POLITECNICO
MILANO 1863

Computer Science and Engineering

Model Checking of Battery-Powered Railway Lines

Formal Method for Concurrent and Real Time Systems Project

Academic year 2021 - 2022

24 June 2022

Authors:
Lorenzo IOVINE - 10633107
Nicola LANDINI - 10666325
Francesco LEONE - 10659809

Istruttors:
Prof. Pierluigi SAN PIETRO
Dr. Livia LESTINGI

Deliverable:	Formal Method for Concurrent and Real Time Systems project
Title:	Model Checking of Battery-Powered Railway Lines
Authors:	Lorenzo Iovine, Nicola Landini, Francesco Leone
Date:	24-June-2022
Copyright:	Copyright © 2022, Lorenzo Iovine, Nicola Landini, Francesco Leone – All rights reserved

Contents

Table of Contents **3**

1 Design **4**
1.1 Purpose 4
1.2 High Level Model Description 4
1.3 Initializations 4
1.4 Design assumptions 5
1.5 Design Choices 5

2 UPPAAL Model **6**
2.1 Global Declarations 6
2.1.1 System Parameters 6
2.1.2 System Variables 6
2.1.3 System Channels 6
2.2 Templates 7
2.2.1 Station 7
2.2.2 Train 8

3 Analysis and Results **10**
3.1 Property Verification 10
3.1.1 Mandatory Properties 10
3.1.2 Additional Properties 10

4 Conclusion **11**

5 References **11**

1 Design

1.1 Purpose

The fight to reduce greenhouse gas emissions is bringing together researchers and manufacturers from all over the world. In particular, rechargeable batteries as a source of power in place of fossil fuels are already widespread in cars and making their way into the rail transport sector. Battery-powered trains are already operative in several countries like Japan, Austria, and Britain. Italy is also planning on producing and deploying fully-electric trains starting mid-2022, thanks to a deal with Hitachi Rail.

Like any electric vehicle, trains can cover a limited distance running only on battery power before needing to recharge. In this project, we will model a railway line in which electric trains can recharge in a station. Nevertheless, trains must still reach the following station on time; in case of excessive delay, the company is obliged to issue monetary compensation to the passengers.

Precisely, given a set of simplifying assumptions, we will model the main actors of the system as a network of **Timed Automata (TA)** whose behavior depends on specific key parameters.

1.2 High Level Model Description

We created two different configurations for the railway model. Both of them include 4 trains and 3 stations.
The first one represents the main configuration of the system and verifies all the properties. The railway model is set as follows:

The second configuration doesn't verify the mandatory properties of the project and it is set as follows:

1.3 Initializations

The stations have the following initial configurations:

- **Station 0:** 2 tracks, 1 available
- **Station 1:** 3 tracks, 2 available
- **Station 2:** 2 tracks, 0 available

The trains that we designed have constant speed set to 120 km/h. They are initialized as follows:

- **Train 0 - charge 100:** starts from station 0 with station 2 as destination
- **Train 1 - charge 100:** starts from station 1 with station 2 as destination
- **Train 2 - charge 100:** starts from station 2 with station 0 as destination
- **Train 3 - charge 100:** starts from station 2 with station 0 as destination

1.4 Design assumptions

In order to efficiently describe the model, we decided to make the following assumptions:

- Every station has less tracks than the total number of trains.
- A clock unit is equal to a minute.
- For each train the destination is the last station, except for the one that start from the last one who has as destination the first station.
- The lower bound to allow passengers to get on and off the train is of 4 clock unit.
- We set a charging multiplier and two different discharging multiplier, one for the waiting and one for the travel.

1.5 Design Choices

- We decided not to design the railway with a dedicated template. That's because the railway's most important features are implicitly designed and verified, without creating additional variables.
- In order to save time when checking properties we avoid redundant clocks for operations that are not issued in parallel.
- Our *Recharge Policy* is based on a control made in function *chargingTime* in the *Train Template*, that allows to recharge the train at least for the lower bound (described before). In case the train needs more time to recharge in order to get to the next station, the time spent in the station is the mean value between the lower bound and the upper bound (calculated as follows: $MaximumDelay - \frac{DistanceToNextStation}{trainSpeed}$) in order not to overcome the maximum delay. We thought that this is a good compromise between charging the train and have some delay in reaching the next station.
- In order to model the station, the distances and the maximum delays between stations we used two matrices. It is enough to change the number of trains/stations, initialize them and update the matrices, and the system will "adapt" to the new configuration.
- All the variables that could be declared as constant, are declared as constant, in order to save time when checking properties.

2 UPPAAL Model

2.1 Global Declarations

2.1.1 System Parameters

Parameter	Description
nStations	Number of stations in the system
nTrains	Number of trains in the system
chargingSpeed	A multiplier for the battery charging speed
journeyDischargeMultiplier	A multiplier for the battery discharge during the journey
waitingDischargeMultiplier	A multiplier for the battery discharge during the waiting
MIN_TIME_PER_STATION	Lower bound to allow passengers to get on and off the train
trainSpeed	Trains' speed
stationNumOfTracks	Array containing the number of tracks of each station
railLine_distance	Matrix containing the distance between stations
railLine_delay	Matrix containing the maximum delay between stations

2.1.2 System Variables

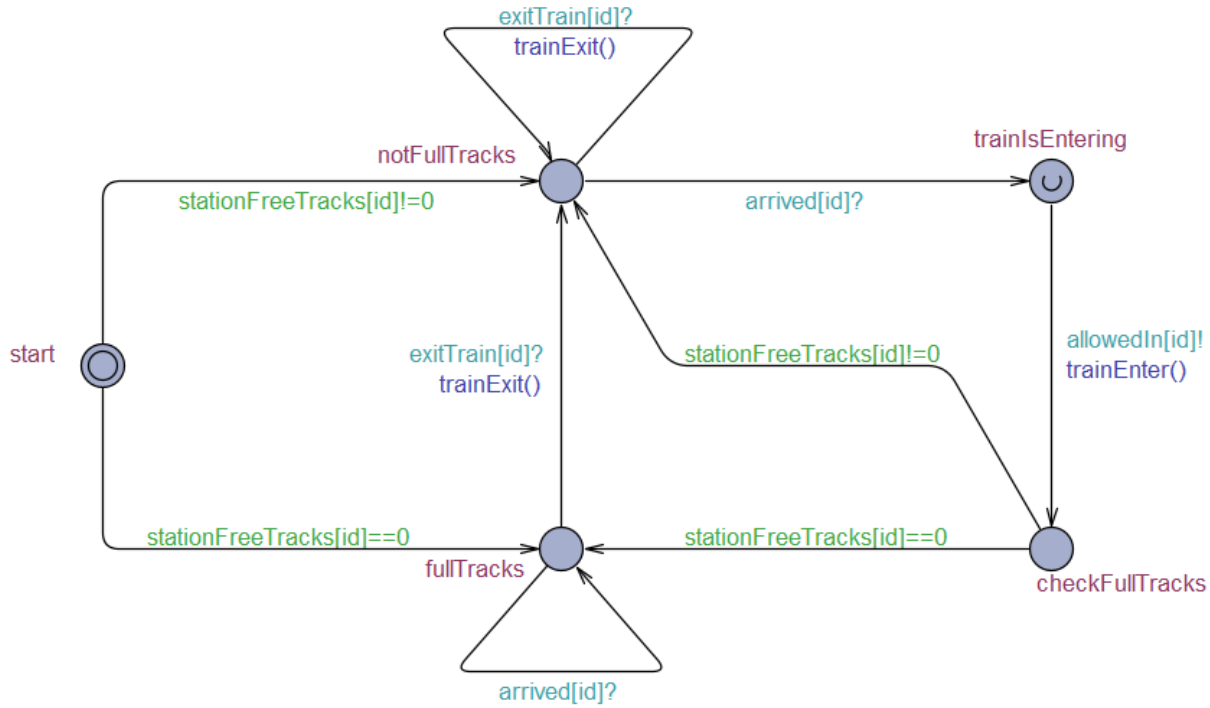
Variable	Description
trainStation	Array with the current station of each train
trainDestination	Array with the destination of each train
nextStop	Array with the next station of each train
stationFreeTracks	Array with the current number of free tracks of each station
chargeOfTrain	Array with the current charge of each train battery

2.1.3 System Channels

Channel	Description
exitTrain	Channels through which a train inform its current station that it is leaving
full	Channels through which a station communicate to an incoming train that it has no available tracks
allowedIn	Channels through which a station grant the access to a train that is waiting
arrived	Channels through which a train inform its next station that it is arrived

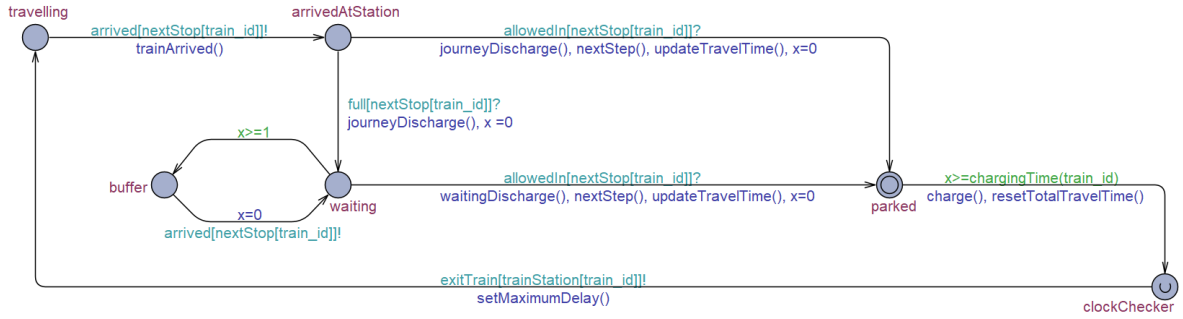
2.2 Templates

2.2.1 Station

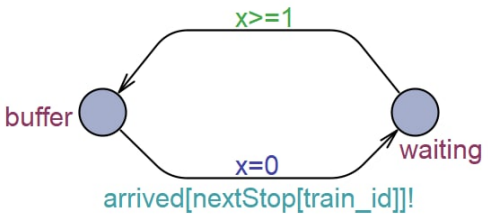


- **start:** this is the initial state of the Station Template. In case the number of available tracks is equal to zero the next state is notFullTracks, otherwise the next state is *fullTracks*.
- **notFullTracks:** when this state receives a message on the *exitTrain* channel, it updates his *stationFreeTracks* variable and the current state doesn't change. Upon receiving a message on the *arrived* channel the current state moves to *checkFreeTracks* state.
- **fullTracks:** when this state receives a message on the *arrived* channel the current state doesn't change. Instead, if this state receives a message on the *exitTrain* channel, it updates his *stationFreeTracks* variable and the current state moves to *notFullTracks* state.
- **trainIsEntering:** this is an urgent state created to describe when a train is joining the station.
- **checkFullTracks:** this state is reached immediately after the update of the *stationFreeTracks* variable and checks if the station has available tracks. In this case the current state becomes *notFullTracks*, otherwise it becomes *fullTracks*.

2.2.2 Train



- **parked:** this is the initial state of the Train Template. It represents the situation in which the train is stopped in a station. The current state changes only if the clock is greater or equal to the needed charging time. Before the current state changes in *clockChecker* we compute the *charge* method that increases the train battery linearly with the time spent in the station.
- **clockChecker:** this is an urgent state created to send a message on *exitTrain* channel and to set the maximum delay to reach the next station.
- **travelling:** this state represents the situation in which the train left the previous station and heads to the next one. Before reaching the next station, we send a message on *arrived* channel to inform the station of the arrival and update train's current station.
- **arrivedAtStation:** this state checks if the train has to wait before entering or if it is allowed to join the station. In the first case the current state becomes *waiting* after receiving a message on *full* channel, otherwise the system comes back to the initial state after receiving a message on *allowedIn* channel. Before reaching one of the next state we compute the *journeyDischarge* method that decreases the train battery linearly with the travel time.
- **waiting:** this is the state in which the train waits until the station, throw *allowedIn* channel, allows him to join. Before entering the station we compute the *waitingDischarge* method that decreases the train battery linearly with the waiting time.
- **waiting-buffer:** this is a snippet of Train Template that describes the loop in which the train is while waiting to be allowed to join the next station. In this loop we send a message through *arrived* channel to the station every clock time unit.



Train local variables	**Description**
x	This is the clock used to temporalize the system
totalTravelTime	An int that represents the overall time to reach the next station
parkedTime	An int that represents the time spent in a station
travelTime	An int that represents the time spent travelling
waitingTime	An int that represents the time wasted until the station authorizes the access
actualMaximumDelay	An int used to save the maximum delay allowed to reach the next station

3 Analysis and Results

3.1 Property Verification

3.1.1 Mandatory Properties

In this section we will describe the analyzed properties. The first two properties analyzed are the compulsory ones:

- $\forall\Box(chargeOfTrain[Train\_id] > 0)$
- $\forall\Box(train.totalTravelTime <= train.actualMaximumDelay)$

We have created two configuration of the system: in the first one the two properties are satisfied, in the second one not. The difference between the two configuration is the max delay allowed.

Configuration1 In this configuration the properties are satisfied

- Max Delay Station 1-2 : 50
- Max Delay Station 2-3 : 60

Configuration2 In this configuration the properties are not satisfied

- Max Delay Station 1-2 : 25
- Max Delay Station 2-3 : 30

The two properties could be satisfied even in this configuration if we increase the *chargingMultiplier*

3.1.2 Additional Properties

We have written two additional properties, the first one to check that the number of free Tracks in a station is always greater or equal to 0, the second one to check that every train eventually reaches its destination.

- $\forall\Box(stationFreeTracks[station\_id] >= 0)$
- $\forall\Diamond(trainStation[Train\_id] = trainDestination[Train\_id])$

Both properties are satisfied in both configuration of the system.

4 Conclusion

In this report we presented an analysis of the problem of the Model Checking of Battery Powered Railway Lines. The aim was to check a realistic configuration in order to understand the validity of the model.
From the results showed, we succeeded in finding a configuration where all the properties to be checked are satisfied. We also tried to increase verification performances by reducing the state space, in order to decrease processing time. The main characteristic that emerged are the recharge policies. The real bottleneck of the system is the fact that there are stations with a number of tracks lower than trains in the system. It is important to find a consistent recharge policy in order to efficiently describe the behavior of the system without letting the parameters to be inconsistent.
This project allowed us to improve our team working and knowledge about the subjects of the course.

5 References

- *A Tutorial on UPPAAL 4.0* by Gerd Behrmann, Alexandre David, and Kim G. Larsen
- *UPPAAL SMC Tutorial* by Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen
- `https://docs.uppaal.org/` UPPAAL tool documentation

