Programming Project 2:
Control system specification

Mathis Arthur Boumaza
0555462
Mathis.Arthur.Boumaza@vub.be
Preperatory programme in Computer Science
Acedmic year 2023-2024

March 17, 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.

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.

In total there are 9 ADTs, 6 of which represent components of the track. More specifically the switch, light, barrier, train, and detection block 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

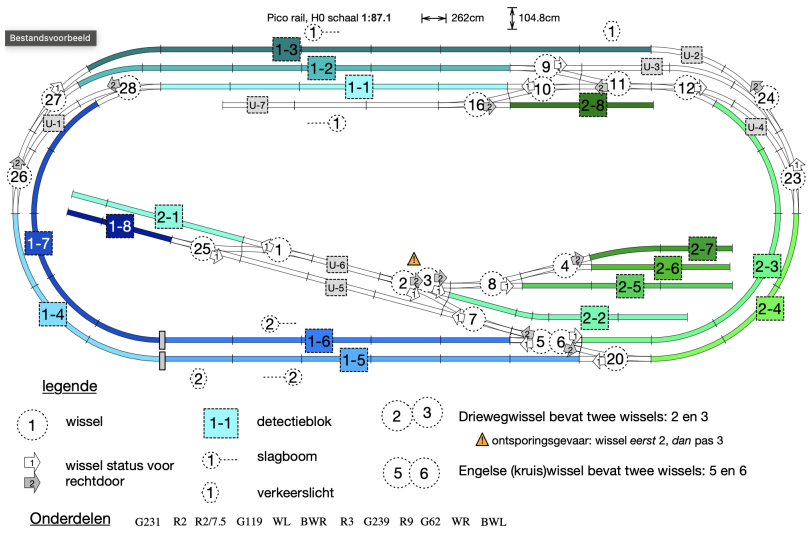

Figure 1: Railway track.

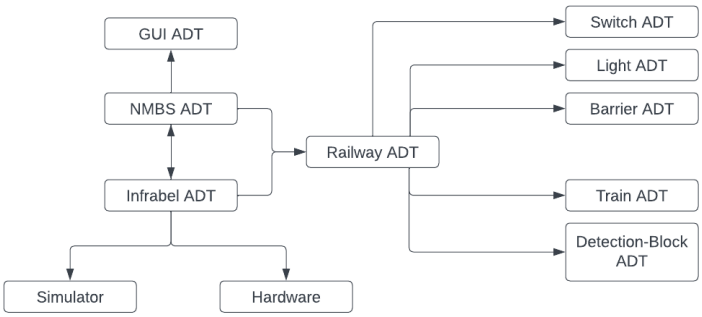

Figure 2: Dependencies between the different components.

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 as well as ensure safety. 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.

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.

name	signature
make-barrier-adt	(symbol → barrier)
get-name	(∅ → symbol)
open!	(∅ → ∅)
close!	(∅ → ∅)
open-barrier?	(∅ → boolean)

Table 1: The operations of the barrier ADT

- *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 → detection-block)
get-name	(∅ → symbol)
get-presence	(∅ → boolean)
change-presence!	(∅ → ∅)
reserve!	(symbol ∪ {#f} ∪ → ∅)
get-reservation	(∅ → symbol ∪ {#f})

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.
- *reserve!* and *get-reservation* respectively reserve the detection block(or unreserve) and get the current reservation status from the detection block object.

3.3 Light ADT

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

name	signature
make-light-adt	(symbol → light)
get-name	(∅ → symbol)
get-state	(∅ → symbol)
change-light!	(symbol → ∅)

Table 3: The operations of the light ADT

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 → switch)
get-name	(∅ → symbol)
current-position	(∅ → number)
change-position!	(number → ∅)
current-comp	(∅ → symbol)
possible-comp-states	(∅ → pair)

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.
- *current-comp* gives back the component the switch object is currently 'pointing' to.
- *possible-comp-states* gives back the components the switch object can possibly point to.

3.5 Train ADT

The Train ADT represents the trains on the railway and does not depend on any other 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* and *change-speed!* respectively give back the current speed of the train object and changes the current speed of the train object

name	signature
make-train-adt	(symbol, symbol, symbol → train)
get-name	(∅ → symbol)
get-initial-track	(∅ → symbol)
get-initial-track-behind	(∅ → symbol)
get-current-speed	(∅ → number)
change-speed!	(number → ∅)
get-trajectory-state	(∅ → pair)
change-trajectory-state!	(pair → ∅)
get-current-track	(∅ → symbol)
get-track-behind	(∅ → symbol)
change-current-track!	(symbol → ∅)
change-current-track-behind!	(symbol → ∅)
get-destination	(∅ → symbol)
change-destination!	(symbol → ∅)

Table 5: The operations of the Train ADT

- *get-trajectory-state* and *change-trajectory-state!* respectively gets the current trajectory the train object is undergoing and changes this trajectory.
- *get-current-track*, *get-track-behind*, *change-current-track!*, and *change-current-track-behind!* respectively gets the current track, the track behind the current track, changes the current track, and changes the track behind the current track of the train object.
- *get-destination* and *change-destination!* respectively gets the current destination of the train object and changes this current destination.

3.6 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, 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. More specifically it consists of two graphs, one will be a graph of detection blocks (called the simplified graph) whilst the other will also include switches in the graph (called the complex graph).

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.

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)
change-train-trajectory-state!	(symbol, number $\rightarrow \emptyset$)
get-train-trajectory-state	(symbol $\rightarrow$ pair)
get-train-track	(symbol $\rightarrow$ symbol)
get-train-track-behind	(symbol $\rightarrow$ symbol)
change-train-track!	(symbol, symbol $\rightarrow \emptyset$)
change-train-track-behind!	(symbol, symbol $\rightarrow \emptyset$)
get-train-destination	(symbol $\rightarrow$ symbol)
change-train-destination!	(symbol, symbol $\rightarrow \emptyset$)
get-switch-comp-state	(symbol $\rightarrow$ symbol)
get-switch-possible-comp-states	(symbol $\rightarrow$ pair)
detection-block-reserve!	(symbol, symbol $\cup$ {#f} $\rightarrow \emptyset$)
get-detection-block-reservation	(symbol $\rightarrow$ symbol)
get-all-detection-block-reservation-states	($\emptyset \rightarrow$ pair)
compute-path-complex	(symbol, symbol $\rightarrow$ pair)
compute-path-simplified	(symbol, symbol $\rightarrow$ pair)
get-track-neighbour	(symbol $\rightarrow$ pair)

Table 6: The operations of the Railway ADT

- 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.
- *change-train-trajectory-state!* and *get-train-trajectory-state* respectively changes the train trajectory of the train object and gets the trajectory.
- *get-train-track*, *get-train-track-behind*, *change-train-track*, and *change-train-track-behind!* respectively gets the track of the train object, gets the track behind, changes this track and changes the track behind.
- *get-train-destination* and *change-train-destination!* respectively gets the train object his current destination and changes it.
- *get-switch-comp-state* and *get-switch-possible-comp-states* respectively gets the switch object, the current component it is 'pointing' to and the possible components it can 'point' to.
- *detection-block-reserve!*, *get-detection-block-reservation* and *get-all-detection-block-reservations-states* respectively reserves the detection block object, gets the object his reservation status, and gets all the detection blocks that are reserved.
- *compute-path-complex* and *compute-path-simplified* respectively compute a path from the start and destination tracks using a graph with both switches and detection blocks, and only detection blocks.
- *get-track-neighbour* gets the neighbour of a given detection block.

3.7 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.

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.

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 7: The operations of the GUI ADT

- 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.

3.8 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	($\emptyset \rightarrow$ nmbs)
retrieve-all-switches	($\emptyset \rightarrow$ pair)
apply-trains!	(symbol, symbol, symbol $\rightarrow \emptyset$)
retrieve-all-barriers	($\emptyset \rightarrow$ pair)
retrieve-all-lights	($\emptyset \rightarrow$ pair)
retrieve-all-trains	($\emptyset \rightarrow$ pair)
update-detection-blocks!	(pair $\rightarrow \emptyset$)
add-trajectory!	(symbol, symbol $\rightarrow \emptyset$)
retrieve-trajectories	($\emptyset \rightarrow$ pair)
compute-trajectory	(symbol, symbol $\rightarrow$ pair)

Table 8: 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.
- *add-trajectory!* adds a trajectory to the trajectories to be executed by the train objects. It does this by getting the train name and destination and then computing the trajectory and then adding it to the to be executed trajectories.
- *retrieve-trajectories* returns a list of all the trajectories that need to be executed by the train objects.
- *compute-trajectory* computes a trajectory from the start to the destination.

3.9 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)
retrieve-all-switches	($\emptyset \rightarrow$ pair)
add-trajectories!	(pair $\rightarrow \emptyset$)
update-trajectories!	($\emptyset \rightarrow \emptyset$)
update-train-positions!	($\emptyset \rightarrow \emptyset$)
retrieve-all-trains	($\emptyset \rightarrow$ pair)
retrieve-DB-reservations	($\emptyset \rightarrow$ pair)

Table 9: 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.
- *add-trajectories!* adds a list of trajectories with their associated trains to a hashmap of trajectories to be processed.
- *update-trajectories!* updates the trajectories that were executed or to be executed, executing the rest of the trajectory.
- *update-train-positions!* updates the positions of the train. More specifically, detecting where they are and then changing the train objects their current track and track behind.
- *retrieve-all-trains* and *retrieve-DB-reservations* respectively get all the trains with their data and the detection blocks with their reservation data.

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. The system has already been made foolproof to avoid users colliding trains against each other.