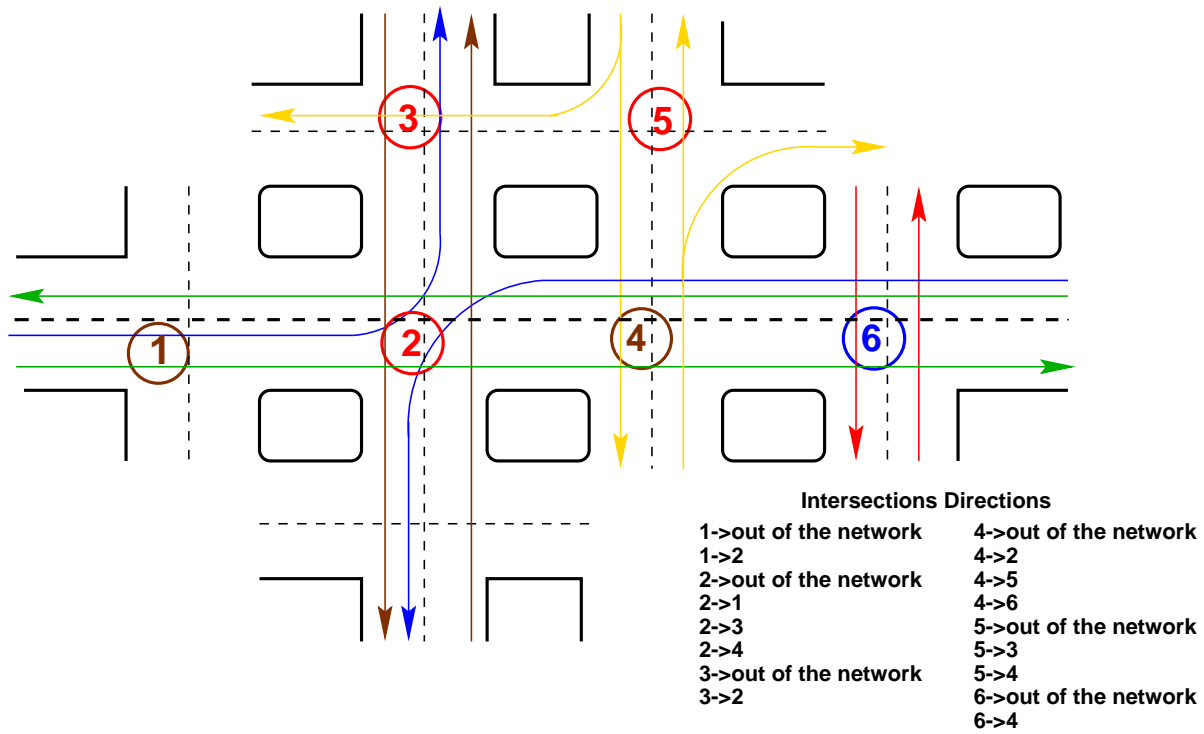


Figure 3: Example of a traffic network

simple is known as Webster's Method. It indicates how to divide the amount of green time in a periodic cycle between opposing flows of traffic at an intersection. Optimal sequences of green time are determined using predicted arrivals of vehicles or blocks. For more details the interested reader may refer to [8, 9, 1, 10, 13].

4 Design of a Traffic Control System

The problem of the design of a traffic control system consists of elaborating a model of a traffic network that allows the control of the traffic lights, guaranteeing that conflicting routes (in one particular intersection) do not have green signs at the same time, as well as modeling the dynamics of the stream of vehicles (or blocks) through the intersection. Also, a block of cars always stop as minimum as possible when in arterial traffic. The vehicles passing through the network may come from each one of the existing intersections or they may come from another traffic network. Thus, it is necessary to represent the interaction of this network with the environment, modeling the input of vehicles by the edges of the network and the output of vehicles after crossing it.

An example of a network composed of six intersections is shown in Figure 3. The

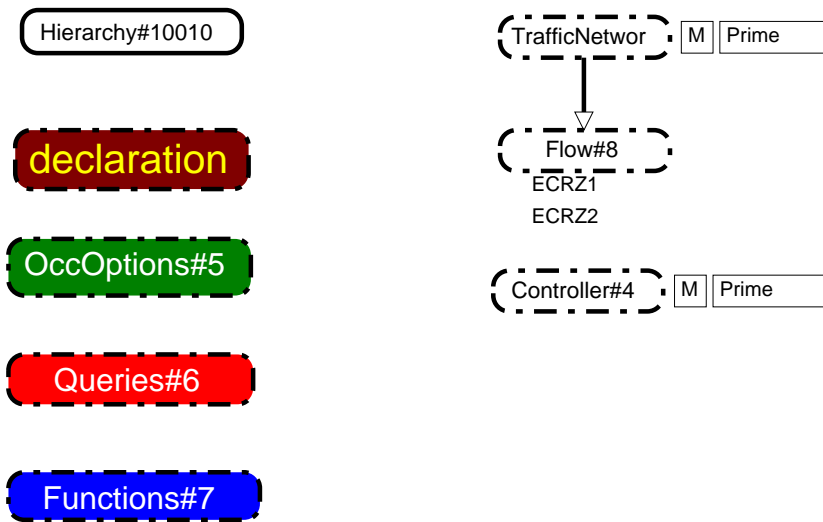


Figure 4: The hierarchy page for the model

routes are represented by directed arcs. We assume that the controllers for each intersection have the same structure.

A Hierarchical Coloured Petri net model was built and the hierarchy page is shown in Figure 4. For this model three pages were built: the traffic network page, the control page, and the flow page. Also, it is shown a page where the CPN-ML functions, queries, and occurrence graph options for the model are defined.

The traffic network page models the network connecting all the intersections. The control page models the controllers implementing the sequence of allowed phases for each intersection and the timing for the phases. The flow page represents the movement of blocks through the intersection including timing parameters for the phases, as well as how long a block takes to move through an intersection.

In what follows we present the declaration node including the color sets and functions and some details related to each one of the pages for the model.

4.1 Color Sets

The Color Sets for the entire model is shown below.

```
color MAX_T = int with 8..13;
color YELLOW_T = int with 3..7;
color INTERSECTION = with c1|c2|c3|c4|c5|c6 timed;
color PHASE_N = with ph1 | ph2 | ph3 | ph4;
color CR_MINT_YELT = product INTERSECTION*MIN_T * YELLOW_T timed;
```

```

color CR_YELLOWT = product INTERSECTION*YELLOW_T timed;
color CR_YELT = product INTERSECTION*YELLOW_T;
color CR_MAXT= product INTERSECTION * MAX_T timed;
color PHASES_SET = product PHASE_N*MAX_T*MIN_T*YELLOW_T;
color CR_PHS = product INTERSECTION*PHASES_SET;
color PHSET_LIST = list PHASES_SET;
color CR_PH = product INTERSECTION*PHSET_LIST;
color CR_PHASE = product INTERSECTION*PHASE_N;
color CROS_PH_LIST = list CR_PHASE;
color B = with b;
var x : PHASES_SET;
var max_t : MAX_T;
var min_t : MIN_T;
var yellow_t : YELLOW_T;
var phase_n : PHASE_N;
var ph : PHASE_N;
var vph1 : PHASE_N;
var vph2 : PHASE_N;
var cr : INTERSECTION;
var cr1 : INTERSECTION;
var cr2 : INTERSECTION;
var cr3 : INTERSECTION;
var cr4 : INTERSECTION;
var cr5 : INTERSECTION;
var cr6 : INTERSECTION;
var last_cr : INTERSECTION;
var phset_list : PHSET_LIST;
var cros_ph_list : CROS_PH_LIST;
var cros1_ph1_list : CROS_PH_LIST;
var cros_ph : CR_PHASE;
var cros1_ph1 : CR_PHASE;

```

Some of the color used in the model are explained as follows:

PHASE_N is a 4-tuple, where each element indicate a possible phase that will be in use in the intersection, at sometime. Phases for example are as shown in Figure 5.

PHASES_SET is a 4-tuple. The first element (PHASE_N) indicates the phase that will be, or is, currently, active for the intersection; the second (MAX_T) and the

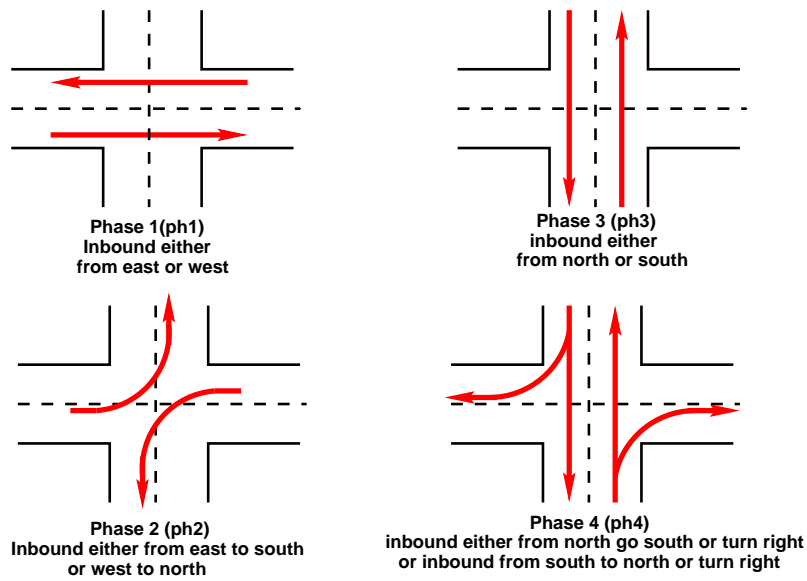


Figure 5: Phases for the example

third (MIN_T) elements indicate, the maximum time and the minimum time that a phase remains active, respectively. And the fourth element (YELLOW_T) indicates the time that the yellow light will remain active.

MAX_T represents the maximum green time for the traffic light.

MIN_T represents the minimum green time for the traffic light.

YELLOW_T represents the yellow time for the traffic light, it varies from phase to phase.

INTERSECTION represents the intersections of the traffic network.

CR_PHASE is a pair defining the active phases for the intersections.

CROS_PH_LIST represents is a list defining pairs specifying the intersection and the phases (CR_PHASE) of a path for a block.

CR_PH represents a pair defining the intersection and a list of possible phases for the intersection with the maximum and minimum green time and the yellow time for the phase.

4.2 Controller Page

The controller page is shown in Figure 6. This CPN models all the controllers of the system. Each controller is represented by a token of the color set CRPH (see Section 4.1) in place `IntersectionsAndCorrespondingPhases`. The red time of the traffic light was not stored since it is not considered in this model. The firing of transition `MakePhases`, models the beginning of a phase for each controller. The green time can vary from a minimum value to a maximum one, depending on the existence or not, of a block of cars crossing the intersection. If the minimum time was consumed, `WaitMinTime` fires, putting a token in place `MinimumTimeExpired`. If there is no block of cars crossing the respective intersection, in other words, there is a token in place `Intersections1`, corresponding to this intersection, `ChangePhase2` fires and the controller will change the traffic light to yellow, on the contrary the light remains green until `max_t`, after that transition `WaitMaxTime` fires. A token in place `YellowTime` models the amount of time that the traffic light must remain on yellow. A token in `Intersections2`, represents the end of a phase.

4.3 Flow Page

The Flow page is shown in Figure 7. A token in place `BlockOfCarsItinerary` represents a block of cars arriving at the intersection or a block of cars waiting for a specified phase. To cross the intersection, blocks of cars wait at the intersection, represented by tokens in place `BlockOfCarsItinerary`, until the specified phase is the active one for the corresponding intersection and there is no block of cars crossing it. The active phases are defined by tokens in fusion place `CrossingPhase` (see Controller page description). When these conditions occur, transition `EnteringCrossing` fires putting a token in place `BlockOfCars1` and removing one token, corresponding to the current intersection, from place `Intersections1`. This allows only one block of cars to cross the intersection at a time. The timed transition `BlockCrossingTime` models the time spent by a car to move through the intersection. Observe that, this amount of time may vary from block to block, but for simplicity in this model we are not taking this into account. When transition `BlockCrossingTime` fires, a token corresponding to the actual intersection for a block is put in place `Intersections1`, and another is put place `IntinerartOfBlockOfCars2`. The token representing the path is a list of pairs for the intersection and the phase for the block, when a token is removed from place `IntinerartOfBlockOfCars1` the head of the list is removed, thus the token put in place `IntinerartOfBlockOfCars2` defines the rest of the path.

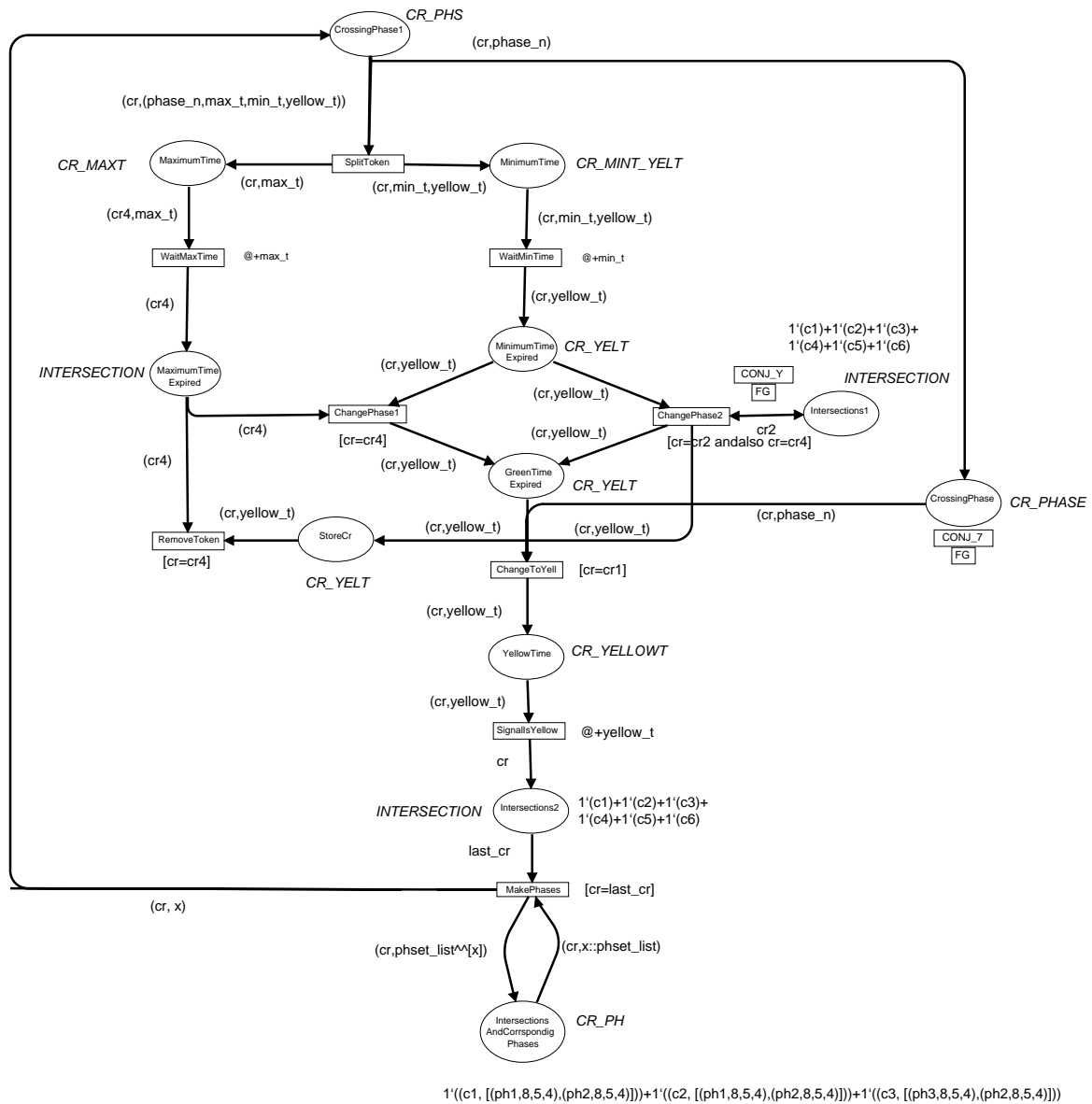


Figure 6: CPN page for the intersection controllers

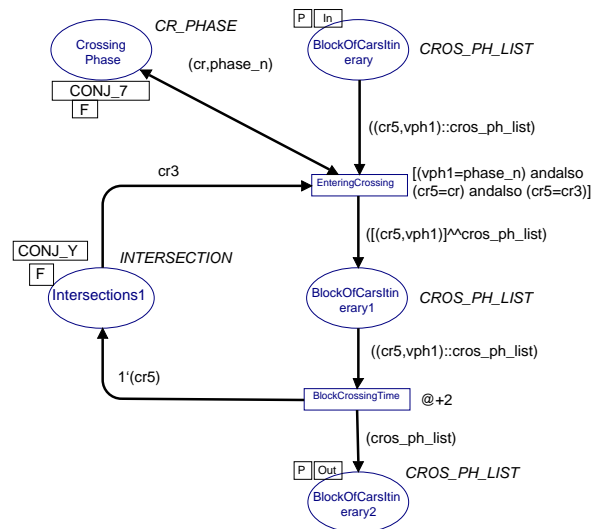


Figure 7: CPN page for the flow of blocks of cars in an intersection

4.4 Traffic Network Page

The Traffic Network page is shown in Figure 7. This is the model the Traffic Network itself, as shown in Figure 3. A token in any of the places *Intersections_i* ($i = 1,2,3,4,5,6$) represents a block of cars entering in the intersection. Each transition *EnteringInter_i* ($i = 1,2,3,4,5,6$) is a substitution transition, modeling the flow page for each intersection. The input places of these transitions correspond to place *BlockOfCarsItinerary*, modeling the arrival of a block of cars in an intersection as well as a block of cars waiting for a specified phase. When a block of cars is at the intersection a token is put at place *ChooseDirection_i* ($i = 1,2,3,4,5,6$). The information represented by this token indicates if a block of cars continues in the system, as for example a token such that $1'([(c2,ph2),(c3,ph3)])$ defines that the block of cars goes to the intersection 2, in this case the transition *LeavingInter₂* fires. Otherwise the block of cars leaves the system, the list specifying the path is empty or the next specified intersection is not immediately reachable. For these cases any of the transitions *ExitingSystem_i* ($i = 1,2,3,4,5,6$) fire.

In the following section we introduce the main concepts related to the animation of CPN models.

5 Animation Environment

The animation approach introduced in this paper is based on a Java applet that receives as input a textual description of the animation environment and control information for

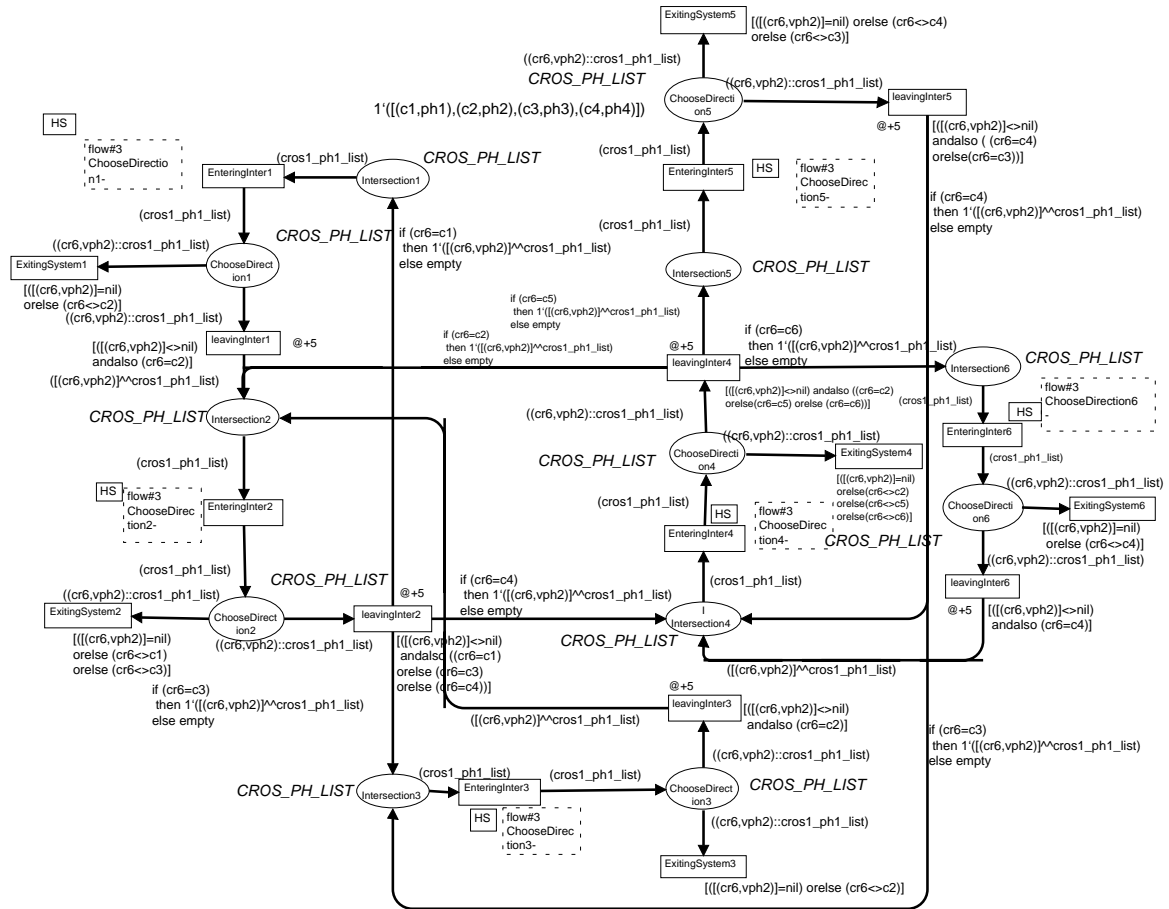


Figure 8: CPN page for the flow of blocks of cars in the network

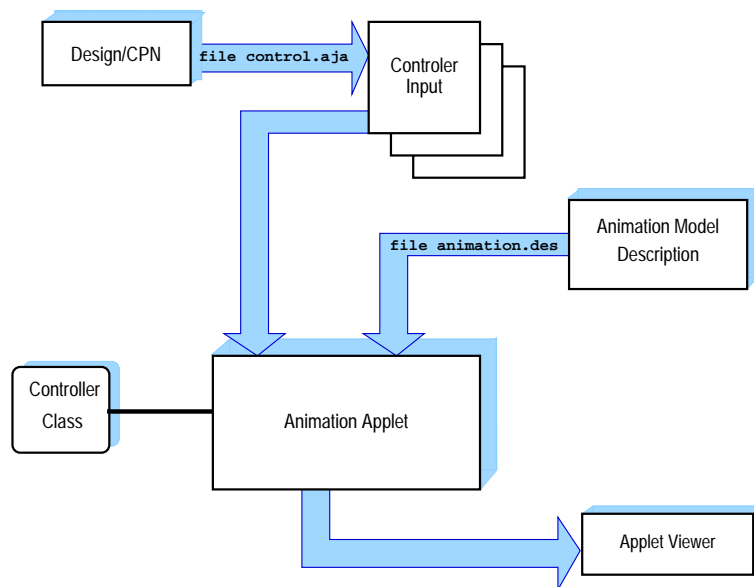


Figure 9: Block diagram for the animation environment

animation purposes from a file. This control file is generated based on the occurrence graph of Design/CPN as explained in Section 2. In Figure 9 a block diagram for the animation environment is shown.

A description of the animation environment follows a grammar that defines all the components taking part in a specific animation. Basically the components are: fixed and mobile objects, a *board* defining the grid where the animation takes place, the routes for the mobile objects, and the controllers for the shared regions in the grid.

Two different files are used for animation purposes: one describing the objects, and a control data file (or stream) Indeed, the control file is a set of vectors (markings) including the specific markings for the controller or controllers defined for the target system. In this paper each position of a vector in the control file defines the phases for the set of signals for each intersection for the traffic network shown in Section 4.

The *Animation Applet* is a set of classes to manipulate the interface actions so that the visual objects can behave according to some basic rules, e.g. objects cannot be in the same area at the same time or they cannot enter in shared regions that are blocked to them. This can be used for example to stop a block of cars in an intersection.

The main characteristic of the implementation is the use of an applet based on a generic controller class that can be defined depending on the system being modeled and animated. For example, in the case of the traffic signal controllers the control file defines the phases and the timing for the controllers of each intersection in the traffic network.

(1`a+2`b,[5,6]);	(1`e+3`s,[3,9]);	(4`g+5`k+9`t,[3,8]);	(4`r+6`w,[7,9]);	(3`d+1`q,[1,4])
(2`a+1`c,[2,6]);	(4`d+2`s,[3,8]);	(6`d+7`f+8`a,[4,6]);	(2`y+3`v,[4,6]);	(4`e+2`g,[5,10])
(1`a+2`b,[5,6]);	(1`e+3`s,[3,9]);	(4`g+5`k+9`t,[3,8]);	(5`s+1`t,[4,5]);	(3`d+1`q,[1,4])
(2`a+1`c,[2,6]);	(4`d+2`s,[3,8]);	(6`d+7`f+8`a,[4,6]);	(4`r+6`w,[7,9]);	(4`e+2`g,[5,10])
(1`a+2`b,[5,6]);	(1`e+3`s,[3,9]);	(4`g+5`k+9`t,[3,8]);	(2`y+3`v,[4,6]);	(3`d+1`q,[1,4])
(2`a+1`c,[2,6]);	(4`d+2`s,[3,8]);	(6`d+7`f+8`a,[4,6]);	(5`s+1`t,[4,5]);	(4`e+2`g,[5,10])
(1`a+2`b,[5,6]);	(1`e+3`s,[3,9]);	(4`g+5`k+9`t,[3,8]);	(4`r+6`w,[7,9]);	(3`d+1`q,[1,4])
(2`a+1`c,[2,6]);	(4`d+2`s,[3,8]);	(6`d+7`f+8`a,[4,6]);	(2`y+3`v,[4,6]);	(4`e+2`g,[5,10])
(1`a+2`b,[5,6]);	(1`e+3`s,[3,9]);	(4`g+5`k+9`t,[3,8]);	(5`s+1`t,[4,5]);	(3`d+1`q,[1,4])
(2`a+1`c,[2,6]);	(4`d+2`s,[3,8]);	(6`d+7`f+8`a,[4,6]);	(4`r+6`w,[7,9]);	(4`e+2`g,[5,10])
Specification of the Behaviour for Controller 1	Specification of the Behaviour for Controller 2	• • •	Specification of the Behaviour for Controller 5	

Figure 10: Control information for file `control.aja`

The class for the controller, the control file, and the description of the animation file, are integrated by the *Animation Applet*. Each row in the control file represents the states of all the controllers, and the state of each controller is in a specific position of this row. This information is extracted from the occurrence graph. In Figure 10 we illustrate an example for the control file specifying the behavior of five controllers. It is important to point out that at this moment we do not assume that the structure of the controllers are the same. Different classes for each controller can be *plugged* in the animation applet. For the example of the traffic light controllers the behavior is a simple finite state machine that controls sequential timed changes of phases for each intersection.

5.1 Generic Controllers

The idea of generic controllers is based on the possibility of defining abstract classes and interfaces in Java. The generic behavior of a controller is specified by an abstract class named *Controller*. Any controller belonging to the class *Controller* must implement the following basic functionalities:

- take as input a list, as shown in Figure 10, specifying the reachable states.
- have a set of methods to manage the list and take the appropriate control actions.
- implement synchronization mechanisms to support different threads of execution.
- to turn public its status.

For the example of traffic lights the list have the phase and timing information for each controller. Therefore, for each class the designer must define a method to extract this information from the control file. In the next section we shown an example for a simplified traffic network controller.

The controllers operate over shared structures or regions that are accessed by moving objects during the animation. For animation purposes this corresponds to avoid that visual objects occupy the same area on the screen. For example if a traffic light is closed in a given direction the block moving towards that direction must stop. Since the control objects as well as visual moving objects are controlled by different threads of control operating over different critical regions, it is possible to implement a parallel animation, what would not be possible otherwise [6].

5.2 Grammar for the Input Control File

As said before the information needed to describe the animation model is stored in a file. In the following the grammar for description of a model is presented.

```

<animation model> := <input><grid><routes><objects><controllers>
<input>           := input = <path>;
<routes>         := <route><routes> | <route>
<route>          := route IDENTIFIER = <points>;
<objects>        := <object> <objects> | <object>
<object>         := object IDENTIFIER = <size>,<speed>,<quantity>,<id_route>[,<pos>];
<grid>           := grid = <size>,<{active}>;
<controllers>   := <controller><controllers> | <controller>
<controller>    := controller IDENTIFIER = <points>,<pos>;
<points>        := <point>,<points> | <point>
<point>         := (INT,INT)
<size>          := <point>
<active>        := <lines>
<lines>         := <line>,<lines> | <line>
<line>          := <row> | <column>
<row>           := r(INT,INT,INT)
<column>        := c(INT,INT,INT)
<pos>           := INT
<path>          := IDENTIFIER
<speed>         := INT
<quantity>      := INT
<id_route>      := IDENTIFIER

```

The terminals are very intuitive and the rules are always very short. Except for the connection between the control file and the controller description, all the other components are basically visual informations. The controller must be associated to a column defining a place of the marking vector for the CPN model representing the state of a controller for the target system. The controllers update their status according to the values defined in each specific column in the marking vector. Based on this grammar one can define the animation in a straightforward way. An example based on the traffic network control is as follows:

```
input = control.file;
grid = (10,10),{r(5,1,10),r(6,1,10),c(5,1,10),c(6,1,10)};
route RT1 = (6,1),(6,10);
route RT2 = (6,1),(6,5),(10,5);
route RT3 = (5,10),(5,1);
route RT4 = (5,10),(5,5),10.5);
object BLOCK1 = (1,1),1,1,RT1;
object BLOCK2 = (1,2),1,1,RT3;
object BLOCK3 = (1,1),1,1,RT4;
controller SEMAPHORES = (5,5)(5,6),(6,5)(6,6),2;
```

For the example shown for the animation file, observe for example the routes RT1,RT2,RT3 and RT4 and the critical region (SEMAPHORE) in Figure 11. For more complex configurations, as in the example presented in Section 4 the definition of the animation model is very simple.

6 A Case Scenario

In this section we present details related to the process to obtain the behavior specification for the controllers for a simplified traffic network with two intersections each one with two phases. For this case the occurrence graph (OG) was obtained in 87 seconds, on a time-shared Pentium II, with 256 Mbytes of main memory, running Linux 2.2.12. The OG has 468 nodes and 1408 arcs.

To obtain the specification of the states for the controllers for the green wave the following steps were executed:

1. Define the initial marking and timing definition. In this case, the traveling time between the intersections was 3 time units, and the crossing time for the intersections was 2 time units. The block of cars generation rate was 12 time units.

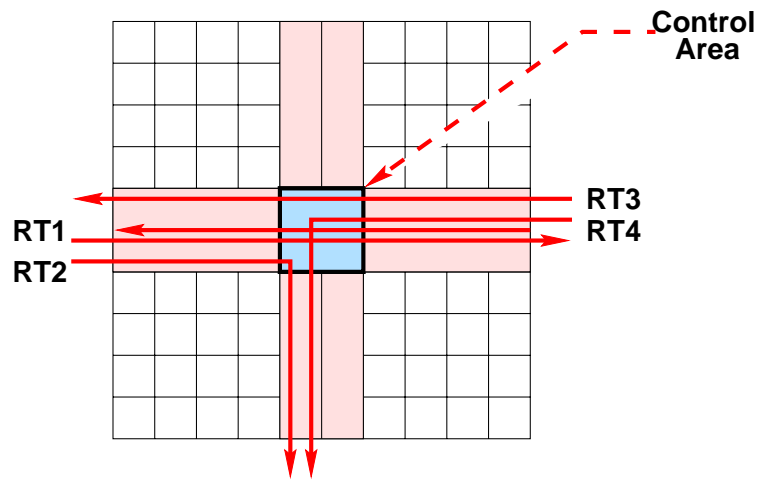


Figure 11: Animation grid in the board

2. Verify whether a green wave exists, if not, then change the timing for the initial marking. To do so, a set of CPN-ML functions were defined and are shown below.
3. Find a set of markings satisfying the green wave property specifying the behavior for the controllers.

(* Find the path matching the green wave *)

```

fun FindThePath ( [] ) = [] |
FindThePath (head::tail) =
let
    val Path = (NodesInPath(InitNode,head)) handle NoPathExists => nil;
in
    if ((Path <> nil) andalso ((length(RemoveDup (Path))) > 1))
    then Path
    else FindThePath (tail)
end;

```

(* Remove duplicated marking from the path (ignoring time stamps) *)

```

fun RemoveDup( [] ) = [] |
RemoveDup ( h::t ) =
let
    val aux = [];
in
    if (memberNode(h,t) orelse

```

```

    (StripTime (Mark.Flow'IntersectionsAndCorrespondingPhases 1 h) == empty))
then RemoveDup (t)
else h::RemoveDup (t)
end;

```

(* Verify if a node is member of a path *)

```

fun memberNode (x,nil) = false
| memberNode (x,h::t) = if (st_Mark.Flow'RemoveDup 1 x) =
    (st_Mark.Flow'RemoveDup 1 h)
    then true
    else memberNode (x, t) ;

```

(* Save the state of the controller for the green wave *)

```

fun SaveMarkings (file,[]) = false |
SaveMarkings (file,h::t) =
    (output(file,(st_Mark.Controller'IntersectionsAndCorrespondingPhases 1 h));
SaveMarkings (file,t); true);

```

The specification for the controllers is a file which contents is as follows:

```

C'ICP 1: 1'(c1,[(ph2,5,3,1),(ph1,6,4,2)]@[0]+ 1'(c2,[(ph1,5,3,1),(ph2,8,6,2)]@[0]
C'ICP 1: 1'(c1,[(ph1,6,4,2),(ph2,5,3,1)]@[6]+ 1'(c2,[(ph1,5,3,1),(ph2,8,6,2)]@[0]
C'ICP 1: 1'(c1,[(ph1,6,4,2),(ph2,5,3,1)]@[6]+ 1'(c2,[(ph2,8,6,2),(ph1,5,3,1)]@[9]
C'ICP 1: 1'(c1,[(ph2,5,3,1),(ph1,6,4,2)]@[10]+ 1'(c2,[(ph2,8,6,2),(ph1,5,3,1)]@[9]
C'ICP 1: 1'(c1,[(ph2,5,3,1),(ph1,6,4,2)]@[10]+ 1'(c2,[(ph1,5,3,1),(ph2,8,6,2)]@[13]
|_____| |____|
(CL1)_____ | _____(Time Stamp)

```

Observe that ICP is the place IntersectionsAndCorrespondingPhases for the CPN for the controller shown in Figure 6. From each line of this file the applet extracts the required information to manage all the controllers at a given moment. For the example shown, to coordinate the behavior of the first controller "c1", the applet takes the tuples of the column CL1 and the Time Stamp. Based on these two information the applet activates the controller at time zero (@[0]) and phase 1 (ph1). Observe that the value of the Time Stamp defines when controller changed to the actual phase. Thus, the phase remained green for 4 time units and yellow for 2 time units. This information is calculated from the difference between the the current time stamp, the previous time stamp, and the yellow time defined for the previous phase, in this case 2 time units.

7 Conclusions

In this paper we presented an approach to the specification, analysis, design and animation of distributed controllers based on Coloured Petri nets and the Design/CPN tool. From the point of view of the animation the main advantage is that the user can observe the concurrent behavior of a given system modeled using the Design/CPN tool. Also the approach introduced for the definition of the behavior of decentralized controllers can be used for adaptive control for systems which behavior can change, as is the case of traffic light controllers for a network.

To illustrate the approach we presented the design of a coordinated traffic supervision, based on decentralized controllers for the intersections. We have used the model for defining the control for the case of a green wave for arterial traffic. It is important to point out that there are other aspects related to coordinated traffic control that can be considered, such as the “early return to green” problem, inefficient green splits, and cycle lengths being too short or too long [10].

In this paper we have used an approach based on ML functions to search for markings expressing the desired properties for the model. The results obtained indicates that the approach for both, the specification of the behavior of the controllers as well as the animation is correct. In the case of the use of model checking it is still necessary to make some changes in the ASKCTL library, so that information such as the identification of node at the defined property was proved or not.

Also, we are currently investigating the applications of the introduced approach to manufacturing and batch systems supervision and animation. Also, we are refining the solution, both the controllers behavior and the animation, so that a more automatic and interactive tool can be developed. Apart from developing the animation, we are investigating the use of the approach introduced for control. Basically, what is necessary to do is to substitute the the animation applet by a controller applet.

References

- [1] D. Bullock and C. Hendrickson. Roadway traffic control software. *IEEE Control Systems Technology*, 2(3):255–264, September 1994.
- [2] K. Jensen. *Colored Petri Nets, basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer-Verlag, 1992.
- [3] K. Jensen. *Coloured petri nets-Basic Concepts, Analisis Methods and Practical Use*, volume 2. Springer-Verlag, 1997.

- [4] K. Jensen. An introduction to the practical use of coloured petri nets. In *Lectures on Petri Nets II: Applications*, volume 2, pages 237–292. Springer, 1998.
- [5] K. Jensen and et. al. *Design/CPN Manuals*. Meta Software Corporation and Department of Computer Science, University of Arthus, Denmark. On-line version:<http://www.daimi.aau.dk/designCPN/>.
- [6] O. Kummer, D. Moldt, and F. Wienberg. Framework for interacting design/cpn- and java-processes. In *CPN'98 Workshop Proceedings*, <http://www.daimi.au.dk/CPnets/workshop98/>. 1998.
- [7] Meta Software Corporation and Department of Computer Science, University of Arthus, Denmark. *Design/CPN-ASKCTL Manual*. On-line version:<http://www.daimi.aau.dk/designCPN/libs/askctl/>.
- [8] I.R. Porche. *Dynamic Traffic Control: Decentralized and Coordinated Methods*. PhD thesis, The University of Michigan, Ann Arbor, MI, USA, 1998.
- [9] I.R. Porche, M. Sampath, Y.-L. Chen, R. Sengupta, and S. Lafortune. A decentralized scheme for real-time optimization of traffic signals. In *IEEE Proceedings of the 1996 IEEE International Conference on Control Applications*, 1996.
- [10] G. Shoup. Traffic signal system offset tuning procedure using travel time data. Master's thesis, West Lafayette, Indiana, USA, 1998.
- [11] M. Silva and E. Teruel. Petri nets for the design and operation of manufacturing systems. In *First International Workshop on Manufacturing and Petri Nets*, pages 31–61, Osaka, Japao, Junho 1996.
- [12] M. Silva, E. Teruel, R. Valette, and H. Pingvad. Petri nets and production systems. In *Lectures on Petri Nets II: Applications*, volume 2, pages 85–124. Springer, 1998.
- [13] J.S. Watson. Reconcilled platoon accomodation at isolated intersections. Master's thesis, West Lafayette, Indiana, USA, 1998.