Vrije Universiteit Brussel

Programeer Project 2

2016-2017

Documentation

Naam:
Carlos Montero

Rolnummer:
0500677

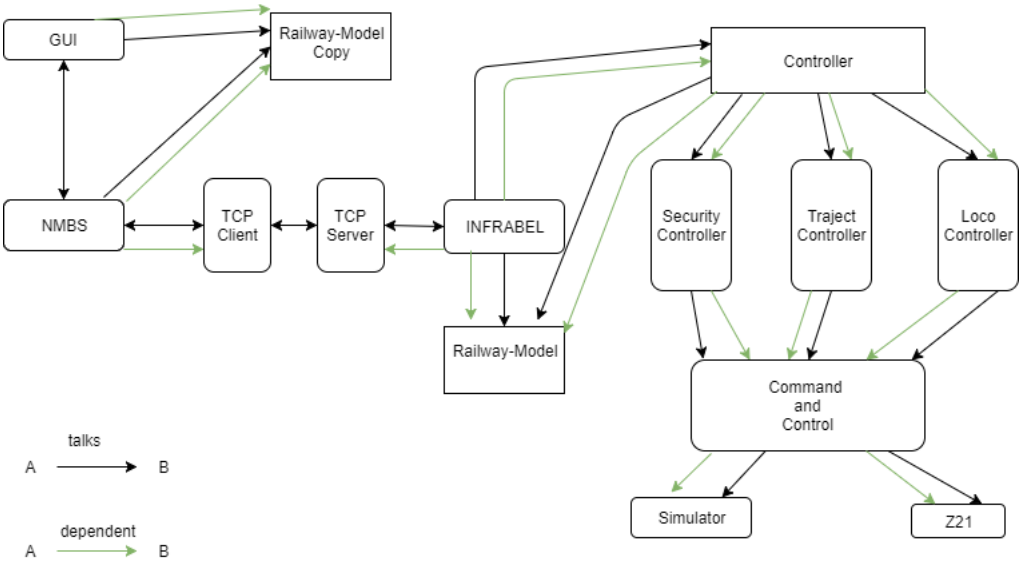
21 Augustus 2017

Contents

1 Introduction

This program simulates the interaction between real life companies NMBS and INFRABEL, where a user can send some requests through a Graphical User Interface. NMBS and INFRABEL will in that scenario communicate with each other to support the requests of the user. INFRABEL has the possibility to control a simulated railway-model or a hardware railway-model.
This documentation has as purpose to inform programmers about the interface of the different ADT's and how the main components communicate with each other.
In the first section we will mention the architecture of all the main components: their dependencies and communication. In section 3, 4, 5 and 6 we will discuss in more detail the interface of each ADT and also how the small components communicates with main components. In the last section we will examine how the main components NMBS, INFRABEL, Railway-model and GUI communicates with each other.

2 Overview of the architecture



3 Railway-Model

The railway-model is one of the main components in the program. It is composed of all the elements that simulates a real railway architecturew, such as locomotives, switch's, tracks etc. For each ADT we show only the most important procedures. The not mentioned procedures are mostly getters and setters (see source code for a complete overview).

3.1 Loco ADT

Loco ADT
(new-loco loco-symbol station-symbol station-symbol) -> dispatch procedure
(loco 'set-distance!)-> (proc number) -> void
(loco 'set-speed!)-> (proc number) -> void

Loco ADT represents a locomotive. It is instantiated based on it's ID, and two stations ID's. The distance variabele can be interpeted as the distance that the loco already made while driving from the first station to second station. The distance is updated based on the driving speed of the loco (see subsection 4.4).

3.2 Station ADT

Station ADT
(new symbol int int)-> dispatch procedure

Station ADT represents a point at the end of a Rail ADT and it's instantiated with an ID-symbol and two integers that corresponds with an (x,y)-coordinate. To avoid confusion the terms node and spot are frequently used instead of station.

3.3 Rails

Rails
(new-track station-symbol station-symbol)-> dispatch procedure
(new-detection station-symbol station-symbol)-> dispatch procedure
(new-switch-track station-symbol station-symbol) -> dispatch procedure
(rail 'reserve!)-> (proc loco-symbol)-> void
(rail 'set-length!)-> (proc float)-> void

Rails is a general concept for a track ADT. We distinguish three different types of rails:

- detection type: a track that can detect if a loco is using it or not.
- switch type: a track that is part of a switch.

- track type: a regular track.

Construction of any type of rail requires two station ID's and returns a dispatch. Construction of a detection track requires an extra parameter: the ID of the detection.
Every rail can be reserved by a loco. This concept will be used by the securityController for safety reasons (see subsection 4.4). A rail has also a "length" that is used by NMBS to construct the fastest traject between two stations.

3.4 Switch ADT

Switch ADT
(new-switch station-symbol station-symbol station-symbol station-symbol-> dispatch procedure
(switch 'set-position!)-> (proc int)-> void
(switch 'reserve!)-> (proc loco-symbol)-> void

A switch ADT is constructed based on four station symbols, where the first parameter represents the middle of the switch. Switch set at position one or two allows sets the passage of the switch to respectively the third or the fourth station-symbol. Switch can also be reserved for safety reasons (see subsection 4.4).

3.5 Signal ADT

Signal ADT
(new-signal symbol)-> procedure dispatch
(signal 'set-status!)-> (proc status)->void

A signal ADT is associated with a detection track and constructed with the samed ID. It can have three different status. Green, orange and red.

- Green: associated detection track is not being used.
- Orange: the next detection track is being used.
- Red: associated detection track is being used.

3.6 Railwaymodel ADT

Railway-model ADT
(new-rwm string)-> Railway-model
(get-graph Railway-model)-> void

Railway-model ADT is constructed based on a file where its path is specified in the String parameter. This ADT combines all above mentioned ADT's and provide getters for each component. The rails and stations are kept in a graph.

4 INFRABEL

4.1 Infrabel

Infrabel Interface
(new-INFRABEL)-> procedure dispatch
(Infrabel 'handle-received-package)->(proc package)-> void
(Infrabel 'handle-traject)> (proc traject)-> void
(Infrabel 'send-update-rwm)-> void

INFRABEL is one of the main components in the program. It will handle upcoming requests from NMBS. Construction of INFRABEL results in a dispatch procedure. Requests to INFRABEL are sent in the form of a package (see subsection 7.1). The requests are:

- handle traject: request to make one of the loco's follow a traject. The request is handled by Controller.
- send update rwm: request to send the current state of the railway-model to the client.

4.2 Traject ADT

Traject ADT
(new-traject list)-> traject
(current-track traject)-> rail
(next-detection traject)-> detection track
(previous-detection traject)-> detection track
(advance! traject)-> void

Traject ADT is instantiated with a list of rails and corresponds with the traject that a loco has to follow. Their exists multiple procedures to navigate through the traject such as current-track, next-track,etc. Traject ADT keeps reference to the next- and previous-detection relative to the current-track.

4.3 Todo ADT

Todo ADT
(new-todo String bag)-> todo
(make-bag traject)-> bag
(time-to-check-delay? bag)-> Boolean
(safe-msg? string)-> Boolean

A todo ADT is needed by the controller to update a loco currently involved in a traject. Every todo is composed of a message and a bag, respectively

the task that needs to be executed and the data used by the Controller to execute the tasks. Examples of todo messages:

- check-safety-msg: Controller needs to check safety.
- detection-msg: a new detection has been reached. Controller needs to make some updates.
- wait-msg: Controller needs to stop the loco and make it wait during a period of time.
- delay-logic-msg: Controller needs to correct the delay of the logic loco with the simulation/hardware loco.

The bag will store information such as the traject, the waited time and time-before-delay. If the loco has to wait due to safety reasons the waited time will be managed. The time-before-delay is a time that indicates whenever the Controller needs to test if there is some delay between the logic loco and the hardware/simulation loco.

4.4 Controller

The Controller is the update loop of INFRABEL. With the help of Security-Controller, Loco-Controller and Traject-Controller, the controller will make sure that the current state of the railway-model matches the hardware state: loco's position, signal colour's, switch's, etc.. will be updated.
The controller is composed of three parts:

- Security-Controller: will update the signals, check security, tests for logic- or hardware-loco delay. Results in a todo message.
- Loco-Controller: will update the position of the logic loco, derive the driving direction, send change-loco-speed commands to commandControl. Results in a todo message.
- Traject-Controller: will reserve and free tracks, update the switch's positions and correct a possible delay in the traject.

5 NMBS

NMBS Interface
(nmbs boolean)-> procedure dispatch
(dispatch 'handle-gui-command)->(proc lst)-> void
(dispatch 'new-traject)->(proc lst)-> void

NMBS is one of the main components in the program. It's construction requires a boolean. The boolean value will be send to INFRABEL. INFRABEL will at that point connect with the simulator for a true value otherwise a connection with the command station Z21 will be established. Whenever a user inserts a command through the GUI, the GUI will inform NMBS to handle the command. NMBS will also built a copy of the railway-model and keep it up to date throught request to INFRABEL. NMBS also provides the possibility to comppose a traject. The composed traject is generated based on Dijkstra algorithm [1].

6 Graphical User Interface

GUI
(new-gui railway-model NMBS)-> procedure dispatch
(GUI 'show-command-box)-> void
(GUI 'show-instructions)-> void
(GUI 'draw-all)-> void

Construction of the GUI requires a reference to the railway-model and NMBS. Requests by the user will be caught on a command-box and then passed to

[1] Algorithms of professor Wolfgang De Meuter have been used. In the source code there is a folder called ExternLib: all third party algorithms.

NMBS for process. Whenever the GUI is started it shows an "instructions window" as a briefly reminder on how to use and interpret the GUI. Draw all will redraw on the canvas all the components from the railway-model.

7 Communication between Components

For this section it is recommended to keep in mind the architecture of the program (see section 2).

7.1 Package ADT

Package ADT
(loco-package loco-symbol station-symbol station-symbol int)->loco-package
(traject-package loco-symbol tracks-package) ->traject-package
(signal-package signal-symbol status)-> signal-package
(switch-package middle-station-symbol int)-> switch-package

Package ADT is the communication data between NMBS and INFRABEL. We can make different types of package:loco-package, switch-package, track-package,etc; There exists one type of package for each component of the railway-model. During a "send-update-rwm" request (see subsection 4.1) a list of packages, that matches current state of the hardware model, will be send to the client.

7.2 TCPserver and TCPclient

TCPclient and TCPserver Interface
(newTCPListener Infrabel)-> server-dispatch
(newTCPConnection)-> client-dispatch
(server/client send-package)-> void

Communication between INFRABEL and NMBS happens through a TCP connection. Whenever newTCPListener is called, by INFRABEL, a listener will be triggered and will wait for a connection. Every request to the server will be handled by a separate thread, who will pass the received package to INFRABEL. Construction of TCPclient, via newTCPListener, will initiate a connection with the server. Both server and client can send a package to each other.

7.3 Command and Control

Command and Control Interface
(start-connection Boolean)-> void
(stop-connection)-> void
(start-loco loco-symbol float)-> void
(stop-loco loco-symbol)-> void
(change-switch! switch-symbol int)-> void
(get-detection-loco loco-symbol)-> detection-id or false

Command and Control is the interface used by the Controller to communicate with the simulator or the hardware. Start-connection will establish a connection with the simulator or the hardware depending on the value of the parameter. True connects with the simulator, false initiates a UDP connection with the command station Z21.

8 Conclusion

The program works fine in a simulated situation as in a real life connection. It lacks though of functionalities, but can easily be extended with the provided interfaces.