Specification and Validation of a Concurrent System: An Educational Project

Gérard Berthelot[1] and Laure Petrucci[2]

[1] CEDRIC-IIE, Institut d'Informatique d'Entreprise,
18 allée Jean Rostand, F-91025 EVRY Cedex
berthelot@iie.cnam.fr
[2] LSV, CNRS UMR 8643, ENS de Cachan
61 avenue du président Wilson, F-94235 CACHAN Cedex
petrucci@lsv.ens-cachan.fr

Abstract. Since several years, we are in charge of a course on specification and validation of concurrent and reactive systems. At the end of this course, the students must carry out a project on a model railway. They have to specify the railway, to validate their model and finally to translate it into a program running the model railway with up to five trains.
In this paper, after presenting the problem, we describe how it is specified and checked, step by step, by the students. We also explain how the analysis results lead to a policy for the switches management. Finally, we give some hints about the implementation.

1 Presentation of the Problem

In this paper, we report about a teaching experience for a group of twenty graduate students, during their second year in engineer school. Their background consist of two years studies in mathematics and physics, followed by one year in computer science. During the previous year they learned the basics of computer programming, operating systems (in particular Unix processes) and concurrent programming concepts. Our course is optional and is composed of several topics: communicating automata, coloured Petri nets, temporal logics, tools for specification and verification (Design/CPN, Spin, Esterel), real-time systems. After 80 hours of lectures and exercise courses, the students have to carry out a project by themselves. They spend roughly 70 hours of personal work, to complete the project from specification to hardware implementation. The students choose among the three previous tools. Here, we will focus on the use of Design/CPN only.

1.1 Goals of the Project

Teaching programming of concurrent systems is not so obvious, as the students are used to programming in a sequential style. In order to alleviate this difficulty, we decided to have them manage a system with intrinsic parallelism. A model

railway, allowing several trains, is altogether well-known, convenient, inexpensive and appealing to students. Moreover, such a system can be used to tackle real-time concepts and tools. Therefore, we decided to create a course including all aspects of parallel programming, from the specification phase to the real-time programming and implementation on the physical model. Nevertheless, this course is still being further developped, we have not yet addressed time aspects. This can be done in the future using interval timed coloured Petri nets as in [dAO94].

1.2 What is Asked from the Students

The students' project was not only designed as an approach to parallel programming, but also to emphasize the benefits of specification and validation prior to programming. In particular, the students were asked to produce a graphical model, having the same aspect as the physical railway. It should be pointed out that this is not demanded for esthetic reasons but it helps a lot to understand whether a configuration of the railway is normal or not. This facilitates a boring and error-prone effort to synthesize a long firing sequence. It represents an important benefit for debugging.

So far, the hierarchical coloured Petri nets ([Jen92]) have been chosen as a model, due to their support of hierarchies, simulation, occurrence graph requirements. Hierarchies allow a structured design, where the top-level net sticks to the hardware layout. The use of high-level nets allows both to capture several cases in a single transition and group the parameters of trains and tracks sections into one place. The use of an ordinary net leads to unreadable intricate graphics. The tool used to support this model is Design/CPN ([CPN96]).

Two views were considered in order to exploit the physical railway.
The first view, which will be named "real railway view", is meant to operate as a real railways, with the same rules. Effectively, the students are asked to design a system where each train is assigned a route. They must prove that some security requirements are satisfied (no collision, no more than one train per section) as well as efficiency (no deadlock). Once the model has been proved correct, the students must figure out how to translate the net into a set of processes, synchronized using semaphores, which can be run on the model railway. The transformation must be systematic, its soundness and the preservation of properties proved must be justified.
The second view is not intended to mimic a real railway system. Rather, it is related to an adaptative routing system, and henceforth will be named hereafter the "adaptative routing system view". The behavior of trains must be adapted to local conditions. Namely, at each switch, their route can be chosen among several tracks and a train may even go back when impossible to continue forward. Although surprising at first glance, such behavior of trains offers several complex routing possibilities and thus appears as a challenge. The students must design a routing policy so that the same security and efficiency requirements as

before are fulfilled. Then they have to deduce from the net the program of a controller, and implement it on the railway system.

2 The Model Railway

The model railway is depicted in figure 1. It consists of about 15 meters of tracks, divided in 16 sections (blocks B1 to B16) plus 2 sidetracks (ST1 and ST2), linked using 4 double or triple switches and one crossing. The way the trains can circulate on the switches and the crossing is indicated by the arrows in figure 1. The traffic on all tracks can go both ways. Although one can notice that switch 1 (and also switch 2) is composed of two elementary ones, it is managed as a single unit, due to the short distance between the two physical components. The railway is connected to a computer via a serial port which allows to read information from sensors and send orders to trains through the tracks or directly to switches. A section is equipped with one sensor at each end, thus allowing to detect the entrance or exit of a train. The orders sent to trains can be either stop or go forward/backwards at a given speed.

3 Specification Using Coloured Petri Nets

In this section, we describe the different steps encountered by a typical student[1] following the second view, i.e. the adaptative routing system described at the end of section 1). We will give hints about the differences with the real railway view in section 3.6.

The student is asked to proceed step by step in the construction of the coloured net representing the model railway. At first, the model only describes the main loop, i.e. the railway without the 2 sidetracks and the crossing. The analysis of this net is performed for three, four and five trains. At first there is an obvious deadlock which is corrected. Then another one arises when adding an extra train. This is repeated until we obtain a model which is satisfactory for five trains. All the corrections made correspond to a policy for managing the four switches and moving between contiguous sections. Then the two sidetracks are added and the analysis is done directly for five trains. Finally, the crossing is added. Due to the large amount of memory (and time spent) to generate the occurrence graph and check properties, the verification is first performed for four and then for five trains.

3.1 Design of a Hierarchical Coloured Petri Net

One of the requirements of such a project is to obtain a model which can easily be understood. This allows, as we will see later on, to facilitate the understanding of errors encountered when analyzing the model with the occurrence graph. Their correction also becomes much more natural.

[1] in fact, each project is conducted by a pair of students.

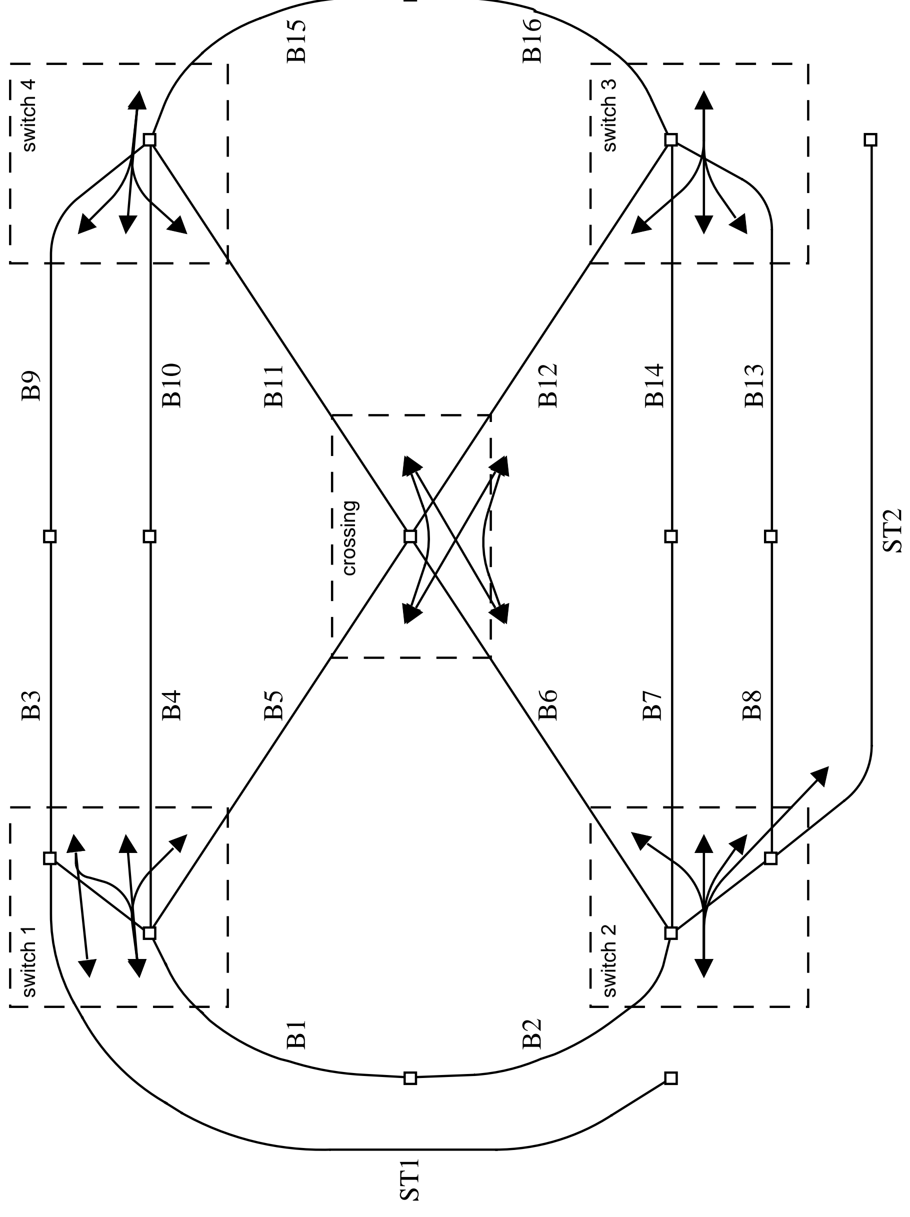

Fig. 1. The tracks of the model railway.

To do so, the use of page hierarchies was required: the prime page represents the whole railway, without any consideration of the policy used to move from one section to the next. This policy is described in subpages, corresponding to the different switches and moves between contiguous sections. A single look at this prime page shows the current state, i.e. where the different trains are located. The similarity between the physical railway model (figure 1) and the prime page (figure 2 for a partial representation and figure 7 for the full one) is easily noticed. The places represent the sections (they have the same names in both figures), while the transitions indicate the possible moves.

3.2 Model of the Main Loop

The students were asked to design their model step by step. Therefore, they started with just the main loop, i.e. the railway without the two sidetracks nor the crossing in the middle.

The prime page of the first net is shown in figure 2. Each place represents the railway section with the same name. It always contains one token, with a value characterizing the state of the section, i.e. either a train is in the section, or the fact that there is none. This is expressed with the union type:

```
color section = union t:train + none;
```

To stick to the real system as much as possible, the student generally chooses to describe trains with both a name and the way they are running, namely clockwise (`cl`) or anti-clockwise (`acl`).

All the transitions are substitution transitions, i.e. they must be substituted by a net with the same input and output places. This can be seen for e.g. transition t1 of figure 2, which has a box next to it with the name of the subpage to be substituted to the transition (changesect) and the correspondence between the names of input and output places of both nets (place B3 of figure 2 corresponds to place P2 of figure 3). The transitions are of two kinds:

- those allowing to move between two contiguous sections with no choice; a reasonable hypothesis is that all transitions of this kind have the same behavior. Therefore, all these substitution transitions will refer to the same subpage. Initially, we will assume that a train can move to the next section if this section is empty, as depicted in figure 3.
- the switches which can permit either to move from a unique section to two different ones or vice-versa. This is described in figure 4, where the net corresponds to one possible direction of trains (switch 1). Another page describes the other way round. These nets could have been merged into a single one, with a place containing the direction used by the switch. That choice would have lead to a less intuitive interpretation of the prime page, and to a unique page used by all the switches. Moreover, when enhancing the model to add the other parts of the railway, the switches become different. Similarly, some students have used two instances of the net in figure 3 to model the simple switch, but then, when changing the policy for moving

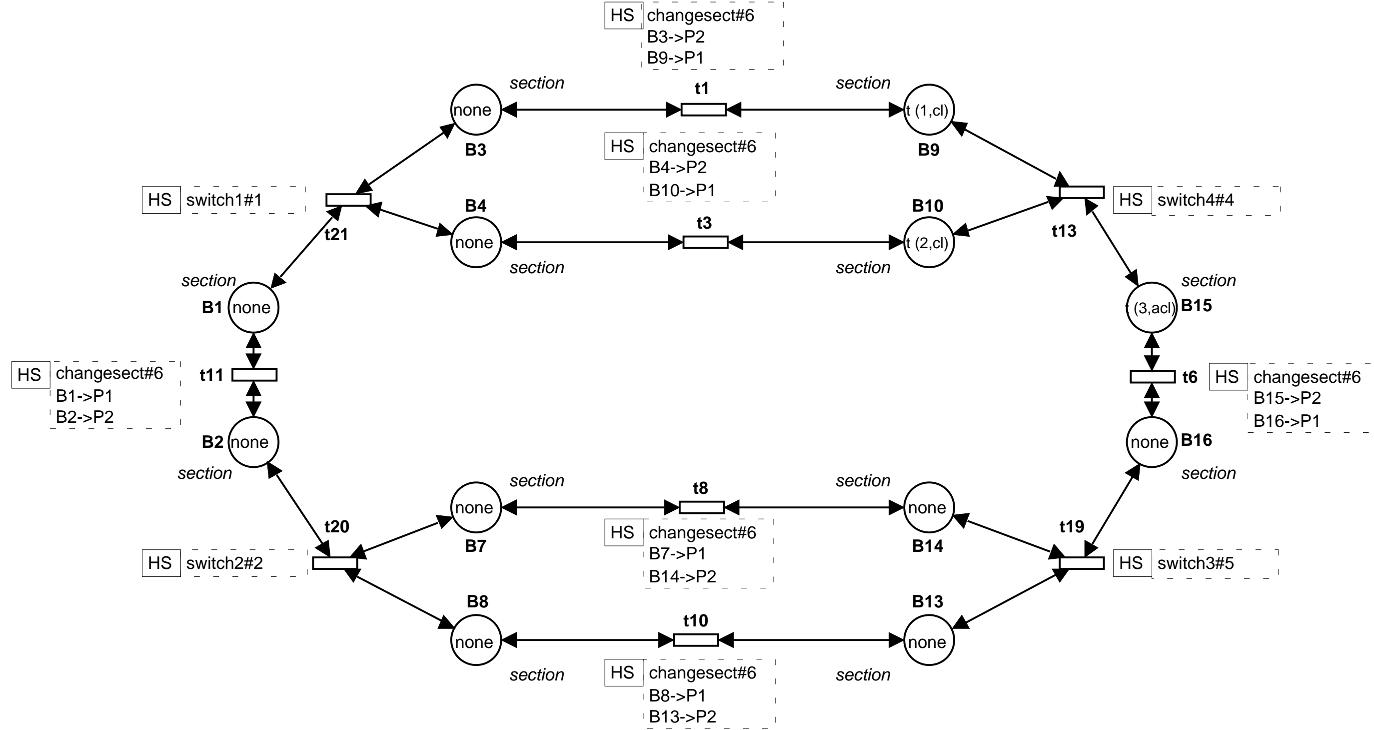

Fig. 2. The prime page of the main loop model.

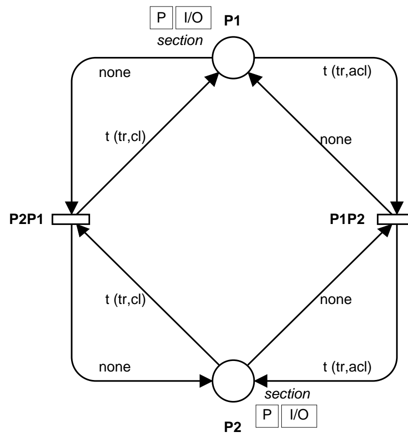

Fig. 3. Moving between two contiguous sections.

from one section to the next, the management of switches was also modified. This is often undesirable.

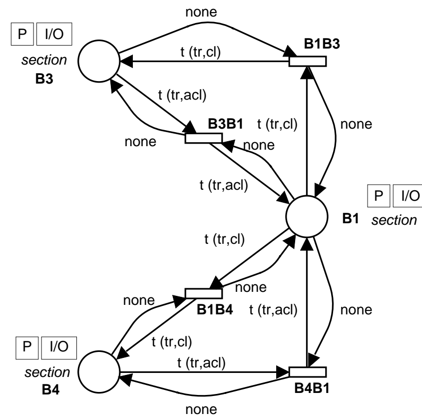

Fig. 4. A simple switch.

Once the net is designed, the students starts examining its behavior. It is obvious that the initial marking (with 3 trains) given in figure 2 is a deadlock. Thus, the student must design a less simple policy for switches, depicted in figure 5.

The switches are (for the moment) composed of two tracks arriving from the

same direction (such as B3 and B4) and one from the other side (as B1). When 3 trains arrive altogether on the switch in a deadlock situation, the new policy consists in having the train on the single track side (e.g. the train on B1) go backwards. Such a possibility takes sense only in the adaptative routing system, but not in a real railway system. This is modeled by transition `retB1` in figure 5.

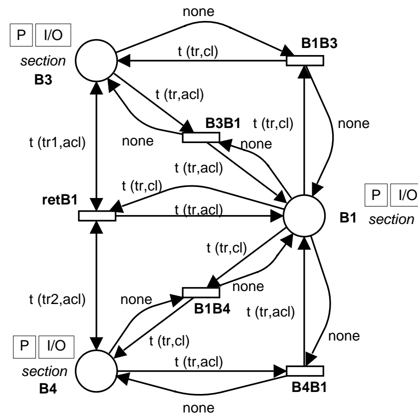

Fig. 5. New policy for switches.

It is easily noticed that the initial marking with three trains will enable only the new transition, and thus only the train going anti-clockwise will change direction. Thereafter, all the trains will go in the same direction, i.e. clockwise. To have interesting results, the student adds an extra train, and analyses his new net.

3.3 Verification with Four Trains

Now, the new policy for switches is used and a fourth train is added, by changing the initial marking of place B3 into `t (4,acl)`, and the value `n=4` in the declaration node.

The analysis of the model is performed by means of Design/CPN occurrence graph tool ([Jen94], [CPN96]). Once the graph is generated, a standard report and some additional properties are checked. This allows to discover nodes satisfying undesired properties, as explained later. To facilitate the understanding of the trains situation, the feature provided by Design/CPN which allows to transform a state of the occurrence graph into the corresponding marking in the simulator is used. Thus, the prime page is updated with the unwanted marking, and the user can see in a glimpse the locations of trains.

Analysis Results After a single occurrence graph generation, faced with the huge number of states, the student often guesses that the trains names highly contribute to the state space explosion, although they are useless, from a verification point of view. Effectively, in the problem considered, it is not necessary to know where a particular train is, but only that there is a train in a particular section, and also where it is heading. This could be formalized using symmetries, but we lack time to teach the theoretical and syntactical aspects. Moreover, the marking equivalence derived from this symmetry is totally obvious.

Occurrence Graph with Trains Names The occurrence graph obtained with four named trains contains 21.574 nodes and 72.026 arcs. It is computed in 334 seconds[2]. The strongly connected components (SCC) graph has 1.243 SCCs and 6.666 arcs. It is calculated in 18 seconds.

Occurrence Graph without Names When removing the trains names, i.e. only the direction the train goes is kept in the token value, the occurrence graph obtained, still with four trains, has 2.166 nodes and 7.157 arcs. It takes 5 seconds to obtain it. The SCC graph has 237 nodes and 1.245 arcs. It is computed in 1 second.

The following step in the project is the verification of the net. The student must formalize what does a correct behavior of the system mean, and indicate the properties that should be satisfied. Having some experience, through exercises, of the properties provided by the standard report of Design/CPN, the student tries to figure out how they can be used in order to prove the correctness of his model. This function allows to obtain in a file a textual result for the usual net properties (e.g. bounds, dead markings, liveness, ...). It turns out that most properties are safety properties and can be checked using only the standard features.
The results and their interpretation are the following :

– all the lower and upper best integer bounds are 1. Thus *there is always exactly one token in each place*;
– all the best upper multi-set bounds are `1‘t(cl)++ 1‘t(acl)++ 1‘none`. This property, together with the previous one, shows that *each section can contain either exactly one train going one way or the other, or none*;
– there are *6 dead markings*. At first, the student is surprised. The evaluation of function `ListDeadMarkings();` provides him with the list of all dead markings. Then, he visualizes each of these markings on the prime page, using the feature of Design/CPN which puts a given marking of the occurrence graph into the simulator. Three of the dead markings have a train going clockwise in `B15` and a train going anti-clockwise in `B16`. The three other ones are similar, with sections `B2` and `B1`.

The student comes to the conclusion that the policy to move between two contiguous sections with no choice has to be improved.

[2] all the results in this paper were obtained on the same machine: a Linux PC Pentium II 450 MHz with 256 Mb of memory. The computation times are those given by Design/CPN.

Policy Adopted In order to avoid the previous situation, the student decides to have both trains go backwards, as modeled by transition `retboth` in figure 6, giving the following arguments:

– if both trains return where they come from, they will not travel in the same direction, thus such a policy should limit or delay the possibilities for all trains to go in the same direction;
– both trains are treated in the same manner, there is no priority.

The new model is then analyzed for four and five trains. The results with five trains will now be discussed.

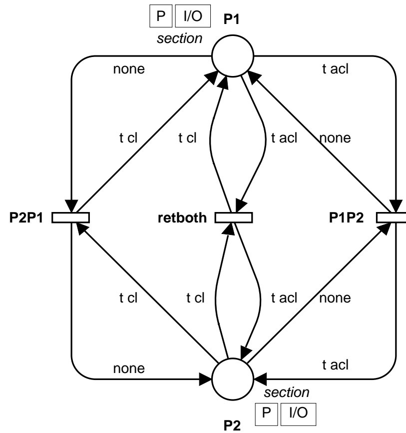

Fig. 6. New policy to move between two contiguous sections.

3.4 Verification with Five Trains

A fifth train going anti-clockwise is added in place `B2`.

Analysis Results The occurrence graph obtained for five trains has 24.556 nodes and 97.020 arcs. It is computed in 430 seconds. Its SCC graph has 615 nodes, 3.128 arcs and was calculated in 18 seconds. The properties obtained from the standard report, show that the bounds are the same as previously, *there is no dead marking nor home marking*.
This last property is quite intriguing for the student. He decides to have a closer look to the terminal SCCs. Therefore, he used the following functions (where the result is given after the arrow):

```
PredAllSccs SccTerminal;        -> [~615,~534]
length(SccToNodes(~615));       -> 792
length(SccToNodes(~534));       -> 792
```

The first function gives the list of all terminal SCCs. They are two, numbered ~615, and ~534. Then, the number of nodes in each of these terminal SCCs is 792. After looking at a marking of each component, the student concludes that this is a normal situation: in SCC ~615, all trains go clockwise while in SCC ~534, all trains go anti-clockwise. The two SCCs remain separated as, considering the routing rules adopted, a train cannot go backwards if it does not meet a train gong in the opposite direction. So, such situations are acceptable w.r.t. the initial requirements.

3.5 The Complete Railway

When the main loop works correctly, the student enhances his model by adding the two sidetracks and the crossing.

Final Model The prime page of the model of the complete railway (see figure 7) has 6 additional places corresponding to the 2 sidetracks and the 4 inner sections, and one extra substitution transition which models the crossing.

A subpage, presented in figure 8, is added to represent the inner crossing in the railway. It takes into account all the possible moves on the inner crossing, as described in section 2. Moreover, the case where four trains want to enter the crossing – and will then be blocked – is treated by forcing the train on `B5` to go backwards.

The four switches subpages are modified to take into account the new possible movements, but their general policy remains the same, just taking into account more possibilities for a train to enter and exit a switch. The nets in figures 9, 10, and 11 give an idea of the complexity of the complete model.

The fourth switch can easily be deduced from the third one, by changing the place names and direction of trains.

Analysis Results As a lot of time is needed to build the occurrence graph of the full model, it was first done with four trains and then with five.

The occurrence graph with four trains contains 48.957 nodes, 228.790 arcs and was calculated in 2.991 seconds (50 minutes). Its SCC graph has 1 node, no arc and is obtained in a bit less than 2 minutes.
The standard report gives us the same integer and multi-set bounds as before. *All the reachable states are home markings.* This is interesting, because it shows that any reachable train distribution can always be reached again. There is no deadlock. But *transition `retB2` of subpage `switch2` is dead.* This means that with these four trains, the situation were a train going anti-clockwise on section `B2` is blocked can never occur. When looking at the design of the railway, the student notices that this is normal as there are 4 possibilities for this train to go forward, and only three can be occupied by the other trains. The question now is: what happens with a fifth train?

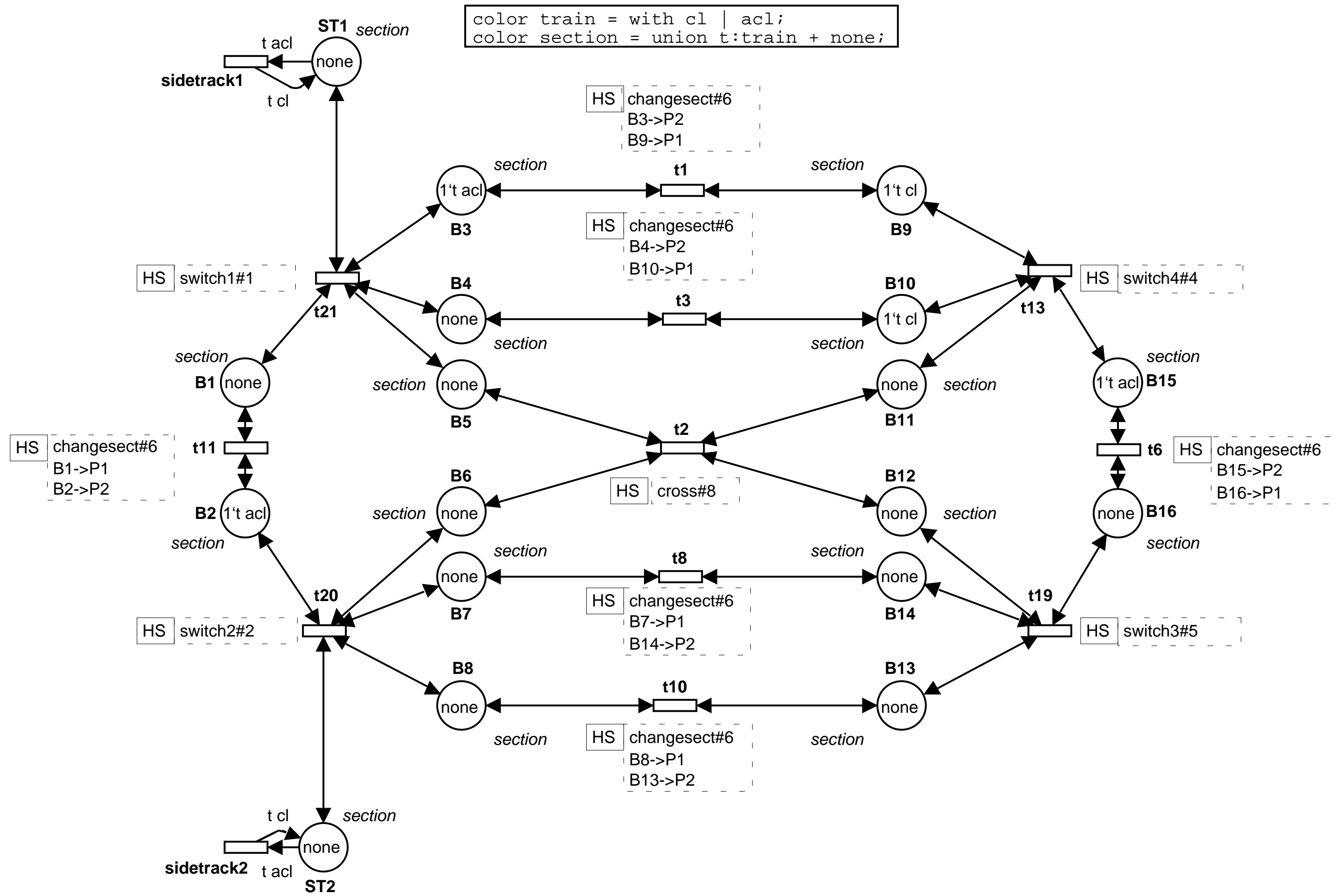

Fig. 7. The prime page of the model railway hierarchical coloured Petri net.

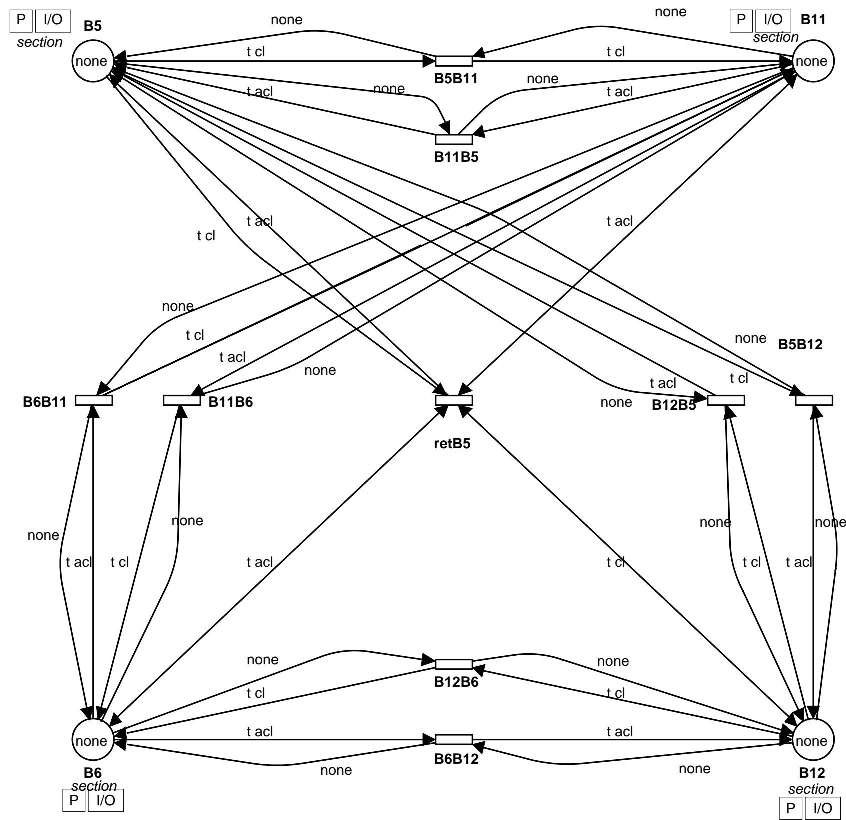

Fig. 8. The subpage modeling the inner railway crossing.

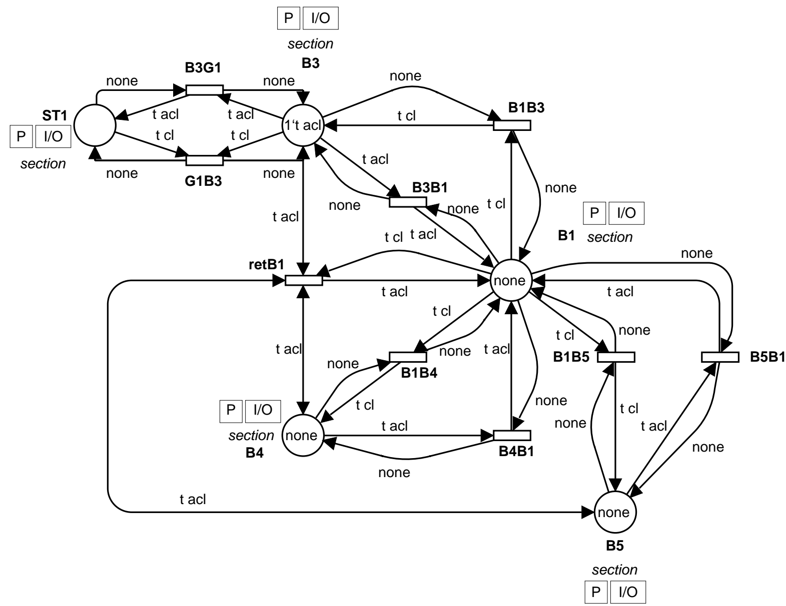

Fig. 9. The first switch (transition t21 of figure 7).

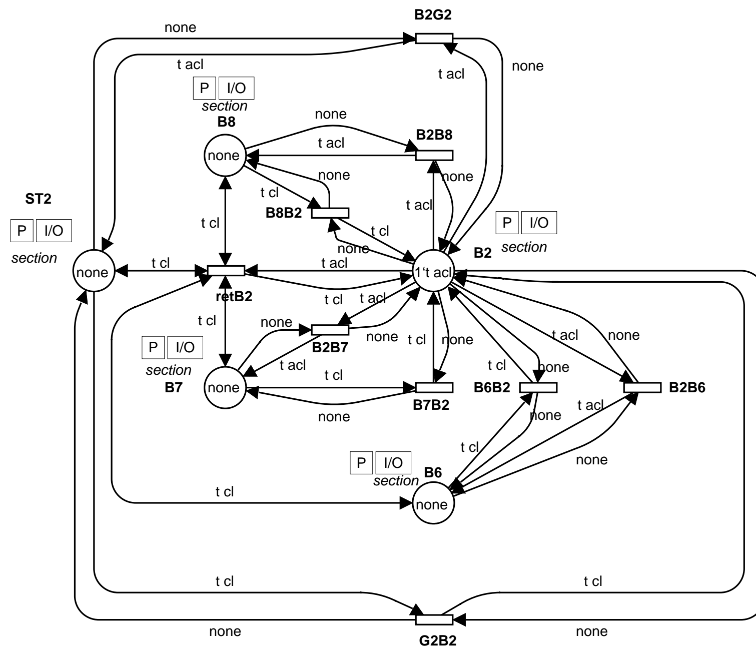

Fig. 10. The second switch (transition t20 of figure 7).

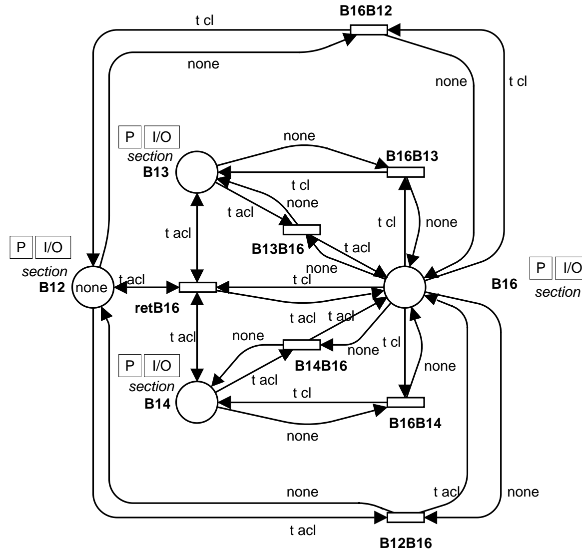

Fig. 11. The third switch (transition t19 of figure 7).

The Occurrence Graph with Five Trains has 274.082 nodes, 1.500.384 arcs and was computed in 103.221 seconds (1 day, 4 hours and 40 minutes)[3], approximately the same time was necessary for the SCC graph which finally contains 1 node and no arc. This informs the student that, as for four trains, all the reachable states are home markings. There is still no deadlock and *all the transitions are live*. Thus the model meets the initial requirements : allow up to 5 trains, avoid collisions and deadlocks.

3.6 Modeling the "Real Railway System"

When the first view (i.e. mimicking a real railway system) is considered, a route must be assigned to each train. This has three consequences: First, when a train enters a switch, it must exit it as specified in its own route, and not by a randomly chosen exit. As a second consequence, trains must keep their identity and cannot become anonymous. Finally, a train may have to book several sections in advance in order to avoid deadlocks: e.g. if a train in B7 wants to enter B2, it should also make the reservation for B1, otherwise it can be blocked by a train coming the other way round. As concerns modeling, a transition must not only be connected by the arcs to places representing the next and/or previous resource, but also to the places representing the resources necessary one or more steps later. Hence, the net is graphically less close to the physical railway, and also it is more difficult to create a hierarchical net as there are special transitions

[3] with intensive use of the 1Gb swap space

associated with each train.
The number of states in the occurrence graph strongly depends on the length and the complexity of the routes designed by the students. Another big difference with the second approach is that verification mainly consists in deadlock detection between trains. It is obvious that complex routes and the simultaneous reservation of four sections can lead to deadlocks very difficult to foretell. To conclude, the complexity of the model and the possibility to fully check it depends a lot on the options chosen by the student. This is the reason for explaining the other view.

4 From the Specification to the Implementation

When the model of the net has been analyzed, and its desired properties proved, the student has to translate it into a `C` program, having a behavior as close as possible to the net's one. To do so, he has roughly two solutions: the first one consists in writing a controller (a Petri net simulator), the second is to split the net into a set of synchronized processes. The student naturally tends towards the first solution, which is closer to sequential programming and automata theory, but, after a deeper insight, some of the students accept to try the second solution. We shall now describe both in more detail.

In the simulator approach, the color sets and associated operators are translated into `C` data structures and functions. Then, each transition is splited into a boolean precondition function and a post function. The former allows to check if the transition is firable, while the latter is called when firing to change the marking and transmit orders to the hardware. The core of the simulator is an endless loop which, at each step, scans the transitions list to see which ones are firable, then fires some of them. It should be noticed that, contrary to what happens in a general simulator, the binding of variables of transitions is quite easy since each place contains exactly one token. Nevertheless, when several transitions are enabled, a fair choice must be provided, using for instance a random number generator. A companion process is in charge of reading the information from the hardware, and update accordingly the state variables.

In the second solution, which is more natural when using the "real railway view" because the trains are seen as independent entities which travel along a route, the set of transitions of the net must be partitioned in order to build up processes which have to synchronize using semaphores. The most intuitive and logical way to do so is to associate each train with a proper process. With this approach, each part of the railway must be considered as a critical resource. The student can directly apply the classical Dijkstra method([Dij65]) to access resources. Each resource is described by a set of state variables. Processes must request resources using a general mutual exclusion semaphore. If a process is blocked, it leaves the critical section and hangs up on a private semaphore until it is awaken by another process. With this approach, no fairness problem appears at first, provided that semaphores have FIFO queues. However such a problem arises when a train arrives at a fork with several possible exits, and several

are compatible with the route. The student must add a mechanism to choose randomly which transition to try. As in the previous approach a companion process handles the hardware inputs.

The program obtained is then downloaded on the PC which monitors the train model and the student may conduct some experiments. The program is often rapidly effective, after some tuning (speed of trains), addition of hardware commands (switches, crossing), and detail improvements (traffic lights). The student can see that no real problem arises, except from bad hardware management (if a sensor does not "see" a train, this one becomes a "crazy train" which must be physically removed as soon as possible). However, most of times a long run results in a periodic behavior, even if the theoretical analysis predicted a less rigid scheme. This is due to the fact that even with a fair program, first come trains are also first served ones at critical points, and the delay to run through a block depends mainly on its length. So, the model train can adopt only a subset of behaviors of its theoretical model. The exact analysis would require to take into account time aspects, but it was not tried yet.

5 Conclusion

In this paper, we have reported a teaching experience using a train model and high-level nets. A similar experience for modeling a train system is described in [HURK98], but the students could not achieve verification.
In spite of its childish aspect, the train model forces students to touch, master and manage all the notions rendering parallel programs error prone: critical section, deadlock, fairness, time and combinatorial explosion. Moreover, it is a very strong evidence that such programs cannot be developed from scratch (bottom-up style) as students tend to do, but must be rigorously modeled and analyzed before implementation. An additional benefit from this project consists in tackling a process control problem.
From students' opinion, it appears that handling a real system helps and motivates both to model the system and to formalize the properties that it should satisfy. However, they are disappointed by the state space explosion problem which arises quite early in the verification. Thus only restricted or more abstracted problems can be validated.

A further step would be to manage time aspects. An automatic code generation feature would have helped a lot for the last step of the project, i.e. the implementation part.

References

[CPN96] META Software and Aarhus University. *Design/CPN 3.0*, 1996. Also available as: `http://www.daimi.au.dk/designCPN`.

[dAO94] W. M. P. Van der Aalst and M. A. Odijk. Analysis of railway stations by means of interval coloured Petri nets. *Real-time systems*, 9:1–23, 1994.

[Dij65] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9):569, september 1965.

[HURK98] W. Hielscher, L. Urbszat, C. Reinke, and W. Kluge. On modelling train traffic control in a model train system. In *Proc. 1st CPN Workshop*, DAIMI PB 532, pages 83–101. Aarhus University, 1998.

[Jen92] K. Jensen. *Coloured Petri Nets: Basic concepts, analysis methods and pratical use. Volume 1: basic concepts.* Monographs in Theoretical Computer Science. Springer, 1992.

[Jen94] K. Jensen. *Coloured Petri Nets: Basic concepts, analysis methods and pratical use. Volume 2: analysis methods.* Monographs in Theoretical Computer Science. Springer, 1994.