# On Modelling Train Traffic in a Model Train System

Wolfgang Hielscher, Lars Urbszat, Claus Reinke, and Werner Kluge
Department of Computer Science
University of Kiel
D–24105 Kiel, Germany
E–mail: wk@informatik.uni-kiel.de

May 15, 1998

**Abstract**

*The paper describes the design of a coloured Petri net model for a rather complex model train system. The purpose of this system is to teach graduate CS students net modelling and analysis techniques, and the systematic concersion of non–trivial net models into fully operational real systems.*

*The track layout of this system currently includes three main cyclic tracks, each subdivided into several sections, three switchyards of several sidings, and also interconnecting tracks via which trains may change main tracks and directions.*

*The idea is to equip each of several trains - currently up to ten - with its own travel plan. It specifies a sequence of tracks through which the train must be routed in the given order. Execution of these plans must be dynamically coordinated based on locally made decisions about the allocation of track sections to requesting trains so that essential safety and liveness properties are met.*

*The paper first introduces the basic net components necessary to model train movement along track sections and across branching and merging switches, then describes the composition of a complete track model from these components, including the controls necessary to enforce an orderly behaviour, and then outlines the composition of the complete system model. It also addresses some of the as yet unsolved problems of deadlock prevention in the system.*

## 1 Introduction

This paper relates to the organization of an orderly train traffic in a complex model train system which serves to teach graduate students how to model, by means of coloured Petri–Nets, the dynamic behaviour of non–trivial real life systems with concurrent activities and how to translate these models into working control programs.

The track layout of this train system is depicted in fig. 1. It consists of

- three main circular tracks called the outer circle (labeled OC_LN), the inner circle (labeled IC_LN), and the kicking horse pass [1] (labeled KH_LN) along which trains may move counterclockwise, clockwise, and in both directions, respectively, as indicated by the arrows;
- switchyards of three to five sidings included in each of the main tracks (the outer circle station (labeled OC_ST), the inner circle station (labeled IC_ST) and the kicking horse pass station (labeled KH_ST)), and also

[1] This name is adopted from a section of the Canadian Pacific Railways mainline in the Rocky Mountains which includes a similarly spiral–shaped track layout to negotiate a rather steep uphill / downhill passage [Po95].

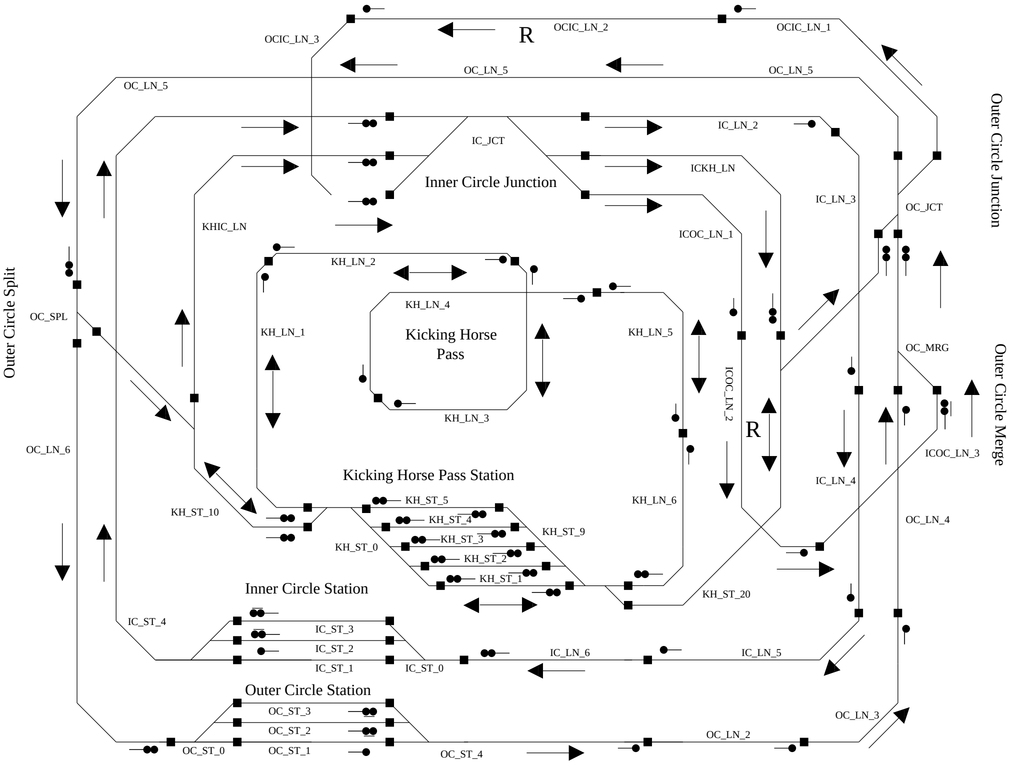


Figure 1: Complete track layout of the system

- interconnecting tracks, labeled OCIC_LN and ICOC_LN, through which trains may change main tracks and thereby also change directions without needing to shunt engines;

Each of the tracks is subdivided into several sections (or blocks), in the figure separated by little square–shaped dots. Light signals, symbolized as little dots extended by short bars, visualize permission to cross section boundaries. Actual train positions are detected by magnetic sensors placed along the tracks, and the speed of trains is controlled by applying appropriate voltage to track sections.

Sensor data are, by a dedicated digital interface, scanned and passed on, as a stream of 256 bytes, to a SUN-Workstation. These sensor data are translated into a description of the current system state, from which a control program executed by the SUN derives the control signals that effect transition to a next state and issues them, through the same interface, as a stream of 256 bytes to the switches, signals and the voltage supplies. This control cycle is executed in periods of a few milliseconds to react quickly enough to critical situations, with trains moving at most some 10 mm during that time.

The track system is laid out on an area of $4.5 * 3.5$ square meters. It includes 145 meters of HO gauge tracks, subdivided into 37 sections and sidings, 28 two-way switches and 51 signals. The inner and outer circles are configured as intertwined loops of about 28 meters length each, running in parallel, and the kicking horse pass includes two double spirals on which trains climb up to (and down from) about 30 centimeters above ground level on a track length of about 25 meters. It takes about two minutes for trains to complete a single lap on the inner and outer circle, and about two and a half minutes over the kicking horse pass.

The photograph of fig. 2 gives an areal view of the track layout, showing the harp-like structures of the station sidings on the left, the tracks of the outer and

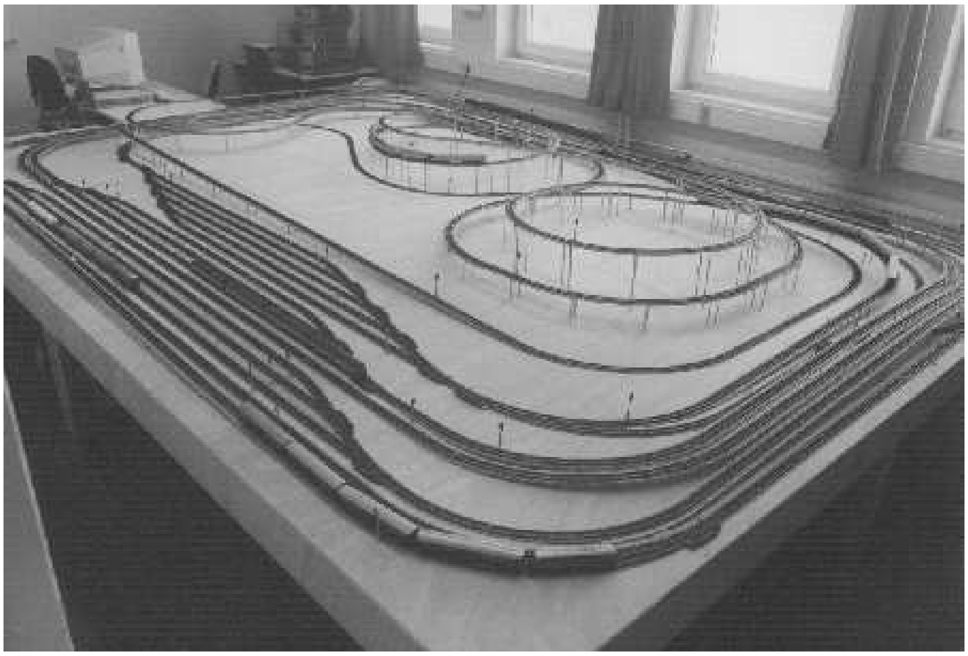

Figure 2: A birds eye's view of the system

inner circles running along the periphery, and the spirals of the kicking horse pass in the center part.

The design of this track system was guided by the objective of providing, within the confines of limited real estate, sufficient complexity with regard to train movement - currently up to 10 trains can be operated simultaneously - so that all the phenomena that typically occur in systems with concurrent activities can be studied and experimented with in a real-life setting.

To this end, train movement is governed by travel plans which for each train individually specify which (sections of) tracks it has to pass through in which order at which speed, and at which stations it has to stop for some time. Each plan sets out with some initial positioning of the train in a specific station siding, and after some finite number of passages through main tracks terminates with the train arriving at the same or some other siding [2]. A typical plan for a train that passes through all three main tracks is given in fig. 3.

The control program that receives these travel plans as input must coordinate their execution 'on the fly', based on local decision making as conflicts and potential deadlock situations arise. In doing so, it should obey first principles of an orderly system behavior, defined in terms of essential safety and liveness properties:

- each track section must be occupied by at most one train at a time (the mutual exclusion principle);
- a train that occupies a particular section of track, including station sidings, must always find a way out unless it has reached its terminal position (completed its plan);
- no train must be unduly delayed in pursuing its travel plan: entry into some track section must be granted eventually to all trains competing for it;

[2] Choosing the same siding as the starting and terminal position has the advantage that the same plan can conveniently be repeated several times in succession (or even forever).

```
train "freight train #17":
   operation speed   6
   minimum   speed   1
   priority          10

   # initial position: siding #1 in the kicking horse Pass station
   # initial heading:  clockwise
   from KH_ST_1 CLK

   # Move to Inner Circle Station and stop there for (at least) 10 seconds.
   goto IC_ST  stop 10 sec

   # Take 3 turns on the Inner Circle without stopping. Upon
   # completing the last turn, stop for 15 seconds.

   loop 2
      goto IC_ST
   endloop
   goto IC_ST  stop 15 sec

   # Go to the kicking horse Pass Station heading counterclockwise,
   # go over the pass once, then move to the Outer Circle Station,
   # stop there for 17 seconds, and finally return to the initial
   # position KH_ST_1.

   goto KH_ST CNTCLK
   goto KH_ST CNTCLK
   goto OC_ST         stop 17 sec
   and return
```

Figure 3: Travel plan for a single train

- a train requesting entry into a track section must be permitted to proceed immediately if no other train competes for it.

A system which meets these essentials can usually be modelled as an ordinary Petri–net and by proving that it satisfies certain invariance properties [GeLaTh80, KlLa82]. However, things are not that simple if, in addition to these essentials, individual travel plans and other train–specific parameters have to be included into the model. Such parameters may be priorities, e.g., of passenger trains over freight trains, time tables which need to be followed (with trains falling behind getting their priorities dynamically stepped up), or pre–specified speed profiles along the travel routes. It takes higher–order Petri–nets [GeLa78, GeLa81, Rei85, Jen90, Jen92] to represent trains as tokens which carry along with them inscriptions to this effect and to have the firing of transitions in some consistent form controlled by these parameters, and the parameters changed if necessary.

The sequel outlines the design of such a net model for the entire system, using the Design/CPN tool version 3.0.4 of Aarhus University for coloured Petri–nets. The design is described from the bottom–up, setting out in the next section with basic net components which model train movement across section boundaries and across merging and branching switch configurations. Section 3 describes in some detail the net model for the kicking horse pass, including the station and the track sections that lead into and out of it, and section 4 briefly explains how the complete system model is assembled from net abstractions (substitution transitions) for the inner and outer circle tracks and for the kicking horse pass track. Section 5 discusses some open problems of deadlock prevention within the system, and the conclusion summarizes some of the experiences we had with the Design/CPN tool.

Since the system model is rather complex – its full net specification requires some 28 pages and a total of about 400 places – the paper merely attempts to convey some basic features of its construction and its interpretation (semantics), but it cannot be very specific on all details. In fact, the net models of the kicking horse pass and of the complete system have been cleaned of some components which relate to rather special control problems in order to get the underlying ideas across more clearly.

# 2 The Basic Net Components

This section is to introduce the net components by which the basic building blocks of the track system and the train movement therein can be modelled.

## 2.1 Track Sections and Train Descriptions

For purely organizational purposes it suffices to model each section of track by a place with a color specifying the type of tokens it may carry. As these tokens must represent the presence or absence of trains in the section, and a train must be characterized by a number of attributes (or parameters), the color `block_descr` associated with these places must be of the general form

$$\mathsf{color}\ \mathtt{block\_descr} = \mathtt{union}\ \mathtt{u\_train} : \mathtt{train\_descr} + \mathtt{no\_train},$$

where the colors `train_descr` and `no_train` respectively specify the type of a train description and the absence of a train in the section. For reasons that will become clear when specifying the net component which controls train movement from one section to the next it must be guaranteed that a token of either type resides in such a place under all markings (token distributions) of the net. The color `train_descr`, in turn, is defined as [3];

$$\mathsf{color}\ \mathtt{train\_descr} = \mathtt{product}\ \mathtt{t\_state} * \mathtt{t\_train\_id} * \mathtt{t\_dir} * \mathtt{t\_sched}$$

with

- `color t_state = with moving | stopped` distinguishing between the train moving through or temporarily stopping in the section;
- `color t_train_id = ( 1 .. n )` serving as indices to distinguish the `n` trains moving about the tracks;
- `color t_dir = with clk | co_clk` distinguishing between clockwise and counterclockwise movement along the tracks;
- `color t_sched = list t_dest` defining the trains travel plan in terms of switch positions of type `color t_dest = with left | straight | right` .

Specifying travel plans as lists of switch positions which must be interpreted from head to tail is a low–level description which suffices to route trains from start to destination: choices among alternative routes must only be made at sections where tracks branch into two or more, otherwise next sections are unique [4].

The color `train_descr` is minimal, in terms of numbers and types of its components (attributes), with respect to an orderly coordination of the train traffic about the track system. No provisions are as yet made to specify priorities or speed profiles.

[3] The prefixes `t_` are to distinguish the colors (types) within the type product from the variables used in the nets to identify the components of train descriptors.

[4] These low–level travel plans can be readily derived from high–level specifications as exemplified in fig. 3.

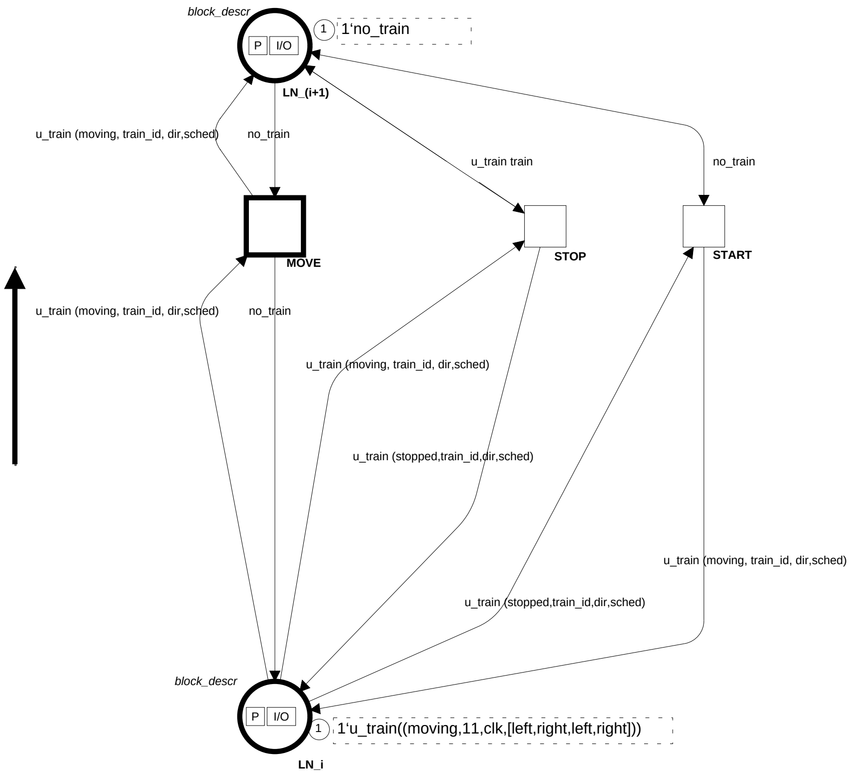


Figure 4: Net component for unidirectional train movement between adjacent track sections

## 2.2 Moving Trains from one Section to the Next

Modelling the unidirectional movement of a train along track sections requires a net component as shown in fig. 4. It includes two places LN_$i$ and LN_$(i+1)$ which model the sections from where to where a train is supposed to proceed, and three transitions which analyze the tokens in both places to take appropriate actions [5] . In particular, the transition

- MOVE is enabled if a train is in section LN_$i$ and moving, and no train is in LN_$(i+1)$, as indicated by the token inscriptions, in which case the transition fires and exchanges the two tokens (i.e., the train moves on);
- STOP is enabled if a train moves through section LN_$i$ and section LN_$(i+1)$ is occupied by another train, in which case the state of the train in LN_$i$ must be changed to `stopped`.
- START returns the status of a stopped train in section LN_$i$ back to `moving` if the train that occupied section LN_$(i+1)$ has moved on and left.

There can be at most one of these transitions enabled as the respective markings are mutually exclusive.

A minor extension of this net component is necessary to deal with track sections in which trains are not allowed to stop. They include the inner and outer circle junctions through which trains may change main tracks, but also other sections

[5] Three transitions are used for structural clarity here. They can, of course, be folded into one in which the three situations that need to be taken care of are distinguished by appropriate inscription.

with switches in them. Here it is necessary to make sure that not only the no-stop section but also the section which the train must reach after it are free. This can be accomplished by means of a so-called look–ahead place, denoted as LA_ ..., of `color t_la = with free | occ` connected to a CTR_ ... transition which imposes an additional condition on its firing: it is disabled if the token in this place carries the color `occ` and, depending on the tokens in the places LN_$i$ and LN_$(i+1)$, may be enabled if the color is `free`. The token that actually resides in place LA_... is generated by some look–ahead subnet which inspects the tokens in the relevant section places.

To model adjacent track sections along which trains may move in both directions (as in the kicking horse pass), the transitions MOVE, STOP and START must simply be duplicated, with one set each taking care of clockwise and counterclockwise direction.

### 2.3 Controlling Train Movement Across Switches

Switches in both unidirectional main tracks (the inner and outer circles) are operated as either

- branching switches through which trains are routed from an incoming track section to one of several outgoing track sections;
- merging switches through which trains are routed from one of several incoming sections to one outgoing section.

Switches within the kicking horse pass are operated both ways, depending on the direction in which trains are going. Inner and outer circle junctions are made up from a set of merging switches followed by a set of branching switches. The switches themselves belong to no-stop sections between the incoming and outgoing sections [6].

In the simple setting that nothing other than the presence (or absence) of trains matters, merging switches must be operated as follows: If there is only one train in any of the incoming sections and both the no-stop switch section and the outgoing section are free, then the train may move on without delay by setting the switch(es) accordingly. If there are two or more trains in the incoming sections competing for passage, then the conflict is resolved by arbitration. A typical example is the net component of fig. 5 which models the merge switch through which trains leave the sidings of the outer circle station [7]. It consists of a place each for the three incoming sections (the sidings) OC_ST_1, .. , OC_ST_3, for the no-stop section OC_ST_4, and for the outgoing section OC_LN_2. The transitions CTR_OC_ST_1, .. , CTR_OC_ST_3 which are to swap train tokens between the sidings and the no-stop section are controlled by look–ahead places LA_OC_ST_I which receive control tokens from some look–ahead subnet encapsulated in the substitution transition LA_OC_ST[123]. This subnet monitors whether there are trains in any of the sidings and both the no-stop section and the outgoing section are free. In this case, the look–ahead subnet injects a `free` token as a selector in one of the look–ahead places which has the respective siding occupied by a train, whereas all other look–ahead tokens remain set to `occ`. This enables the selected train token to proceed to the no-stop section (whereupon the look–ahead token is reset to `occ`) and, via the MOVE transition, on to the track section OC_LN_2. The look–ahead subnet selects competing trains based on fair simulation in order to prevent waiting trains from being unduly delayed. As an additional safety measure which relates to implementation details of the controls,

[6] $n$–fold switches of either kind are technically implemented as cascades of $n-1$ two–way switches.

[7] In this and all the following nets the tokens are omitted since inscriptions that are readable would take up too much space.

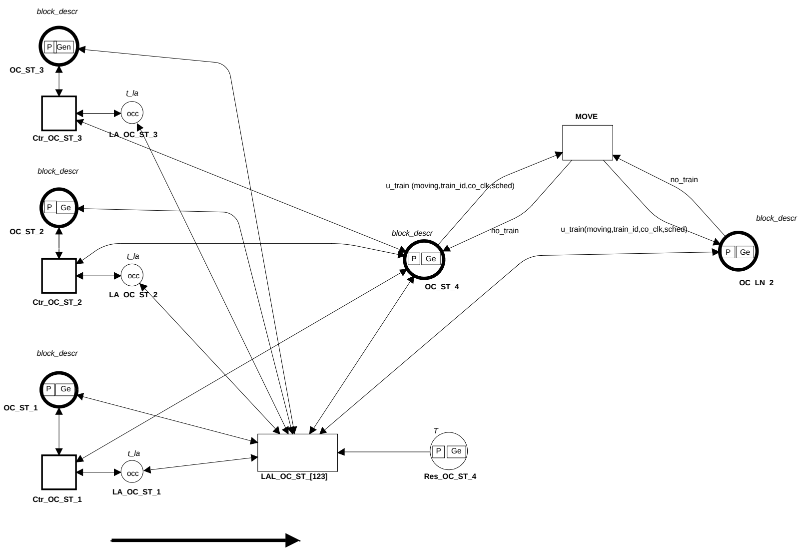


Figure 5: Net model of a three–way merge switch

the firing of the look–head transition is also made dependent on the consumption of a plain control token from the place RES_OC_ST_4, which is returned if the train token is removed from place OC_LN_2, i.e., the train leaves the outgoing section. This measure ensures that no train is in either the no-stop or the outgoing track section before one of the CTR_OC_ST_$i$ transitions can be enabled.

Branching switches come in a non–deterministic and in a deterministic version. The former is to control entry of trains into any of the free sidings of a station where it does not matter which one is actually chosen, the latter is to control selection of a specific outgoing track.

Consider, as an example, the net component for the branching switch as depicted in fig. 6 which non–deterministically controls entry of trains into the outer circle station. It comprises a place each for the incoming track section OC_LN_6, for the three outgoing track sections (the sidings) OC_ST_1, .. , OC_ST_3, and for the no-stop switch section OC_ST_0. The firing of the entry transition CTR_OC_LN_6 is made dependent on a look–ahead place LA_OC_LN_6 whose marking is worked out by some look–ahead logic inside the substitution transition LA_OC_ST_0. This logic checks whether any of the outgoing sections `and` the no-stop section are free. This being the case, it injects a `free` token into LA_OC_LN_6 to allow the train token to proceed to OC_St_0 and from there across one of the enabled MOVE transitions to a free siding; otherwise it is not enabled, i.e., the `occ` token remains in place, which stops the train in the incoming track section.

An example for the net model of a deterministic branching switch is the one that allows trains to either change from the track section OC_LN_5 of the outer circle to section KH_ST_10 of the kicking horse pass station or to continue along the outer circle track to section OC_LN_6 (see fig. 7). Apart from different labels for places and transitions, it basically differs from the net model for the non–deterministic branching switch in that it includes another place DEST which is fed with a selector token `left` or `right` extracted from the train token as it enters OC_LN_5 (which is not included in this subnet). This token enables either of the transitions CTR_SPL_*left* or CTR_OC_SPL_*right* to route the train token in OC_LN_5 to either KH_ST_10 or OC_LN_6, respectively.

Deterministic switches are in the entire track system used only unidirectionally. Non–deterministic switches may also be used, in reverse direction, as merging

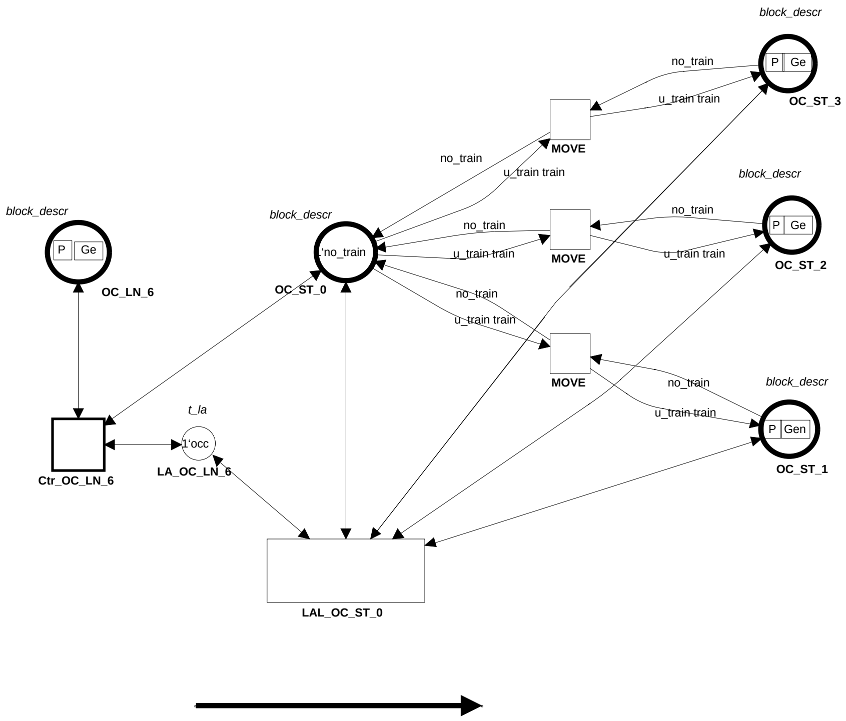


Figure 6: Net model of a three–way non–deterministic branching switch

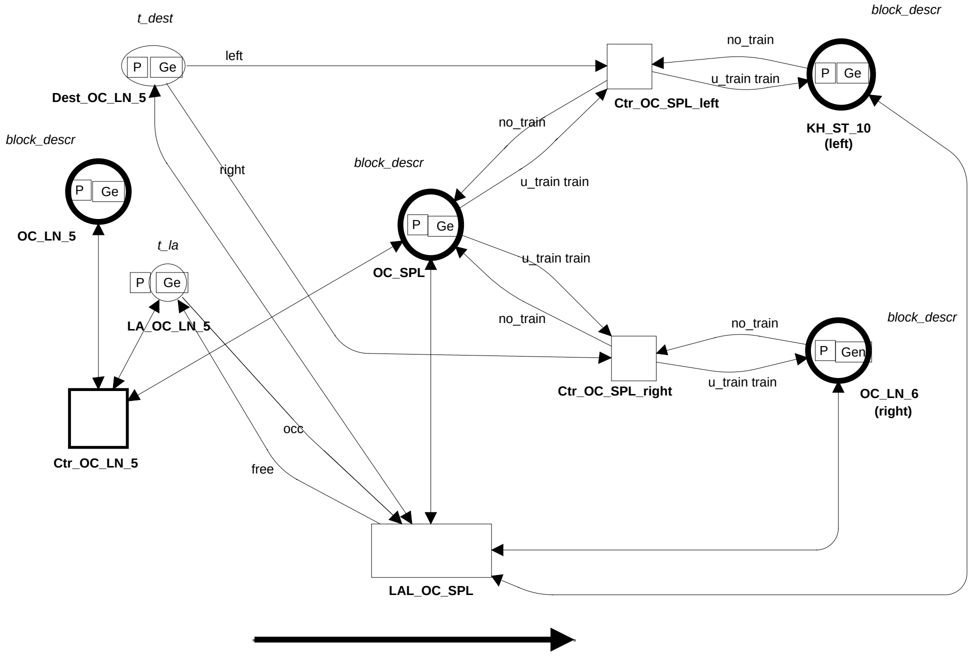


Figure 7: Net model of a two–way deterministic branching switch

switches, and vice versa as, for instance, at both sides of the kicking horse pass station. All it takes to operate these switches both ways is to equip the respective nets with the controls of both figs. 5 and 6.

# 3 Modelling the Kicking Horse Pass

With the net components as introduced in the preceding section, it is fairly straightforward to compose larger subnets which model, say, complete circular tracks, including stations and switch configurations through which trains may change main tracks, and the necessary controls.

The most interesting of these tracks is the kicking horse pass as it allows trains to travel in both directions and its station, besides being connected to the pass itself, includes entries from and exits to the other two main tracks (compare fig. 1). Moreover, train movement over the track must be organized so that

- there are never two trains in the track that go in opposite directions (to prevent deadlocks);
- there may be several trains moving in the same direction over the pass (each occupying one of the six track sections);
- some measure of fairness is enforced which prevents the monopolization of the track by trains going in one direction, while trains trying to go in the other direction are starving.

Controls to this effect may be realized by means of ordinary Petri–nets attached to the station model as they merely require keeping track of the number of trains that have left the station to cross the pass in one of the two possible direction (and have not yet returned) [Kl98].

Fig. 8 shows the net model of the complete pass, with the station abstracted to a substitution transition which is connected to places modelling

- the pass sections KH_LN_1 on the left and KH_LN_6 on the right;
- the sections KH_ST_10 on the left and KH_ST_20 on the right through which trains may move in from and out to both the inner and outer circles;
- the sections KHIC_LN and ICKH_LN to which trains may proceed past KH_ST_10 on their way out to the inner circle and from which they may enter section KH_ST_20 on their way in from the inner circle, respectively.

The types of tokens that are actually in these places are used by the controls inside the substitution transition to decide which of the trains in these sections or in one of the sidings inside the station gets permission to proceed.

The track over the pass is modelled by the six places KH_LN_*i* representing the six sections and by the six substitution transitions CTR_KH_LN_*i* which pass train tokens along the track in both directions.

The internal structure of the HS–transition that substitutes for the kicking horse station is depicted in fig. 9. It interfaces with the surrounding places through the substitution transitions KH_ST_0 and KH_ST_9 which represent the five–way switch configurations through which trains may enter and leave the station on both sides. Both transitions are interconnected through four places which model various aspects of the occupation by trains of the five sidings inside the station. In particular, the place

- KH_ST_[12345] contains tokens inscribed with `train_descriptors` paired with the indices of the sidings which they actually occupy;

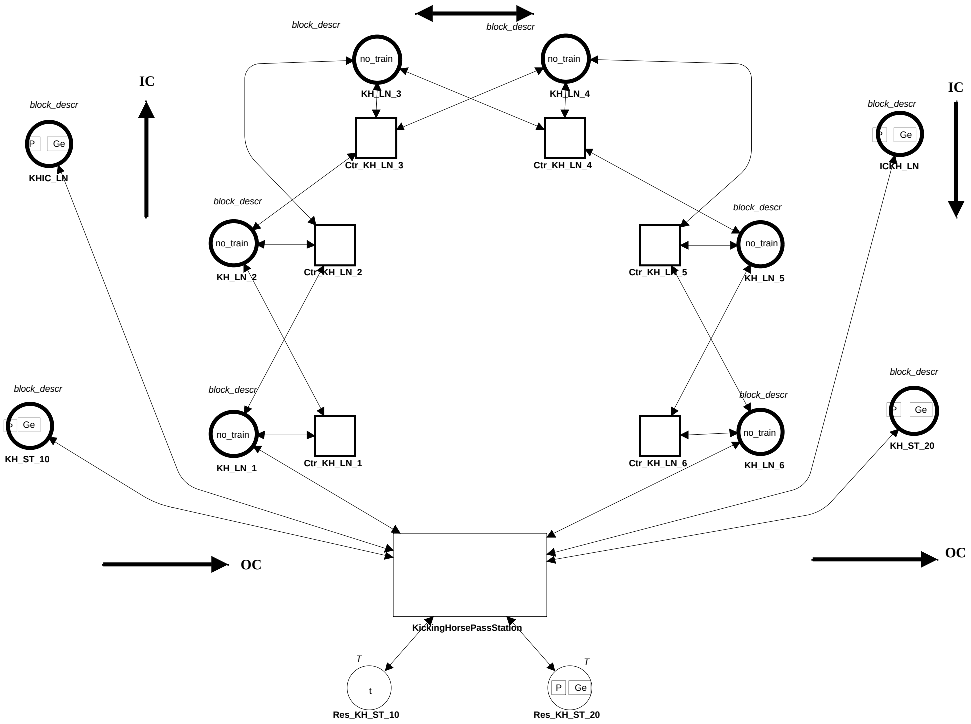


Figure 8: Net model of the kicking horse pass track

- RES_KH_ST_[12345] contains tokens inscribed with indices $i \in [1 \ldots 5]$ which represent free sidings;
- DEST_KH_ST_[12345] contains switch selector tokens paired with siding indices which determine in which direction trains in the particular sidings have to leave the station (up the pass or alternatively through section KH_ST_10 on towards the inner circle when going clockwise or through section KH_ST_20 and on towards the outer circle when going counterclockwise);
- LA_KH_ST[12345] contains look–ahead tokens generated by either of the HS–transitions which determine the siding(s) from where train(s) may leave the station next (with a train each being able to leave concurrently in either direction);

Four more places RES_KH_ST_ 0|10 | 20| 9 for plain tokens are necessary to ensure that trains arriving from or leaving towards the inner or outer circles have free passage through the track sections KH_ST_10 and KH_ST_20 , and that incoming trains from either of these circles find free sidings before entering these sections in order to prevent potential deadlocks with trains trying to use them in the opposite direction. These places are also connected to controls, specifically look–ahead subnets, for the respective parts of the inner and outer circle net models (not shown here) with which the HS–transitions KH_ST_0 and KH_ST_9 need to interact.

Another important part of this net model is the subnet contained in the substitution transition at the bottom of fig. 9 which exercises control over the direction in which trains are permitted to move across the pass and also enforces fairness between trains trying to go in opposite directions.

Roughly, these controls work as follows: A conflict between trains trying to enter the empty track in the same or opposite direction(s) is generally resolved by arbitration. Once permission is given to a train to leave the station in one

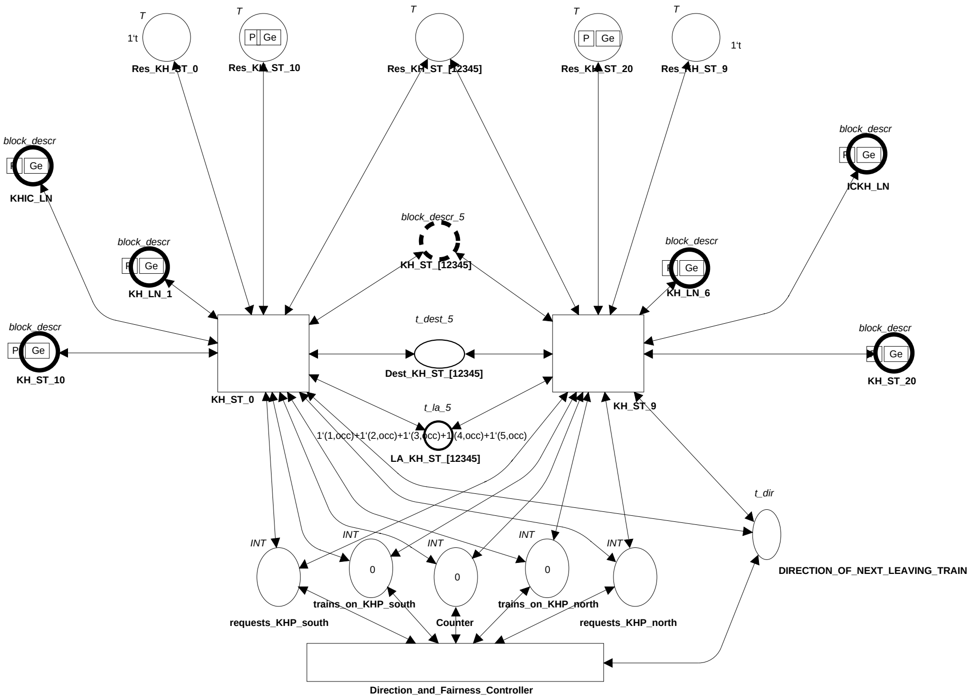


Figure 9: Net model of the kicking horse Pass station

direction, a lock is set for trains waiting to go over the pass in the other direction. A train trying to enter the empty track without competitor in either direction gets permission to leave immediately, thereby again locking the other direction. Once a train moves along the track, more trains may follow in the same direction, while the other direction remains blocked. Fairness is enforced by putting an upper limit on the number of trains which in one direction may move over the pass in succession. If this limit is exhausted, further trains are blocked. As soon as all trains have left the track, the direction is reversed, and trains waiting in the station to go in the new direction are given permission to do so. Directions may also freely be changed if the track is empty and neither of the limits is exhausted. This measure in fact establishes a finite synchronic distance between trains going in opposite directions [GeLaTh80, KlLa82].

To this effect, the DIRECTION_AND_FAIRNESS_CONTROLLER transition is connected through a place for tokens of type (color) `t_dir` and five places for tokens of type `t_integer` with the HS–transitions that control entrance to and exit from the sidings for trains scheduled to go over the pass. These places serve the following purposes:

- the token in DIRECTION_OF_NEXT_LEAVING_TRAIN, as the name indicates, specifies the direction in which the next train is permitted to move across the pass;
- REQUESTS_ON_KHP_CLK | _CO_CLK keep track of the number of trains requesting entry into the track in clockwise and counterclockwise direction, respectively;
- TRAINS_ON_KHP_CLK | _CO_CLK follow up on the number of trains that are actually in the track in either direction (one of these counter values must always be zero);

- COUNTER counts the trains which in succession have left the station in the direction indicated by the token in place DIRECTION_OF_NEXT_LEAVING_TRAIN (of which some may already have completed their passage over the pass, others may still be in the track).

A control structure similar to the one that needs to be realized inside the DIRECTION_AND_FAIRNESS_CONTROLLER transition is, on the basis of an ordinary Petri–net model, described in [Kl98].

In a larger context, the complete kicking horse pass model can be abstracted to another substitution transition KICKINGHORSEPASS which interfaces with its surroundings, the inner and outer circle net models, essentially via the places KHIC_LN, ICKH_LN and KH_ST_10, KH_ST_20, respectively. These interfaces must, of course, be complemented by small subnets which model the look–ahead controls for the passage of train tokens across these places. However, as these subnets feature basically the same elements as in figs. 5, 6 and 7, they have not been included in the nets of figs. 8 and 9 to keep them clear of details that add nothing new.

# 4 The Complete System Model

The net models for both the inner and outer circle tracks, including the stations, the inner and outer circle junctions and the two tracks by which they are interconnected, are quite similar to each other and slightly less complicated than the kicking horse pass model since trains (train tokens) are allowed to move only in one direction in both of them.

Fig. 10 shows, merely for illustration purposes and without further explanation, how the net for the outer circle looks like. Of primary interest for the construction of the full system model are the places KH_ST_10, KH_ST_20 and ICOC_LN_3, OCIC_LN_3 through which this net respectively interfaces with the kicking horse pass net and with the net for the inner circle.

Likewise, the net for the inner circle interfaces with the kicking horse pass through the places ICKH_LN (for trains leaving towards the pass) and KHIC_LN (for trains arriving from the pass) and with the outer circle through the places ICOC_LN_3 (for trains leaving) and OCIC_LN_3 (for trains arriving).

When abstracting both nets to substitution transitions OUTERCIRCLE and INNERCIRCLE, and using the HS–transition KICKINGHORSEPASS, the net model of the entire system can simply be constructed by overlapping the respective interfacing places of these three components, as is shown in fig. 11 (the look–ahead controls associated with the interfaces between the KICKINGHORSEPASS transition on the one hand and the OUTERCIRCLE and INNERCIRCLE transitions on the other hand are again omitted here).

There is some asymmetry, though, between the interfaces of the OUTERCIRCLE and INNERCIRCLE transitions with the KICKINGHORSEPASS, which is not visible on this level of abstraction. It relates to the chosen modus of operation for the track sections KH_ST_10 and KH_ST_20 which get involved in all train movements between the kicking horse station and both the inner and outer circles. Trains moving into this station from the outer circle or leaving it towards the outer circle may be stopped in these sections, whereas for trains that are being exchanged with the inner circle they are operated as no-stop sections. This implies that in the former case both sections may be allocated directly if trains demand entry and they are free, whereas in the latter case they must be reserved by look–ahead controls which ensure that subsequent sections are also free. The controls to this effect are in large parts integrated into the HS transitions KH_ST_0 and KH_ST_9 which model the five–way switches of the station (compare fig. 9).

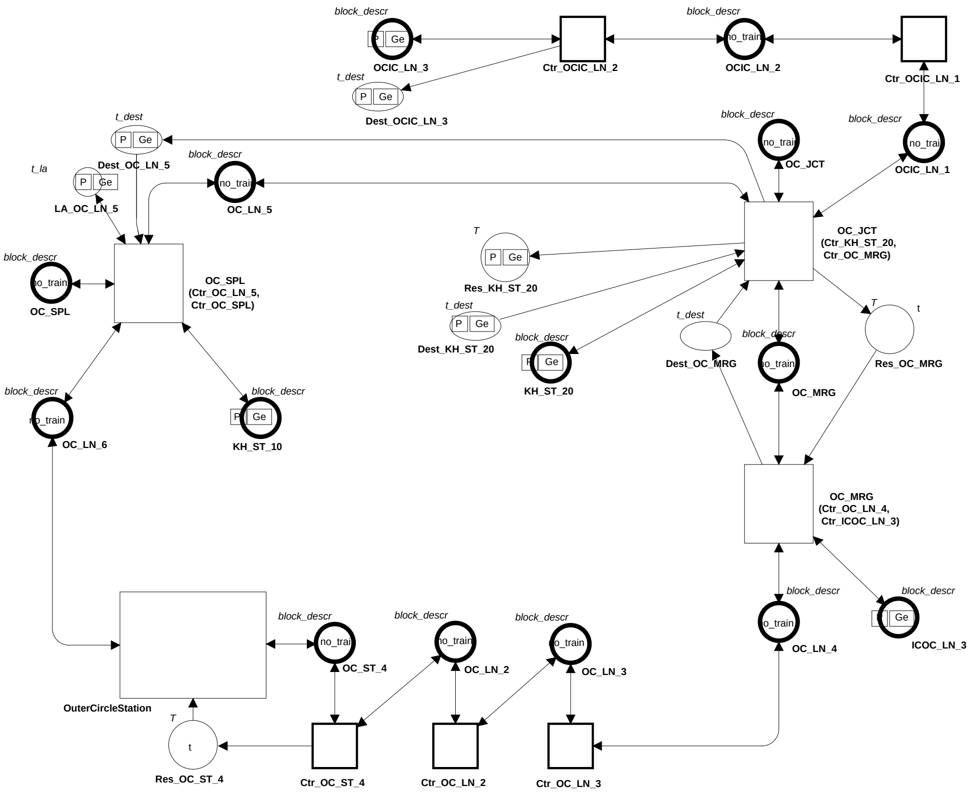


Figure 10: Net model of the outer circle track

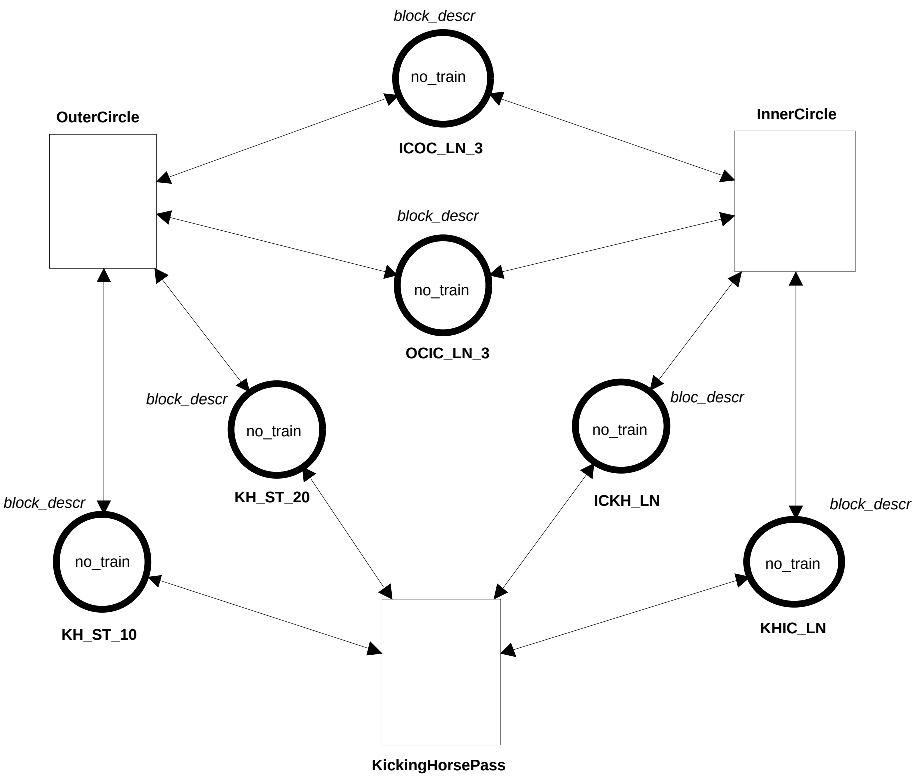


Figure 11: Complete net model of the entire track system

## 5 Discussion

The net model outlined in this paper was a first attempt to explore and analyze the dynamic aspects of the train system, in which several concurrent activities - the routing of up to ten trains as prescribed by individual travel plans - have to be coordinated ‘on the fly’. Though not exactly a blueprint, it also served as a guideline for the development of a prototype control program, written in C, which runs the trains in the real system. This program builds on an internal representation, in the form of a large data structure, of the system state in terms of travel plans, actual train positions (sections). the acctual settings of switches and light signals, and of speed levels (voltages). The program itself consists of a set of C–functions which more or less directly implement the various substitution transitions. They are called upon receiving sensor signals triggered by trains crossing section boundaries to inspect and update the parts of the state representation actually affected and to issue the control data necessary to set switches and light signals, and to apply voltages to track sections.

As several people were involved in various aspects of the project (track design and construction, development of hard- and software, net modelling), and not all of them had a background in Petri nets, the basis for communication was the schematic track layout given in figure 1. In the early stages of the project, the CPN model helped to identify fairly quickly flaws in the conceptual design and potential trouble spots by generating extreme system states (token distributions representing trains in track sections), rather than doing these tests in the real system where critical situations might be difficult and rather time–consuming to bring about and to reproduce. Also, a C implementation of software that controls concurrent activities under real-time constraints includes too many low–level details to permit an abstract view of the (idealized) system dynamics. In contrast, translations between the CPN model and the track layout are straightforward, i.e., problems found in the model could be communicated using the layout, and issues unclear in the layout could be investigated in the model.

A major issue that cannot be properly addressed using the track layout alone is the identification of potential deadlocks (in the non-technical meaning of the term). The layout could easily be mapped to a condition-event net, but this would fail to account for the individual travel plans. In general, none of the trains “sees” the complete track layout, but rather each of them has its own partial view, or its own private net model, of the system as defined by its travel plan.

To convey the nature of the problem, consider as an example the condition-event net of figure 12 which models just the outer and inner circles without sidings and, in rudimentary form, the two interconnecting tracks. The token distribution depicts 6 trains on the inner circle and another 7 trains on the outer circle, and it is assumed here that no trains can move in or out of this particular subsystem (tokens may neither be added nor disappear).

Trains can obviously move along both tracks – though not very smoothly in reality – as there is one free section in each of the circles. However, if the travel plan of one of the trains on the outer circle would prescribe changing into the inner circle but none of the travel plans of the trains already in it would prescribe changing in the opposite direction, the last free slot in that track would be occupied, and all trains would come to a halt. Trains could still move along the outer circle, but if further trains would try to follow their travel plans into the – now blocked – inner circle, the deadlock would spread out over the outer circle as well. Thus, as far as these particular travel plans are concerned, there exists just the interconnecting path from the outer to the inner circle through the place OCIC_LN, but not the path through the place ICOC_LN, i.e., the inner circle constitutes a classical structural deadlock.

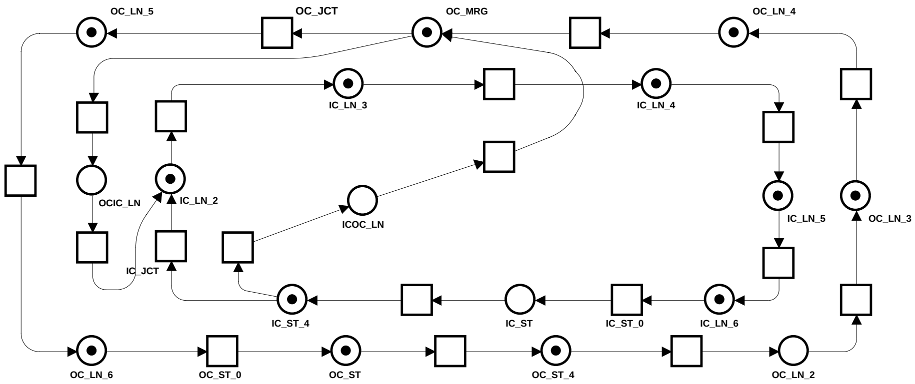


Figure 12: Illustration of a deadlock problem

The constraints imposed on the net by the travel plans do not necessarily increase the potential for deadlocks, even though they do decrease the available resources in terms of usable track sections. In our example, if the travel plans of the trains in the outer circle would not include the connection to the inner circle, the path through the place OCIC_LN would become invisible as well, leaving two separate and deadlock-free circles.

A conservative approach to deadlock avoidance would limit the number of trains on each track. With $n$ denoting the number of sections per track, this seems to allow for $n-1$ trains to move along each track at the maximum. In a worst case scenario, however, the individual travel plans could be such that all trains try to get into the same track at about the same time to make some turns there before leaving again. Therefore, the restrictions would have to include those trains that just pass through a track while following their travel plan, leaving only $min\{n_{IC}, n_{OC}\} - 1$ trains moving about inner and outer circle. The same consideration applies if the kicking horse pass is included, so that the total number of trains that can be guaranteed to move freely about the entire track system without causing deadlocks would have to be limited to $min\{n_{IC}, n_{OC}, n_{KH}\} - 1$. Other circular tracks of restricted capacity are formed by the interconnecting tracks between the inner and outer circles and the kicking horse pass, bringing down the number of trains in the complete track system to no more than about five trains.

With more trains, deadlock prevention requires that some restrictions be placed on specifying individual travel plans, e.g., by allowing each of the circular tracks to be used in at most $n_{track} - 1$ of them. With ten trains to deal with, this could for instance be accomplished by travel plans which allow three trains to use all three main tracks in any order, while of the remaining seven trains two each could cycle just over the kicking horse pass and just around the inner circle, and three trains could make turns just around outer circle.

Travel plans in which all ten trains use all three main tracks in some arbitrarily chosen sequences may or may not end up in deadlocks, depending on the order in which trains happen to get into critical situations. Preventing these deadlocks requires sophisticated global controls which must monitor actual distributions of trains over track sections and next actions prescribed by travel plans to decide which trains may proceed without getting trapped. This is still an open problem which requires further simulation and analysis to find a satisfactory solution.

Another source of deadlocks are trains that have completed their journeys and arrived at pre-defined terminal positions (station sidings). The simpler of these situations may occur at stations of the inner or outer circles, along which trains are moving in just one direction. If all three sidings in these stations are terminal positions and the respective trains have ended up there, no other trains still on the move can pass through, i.e., they inevitably deadlock. The way out of this problem consists either in using only two of the three sidings as terminal positions, or in devising a travel plan for one of the three trains ending there which can be expected to terminate after all others trains have passed through.

Deadlocks that involve the kicking horse station, due to trains moving in both directions, are more difficult to prevent. There may be trains arriving in the station scheduled to go, say, counterclockwise over the pass while other trains are actually moving clockwise over the pass. If there are more such trains than there are sidings left in the station, the entire pass deadlocks. The same happens if a train arriving from the pass to end its journey in the station finds its terminal position occupied by another train scheduled to go over the pass in the opposite direction.

These situations could be brought about quite frequently by simulating various travel plans in the net model but also by running them in the train system itself. The remedies found sofar are not very satisfactory from a systems design perspective as they either require careful coordination of travel plans or putting rather conservative restrictions on train movement which often bring to a grinding halt all but one or two of the ten trains.

As all of this is only the prelude to the problems to be solved when designing and implementing the control software for the real system, the reader may wonder why the information gathered from the CPN model was not used to simplify the track layout. The rationale for this comes from the teaching context of this project: the system has to have rough edges for the modelling process to be interesting, and while minor corrections were made to the initial layout, most trouble spots had to remain, so that the students can employ their theoretical knowledge about Petri Nets in practice to find them. We do not claim that we have consciously designed the track system with all its problems in mind, but the current system seems to provide just the right level of complexity. The non-local interactions between its parts make it challenging even for good students, but a great deal can be achieved by finding the right abstractions and by looking for modular solutions.

# 6 Conclusion

The entire net model was designed by two graduate students (the first two coauthors of this paper) as a term project accompanying a graduate course on Petri–nets. Having neither been familiar with the Design/CPN tool nor having had any prior experience with system architecting and modelling, i.e., starting more or less from scratch, it took them about four months of hard work to complete the model and to run some simulations. A considerable part of this time was spent on getting used to handling the tool rather than on the net design itself.

One problem area is editing. It takes pages after pages (and diagrams and menu screen dumps) of a rather voluminous tutorial (which would take students the better part of a term to digest) to explain things that should be self–explanatory with a decently designed graphical interface; and although the manuals are available online, there is no help functionality integrated into the tool. Another problem is the need to switch back and forth between the editor and the simulator which takes way too much time (even for an early and incomplete prototype of the full net model it took some 10 minutes on a SUN–SPARC equipped with 48 MBytes of memory, 4 minutes if the memory was stepped up to 64 MBytes, and still 2 minutes on an Ultra–SPARC with 512 MBytes of main memory). For all practical purposes, this rules out going repeatedly through cycles of designing nets in incremental steps, testing and modifying them until an acceptable solution is found, as one would often wish (or need) to do it in early phases of architecting complex systems.

Executing the simulator in a stepwise manner is equally frustrating since changes of the token distribution are very hard to recognize on the graphical display and, what is even worse, each step requires several seconds, sometimes even minutes, to complete. It renders testing large nets in this way a time–consuming affair. Surprisingly, most of the time is spend in the graphic updates, whereas the simulation itself seems to be reasonably fast. As a consequence, automatic simulation has to be used in practice instead of the animated, interactive simulation.

These deficiencies render Design/CPN in its current form a tool that can hardly be used for teaching as students are easily turned off.

What is also (as yet ?) missing are means to verify net designs with regard to essential safety and liveness properties. It would already help if the existence of S– and T–invariants could be verified by analytical methods at least on the level of plain Petri–nets, i.e., without regard for the constraints imposed by inscriptions (which would be partially sufficient in the case of our train system model and presumably also in many other areas, e.g., process coordination languages [CaGe89, Ass95]). This being a problem which, for complexity reasons, may be hard to crack if the nets become large, one could alternatively think of simply checking whether or not invariants pre–specified by the designer do indeed hold.

While the improvements planned for the new CPN simulator (see the respective web page) seem to point in the right direction, doing away with at least some of the performance problems, it is not at all clear why memory demand still remains excessive, why compilation of complete nets still takes several minutes, and why users are not given the choice of compiling to an SML implementation of their own.

# References


[Ass95] Aßmann, C.: *A Coordination Language for Systems of Cooperating Processes*, Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'95), pp. 738 - 747

[CaGe89] Carriero, N.; Gelernter, D.: *Linda in Context*, CACM Vol. 32, No. 4, 1989, pp. 444–458

[GeLa78] Genrich, H.J.; Lautenbach, K.: *The Analysis of Distributed Systems by Means of Predicate/Transition-Nets*, in: Semantics of Concurrent Computation, Lecture Notes in Computer Science, No. 70, Springer, 1979, pp. 123–146

[GeLaTh80] Genrich, H.J.; Lautenbach, K., Thiagarajan, P.S.: *Elements of General Net Theory*, in: Net Theory and Applications Lecture Notes in Computer Science, No. 84, Springer, 1980, pp. 21–163

[GeLa81] Genrich, H.J.; Lautenbach, K.: *System Modelling with High–Level Petri–Nets*, Theoretical Computer Science 13, 1981, pp. 109–136

[Jen90] Jensen, K.: *Coloured Petri Nets: A High-Level Language For System Design and Analysis*, in: Rozenberg, G. (Ed.): Advances in Petri Nets 1990, Lecture Notes in Computer Science, No. 483, Springer 1991, pp. 342–416

[Jen92] Jensen, K.: *Coloured Petri Nets*, Monographs on Theoretical Computer Science, Springer 1992

[KlLa82] Kluge, W., Lautenbach, K.: *On the Orderly Resolution of Memory Access Conflicts among Competing Channel Processes*, IEEE–TC Vol. C–31, No. 3, 1982, pp. 191–207

[Kl98] Kluge, W.: *The Kicking Horse Pass Problem*, to be published in Petri–Net News Letters

[Po95] Pole, G.: *The Spiral Tunnels and the Big Hill*, Altitude Publishing Canada Ltd. 1995

[Rei85] Reisig, W.: *Petri Nets*, EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer, 1985