Formal Methods For Concurrent and Real Time Systems
Homework report

Giuseppe Calderonio, Marco Colombo, Paolo Corsa

Instructors
Prof. Pierluigi San Pietro
Dr. Livia Lestingi

A.Y 2022-2023

POLITECNICO
MILANO 1863

Abstract

In this report is presented an UPPAAL model for railway networks based on rechargeable batteries installed on trains as source of energy, with a special focus on the design choices, extensibility, property verification and limitations.
Special emphasis is placed on the train and how they can flow in the line without losing charge, how they approach a station and how they are allowed to get in, how they can move from a station to another without exceeding the specific line they belong to, and how a station can handle the flow of multiple trains, having a limited number of tracks.

1 Problem Description

Before analyzing the model and its structure, it's necessary to give a brief but detailed description about the problem, with a special focus on the characteristics that are object of design choices.

1.1 High Level Description

The problem comes from the real world scenario of fighting greenhouse gas emissions with electronic vehicles, specifically trains for the scope of this project.
The most common use case could be a public transport company that provides transportation services; this can be considered as a safety-critical environment because of the presence of passengers in the trains:

- The trains should not crash
- The trains should not run out of battery
- Passengers should be able to get in and out of the train safely

Moreover, other guarantees about the service has to be done in order to be competitive in the business of public transportation, such as respecting deadlines.

In order to model some of this properties, a model checking approach with UPPAAL is proposed.

2 High Level Model Description

The main entities of our model are:

- **Line**: modeled not as a template, differently from all the others, but as a set of globally declared data structures that can be accessed by other entities in order to allow correct execution and simulation.
  Data structure modeling, access, and extensibility are the main design choices of this entity.
- **Station**: modeled as a template, that works by only handling the available tracks and the synchronization with trains.
  Synchronization and tracks management are the main design choices of this entity.
- **Train**: the main entity, modeled as a template, that should logically traverse the lines, and properly synchronize with stations in order to do it.
  Synchronization and charge management are the main design choices that characterize this entity.

2.1 Measurement Units

The decision of writing an entire section about measurement units comes from the fact that this design choice impacts the level of precision of the whole system and consequently the made assumptions.
In particular, the project is focused more on modeling the recharge of the trains and their flow on the system instead of simulating the interaction between trains and customers; this brought us to use minutes as the smallest time unit.
For this reason, we approximate each interaction that in practice would require less than half a minute as instantaneous: as a consequence the messages between stations and trains are not fully representative of a real world scenario, even if we assume the interaction happens in the order of milliseconds

```
/*
    unit of measure used in the system

    distance in m
    time in minutes
    speed in km/h
*/
```

Figure 1: Measurement units used

(e.g. a network message in a LAN) and thus can be approximated to zero.
However, it is guaranteed by the model that this approximation is not dangerous for the safety of the trains and passengers.
Differently, in order to model the speed of the train, km/h has been chosen; the reason for this choice is simple and intuitive: it fits better with a real world scenario, since in practice speed in Europe is measured in km/h.
Finally, the distance is measured in meters, for precision reasons.
In order to explain this better, it's mandatory to understand how computations and conversions are implemented.

```
/*
    This method returns the time in minutes required to travel
    from the current station and the next one (according to the direction)
*/
int computeTravelTime(){

    return minutes(meters(computeDistance(position))/V)/1000 + 1;

}
```

Figure 2: Compute travel time function (Train template)

As shown in Figure 2, to compute the time required to travel from the current station to the next one, we have to compute $\frac{D_i}{V}$, where D_i is the distance, and V is the speed (assumed constant).

However, this value has to be integer, because it is used as operand for a comparison with a clock, and thus cannot be floating point; for this reason UPPAAL, as any other programming language, does $t = \lfloor \frac{D_i}{V} \rfloor$ to properly cast it.
Here there is the main problem: **approximating 1.9 meters to 1 meter is much more precise than approximating 1.9 Km to 1 Km.**
In this way, we significantly reduce the information lost due to the approximation and we consider that negligible. As a last remark, a conservative approach was used in the computations, keeping always an upper bound of the result approximated (always takes $\lfloor \frac{D_i}{V} \rfloor + 1$) .

The biggest drawback of this design choice is that UPPAAL integers are only 16 bits wide [-32768, 32767], so the maximum distance supported is 32 Km.

To summarise, the measurement units are a trade off between 2 factors: the decision to focus on the travelling aspects and the precision that always has to reflect a real world scenario.

3 Components Description and Design Choices

3.1 Line

3.1.1 Component Description

A line is a logical list of stations that can be traversed in both directions. To model this behavior, in our system a line is fully represented by global variables and data structures:

- Nlines : number of train lines
- Nstations : total number of stations in the whole train network
- MaxNStation : maximum number of stations that a line can have
- railwayNetwork[NLines][MaxNStation] : This matrix represents the entire train network where:
  - each row represents a train line (a set of consecutive stations)
  - each column index represents the relative position of the stations in the line
  - each element of the matrix represents a unique identifier associated with a station

  In order to share a station among multiple lines it is enough to put the same station identifier in different train lines (rows of the matrix).

  If the line has less stations than the maximum number of stations that a line can have, the remaining identifiers are negative (-1) and do not match with any real station.
- distance[NLines][MaxNStations-1] : This matrix represents the distances between the (consecutive) stations of the train network.
  Given a row (which represents a train line) at column index i (station at position i), the element of the matrix represents the distance between station i and station i+1 (that explains why there is MaxNStations - 1).

  If the line has less stations than the maximum number of stations that a line can have, the remaining values are negative (-1) and do not match with any real distance.
- length[NLines] : The elements of this vector at index i (which represents train line i) contains the number of stations of the line.
  It is mostly used by the train belonging to the corresponding line not to go out of bounds of the line

3.1.2 Design Choices

The main reason for which this approach has been used to model this entity is scalabilty. In fact, in this way we centralize the complexity of the model as a problem of parameter selection and coherency and, once the parameters has been selected, to validate the model (to understand whether it fulfils the requirements), it's enough to let the verifier check the properties we are interested in.
Of course, each design choice has pros and cons, and in this case the main ones are two:

- The code of the train template is not so trivial, since it has to handle a big set of data structures.
- Some elements of the matrices could be mostly empty; in particular, if the average number of stations for line is much smaller than the number of stations of the longest line (MaxNstations), this approach could be inefficient.

3.2 Station

A station in our model is represented as a template that interacts and is synchronized with the trains that request to enter and leave.

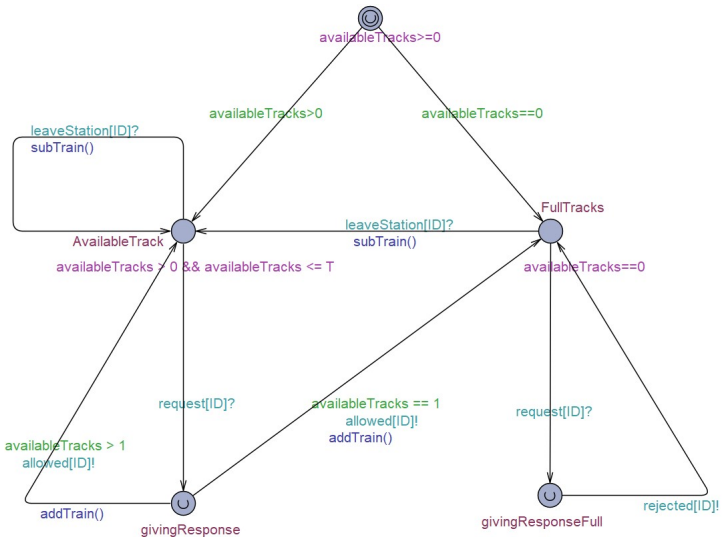

Figure 3: Station template

3.2.1 Component Description

In figure 3 is reported the Timed Automata of the station.

The constructor parameters are:

- int ID : unique identifier associated with each station (those identifiers are the elements of the matrix railwayNetwork).
- const int T : number of track in the station, or equivalently the maximum number of trains that the station can host.
- const int initTrains : number of initial trains in the station; we assume that at system startup time, we can decide for each station how many trains start from there.

The behavior of the Timed Automata is not very complicated :

- First of all, based on how many trains the station hosts at startup time, it decides instantly (0 time) if switching to "AvailableTracks" state or "FullTracks" state.
- When the train is in "AvailableTracks" (state constraint is that has at least one free track):
  - if a train leaves the station, the station increases the number of available track, and does not leave this state.
  - if a train asks the permission to enter, the station always allows the train to join, updates the number of free tracks (addTrain()), and if, after the update, the station is full of trains, it switches to "FullTrack" state.
- When the train is in "FullTrack" (state constraint is that there are no free tracks):
  - if a train asks the permission to enter, the request is always denied.
  - if a train leaves the station, the station decreases the number of available tracks, and switches to "AvailableTracks" state.

3.2.2 Design Choices

The real complexity of this entity resides on the channel management; in fact, **each channel is associated with one station**. Each channel is urgent, since as already mentioned the messages latency is considered negligible for the orders of magnitude of this project, and each channel is represented as an element of a vector of channels, indexed by ID. This implies though that each station has to

be always aware of the ID of the next station it is going towards (actually, this assumption is easily satisfied in a real world environment): in the code this is possible thanks to the RailWayLine matrix, that contains in every element the ID of the next station. Another design choice made is to model the entity as "stateless" as possible; the station indeed only updates its own number of available tracks, that are modeled as simple integers (and not as complex entities that, for instance, could take care to provide energy to the trains), is not aware about the trains that request for entering and just replies to them based on the number of available tracks, and **each station can be considered as completely independent from other stations**.

3.3 Train

A train in our model is represented as a template that interacts and is synchronized with the stations of the line it belongs to.

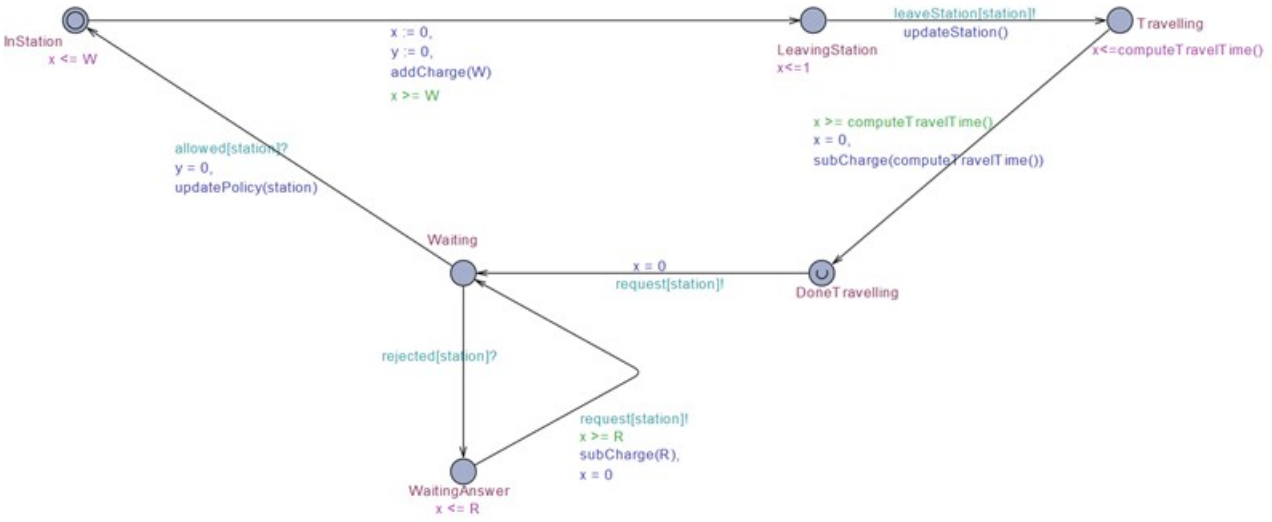

Figure 4: Train automata.

3.3.1 Component Description

The constructor parameters are:

- const int Cmax : Maximum Charge of the train
- const int Cdis : Negative angular coefficient of the linear dependency of the DISCHARGE [meters/minutes]
- const int Crec : Positive angular coefficient of the linear dependency of the CHARGE [meters/minutes]
- const int V: train speed in km/h
- const int initialStation : initial relative position of the train in the line it belongs to
- const bool initialDirection: true = left-right, false = right-left (w.r.t. matrix rows)
- const int Wmin : Minimum stop time in the station
- const int line : index of the line which the train belongs to

The Timed Automata has the following locations:

- InStation: this is the initial location of the automata. When a train is in this location, it means that it is into a station, waiting for the passengers to get in/off and also recharging. After an amount of time which is decided by the recharge policy (that cannot be lower than Wmin), the train will go to the next location of the automata.
- LeavingStation: this location represents the fact that the train is leaving the station, but it is still in the track of the station itself. When the train actually leaves the station, it tells the station that the track is now free.
- Travelling: this location represents the fact that the train is travelling from a station to the next one (according to the direction and the line). While travelling the charge of the train decreases. This location is left after the amount of time necessary to arrive to the next station.
- DoneTravelling: the train has reached the next station and it asks the station the permission to enter.
- Waiting: when the train is in this location, it is waiting for the response of the station. Two things may happen:
  - if the station has an available track, the request is accepted and the train can enter.
  - if the station is full, the request is rejected, and the automata goes to location "WaitingAnswer".
- WaitingAnswer: when the train is in this location, it waits R minutes (the time to re-submit the request) and sends a new request to the station, coming back to the location "Waiting".

3.3.2 Design Choices

Two main design choices are important to be specified:

- Instead of updating the charge every time unit, it is added or subtracted only once at the end of the entire time interval. This happens in three cases: when the train is charged, when the train consumes charge while travelling and when the train is waiting to enter a station. Although this implies a significant decrease of the model complexity and thus of the verification process time, this behaviour relies on the assumption that the speed of the train is constant and the time to wait before resubmitting a new request is fixed, representing a possible drawback of our system that, indeed, does not model exactly a real world scenario; we have decided anyway to include this choice in our model.
- As mentioned before, the messages that trains send to and receive from stations (modeled with urgent channels) are assumed to be instantaneous, since their latency is of several orders of magnitude smaller than the time unit used in this model (already discussed).

4 Properties

In this section are described the main properties that the model is expected to verify.

- P1 :(Mandatory) It never happens that a train reaches the destination after the deadline.
- P2 : (Mandatory) It never happens that a train runs out of power.
- P3 : The System never deadlocks.

  Note: since train can travel independently from the charge, this property is verified even when trains reach 0 charge (above property checks for this condition).
- P4 : For all states eventually happens that at least one train will be in location WaitingAnswer

  This property does not necessarily imply a badly designed model. For example, if the system has no bottleneck stations (stations with less tracks than the total number of trains in the lines it belongs to)

- P5 : For all states eventually happens that a specific station becomes full

  Sometimes we design a model expecting to introduce complexity in specific stations.

- P6 : Trains never have more charge than the maximum allowed.

- P7 : The trains have enough charge to cover the distance between the current station and the next one.

  This property can be useful to give a lower bound to the recharge policy (if not verified, it does not recharge enough the trains), or to understand if C_{max} is too small.

- P8 : This property verifies that each train sends a new request to enter into a station no later than R minutes.

  Assuming a blackbox approach, this property is worthy to be verified because it represents a constraint of the system. However, by looking at the timed automata of the train, it is easy to see that it always holds since there is a guard on the location WaitingAnswer.

- P9 : The number of available tracks of each station is within the right bounds (greater or equal than zero and less or equal than the number of tracks instantiated in the station).

5 Configurations and results

We have built some configurations to verify all the properties, others to deny at most three of them.

5.1 Configuration_line

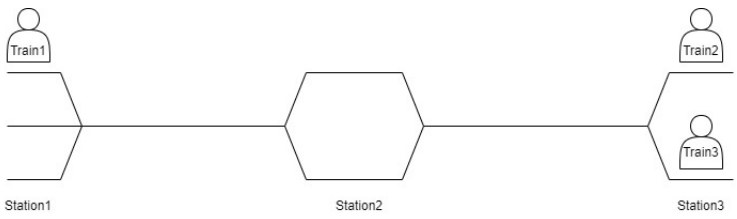

Figure 5: Configuration with a Line, 3 stations and 3 trains

Configuration_line is represented by three stations *Station1, Station2, Station3* on a single line: *Station1* is the terminus station on the left, *Station2* is the station in the middle and *Station3* is the terminus on the right.

In order to give a more realistic representation to the parameters selected, we assume that some trains are older than others , and for this reason, their batteries are degrading therefore the angular coefficient of the discharging function varies by a few digits, also their initial charge and their speed differs.

```
Cdis1 = 1284;
Cdis2 = 1296;
Cdis3 = 1300; (this is Train3, the oldest.)
Cmax1 = 25;
Cmax2 = 23;
Cmax3 = 27; (this is Train3, the oldest.)
V1 = 90;
V2 = 110;
V3 = 90;
```

In this specific configuration **all** the properties are satisfied.

5.2 Configuration_line_bad1

This configuration has the same construction of the stations as the one mentioned above but differs for the parameters of the trains. The assumption taken in this configuration is that *Train1* is one train that is very old w.r.t others in fact it has a poor *Cmax* (23), a moderate *V* (70 Km/h) and a very high value of *Cdis* (1730).

In this configuration **P2** is not satisfied while all the others are, because:

- *Cmax1* is too small to support the longest path in the line (also considering the *meanWaiting-Time*); indeed **P2** is not satisfied **but P7** is.
- *V1* is lower compared to others trains.
- *Cdis1* is too big.

In order to have higher granularity we have checked that only the first train runs out of power.

5.3 Configuration_line_bad2

It has the same construction of **Configuration_line** but it differs only for the recharge policy. The new recharge policy takes the maximum value among:

- *Wmin*
- the time required to recharge the train with enough power to cover the maximum distance between two stations of the line

This policy is not well designed because, even if it considers the maximum distance, it does not take into account the average time that a train waits outside a station before entering.

As a consequence **P2** and **P7** are not satisfied: in particular **P7** is not verified because the train waits (at most) one minute to leave the station, so it is a bit less than the actual distance.

5.4 Configuration_cross

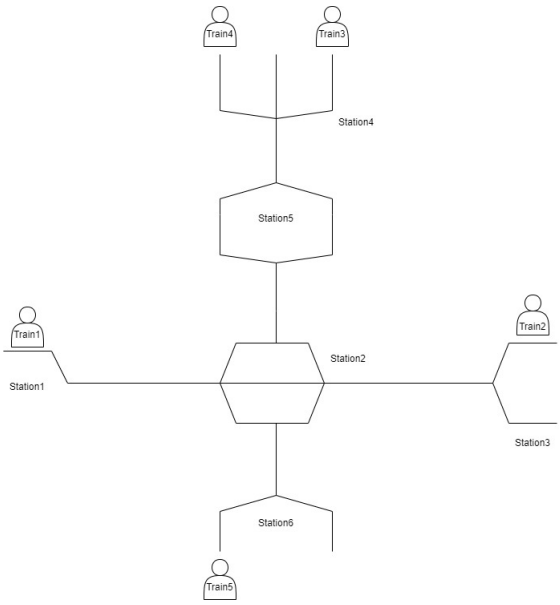

Figure 6: Configuration with two Line, six stations with one shared among the lines, and three trains per line

The topology of this configuration consists in two lines that intersects in one station, *Station2*, that can be a potential bottleneck.

On the two lines are distributed five trains with this initialization:

```
Train1 = Train(32, 1530, 6250, 90, 0, true, 2, 0);
Train2 = Train(32, 1530, 6250, 90, 2, false, 2, 0);
Train3 = Train(30, 1530, 6250, 90, 0, true, 2, 1);
Train4 = Train(30, 1530, 6250, 90, 0, true, 2, 1);
Train5 = Train(30, 1530, 6250, 90, 3, false, 2, 1);
```

We have designed this configuration and parameters such that **all** the properties are verified.

5.5 Configuration_cross_bad

The topology and the construction of the templates are exactly the same as **Configuration_cross**, but we made a variable change in fact

```
int meanWaitingTime = 6; (vs. meanWaitingTime = 2;)
```

so the train takes more time to recharge.

As a consequence trains wait more time before entering a station. The value of *meanWaitingTime* in this specific configuration is too big and even if the trains have enough charge, they don't reach the destination in time, thus violating the first mandatory property **BUT** not the second one.

6 Conclusion

We managed to build our model in order to be as scalable as possible. Indeed it can feature an arbitrary number of lines and trains, and at the same time keeping the single component as simple as possible while fulfilling all the requirements.

In fact, under some assumptions, it should not be too hard to model a real railway network but for the parameters selection.

We focused also on designing "bad models" to highlight how the parameters impact on the verification of each property: finding the right trade off between them was both the real challenge and the target of their design.