Railsimulator

Faur Ioan-Aurel

Abstract

În cadrul orei de Sisteme de Control Distribuit sa pus problema implementării unui simulator de reţea ferată. Acesta, pornind de la o hartă feroviară şi un tabel cu id-urile trenurilor care vor circula pe această hartă, va trebui să construiască dinamic orarul trenurile, specificând în prealabil ruta trenurilor şi timpul minim şi maxim în care trenul va parcurge o anumită locaţie din ruta specificată. Sistemul de simulare va trebui sa permită introducerea unor întârzâieri pe o anumită rută şi vizualizarea lor. Deasemenea utilizatorul ar trebui să fie atenţionat cu privire la aceste modificări.

Pentru această problemă am propus un sistem software cu o arhitectură complet distribuită care va raspunde cerinţelor de mai sus.

Introducere

Sistemul software propus să rezolve problema de mai sus se numeşte *Railsimulator* (Fig 1.) şi prin abordarea pe care o are asupra cerinţei, poate să îndeplinească toate obiectivele de mai sus.

Metoda propusă este de a împărţi sistemul în mai multe aplicaţii mai mici, fiecare cu un scop bine definit şi care să permită o arhitectură distribuită. Am ajuns astfel la un sistem compus din una sau mai multe *interfete grafice* care nu fac altceva decât să deseneze o hartă feroviară, sau o parte din ea, şi să interpreteze comenziile transmise de o altă componentă numită *simulator.* Acesta este o mică aplicaţie care primeşte un orar al trenurilor de la un *controler* şi la o anumită unitate de timp bine definită construieşte o comandă de simulare pe care o trimite interfeţei grafice la care este conectat.

Unitatea de timp este dată de un *server de sincronizare* care la un moment bine definit trimite o simulare a unui semnal de ceas simulatoarelor conectate la el. Se consideră ca trimiterea şi primirea acestui semnal are loc aproape instantaneu. În realitate lucrurile stau cu totul altfel, astfel că sunt folosite metode mult mai avansate de sincronizare dar care sunt mai greu de implementat sau chiar mai

scumpe.

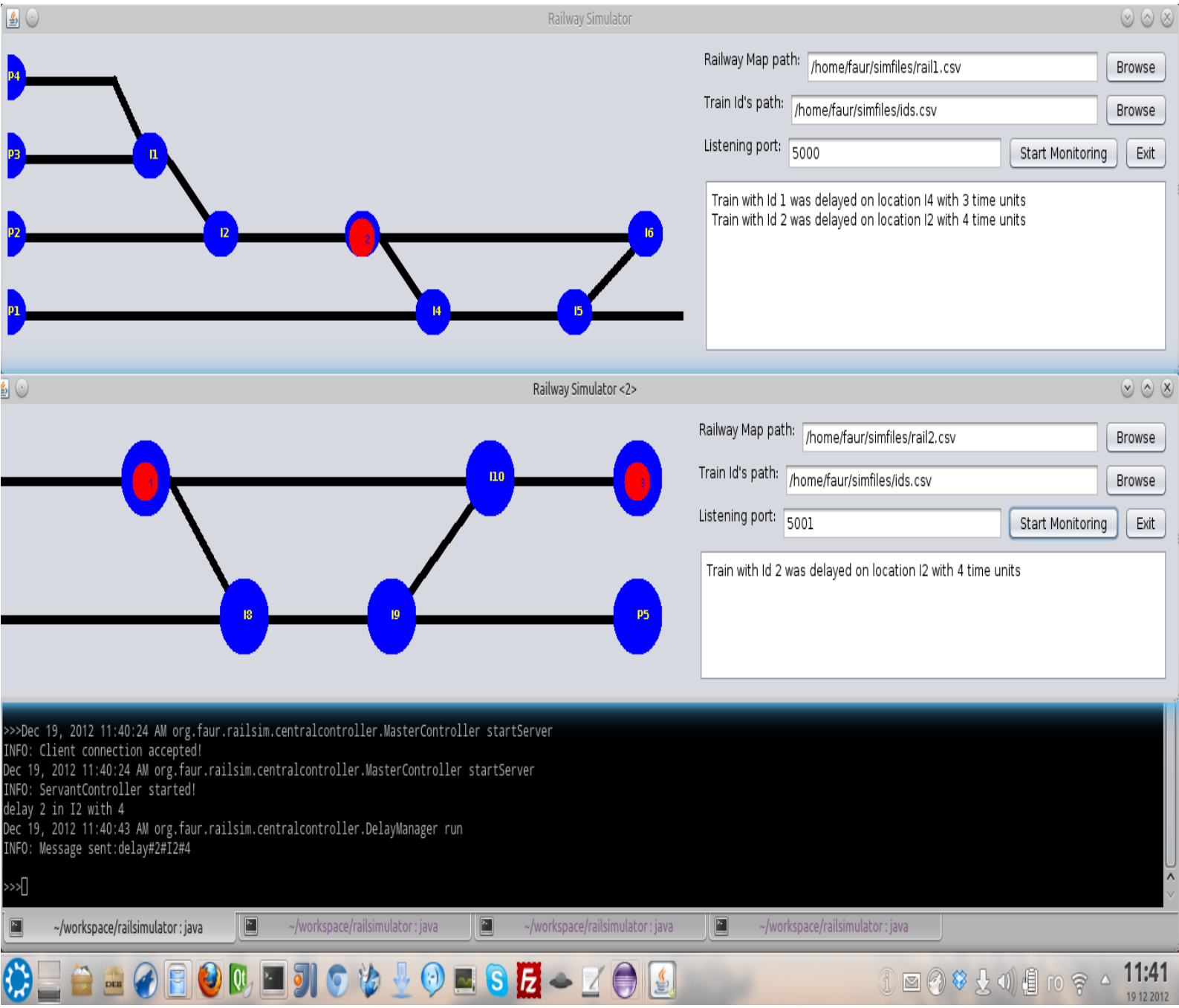


Fig. 1: Railsimulator

Componentele aplicaţiei *Railsimulator* după cum se arată şi în Fig. 2 sunt

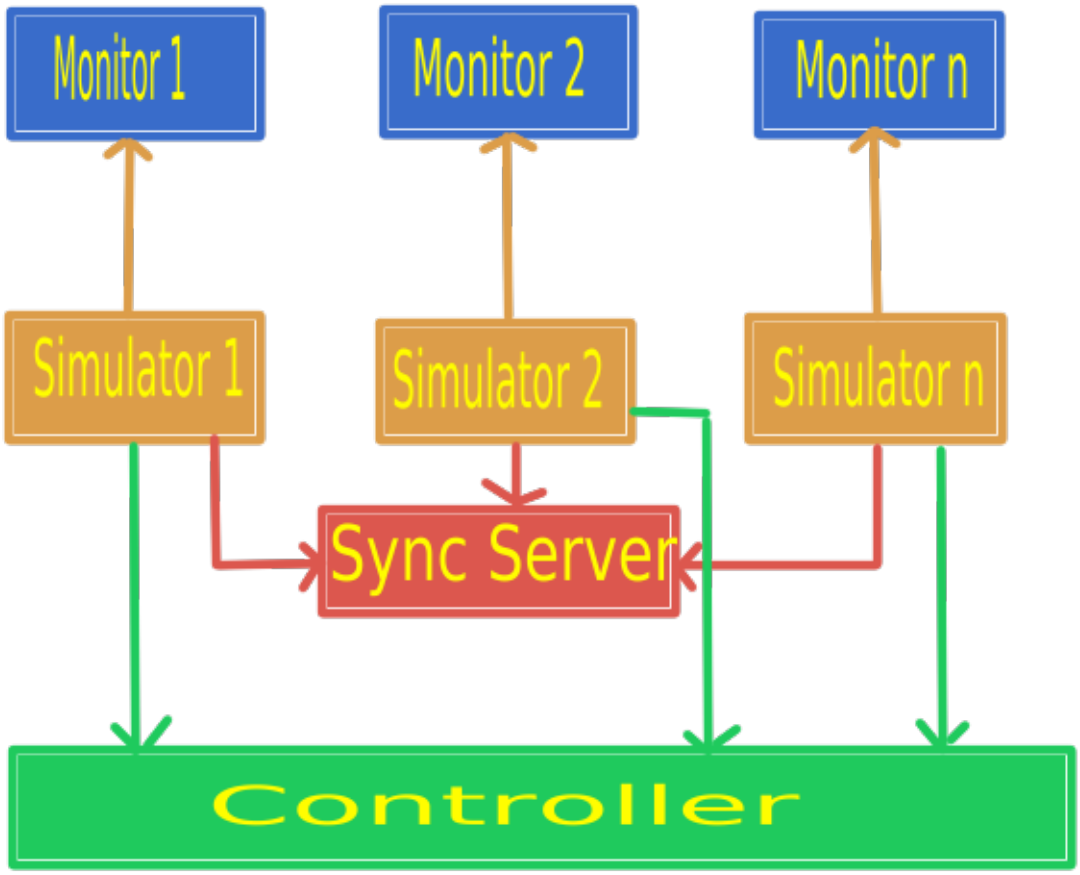


Fig. 2: Arhitectura aplicatiei Railsimulator

interfeţele grafice, denumite Monitor, Simulato-arele, Serverul de Sincronizare, şi Controlerul.

Fiecare din aceste componente este o aplicaţie de sine stătătoare iar numărul instanţelor pentru fiecare aplicaţie poate să fie oricât de mare. Ele pot fi rulate în locaţii diferite, comunicaţia între ele având loc prin intermediul protocolului TCP/IP.

După cum se poate observa în Fig. 2, există o legătură univocă între instanţele aplicaţiei *Monitor* şi instanţele aplicaţiei *Simulator.* Astfel un singur *Simulator* este conectat doar la o singură interfaţă grafică. Toate instanţele simulatoarelor de reţea ferată sunt conectate prin TCP/IP la un *Server de Sincronizare* (Sync Server din Fig. 2). Acesta din urmă va asigura execuţia simultană de către simulatoare a orarului specificat de catre controler.

Tot din Fig. 1, observăm faptul că fiecare instanţă de simulator este conectată la aceeaşi instanţă de controler. Deşi nu este necesar acest lucru, legarea tuturor simulatoarelor la un singur controler ne asigură că fiecare dintre aceste instanţe va prelucra acelaşi orar al trenurilor.

Arhitectura

În cazul în care proiectul *Railsimulator* nu a fost ataşat acestei documentaţii, vă rog consultaţi pagina de internet https://github.com/fioan89/railsimulator

Tot aici se va putea găsi versiunea la zi a acestui sistem software.

Monitor

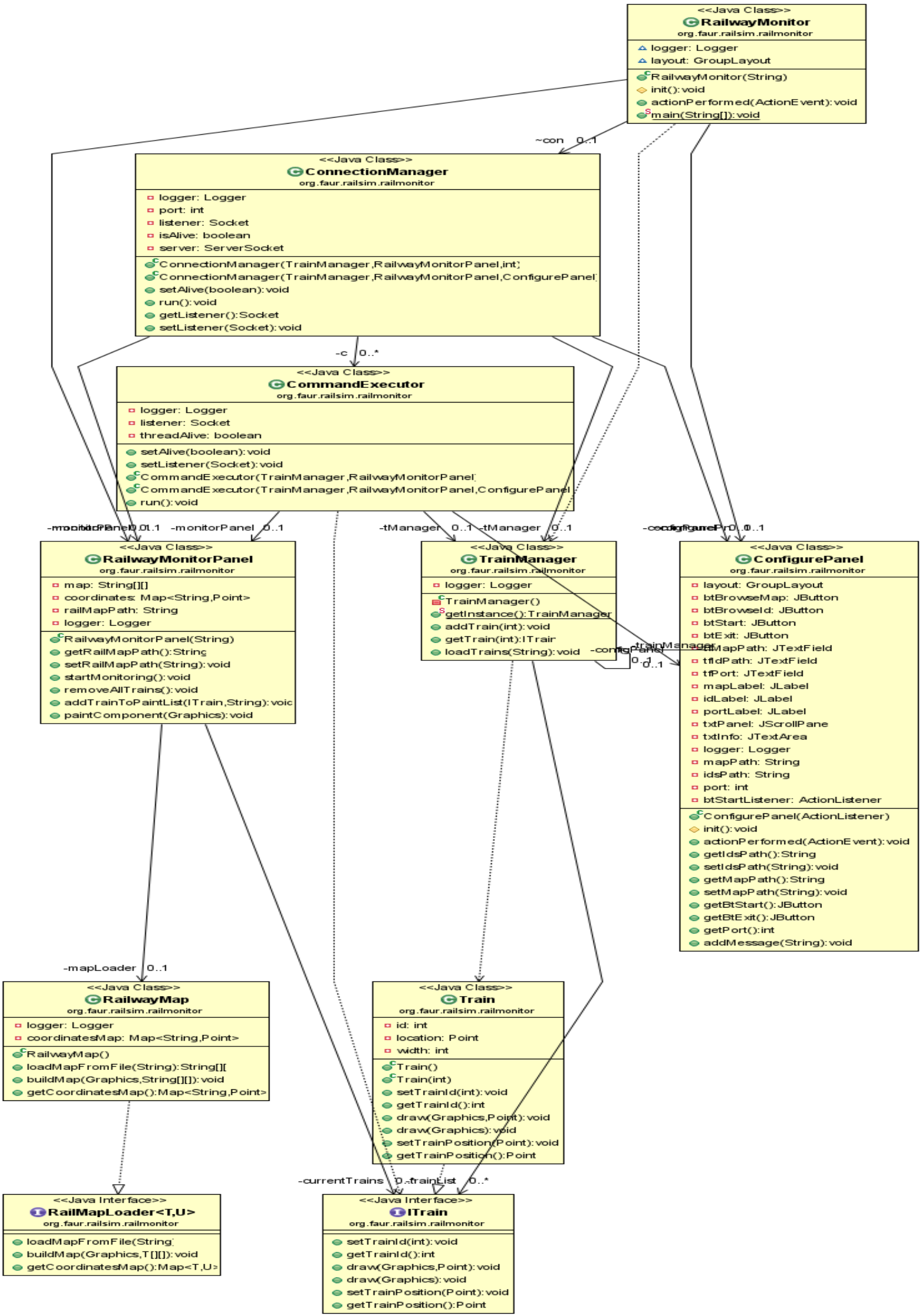
Aplicaţia Monitor este definită în pachetul *org.faur.railsim.railmonitor* din proiectul ataşat acestei documentaţii. Rolul principal al acestui pachet software este de a simula grafic reţeaua feroviară. Această simulare cuprinde desenearea sistemului feroviar, primirea comenziilor de la simulator, interpretarea şi randarea lor pe interfaţa grafică. Deasemenea *Monitor*-ul implemenetează un mecanism simplu prin care anunţă utilizatorul de întârzâier-ile apărute în reţeaua feroviară.

Interfaţa grafică este foarte flexibilă dând-ui posibilitatea utilizatorului să încarce o hartă feroviară customizată dar şi o listă cu trenurile care au voie să circule pe această hartă. Aceste opţiuni trebuie să fie date din fişiere în format *.csv* . Pentru mai multe detalii consultaţi fişierele *route1.csv* şi *ids.csv* din directorul *simfiles* aflat în directorul cu documentaţia curentă. Atenţie, pentru a vizualiza corect fisierul *.csv* trebuie ca editorul care deschide fişierul să folosească encoding-ul *UTF-8* iar ca şi line spacing caracterul spaţiu.

Comanda prin care se porneşte o instanţă a monitorului este:

```
java -cp .:./target/classes:./target/classes/commons-cli-1.2.jar org.faur.railsim.railsimulator.RailwayMonitor
```

Important de notificat este faptul că interfaţa grafică trebuie să fie instanţiată înaintea simulatorului.


RailwayMonitor
org.faur.railsim.railmonitor
logger: Logger
layout: GroupLayout
RailwayMonitor(String)
init(): void
actionPerformed(ActionEvent): void
main(String[]): void
~con 0..1
ConnectionManager
org.faur.railsim.railmonitor
logger: Logger
port: int
listener: Socket
isAlive: boolean
server: ServerSocket
ConnectionManager(TrainManager,RailwayMonitorPanel,int)
ConnectionManager(TrainManager,RailwayMonitorPanel,ConfigurePanel)
setAlive(boolean): void
run(): void
getListener(): Socket
setListener(Socket): void
-c 0..*
CommandExecutor
org.faur.railsim.railmonitor
logger: Logger
listener: Socket
threadAlive: boolean
setAlive(boolean): void
setListener(Socket): void
CommandExecutor(TrainManager,RailwayMonitorPanel)
CommandExecutor(TrainManager,RailwayMonitorPanel,ConfigurePanel)
run(): void
-monitorPanel 0..1
-monitorPanel 0..1
-tManager 0..1
-tManager 0..1
RailwayMonitorPanel
org.faur.railsim.railmonitor
map: String[][]
coordinates: Map<String,Point>
railMapPath: String
logger: Logger
RailwayMonitorPanel(String)
getRailMapPath(): String
setRailMapPath(String): void
startMonitoring(): void
removeAllTrains(): void
addTrainToPaintList(ITrain,String): void
paintComponent(Graphics): void
TrainManager
org.faur.railsim.railmonitor
logger: Logger
TrainManager()
getInstance(): TrainManager
addTrain(int): void
getTrain(int): ITrain
loadTrains(String): void
-configPanel 0..1
-trainManager 0..1
ConfigurePanel
org.faur.railsim.railmonitor
layout: GroupLayout
btBrowseMap: JButton
btBrowseId: JButton
btStart: JButton
btExit: JButton
tfMapPath: JTextField
tfIdPath: JTextField
tfPort: JTextField
mapLabel: JLabel
idLabel: JLabel
portLabel: JLabel
txtPanel: JScrollPane
txtInfo: JTextArea
logger: Logger
mapPath: String
idsPath: String
port: int
btStartListener: ActionListener
ConfigurePanel(ActionListener)
init(): void
actionPerformed(ActionEvent): void
getIdsPath(): String
setIdsPath(String): void
getMapPath(): String
setMapPath(String): void
getBtStart(): JButton
getBtExit(): JButton
getPort(): int
addMessage(String): void
-mapLoader 0..1
RailwayMap
org.faur.railsim.railmonitor
logger: Logger
coordinatesMap: Map<String,Point>
RailwayMap()
loadMapFromFile(String): String[][
buildMap(Graphics,String[][]): void
getCoordinatesMap(): Map<String,Point>
Train
org.faur.railsim.railmonitor
id: int
location: Point
width: int
Train()
Train(int)
setTrainId(int): void
getTrainId(): int
draw(Graphics,Point): void
draw(Graphics): void
setTrainPosition(Point): void
getTrainPosition(): Point
-currentTrains 0..*
-trainList 0..*
RailMapLoader<T,U>
org.faur.railsim.railmonitor
loadMapFromFile(String
buildMap(Graphics,T[][]): void
getCoordinatesMap(): Map<T,U>
ITrain
org.faur.railsim.railmonitor
setTrainId(int): void
getTrainId(): int
draw(Graphics,Point): void
draw(Graphics): void
setTrainPosition(Point): void
getTrainPosition(): Point

Simulator

Aplicaţia se găseşte în pachetul *org.faur.railsim.railsimulator* şi este responsabilă cu interpretarea tabelului primit de la controler. Acest tabel are o formă destul de simplă, pe prima linie fiind reprezentate toate locaţiile prezente pe harta feroviară generală. Pe urmatoarele linii sunt introduse id-urile trenurilor care se află la un anumit moment pe locaţia respectiva.

Un astfel de tabel are forma:

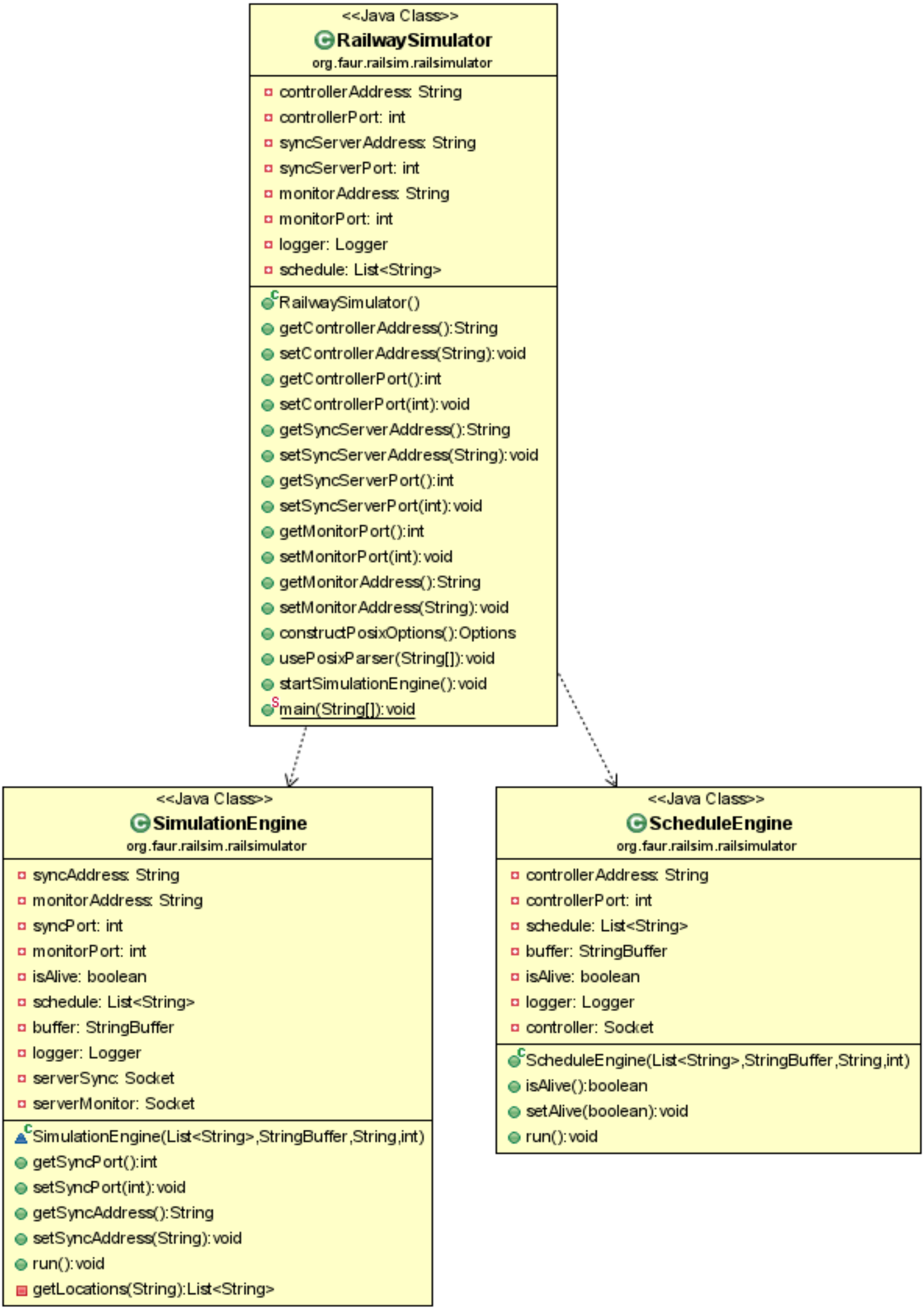
```
P1 I4 I5 I6 I7 I8 I9 P5 P4 I1 I2 I3 P6 I1P3
1  #  # # # #  # # # # # # # # # #
1  #  # # # #  # # 2 # # # # # # #
1  #  # # # #  # # 2 # # # # # # #
1  #  # # # #  # # 2 # # # # # # #
1  #  # # # #  # # 2 # # # # # # #
#  1  # # # #  # # 2 # # # # # # #
```

După primirea tabelului simulatorul se conectează la interfaţa grafică şi va începe să transmită comenzi. Aceste comenzi sunt deduse din tabel, spre exemplu pe linia 2, simulatorul va instrui interfaţa grafică să deseneze trenul cu id-ul 1 si trenul cu id-ul 2 în locaţiile P1 respectiv P4, timp de o unitate de timp. Aceste comenzi nu sunt transmise la un moment de timp aleator, ci doar atunci când serverul de sincronizare la care simulatorul este conectat trimite semnalul de tact. Acest semnal de tact este trimis la o anumită unitate de timp specificate de user atunci când porneşte serverul de sincronizare.

Comanda prin care se porneşte o instanţă a simulatorului este:

java -cp .:./target/classes:./target/classes/commons-cli-1.2.jar org.faur.railsim.railsimulator.RailwaySimulator -controllerPort 16000 -controllerAddress localhost -syncServerPort 10000 -syncServerAddress localhost -monitorPort 5000 -monitorAddress localhost

Pentru a vedea lista de parametrii vă rugăm consultaţi documentaţia codului sursă. Important de notificat este că simulatorul este pornit doar după ce interfaţa grafică, serverul de sincronizare şi controlerul au fost instanţiaţi.


<<Java Class>>
RailwaySimulator
org.faur.railsim.railsimulator
controllerAddress: String
controllerPort: int
syncServerAddress: String
syncServerPort: int
monitorAddress: String
monitorPort: int
logger: Logger
schedule: List<String>
RailwaySimulator()
getControllerAddress():String
setControllerAddress(String):void
getControllerPort():int
setControllerPort(int):void
getSyncServerAddress():String
setSyncServerAddress(String):void
getSyncServerPort():int
setSyncServerPort(int):void
getMonitorPort():int
setMonitorPort(int):void
getMonitorAddress():String
setMonitorAddress(String):void
constructPosixOptions():Options
usePosixParser(String[]):void
startSimulationEngine():void
main(String[]):void
<<Java Class>>
SimulationEngine
org.faur.railsim.railsimulator
syncAddress: String
monitorAddress: String
syncPort: int
monitorPort: int
isAlive: boolean
schedule: List<String>
buffer: StringBuffer
logger: Logger
serverSync: Socket
serverMonitor: Socket
SimulationEngine(List<String>,StringBuffer,String,int)
getSyncPort():int
setSyncPort(int):void
getSyncAddress():String
setSyncAddress(String):void
run():void
getLocations(String):List<String>
<<Java Class>>
ScheduleEngine
org.faur.railsim.railsimulator
controllerAddress: String
controllerPort: int
schedule: List<String>
buffer: StringBuffer
isAlive: boolean
logger: Logger
controller: Socket
ScheduleEngine(List<String>,StringBuffer,String,int)
isAlive():boolean
setAlive(boolean):void
run():void

Controler

Controler-ul se găseşte în pachetul *org.faur.railsim.centralcontroller* şi este este cel mai complicat sistem software din întregul sistem *Railsimulator.* Asta se datorează algoritmilor necesari pentru crearea tabelului doar dintr-o lista de rute, modificarea lui atunci când utilizatorul o cere şi trimiterea către simulator a orar-ului de fiecare dată când a fost modificat.

Primele operaţii pe care le face controler-ul sunt citirea routelor stocat într-un fişier *.csv (*vezi *routes.csv)* în următoarea formă:

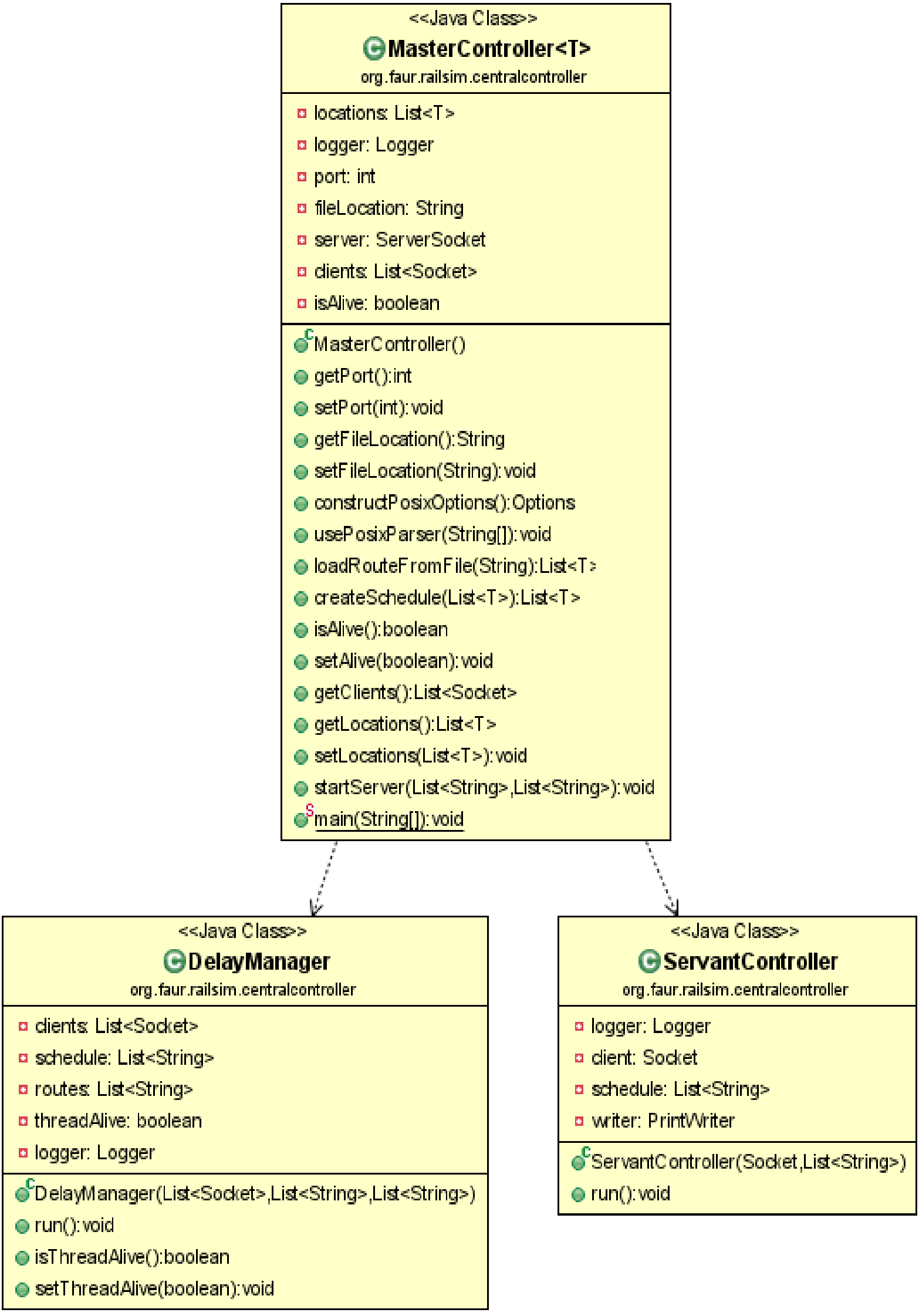
idTren
P1 I4 I5 I6 I7 I8 I9 P5
[5,7] [3,5] [2,4] [1,4] [2,3] [1,4] [1,2] [4,5]
idTren
P4 I1 I2 I3 I6 I7 I8 I9 P5
[5,8] [3,5] [2,4] [2,3] [1,5] [1,4] [1,3] [3,4] [4,6]

După citirea tabelului are loc algoritmul de rutare care stabileşte fiecărui tren în parte ora la care pleacă din gară în funcţie de ruta stabilită şi timpii necesari pentru parcurgerea fiecărei locaţii. La sfârşitul operaţiei de rutare se formează un tabel (vezi Simulator) care este trimis fiecărui simulator/client în parte.
În urmatorul pas pe care controlerul îl execută, se verifică dacă trenurile au întârzâiat într-o anumită locaţie. Dacă se întâmplă acest lucru tabelul este refăcut şi retransmis clienţilor. Întârzâierea este introdusă de la tastatură printr-o comandă care are forma: *delay <trainId> in <location> with <x>*

Comanda prin care se porneşte o instanţă a controlerului este:

java -cp .:./target/classes:./target/classes/commons-cli-1.2.jar
org.faur.railsim.centralcontroller.MasterController -port 16000 -file /patjh/to/routes.csv

Pentru a vedea lista de parametrii vă rugăm consultaţi documentaţia codului sursă. Important de reţinut este că, controlerul, este pornit înaintea simulatorului.


<<Java Class>>
MasterController<T>
org.faur.railsim.centralcontroller
locations: List<T>
logger: Logger
port: int
fileLocation: String
server: ServerSocket
clients: List<Socket>
isAlive: boolean
MasterController()
getPort():int
setPort(int):void
getFileLocation():String
setFileLocation(String):void
constructPosixOptions():Options
usePosixParser(String[]):void
loadRouteFromFile(String):List<T>
createSchedule(List<T>):List<T>
isAlive():boolean
setAlive(boolean):void
getClients():List<Socket>
getLocations():List<T>
setLocations(List<T>):void
startServer(List<String>,List<String>):void
main(String[]):void
<<Java Class>>
DelayManager
org.faur.railsim.centralcontroller
clients: List<Socket>
schedule: List<String>
routes: List<String>
threadAlive: boolean
logger: Logger
DelayManager(List<Socket>,List<String>,List<String>)
run():void
isThreadAlive():boolean
setThreadAlive(boolean):void
<<Java Class>>
ServantController
org.faur.railsim.centralcontroller
logger: Logger
client: Socket
schedule: List<String>
writer: PrintWriter
ServantController(Socket,List<String>)
run():void

Server sincronizare

Această aplicaţie se găseşte în pachetul *org.faur.railsim.serversynchronizer* iar scopul ei este de a sincroniza clienţii conectaţi la acest server. Fără acest server care să coordoneze operaţiile, clienţii (vezi Simulator) ar lucra haotic, iar noţiunea de simulator nu s-ar mai potrivi întregului sistem.

În general paşii pe care-i face server-ul de sincronizare sunt destul de simpli:

1. porneşte cu un contor de la valoarea 1. Acest contor reprezintă o unitate de timp.

2. trimite clienţiilor valoare contorului.

3. opreşte firul un anumit număr de milisecunde specificat de catre utilizator la pornirea serverului.

4. incrementează contorul.

5. dacă contorul a trecut peste o anumită valoare prag specificată de către utilizator la pornirea serverului, atunci resetează contorul la valoarea 1.

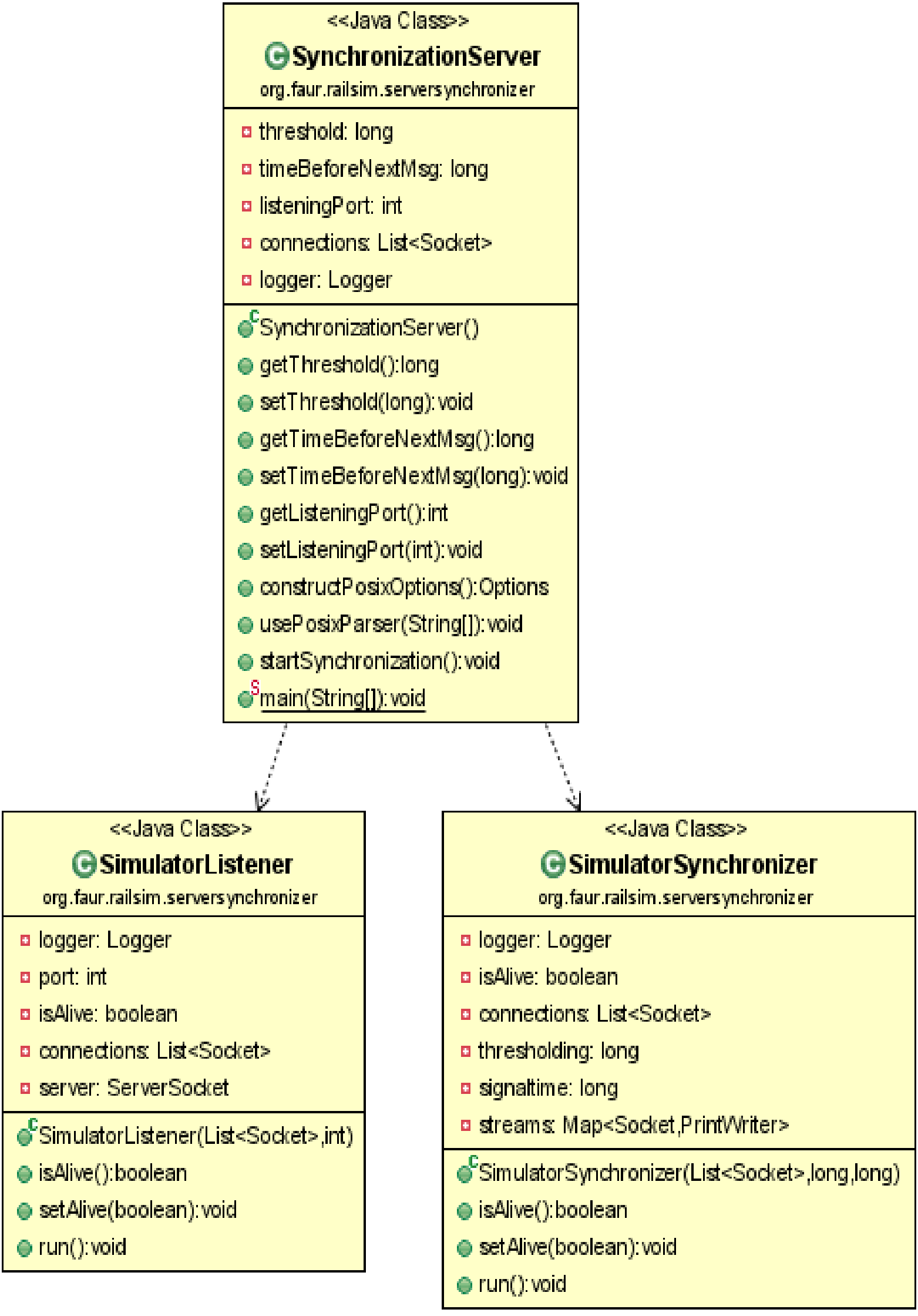
6. revino la pasul 2.

Comanda prin care se porneşte o instanţă a *serverului de sincronizare* este:

java -cp .:./target/classes:./target/classes/commons-cli-1.2.jar

org.faur.railsim.serversynchronizer.SynchronizationServer -port 10000 -threshold 10000 -signaltime 1500

Pentru a vedea lista de parametrii vă rugăm consultaţi documentaţia codului sursă. Important de reţinut este că, serverul de sincronizare este pornit înaintea simulatorului.


<<Java Class>>
SynchronizationServer
org.faur.railsim.serversynchronizer
threshold: long
timeBeforeNextMsg: long
listeningPort: int
connections: List<Socket>
logger: Logger
SynchronizationServer()
getThreshold():long
setThreshold(long):void
getTimeBeforeNextMsg():long
setTimeBeforeNextMsg(long):void
getListeningPort():int
setListeningPort(int):void
constructPosixOptions():Options
usePosixParser(String[]):void
startSynchronization():void
main(String[]):void
<<Java Class>>
SimulatorListener
org.faur.railsim.serversynchronizer
logger: Logger
port: int
isAlive: boolean
connections: List<Socket>
server: ServerSocket
SimulatorListener(List<Socket>,int)
isAlive():boolean
setAlive(boolean):void
run():void
<<Java Class>>
SimulatorSynchronizer
org.faur.railsim.serversynchronizer
logger: Logger
isAlive: boolean
connections: List<Socket>
thresholding: long
signaltime: long
streams: Map<Socket,PrintWriter>
SimulatorSynchronizer(List<Socket>,long,long)
isAlive():boolean
setAlive(boolean):void
run():void

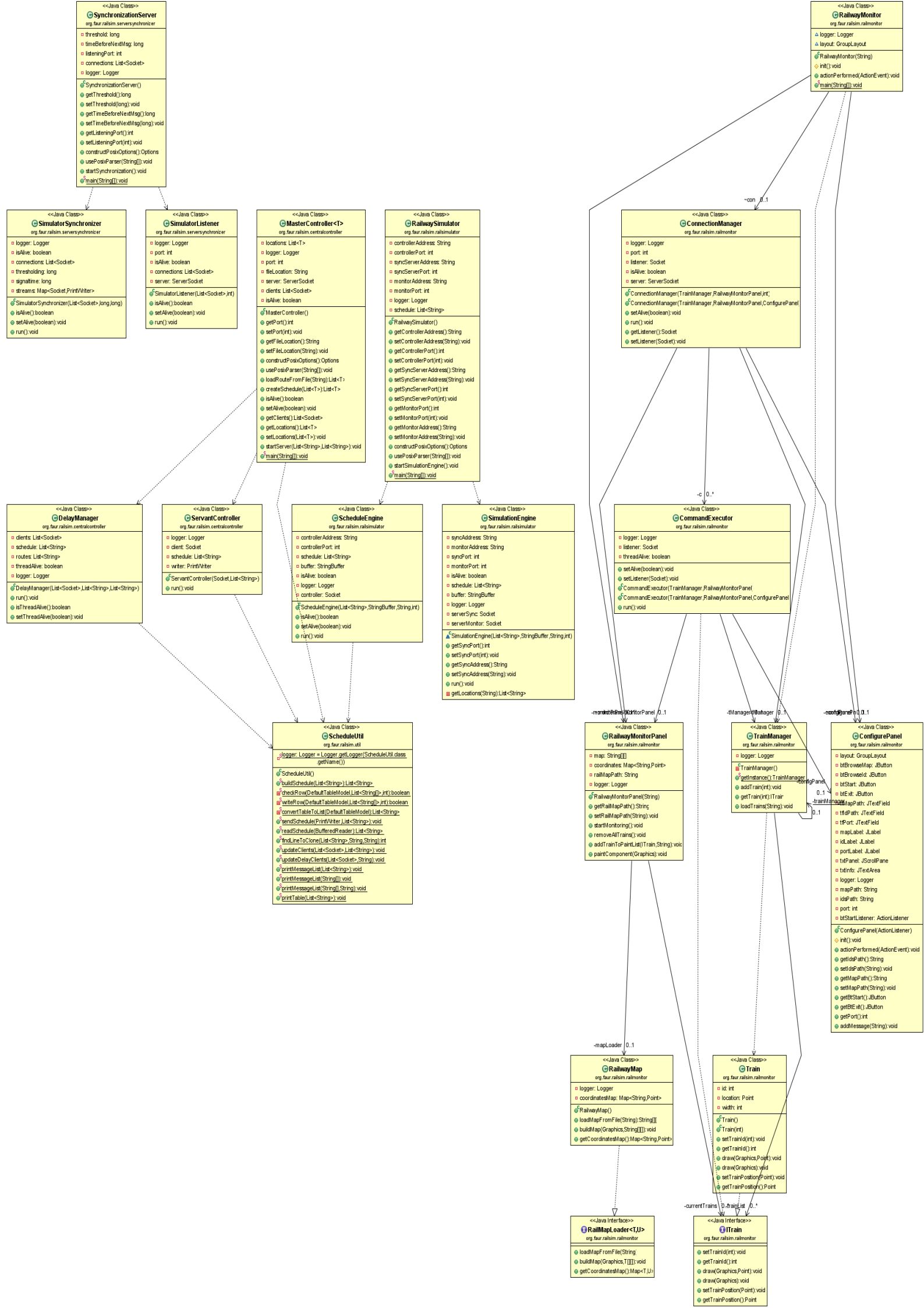
Concluzii

Railsimulator **prin arhitectura sa distribuită reuşeşte să îndeplinească destul de bine cerinţele enumerate la începutul acestui document, în secţiunea *Abstract* şi *Introducere.* Ba mai mult, prin sistemul modular folosit, Railsimulator poate fi foarte uşor modificat şi adaptat, unor noi cerinţe sau de ce nu, înlocuirea anumitor componente cu altele care fac acelaşi lucru doar într-un mod mai eficient. Bineînţeles respectând protocolul impus în acest sistem.**

Pentru o mai bună înţelegere a întregului sistem şi a algoritmilor folosiţi vă rugăm consultaţi codul sursă, împreuna cu documentaţia acestuia.

TO DO

- **În momentul când controlerul identifică o întârzâiere, un nou tabel este creat şi trimis simulatorului. După primirea lui, simulator-ul se ocupă imediat de execuţia lui din momentul specificat de serverul de sincronizare. Ar trebui ca simulator-ul să-şi termine execuţia vechiului tabel şi doar după aceea să pună în aplicare noul tabel.**
- **Când monitorul primeşte o înştiinţare prin care este anunţat că un tren este întârzâiat , acesta afişează un mesaj pe interfaţa grafică. Ar fi frumos ca trenul respectiv care este întârzâiat să fie desenat cu o alta culoare.**
- **Controlerul ar trebui să citească întârzâier-ile de la un senzor. Momentan se face direct de la tastatură. Acest senzor ar trebui să fie un server care să comunice prin TCP/IP cu controler-ul.**


SynchronizationServer
org.faur.railsim.serversynchronizer
threshold: long
timeBeforeNextMsg: long
listeningPort: int
connections: List<Socket>
logger: Logger
SynchronizationServer()
getThreshold():long
setThreshold(long):void
getTimeBeforeNextMsg():long
setTimeBeforeNextMsg(long):void
getListeningPort():int
setListeningPort(int):void
constructPosixOptions():Options
usePosixParser(String[]):void
startSynchronization():void
main(String[]):void
RailwayMonitor
org.faur.railsim.railmonitor
logger: Logger
layout: GroupLayout
RailwayMonitor(String)
init():void
actionPerformed(ActionEvent):void
main(String[]):void
SimulatorSynchronizer
org.faur.railsim.serversynchronizer
logger: Logger
isAlive: boolean
connections: List<Socket>
thresholding: long
signaltime: long
streams: Map<Socket,PrintWriter>
SimulatorSynchronizer(List<Socket>,long,long)
isAlive():boolean
setAlive(boolean):void
run():void
SimulatorListener
org.faur.railsim.serversynchronizer
logger: Logger
port: int
isAlive: boolean
connections: List<Socket>
server: ServerSocket
SimulatorListener(List<Socket>,int)
isAlive():boolean
setAlive(boolean):void
run():void
MasterController<T>
org.faur.railsim.centralcontroller
locations: List<T>
logger: Logger
port: int
fileLocation: String
server: ServerSocket
clients: List<Socket>
isAlive: boolean
MasterController()
getPort():int
setPort(int):void
getFileLocation():String
setFileLocation(String):void
constructPosixOptions():Options
usePosixParser(String[]):void
loadRouteFromFile(String):List<T>
createSchedule(List<T>):List<T>
isAlive():boolean
setAlive(boolean):void
getClients():List<Socket>
getLocations():List<T>
setLocations(List<T>):void
startServer(List<String>,List<String>):void
main(String[]):void
RailwaySimulator
org.faur.railsim.railsimulator
controllerAddress: String
controllerPort: int
syncServerAddress: String
syncServerPort: int
monitorAddress: String
monitorPort: int
logger: Logger
schedule: List<String>
RailwaySimulator()
getControllerAddress():String
setControllerAddress(String):void
getControllerPort():int
setControllerPort(int):void
getSyncServerAddress():String
setSyncServerAddress(String):void
getSyncServerPort():int
setSyncServerPort(int):void
getMonitorPort():int
setMonitorPort(int):void
getMonitorAddress():String
setMonitorAddress(String):void
constructPosixOptions():Options
usePosixParser(String[]):void
startSimulationEngine():void
main(String[]):void
~con
0..1
ConnectionManager
org.faur.railsim.railmonitor
logger: Logger
port: int
listener: Socket
isAlive: boolean
server: ServerSocket
ConnectionManager(TrainManager,RailwayMonitorPanel,int)
ConnectionManager(TrainManager,RailwayMonitorPanel,ConfigurePanel)
setAlive(boolean):void
run():void
getListener():Socket
setListener(Socket):void
DelayManager
org.faur.railsim.centralcontroller
clients: List<Socket>
schedule: List<String>
routes: List<String>
threadAlive: boolean
logger: Logger
DelayManager(List<Socket>,List<String>,List<String>)
run():void
isThreadAlive():boolean
setThreadAlive(boolean):void
ServantController
org.faur.railsim.centralcontroller
logger: Logger
client: Socket
schedule: List<String>
writer: PrintWriter
ServantController(Socket,List<String>)
run():void
ScheduleEngine
org.faur.railsim.railsimulator
controllerAddress: String
controllerPort: int
schedule: List<String>
buffer: StringBuffer
isAlive: boolean
logger: Logger
controller: Socket
ScheduleEngine(List<String>,StringBuffer,String,int)
isAlive():boolean
setAlive(boolean):void
run():void
SimulationEngine
org.faur.railsim.railsimulator
syncAddress: String
monitorAddress: String
syncPort: int
monitorPort: int
isAlive: boolean
schedule: List<String>
buffer: StringBuffer
logger: Logger
serverSync: Socket
serverMonitor: Socket
SimulationEngine(List<String>,StringBuffer,String,int)
getSyncPort():int
setSyncPort(int):void
getSyncAddress():String
setSyncAddress(String):void
run():void
getLocations(String):List<String>
-c
0..*
CommandExecutor
org.faur.railsim.railmonitor
logger: Logger
listener: Socket
threadAlive: boolean
setAlive(boolean):void
setListener(Socket):void
CommandExecutor(TrainManager,RailwayMonitorPanel)
CommandExecutor(TrainManager,RailwayMonitorPanel,ConfigurePanel)
run():void
ScheduleUtil
org.faur.railsim.util
logger: Logger = Logger.getLogger(ScheduleUtil.class.getName())
ScheduleUtil()
buildSchedule(List<String>):List<String>
checkRow(DefaultTableModel,List<String[]>,int):boolean
writeRow(DefaultTableModel,List<String[]>,int):boolean
convertTableToList(DefaultTableModel):List<String>
sendSchedule(PrintWriter,List<String>):void
readSchedule(BufferedReader):List<String>
findLineToClone(List<String>,String,String):int
updateClients(List<Socket>,List<String>):void
updateDelayClients(List<Socket>,String):void
printMessageList(List<String>):void
printMessageList(String[]):void
printMessageList(String[],String):void
printTable(List<String>):void
-monitorPanel
0..1
RailwayMonitorPanel
org.faur.railsim.railmonitor
map: String[][]
coordinates: Map<String,Point>
railMapPath: String
logger: Logger
RailwayMonitorPanel(String)
getRailMapPath():String
setRailMapPath(String):void
startMonitoring():void
removeAllTrains():void
addTrainToPaintList(ITrain,String):voic
paintComponent(Graphics):void
-tManager
0..1
TrainManager
org.faur.railsim.railmonitor
logger: Logger
TrainManager()
getInstance():TrainManager
addTrain(int):void
getTrain(int):ITrain
loadTrains(String):void
-configPanel
0..1
-trainManager
0..1
ConfigurePanel
org.faur.railsim.railmonitor
layout: GroupLayout
btBrowseMap: JButton
btBrowseId: JButton
btStart: JButton
btExit: JButton
mapPath: JTextField
tfIdPath: JTextField
tfPort: JTextField
mapLabel: JLabel
idLabel: JLabel
portLabel: JLabel
txtPanel: JScrollPane
txtInfo: JTextArea
logger: Logger
mapPath: String
idsPath: String
port: int
btStartListener: ActionListener
ConfigurePanel(ActionListener)
init():void
actionPerformed(ActionEvent):void
getIdsPath():String
setIdsPath(String):void
getMapPath():String
setMapPath(String):void
getBtStart():JButton
getBtExit():JButton
getPort():int
addMessage(String):void
-mapLoader
0..1
RailwayMap
org.faur.railsim.railmonitor
logger: Logger
coordinatesMap: Map<String,Point>
RailwayMap()
loadMapFromFile(String):String[][]
buildMap(Graphics,String[][]):void
getCoordinatesMap():Map<String,Point>
Train
org.faur.railsim.railmonitor
id: int
location: Point
width: int
Train()
Train(int)
setTrainId(int):void
getTrainId():int
draw(Graphics,Point):void
draw(Graphics):void
setTrainPostion(Point):void
getTrainPostion():Point
-currentTrains
0..*
-trainList
0..*
RailMapLoader<T,U>
org.faur.railsim.railmonitor
loadMapFromFile(String
buildMap(Graphics,T[][]):void
getCoordinatesMap():Map<T,U>
ITrain
org.faur.railsim.railmonitor
setTrainId(int):void
getTrainId():int
draw(Graphics,Point):void
draw(Graphics):void
setTrainPostion(Point):void
getTrainPostion():Point