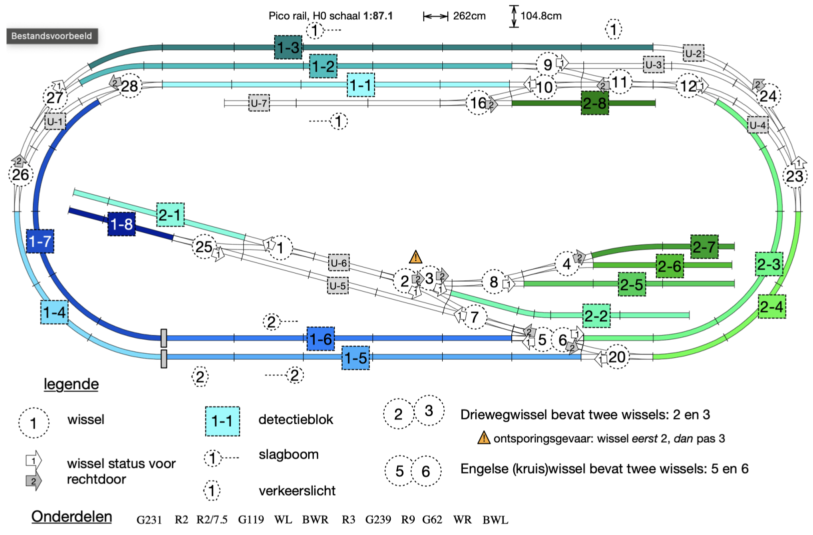
Programming Project 2: Control system specification

Mathis Arthur Boumaza
0555462
Mathis.Arthur.Boumaza@vub.be
Preperatory programme in Computer Science

Academic year 2023-2024

1 Application

The control system application is an implementation of a system that allows to manipulate a model railway, more specifically the railway depicted in figure 1. This hardware has been made available on top of which software was written implementing the specified system. The railway consists of different components which include barriers, switches, tracks, trains, and lights of which their state can be adjusted. In order to implement this system on top of the hardware several modules must be made which should work together. On a high-level the modules consist of a NMBS module, which allow users to interact with the railway and plan trajectories, and an INFRABEL module, which will manage the underlying hardware and avoid collisions of trains. These 2 major modules will run on seperate devices. More specifically, the NMBS module will run on the user's computer device while the INFRABEL module will run on a Raspberry PI. The modules will then interact through a TCP connection, realizing a fully functioning control system. In the following sections a more detailed explanation can be found on the inner workings of the system.



Pico rail, H0 schaal 1:87.1
262cm
104.8cm
Bestandsvoorbeeld
1-1
1-2
1-3
1-4
1-5
1-6
1-7
1-8
2-1
2-2
2-3
2-4
2-5
2-6
2-7
2-8
U-1
U-2
U-3
U-4
U-5
U-6
U-7
legende
wissel
wissel status voor rechtdoor
detectieblok
slagboom
verkeerslicht
Driewegwissel bevat twee wissels: 2 en 3
ontsporingsgevaar: wissel eerst 2, dan pas 3
Engelse (kruis)wissel bevat twee wissels: 5 en 6
Onderdelen
G231 R2 R2/7.5 G119 WL BWR R3 G239 R9 G62 WR BWL

Figure 1: Railway track.

2 Software architecture

In order to develop such an application, differents ADTs must be constructed to implement the previously mentioned modules. Figure 2 shows the different ADTs and how they are linked together. More specifically, an arrow indicates that an ADT is dependent on another ADT.

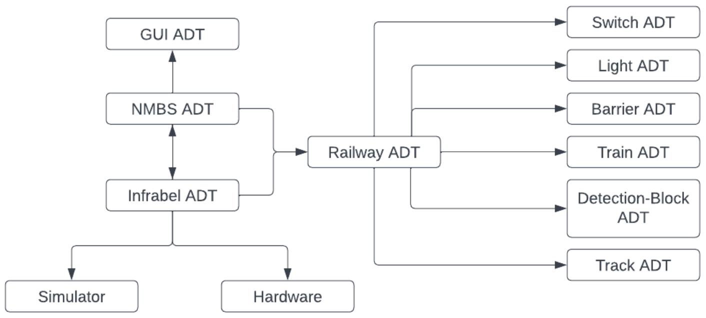


Figure 2: Dependencies between the different components.

In total there are 10 ADTs, 6 of which represent components of the track. More specifically the switch, light, barrier, train, detection block and track ADT represent the different components of the track and are independent of everything, they are stand alone. The railway ADT then uses these ADTs to make a representation of the railway using a graph. This representation of the railway through the railway ADT is then used by the NMBS and Infrabel ADT to have their own representation of the railway. This is necessary if we want to establish a TCP connection between the Infrabel and NMBS ADT. Moreover, the NMBS ADT relies on the GUI ADT to receive input from the user and adjust its railway state. Finally, the Infrabel ADT also relies on the simulator or hardware to effectively reflect the user input on the real railway. In the next section the ADTs and their exact operations will be discussed in more detail.

3 Software modules

3.1 Barrier ADT

The barrier ADT represents barrier objects on the railway and does not depend on any other ADT.

name	signature
make-barrier-adt	(symbol $\rightarrow$ barrier)
get-name	($\emptyset \rightarrow$ symbol)
open!	($\emptyset \rightarrow \emptyset$)
close!	($\emptyset \rightarrow \emptyset$)
open-barrier?	($\emptyset \rightarrow$ boolean)

Table 1: The operations of the barrier ADT

The barrier ADT has the following operations:

- *make-barrier-adt* takes in a name and gives back a barrier object with the given name.
- *get-name* gives back the name of the barrier object.

- *open!* and *close!* respectively opens and closes the barrier object.

- *open-barrier?* checks whether the barrier object is open. It gives back #t when it's open and #f when this is not the case

3.2 detection block ADT

The detection block ADT represents detection blocks on the railway and does not depend on any other ADT.

name	signature
make-detection-block-adt	(symbol $\rightarrow$ detection-block)
get-name	($\emptyset \rightarrow$ symbol)
get-presence	($\emptyset \rightarrow$ boolean)
change-presence!	($\emptyset \rightarrow \emptyset$)

Table 2: The operations of the detection block ADT

The detection block ADT has the following operations:

- *make-detection-block-adt* takes in a name and gives back a detection block object with the given name.

- *get-name* gives back the name of the detection block object.

- *get-presence* gives back the presence of the given detection block object. It either gives back #f when there is no presence of a train object, and #t when there is presence.

- *change-presence!* change the presence state of the detection block to the given input, which should be a boolean.

3.3 Light ADT

The light ADT represents lights on the railway and does not depend on any other ADT.

name	signature
make-light-adt	(symbol $\rightarrow$ light)
get-name	($\emptyset \rightarrow$ symbol)
get-state	($\emptyset \rightarrow$ symbol)
change-light!	(symbol $\rightarrow \emptyset$)

Table 3: The operations of the light ADT

The light ADT has the following operations:

- *make-light-adt* takes in a name and returns a light object with the given name.

- *get-name* gives back the name of the light object.
- *get-state* gives back the state in which the light object is in.
- *change-light!* changes the state of the light object into the given state.

3.4 Switch ADT

The switch ADT represents the switches on the railway and does not depend on any other ADT.

name	signature
make-switch-adt	(symbol $\rightarrow$ switch)
get-name	($\emptyset \rightarrow$ symbol)
current-position	($\emptyset \rightarrow$ number)
change-position!	(number $\rightarrow \emptyset$)

Table 4: The operations of the Switch ADT

The switch ADT has the following operations:

- *make-switch-adt* takes in a name and returns a switch object with the given name.
- *get-name* gives back the name of the switch object.
- *current-position* gives back the current position in which the switch object is in.
- *change-position!* changes the position of the switch to the given switch position. It only allows the numbers 1 and 2 as input, each representing a switch state.

3.5 Train ADT

The Train ADT represents the trains on the railway and does not depend on any other ADT.

name	signature
make-train-adt	(symbol, symbol, symbol $\rightarrow$ train)
get-name	($\emptyset \rightarrow$ symbol)
get-initial-track	($\emptyset \rightarrow$ symbol)
get-initial-track-behind	($\emptyset \rightarrow$ symbol)
get-current-speed	($\emptyset \rightarrow$ number)
change-speed!	(number $\rightarrow \emptyset$)

Table 5: The operations of the Train ADT

The train ADT has the following operations:

- *make-train-adt* which takes in a name, the name of initial track and the track behind the initial track. It gives back a train object on the specified track with the given name.

- *get-name* gives back the name of the train object.
- *get-initial-track* gives back the name of the initial track on which the train object was put.
- *get-initial-track-behind* gives back the name of the track behind the initial track.
- *get-current-speed* gives back the current speed of the train object.
- *change-speed!* changes the speed of the train object to the given speed.

3.6 Track ADT

The track ADT represents the tracks on the railway and does not depend on any other ADT. This ADT is necessary for when NMBS needs to calculate trajectories and adjust switches, barriers, ...

name	signature
make-track-adt	(symbol, symbol, symbol, symbol $\rightarrow$ track)
get-detection-block-name	($\emptyset \rightarrow$ symbol $\cup$ {#f})
get-light-name	($\emptyset \rightarrow$ symbol $\cup$ {#f})
get-switch-name	($\emptyset \rightarrow$ symbol $\cup$ {#f})
get-barrier-name	($\emptyset \rightarrow$ symbol $\cup$ {#f})

Table 6: The operations of the Track ADT

The track ADT has the following operations:

- *make-track-adt* which takes in a detection block, light, switch, and barrier name and gives back a track object.
- *get-detection-block-name* gives back the name of the detection block on the track. It either gives back a symbol representing the detection block name or #f if the track does not contain a detection block
- *get-light-name* gives back the name present around the track or #f if the track does not contain a light around it.
- *get-switch-name* gives back the name of the switch present on a track or #f if the track does not contain a switch around it.
- *get-barrier-name* gives back the name of a barrier present around a track or #f if the track does not contain a barrier around it.

3.7 Railway ADT

The railway ADT models the physical railway on which we build the software. In order to represent the railway it will depend on the train, track, switch, light, detection-block, and barrier ADT to represent the different components of the track. More specifically, hashmaps will be stored linking the name and object of the different components. In addition to that, it will store a graph connecting all the switches and detection blocks which will be necessary for trajectory planning.

name	signature
make-railway-adt	($\emptyset \rightarrow$ railway)
change-train-speed!	(symbol, number $\rightarrow \emptyset$)
get-all-trains	($\emptyset \rightarrow$ pair)
get-train-speed	(symbol $\rightarrow$ number)
change-switch-state!	(symbol, number $\rightarrow \emptyset$)
get-switch-state!	(symbol $\rightarrow$ number)
get-all-switches	($\emptyset \rightarrow$ pair)
check-barrier-open?	(symbol $\rightarrow$ boolean)
get-all-barriers	($\emptyset \rightarrow$ pair)
change-barrier-state!	(symbol, symbol $\rightarrow \emptyset$)
get-light-state	(symbol $\rightarrow$ symbol)
get-all-lights	($\emptyset \rightarrow$ pair)
change-light-state!	(symbol, symbol $\rightarrow \emptyset$)
update-detection-blocks!	(pair, pair $\rightarrow \emptyset$)
get-detection-block-state	(symbol $\rightarrow$ boolean)
get-all-detection-blocks	($\emptyset \rightarrow$ pair)

Table 7: The operations of the Railway ADT

The railway ADT has the following operations:

- *make-railway-adt* makes a railway object.
- *add-train!* takes in the train name, the name of the initial track, and the track behind the initial track and gives back #f when a train object already has this name and #t when the train object has been added to the railway.
- *change-train-speed!* changes the speed of the train object with the given name to the given speed.
- The operations *get-train-speed*, *get-switch-state*, *get-light-state*, and *get-detection-block-state* respectively gives back the train speed, the switch, light and detection block state with the specified name.
- the operations *get-all-trains*, *get-all-switches*, *get-all-barriers*, *get-all-lights*, and *get-all-detection-blocks* respectively gives back an association list of all the trains names, all the possible switch, barriers, light, and detection block names together with their state.
- *change-switch-state!* changes the state of the switch object with the given name to the given state.

- *check-barrier-open?* gives back a boolean, representing whether or not the barrier object with the given name is open or not. It gives back #t when the barrier object is open and #f when it's closed.
- *change-barrier-state!* changes the state of the barrier object with the given name to the given state.
- *change-light-state!* changes the state of the light object with the given name to the given state.
- *update-detection-blocks!* is the operation that takes in a list of occupied detection block names and a list of all the possible detection block names and updates the state of the detection block objects on the railway.

3.8 GUI ADT

The GUI ADT represents the GUI part of the application which is used to receive input from the user. It does not depend on any other ADT.

name	signature
make-gui-adt	((symbol, number $\rightarrow \emptyset)^1$ $(\emptyset \rightarrow$ pair$)^2$ (symbol, number $\rightarrow \emptyset)^3$ $(\emptyset \rightarrow$ pair$)^4$ (symbol, symbol $\rightarrow \emptyset)^5$ $(\emptyset \rightarrow$ pair$)^6$ $(\emptyset \rightarrow$ pair$)^7$ (symbol, number $\rightarrow \emptyset)^8$ (symbol, symbol, symbol $\rightarrow \emptyset)^9$ $(\emptyset \rightarrow$ pair$)^{10}$ $\rightarrow$ gui)

Table 8: The operations of the GUI ADT

The GUI ADT has the following operation:

- *make-gui-adt* takes in 10 callback procedures and makes the GUI object, which will draw everything on the screen. The
  - The first 2 procedures are respectively callbacks for adjusting the switch objects and to retrieve the switch names and their state in an association list.
  - Procedures 3 through 4 are respectively callbacks for adjusting barrier objects and to retrieve the barrier names and their state in an association list.
  - Procedures 5 through 6 are respectively callbacks for adjusting the light objects and to retrieve the light names and their state in an association list.

  – Procedure 7 is a callback retrieving the detection block names and their state in an association list.
  – Procedure 8 through 10 are respectively procedures for adjusting the train objects' speed, adding a train object to the railway, and retrieving the train names and their speed in an association list.

3.9 NMBS ADT

The NMBS ADT represents the companies riding on the tracks. More specifically, it will receive input from the GUI ADT as well as the Infrabel ADT and adjust its railway representation accordingly. This means that the NMBS ADT depends on the GUI, Infrabel and railway ADT. The communication with the Infrabel ADT currently happens with output of their operations being passed as input in the Infrabel's operations. This is to be replaced by a TCP connection.

name	signature
make-nmbs-adt	$(\emptyset \rightarrow \text{nmbs})$
retrieve-all-switches	$(\emptyset \rightarrow \text{pair})$
apply-trains!	$(\text{symbol, symbol, symbol} \rightarrow \emptyset)$
retrieve-all-barriers	$(\emptyset \rightarrow \text{pair})$
retrieve-all-lights	$(\emptyset \rightarrow \text{pair})$
retrieve-all-trains	$(\emptyset \rightarrow \text{pair})$
update-detection-blocks!	$(\text{pair} \rightarrow \emptyset)$

Table 9: The operations of the NMBS ADT

The NMBS ADT has the following operations:

- *make-nmbs-adt* gives back an NMBS object.
- *retrieve-all-switches* gives back an association list of the switch names with their current state.
- *apply-trains!* places a train object on the railway representation the NMBS object uses given the name, initial track name, and the name of the track behind the initial track.
- *retrieve-all-barriers* gives back an association list of the barrier names with their current state.
- *retrieve-all-lights* gives back an association list of all the light names with their current state.
- *retrieve-all-trains* gives back an association list of all the train names with their current speed.
- *update-detection-blocks!* takes in a pair with the occupied detection blocks' names and all the detection blocks' names, and updates the states in the railway representation in NMBS.

3.10 Infrabel ADT

At last, the Infrabel ADT represents the company managing the underlying hardware railway. More specifically, it receives input from the NMBS ADT and changes its representation of the railway, as well as the actual underlying hardware.

name	signature
make-infrabel-adt	($\emptyset \rightarrow$ infrabel)
update-switches!	(pair $\rightarrow \emptyset$)
update-lights!	(pair $\rightarrow \emptyset$)
update-barriers!	(pair $\rightarrow \emptyset$)
update-trains!	(pair $\rightarrow \emptyset$)
update-detection-blocks!	($\emptyset \rightarrow$ pair)

Table 10: The operations of the Track ADT

The Infrabel ADT has the following operations:

- *make-infrabel-adt* gives back an infrabel object.
- *update-switches!* takes in an association list of switch names and their states and updates the railway representation as well as the underlying hardware.
- *update-lights!* takes in an association list of light names and their states and updates the railway representation as well as the underlying hardware.
- *update-barriers!* takes in an association list of barrier names and their states and updates the railway representation as well as the underlying hardware.
- *update-trains!* takes in an association list of train names and their data (the initial track, the track behind the initial track, and the speed) and updates the trains' speed on the railway representation as well as on the underlying hardware.
- *update-detection-blocks!* updates the detection blocks on the railway representation and gives back a pair containing the occupied detection blocks' names and all the detection blocks' names.

4 Conclusion

To conclude, many components are necessary to develop such a control system application, and these components must interact in a specific manner to ensure that we have a robust system. In the future improvements will be made to establish a TCP connection and on top of that additional functionalities will be implemented to make the system more enjoyable and fool proof for the user, such as trajectory planning and collision detection.