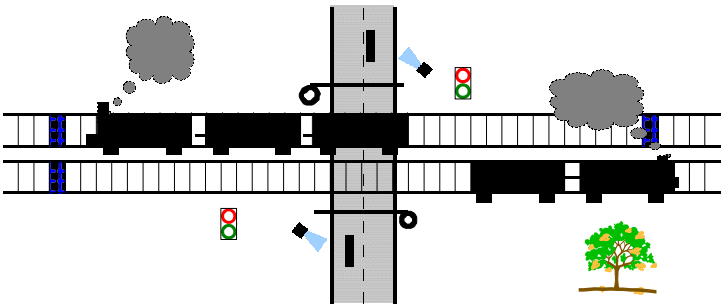
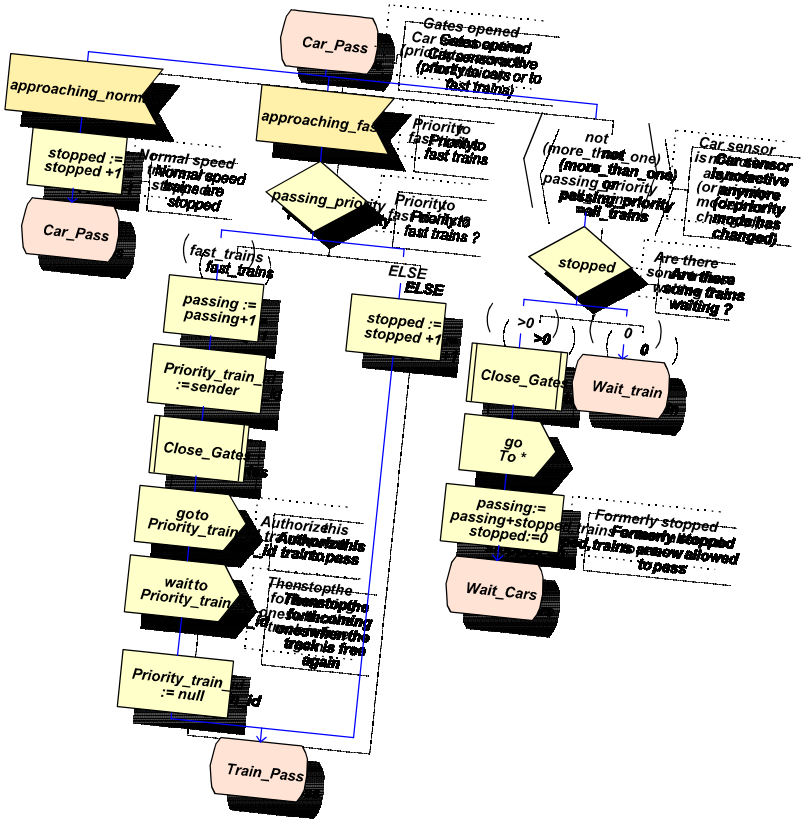


Eric Conquet
Maxime Perrotin

SDL Contest

Railway crossing specification



12 MAY 2002

SAM WORKSHOP

Table of Contents

I. Introduction

This report presents the specification in SDL and MSC of a railway crossing. The system has been developed for the SDL Design Contest of the 3rd SAM Workshop held in June 2002 in Aberystwyth, Wales.

The purpose of the system is to manage a railway crossing which can have several tracks. Each track can be dedicated either to high speed trains, or to normal speed ones. The crossing road can be closed with a gate, and a sensor indicates if more than one car is waiting in front of it. Some sensors tell the controller if a train is approaching or leaving the crossing and the controller has to decide if it allows the train to pass or not.

The first part of the document presents the analysis of the user requirements and the structure of the model. It also includes the assumptions we have made on the architecture of the physical system, such as the location of the sensors and signals.

Then, the second part presents the detailed SDL model specifying the complete system.

Finally, the last part of the report presents the validation and verification of the system, with formal proofs of properties and results of exhaustive simulations.

The SDL, MSC, and GOAL models were made using the ObjectGeode tool from Telelogic.

II. User requirements analysis

1 System environment

The system describes the interactions between the following actors :

- Trains/tracks. There are two kinds of trains : fast speed and normal speed. Each track has two sensors : one that detects if a train is approaching the railway crossing and one that detects that the train has left. Each track also has a stopping signal that can be set in case too many cars are waiting. This signal is a light that can be green if the trains are allowed to pass, or red if not. It is not possible that the light becomes red if there is a train between the two sensors. The system shall be generic in the number of tracks (for fast and normal speed trains).

- Gates. There is one gate on each side of the road. They are used to prevent the cars from crossing the railway while some trains are coming. Both gates are controlled with one signal, and take a constant time to be closed.

- Car sensor. When more that one car is waiting in front of the gates, a sensor sends a signal to the controller. The signal is sent again when the cars have left. This sensor is used to control the traffic in case the operator chose the priority to cars strategy.

- Railway crossing controller : the controller is the main part of the system. It is in charge of managing both the gates for the cars and the stopping signals for each track. The controller can work in automatic or manual mode. In automatic mode, depending on the strategy chosen by the operator, the controller can give the priority to the cars, to the fast trains, or to all the trains. In manual mode, the operator activates himself the gates and the stopping signal for the trains. However, the system shall not allow any unsafe operation, like opening the gates for the cars when a train is approaching the railway crossing.

Various signals are exchanged between the system and its environment :

- from the trains : train approaching, train leaving (both signals for all kind of trains, normal speed and fast speed) ;
- car sensor information ;
- from the controller : open gates, close gates (both gates are controlled at the same time), stopping signal ;
- from the operator : manual open, manual close, manual wait (stopping signal), manual go, set priority.

We can deduct from this preliminary analysis the structure of the system, expressed in a formal way (Figure 2) :

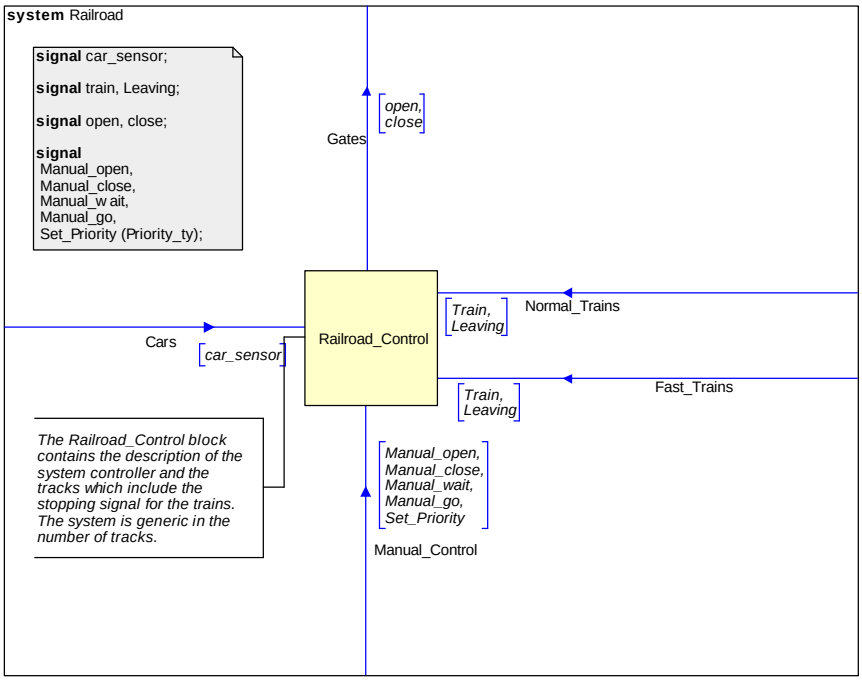


Figure 2 - System structure

Assumptions :

- The approaching train sensor is located before the light.
- Before the initialization of the system, the gates for the cars are closed and the stopping signal is off. The trains are supposed to be stopped by another light before the portion of railway controlled by the system. *This means that when the system is activated, no train should be located between the two sensors*.

The picture below (Figure 3) shows the actual configuration of the system.

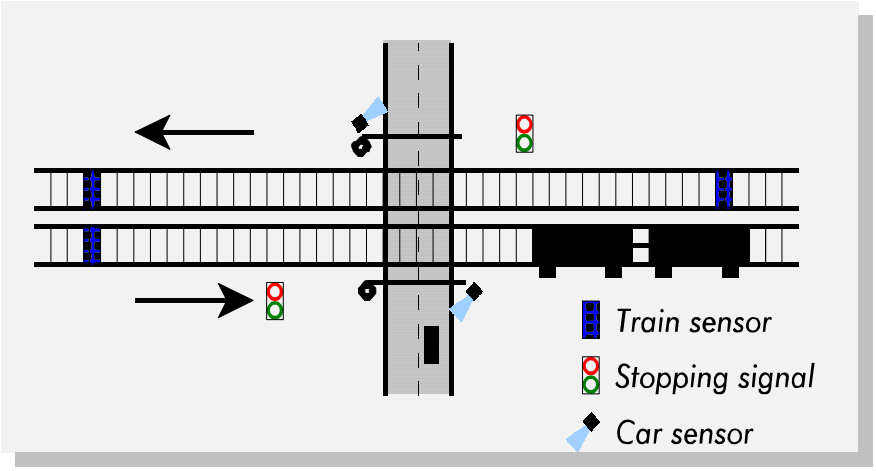


Figure 3- The railway crossing environment

The train sensors are designed in a way that there is not a single signal sent when the trains pass over it but a sequence of signals, each one produced by a wheel of the train. When the sensor stops sending signals after some time, it means that the train has finished passing over it.

2 Strategies to control the crossing

We have seen in the previous section all the actors of the system and the information they carry. We will now see how the operator can choose how it should actually react. It is expected that the system implement different strategies (at least three) for giving priority to the trains or to the cars in an automatic mode. The operator can also choose a manual mode in which he can control both the gates and the stopping signals himself, provided that his actions are safe.

The different modes are therefore the following :

2.1 Priority to cars

In this mode, the car sensor is used to determine if more than one car is waiting at the gates, and if so, the controller should stop forthcoming trains to let the cars pass. A typical scenario of the expected behaviour expected in this mode is shown below.

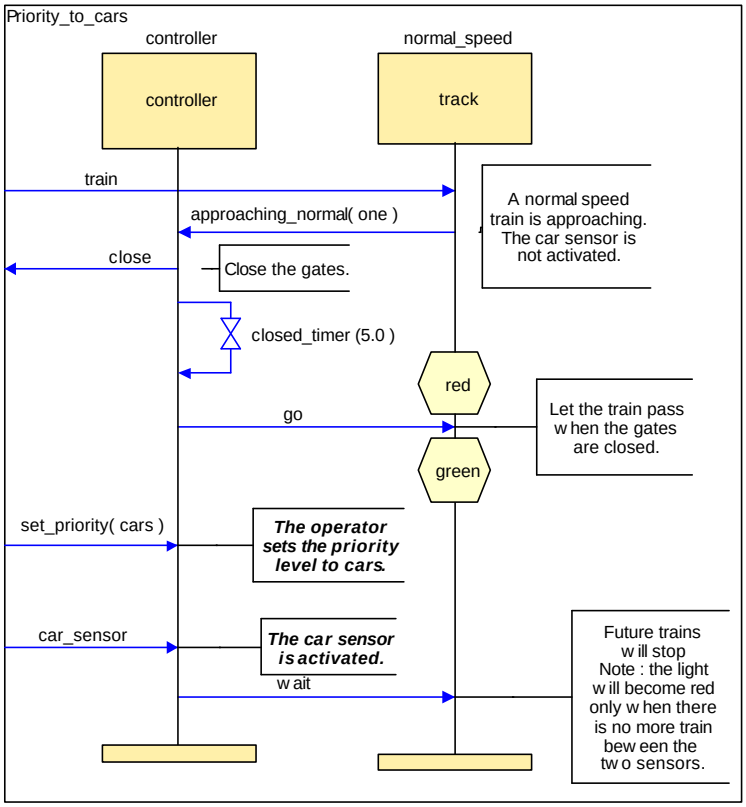


Figure 4 - Typical scenario in “Priority to cars” mode

2.2 Priority to fast trains

Another strategy is to give priority only to fast trains over cars. In that mode, normal speed trains are stopped if too many cars are waiting, but fast trains can always go.

2.3 Priority to all trains

In this mode, all trains have priority over cars, even if too many of them are waiting. It means that whatever the state of the system, if a train is approaching, the gates must be closed to let it pass without having to stop.

2.4 Automatic priority

The priority strategies presented above have one major drawback : they are fully dependant on the traffic. This means that, for example, if the operator has chosen the priority to cars mode and there is a big car traffic jam around the crossing, the car sensor may remain active very long and the trains could never pass.

For that reason, another priority strategy has been thought. In this "automatic priority" mode, a priority level will start decreasing as soon as it has been set. When the priority level reaches zero, the priority changes automatically. This strategy allows both the trains and the cars to have one chance to pass (see Figure 5).

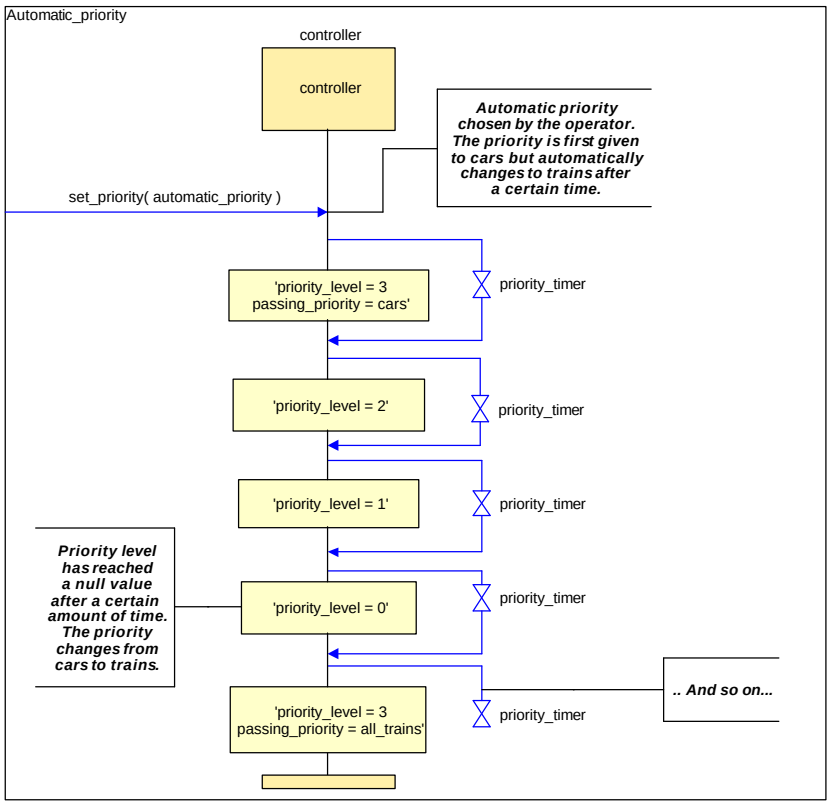


Figure 5 - Automatic priority principle

2.5 Manual mode

Besides all the automatic modes, which give priority to the cars or to the trains, the crossing has to be able to be controlled fully manually. However, some unsafe situations must be rejected :

- Opening the gates when one or more trains are currently between the two sensors ;
- Opening the gates when the stopping signal is not set on all the tracks ;
- Removing the stopping signal when the gates are opened.

The manual mode is the mode reached at the system initialization. When the system is activated, the gates for the cars are closed and no train is between the two sensors of any track.

To exit the manual mode, the operator must choose a priority mode. To enter the manual mode again, he has to use a manual control signal.

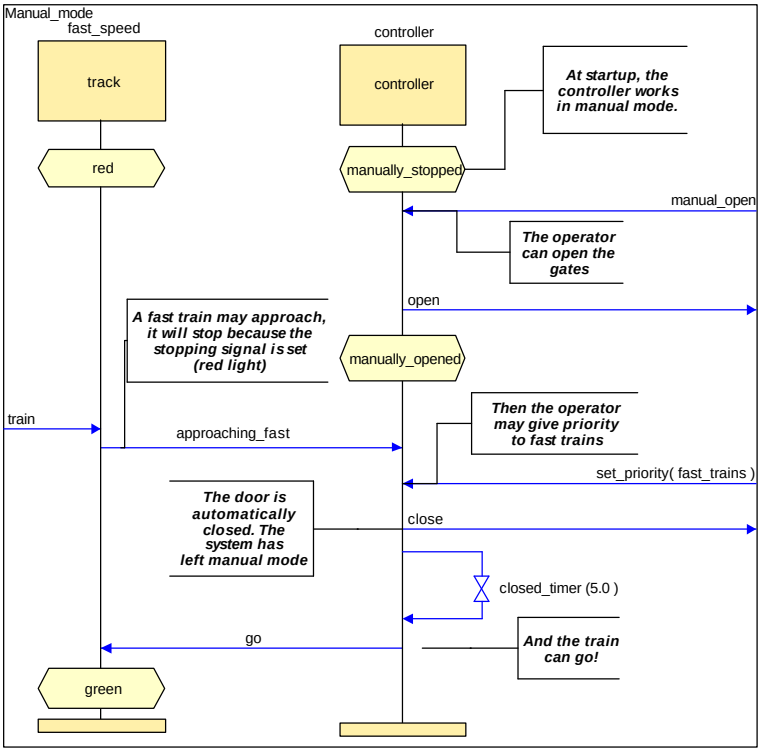


Figure 6 - Manual mode

III. Specification and design

In the previous part, we have seen the structure of the system and its interactions with its environment. The system basically receives information from the trains sensors, the car sensor, and the operator in case of manual management of the crossing. The user requirements related to the way this information should be processed have also been presented. We have presented different priority strategies that can be chosen while the system is running. In this part, we present the implementation in SDL of these requirements. The complete model is available as a separate file, and should be used for step by step simulation to complete these explanations. This document focuses on the most important parts of the specification that are useful to understand how the needs are fulfilled.

1 Inside the structure

The general system structure is shown in Figure 2. If we zoom into this "Railroad_Control" block we can see the heart of the system :

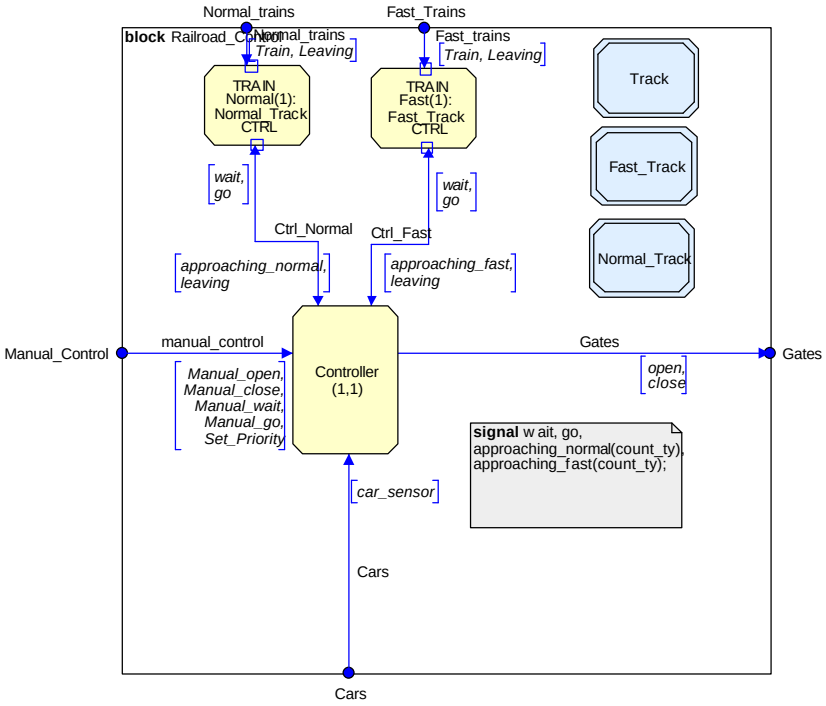


Figure 7 - The Railroad_Control block

This communication view reveals several characteristics of the model :

- Fast tracks and normal tracks are taken into account, the distinction being made with the "approaching" signal which may be "approaching_normal" or "approaching_fast". Processes type "Fast_track" and "Normal_Track" inherits their behaviour from the process type "Track" with the approaching signal as context parameter ;

- The system is generic in the number of tracks : to add or remove some tracks, simply change the number of instances of the corresponding process ;

- The stopping signal is part of the tracks.

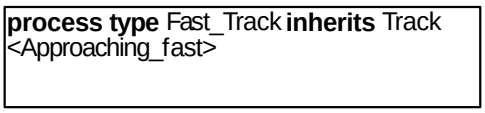
2 The tracks

All the tracks inherit from the behaviour of the process type "Track" which is in charge of two tasks :

a) Setting/removing the stopping signal for the trains when the conditions to do it are fulfilled. The stopping signal is represented by a state called "red" in the track behaviour, and the state called "green" means that the trains that approach are allowed to go ;

b) Managing the reception of the approaching and leaving signals which come from the two sensors located on the track. This means informing the controller when a train is coming or has left the railway crossing section. To do this task, the track must take into account the fact that each time a wheel of the train passes over the sensor, it generates a signal to the system. Therefore, it has to discard all the signals but one, which is transmitted to the controller. In the case of a train approaching, it is the signal generated when the first wheel of the train passes over the sensor, and in the case of a train leaving, it is the signal generated when the last wheel passes.

Note : Two processes inherit from the "Track" process : the "Normal_track" and the "Fast_track" processes. The context parameter signal "approaching" is different in these two processes :



The possible states of the process "Track" are the following :

State	Description
Red	Stopping signal is set
Green	Stopping signal is not set

The logic of the process is to count how many trains are between the two sensors. If there is one or more and if it receives an order to stop the trains from the controller, then it will wait until all the trains have left before going to the red state. In that case, since the stopping signal is not set immediately, another train may approach and therefore pass. This information is then transmitted to the controller so that it always know that some trains are on the railway.

2.1 Data and signals

The following table describes the signals and the important data that is related to a track.

Name	Type	Description
train	signal	From the environment. Indicates that a wheel of a train is currently on the “approaching” sensor.
leaving	signal	From the environment. Indicates that a wheel of a train is currently on the “leaving” sensor.
leaving	signal	To the controller. Indicates that a train has left.
wait	signal	From the controller. Indicates that future trains on that track should stop if no train is currently between the two sensors.
wait	variable	Internal variable to store the order from the controller to set the stopping signal when the last train on the track has left.
go	signal	From the controller. Order to remove the stopping signal.
new_train	variable	Internal variable to distinguish between a new train approaching and a wheel of a train that has already been detected on the sensor.
approaching_timer	timer	Timer to detect that all the wheels of a train have passed over the approaching sensor.
leaving_timer	timer	Timer to detect that all the wheels of a train have passed over the leaving sensor.
train_passing	counter	Variable that stores the number of trains currently between the two sensors. This variable is used to know whereas it is allowed to set the stopping signal or not.
approaching (movement) *approaching_fast, approaching_normal*	signal	To the controller. Indicates that a train is approaching the traffic light. Contains a parameter to precise if the train is stopped or if it is passing.

2.2 Red state

In the red state, it is possible that one train approaches. Then when it sees the red light, it should stop. It is not needed to send a specific signal to the environment to stop the train.

The “new_train” variable is used to determine if it is the first wheel of the train that is over the sensor or not. If so, the signal “approaching” is sent to the controller ; else, it is simply discarded until the timer “approaching_timer” expires. This timer is set each time the “train” signal is received.

2.3 Green state

The “green” state is a little bit more complex because of the transition to the “red” state, which may not be immediate if some trains are located between the two sensors. If the “wait” signal is received, it is first checked that no train is currently between the sensors and in that case only it switches to the “red” state.

The “leaving” signal is treated in a slightly different way as the “train” signal, in the sense that the signal is sent only when the last wheel of the train has passed over the sensor. After the last train has left, it is checked if the stopping order has been received from the controller.

3 The controller

The controller is the main actor of the system. It is responsible for managing all the traffic independently from the number of tracks. It has two main functioning modes : automatic (with different strategies for the priority) and manual (controlled by an external operator).

The states that represent this separation are shown below :

State	Description
Wait_train	Gates are opened, stopping signal is set. Waiting for a train.
Wait_cars	Gates are closed, stopping signal is not set. At least one train is currently between the two sensors.
Train_pass	Gates are closed. Priority is set to cars or to fast trains. At least one train is between the two sensors but the car sensor is active. The stopping signal is set for the tracks where no train is present and will be set everywhere else when all the trains have left the other tracks.
Car_pass	Gates are opened. Priority is set to cars or to fast trains. Stopping signal is set on all the tracks and the car sensor is active.
Manually_stopped	Manual control. All the traffic is stopped : gates are closed and stopping signal is set on every tracks.
Manually_opened	Manual control. Gates are opened and trains are stopped.
Manually_allowed	Manual control. Gates are closed and trains can go.
Manually_closed	Manual control. Gates are closed and stopping signal will be set on all the tracks when the trains between the two sensors have left (similar to “train_pass” state).

3.1 Automatic mode

The automatic mode implements the priority strategies presented in section II.2. It comprises the SDL states “Wait_train”, “Wait_cars”, “Train_pass” and “Car_pass”. In all of these states, it is possible for the operator to choose one of the available strategies.

Figure 9 below shows how the “automatic priority” mode is managed in the controller. As it was described in the MSC (Figure 5), when the automatic priority is chosen, a timer is set and the priority is first given to cars. Then each time the timer expires, the priority level decreases until it reaches a null value. Then the priority is changed to trains and the timer is set again.

The behaviour that results from a specific priority is then described in each state of the automatic mode. For example in state “Wait_cars”, if the priority is set to cars and the car sensor becomes active, then the controller sends the “wait” signal to all the tracks so that when the trains that are currently between the two sensors have left, the stopping signal is set.

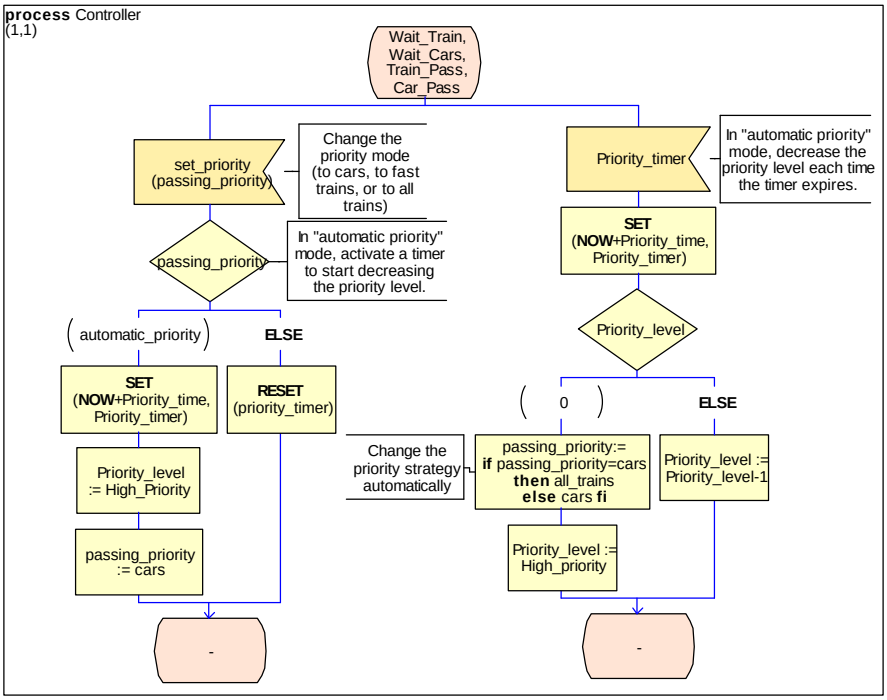


Figure 9 - Priority management in automatic mode

The state “Train_pass” is an unstable state. It is reached when priority is given to the cars or to the fast trains (in this case, cars still have priority over normal trains), and when some trains are between the two sensors whereas the car sensor is active. It is unstable in the sense that it will switch to another state as soon as the last train has left the railway to let the cars pass.

Before reaching this state, the controller has sent the “wait” signal to all the

tracks. However, as it was seen before (see II.2.3), the tracks with some trains between the two sensors keep their light green. It is therefore still possible in that state to have some trains approaching and passing even though the "wait" signal has been sent, because the "wait" signal is not an immediate order nor the stopping signal itself.

To know if the train which is approaching is stopped at the light or passing, the controller analyses the parameter of the signal approaching. It directly gives this information (Figure 10).

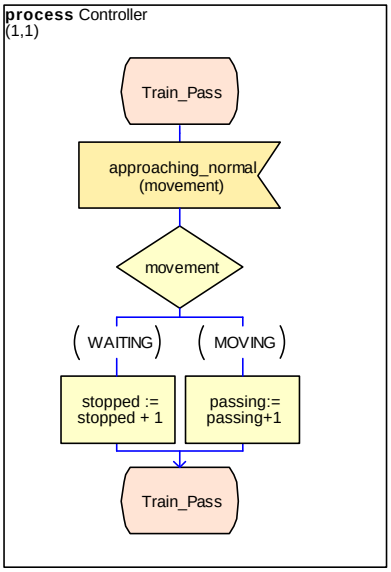


Figure 10 - Movement detection

3.2 Manual mode

The manual mode is first reached at the system activation, since at that time the gates are closed and the stopping signal is set for all the tracks.

In the manual mode, two operations are possible : controlling manually the crossing by activating the gates or the stopping signal, and setting a new priority mode. In that case, depending on the state of the overall system, the controller chooses what action to perform.

When controlling manually the gates and the stopping signal, the operator has to know that if he tries unsafe action, they will not be taken into account by the system. For instance, in the "manually_opened" state, it is not possible to send the signal "manual_go" to the controller.

The manual mode comprises four states, and each state corresponds to one particular configuration of the system, depending on the current state of the gates and of the stopping signals.

IV. Model verification

1 Introduction

Building a model and document it is, to our opinion, only half of the work. The demonstration of correct behaviour has to be performed through simulation sessions. We have used the ObjectGeode simulator in two different modes: interactive and exhaustive. Interactive mode is interesting to show how the model behaves when executing specific scenarios. But proving properties is essential for real-time systems and is the realm of exhaustive simulation.

2 Setting the simulator parameter

2.1 Defining the system environment

The model has its own environment. In our case, the environment can send signals indicating that a train is approaching or leaving, or more than one car is waiting at the gate. Manual commands to control the gate or to set the priority mode are also coming from the environment.

With the ObjectGeode simulator, a powerful feature can be used when it would be hard to completely describe environment behaviour: the feed list. Such a list is made of signals coming from the environment with possibly a set of values.

In our model, the feed list is described by the following lines:

```
feed manual_control set_priority( * ) to controller
feed manual_control manual_close
feed manual_control manual_go
feed manual_control manual_open
feed manual_control manual_wait
feed fast_trains leaving
feed fast_trains train
feed normal_trains leaving
feed normal_trains train
feed cars car_sensor
```

The first line means that any priority level can be set.

2.2 Reducing the size of the state space

Some tips to reduce the size of the state space in exhaustive simulation mode.

The major drawback of exhaustive simulation is the potential explosion of states space. To reduce the size of the whole graph, different techniques can

be used. One of the most efficient consist in resetting the process variables at the end of each transition. Obviously, such an action must be limited to useless variables, i.e. variables whose content has not to be kept. In our model, we have simplified the state graph a lot by using the following goal observer:

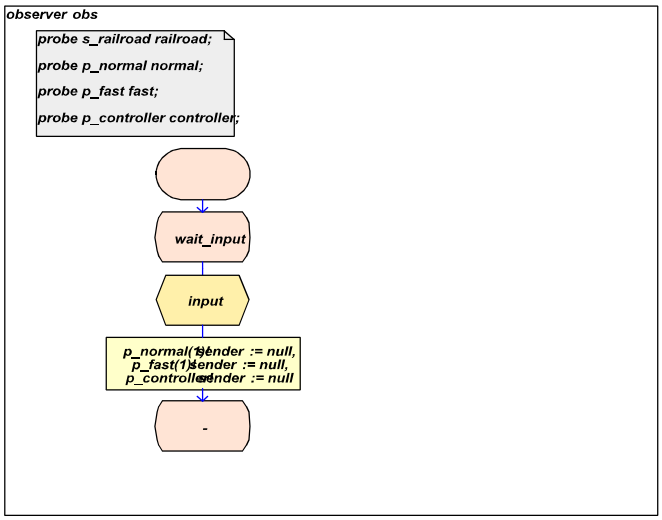


Figure 11 - GOAL observer to limit the state space

Such a GOAL observer reset the sender variable of each process which has an immediate effect: simulation time is divided by 7.

3 Simple interactive simulations

3.1 Presentation of the examples

To exercise the model, different simple simulations have been run :

1) Cars are present before system is initialised in a given priority mode ;

2) Trains are present before system is initialised and pass ;

3) Multiple trains are approaching on the same track and cars arrive in train priority mode.

3.2 Scenario 1

The generated MSC show that cars are taken into account even if they arrive when the system has not be fully initialised.

At the end of the scenario, the gate is reopen and cars can pass.

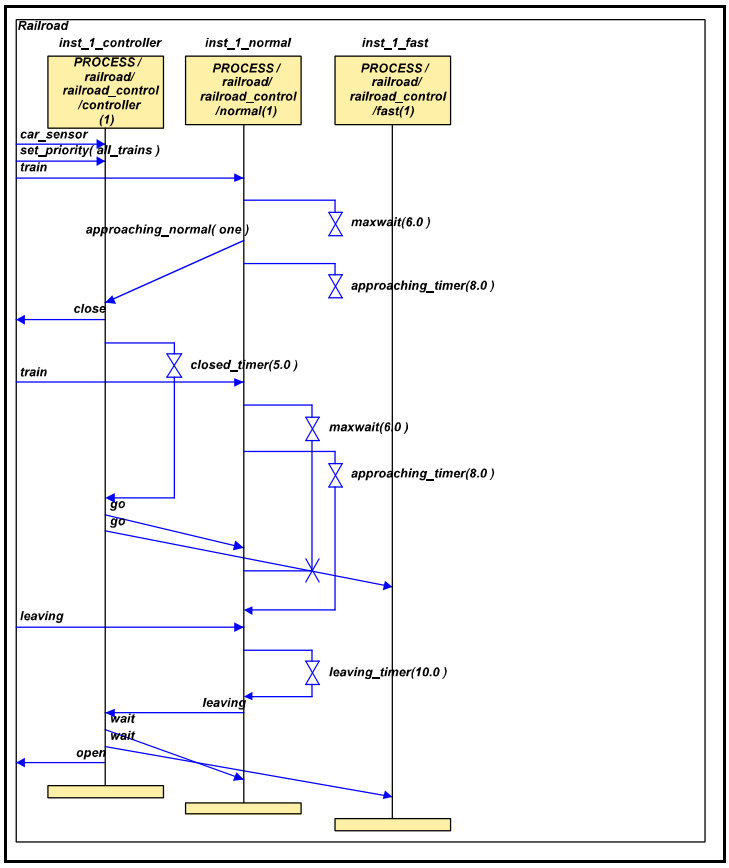


Figure 12 - Scenario 1 generated MSC

3.3 Scenario 2

In that case, the MSC shows that waiting trains are taken into account and pass before the gate is reopen.

Two train signals are received before the priority mode is set to trains. They are taken into account, the gate is closed and trains pass.

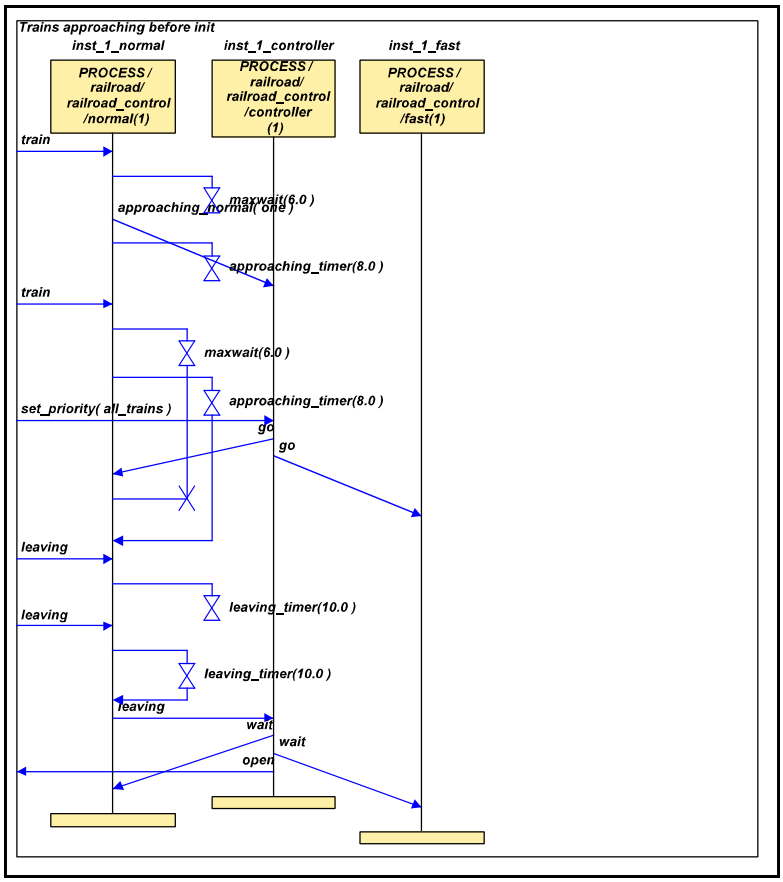


Figure 13 - Scenario 2 generated MSC

3.4 Scenario 3

When multiple trains are approaching on the same track they are clearly identified and the gate is open only when all trains have left the gate.

In that scenario, a first fast train approaches and triggers the approaching signal three times before the leaving signal.

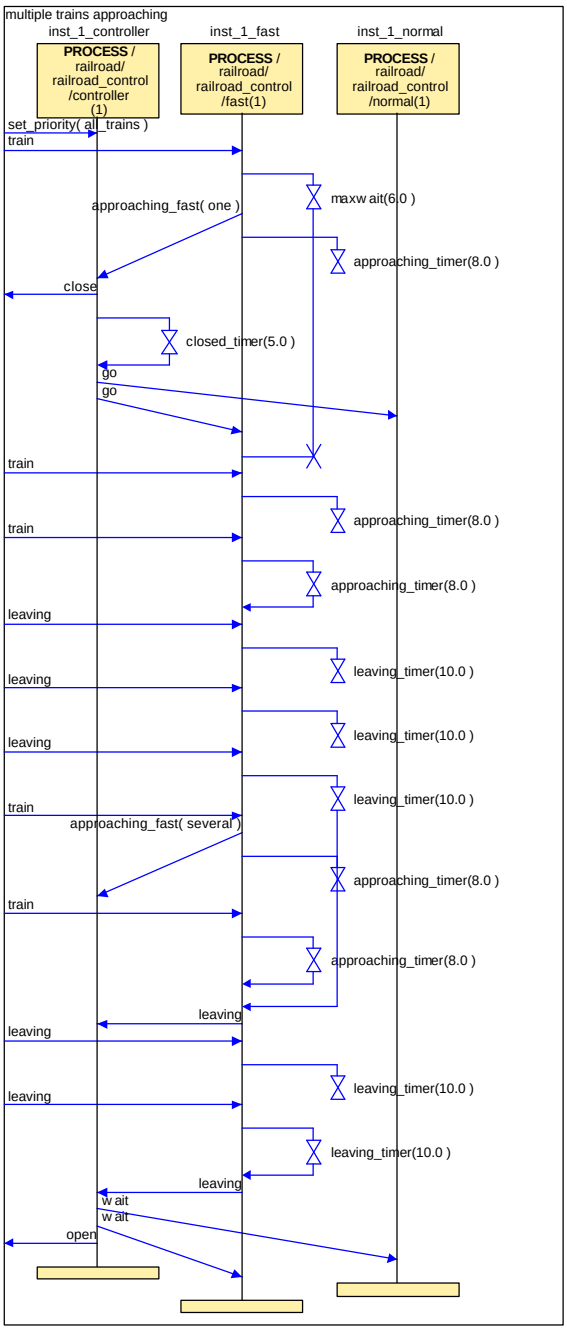


Figure 14 - Scenario 3 generated MSC

4 Property checking

4.1 Introduction

The following properties have been checked :

P1: No leaving signal can occur if the door is open ;

P2: No go signal is sent if the door is open ;

P3: The gate cannot be opened if train are passing ;

P4: Trains do not wait more than time needed to close the gate in train priority mode ;

P5: Fast trains do not wait more than time needed to close the gate in fast train priority mode ;

P6: Cars do not wait forever in car priority mode ;

P7: In manual mode, the gate cannot be opened if a train is passing ;

P8: In manual mode, the gate cannot be opened if not all traffic lights are red ;

P9: In manual mode, a traffic light cannot be set to green is the gate is open.

4.2 Property 1

No leaving signal can occur if the door is open.

The property is described with the following MSC. Such a MSC means " if the gate is open, no leaving signal can occur". The signal close is part of the MSC since the property is violated only if the close signal occurs after the leaving

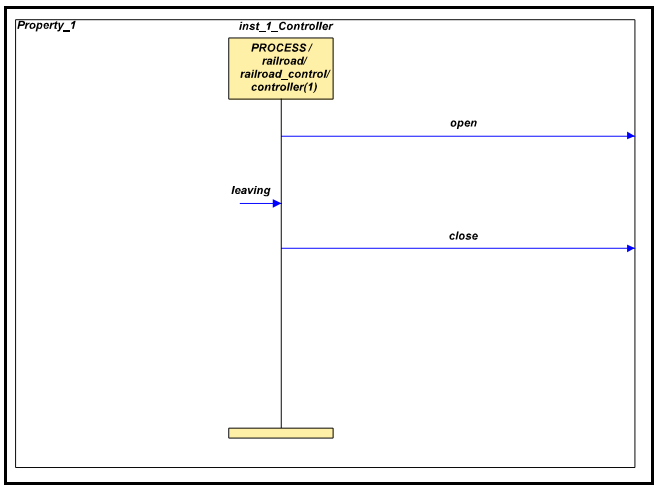


Figure 15 - Property 1

signal.

Initial condition: The system is set in train priority mode. No manual action is allowed since such an action could violate the property and is taken under operator responsibility.

The results show that the property is declared true after 6s of computation time.

```
Number of states : 13890
Number of transitions : 34244
Maximum depth reached : 79
Maximum breadth reached : 508
duration : 0 mn 6 s
Number of exceptions : 0
Number of deadlocks : 0
Number of stop conditions : 0
Transitions coverage rate : 40.00 (45 transitions not covered)
States coverage rate : 63.16 (7 states not covered)
Basic blocks coverage rate : 31.25 (88 basic blocks not covered)
Number of errors : 0
Number of success : 0
  observer obs: 0 errors, 0 success
  observer nogobetweenopenandclose: 0 errors, 0 success
  observer property_1: 0 errors, 0 success
```

4.3 Property 2

No go signal is sent if the door is open,

The property is described with the following MSC. Such a MSC means " if the gate is open, the controller cannot send the go signal which sets a light to green". The signal close is part of the MSC since the property is violated only if the go signal occurs between the open and close signals.

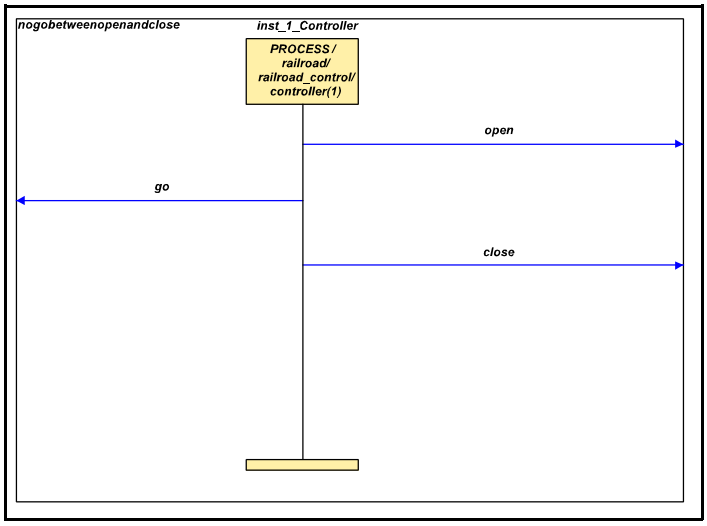


Figure 16 - Property 2

Conditions : The system is set in train priority mode and manual actions are authorised to change the priority mode.

This property has been verified with Property 1 under the train priority mode.

4.4 Property 3

The gate cannot be opened if train are passing.

The property is defined as a simple stop condition:

"Stop if output open and controller ! passing > 0" which means "Stop if the controller sends an open signal when at least one train is passing".

Conditions: The system is set in train priority mode and manual actions are authorised to change the priority mode.

That property has been verified with P1 and P2. See the results given for P1.

4.5 Property 4

Trains do not wait more than time needed to close the gate in train priority mode,

This property aims at verifying that the train priority mode really gives priority to trains. In other words, it means that trains do not wait more than the time needed to close the gate. The waiting time starts when the train passes over the approaching sensor and ends when the lights is set to green by the controller.

The difficulty in verifying this property lays on the fact that timing properties cannot be described easily with observers. In exhaustive mode, the time variable (NOW) is not incremented since it would otherwise led to infinite number of states. As a consequence, time can only be observed with timers. Since observers (MSC or GOALS) cannot contain timers, additional timers have to inserted in the model. In that case, the model is updated to include some mechanisms only required for simulation purposes.

A timer has been added to the track process type. It is set when a train signal occurs when the traffic light is red and is reset when a go is received. If a time out occurs, the track process enters a deadlock state called "time_exceeded".

The property itself is described as a double condition:

"stop if fast!state = time_exceeded"

"Stop if normal!state = time_exceeded"

That property has been verified with P1, P2 and P3. See the results given for P1.

4.6 Property 5

Fast trains do not wait more than time needed to close the gate in fast train priority mode,

This property is very similar to the previous one but only concerns fast trains.

As a consequence, the stop condition is limited to the state of the fast process.

The results shown that the property is verified. Some exceptions are generated which are due to the presence of the deadlock state in the normal process.

```
Number of states : 64465
Number of transitions : 114794
Maximum depth reached : 87
Maximum breadth reached : 2077
duration : 0 mn 22 s

Number of exceptions : 281
Number of deadlocks : 0
Number of stop conditions : 0
Transitions coverage rate : 57.33 (32 transitions not covered)
States coverage rate : 78.95 (4 states not covered)
Basic blocks coverage rate : 50.00 (64 basic blocks not covered)
Number of errors : 0
Number of success : 0
```

4.7 Property 6

Cars do not wait forever in car priority mode

The property checker has proven that this property *is not true* due to the modification C1 in the specification :

"It is not allowed to set the stopping signal for a track when a train is between the two sensors".

It is possible in an extreme situation that an unlimited number of trains follow each other on a track so that the stopping signal is never set.

4.8 Property 7

In manual mode, the gate cannot be opened if a train is passing

That property is similar to P3 with only one difference: here we authorize the manual mode. The model is set in train priority mode and manual actions, excepting a change of the priority mode, are allowed.

Results show that the property is verified. Exceptions have the same cause as above: some normal or fast processes may enter the time_exceeded state since

no priority is managed in pure manual mode.

```
Number of states : 141870
Number of transitions : 301492
Maximum depth reached : 85
Maximum breadth reached : 4441
duration : 0 mn 58 s

Number of exceptions : 1340
Number of deadlocks : 0
Number of stop conditions : 0
Transitions coverage rate : 69.33 (23 transitions not covered)
States coverage rate : 89.47 (2 states not covered)
Basic blocks coverage rate : 51.56 (62 basic blocks not covered)
Number of errors : 0
Number of success : 0
```

4.9 Property 8

In manual mode, the gate cannot be opened if not all traffic lights are red,

That property can be described with the following assertions used as stop conditions:

“Stop if output open and fast!state=green”.

“Stop if output open and normal!state=green”

The result show that the property can be violated as the “wait” and “open” signals are sent by an unique transition. Although that situation violates the property in the formal sense, it does not mean that the system is unsafe since the wait signals have been sent. Exceptions are still present for the same reason as above.

```
Number of states : 141864
Number of transitions : 301480
Maximum depth reached : 85
Maximum breadth reached : 4441
duration : 1 mn 0 s

Number of exceptions : 1340
Number of deadlocks : 0
Number of stop conditions : 4
Transitions coverage rate : 69.33 (23 transitions not covered)
States coverage rate : 89.47 (2 states not covered)
Basic blocks coverage rate : 51.56 (62 basic blocks not covered)
Number of errors : 0
Number of success : 0
```

4.10 Property 9

In manual mode, a traffic light cannot be set to green is the gate is open.

This property is similar to P2 but this time manual operations are allowed.

The results show that property 9 is verified. Exceptions have the same cause as above.

```
Number of states : 181874
Number of transitions : 383240
Maximum depth reached : 85
Maximum breadth reached : 5177
duration : 1 mn 17 s

Number of exceptions : 1940
Number of deadlocks : 0
Number of stop conditions : 0
Transitions coverage rate : 69.33 (23 transitions not covered)
States coverage rate : 89.47 (2 states not covered)
Basic blocks coverage rate : 51.56 (62 basic blocks not covered)
Number of errors : 0
Number of success : 0
  observer obs: 0 errors, 0 success
  observer nogobetweenopenandclose: 0 errors, 0 success
```

5 Conclusion on simulation

In this section, we have shown the use of the interactive and exhaustive simulations modes to verify and validate the behaviour of our specification. The main result is that the model fulfills all the key properties that the user requirements analysis has put into focus.

As a conclusion, we can say that the model matches the user requirements and exhibits a safe behaviour even in manual mode.

One interesting outcome of this simulation activity is that we have been able to measure the complexity of the system. Although the model seems simple, the number of states and possible transitions can go up to more than 3 millions. This number can even be larger if more tracks are added. Nevertheless, with a “divide and conquer” approach, an exhaustive simulation leading to proof of properties is always feasible.

V. Conclusion

The SDL modelling of the railway crossing was an interesting exercise. It demonstrated that an apparently simple system can hide an internal complexity which can only be revealed by means of simulation. With a standard design approach, such a complexity would be revealed later during the coding and testing phases. This is the power of the SDL modelling, which can validate a specification and a design before writing any line of executable code. We could resume this by opposing the “validate and code” approach to the “code and bug-hunting” approach, which is unfortunately still used in most projects.