Train Control System

Alloy Design Report

Pedro Barbeira
Ricardo Ribeiro
Tiago Marques

U. PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Formal Methods for Critical Systems

2023/2024 - Second semester

Contents

1 Introduction

This project, in the context of Formal Methods for Critical Systems of the Masters in Informatics Engineering at FEUP, aims to model a simplified version of the Level 3 of the European Train Control System (ECTS) protocol using Alloy Analyzer. The system aims to manage train movements based on virtual sub-sections (VSS) and onboard train information.

2 Problem Description

Trains possess onboard devices which keep information regarding structural integrity and position. This information is regularly sent to the Movement Authority, leading to updates in the Virtual Sub-Sections of the train track.

A VSS can be in one of three states: Free, Occupied or Unknown.

A VSS is free if the Movement Authority (MA) knows for sure that it is free, ergo, it knows the position of the train, and the train is not occupying the VSS. Alternatively, a VSS is occupied if the Movement Authority knows a train is occupying it. Lastly, a VSS is Unknown if the Movement Authority cannot safely assert if it is Free or Occupied. This may happen due to malfunction in the train communication systems or loss of train integrity, among other reasons.

For our project, we decided to focus on modelling both of these cases. We consider only non-cyclical tracks, without intersections.

3 Structural Design

To further explore our project, the first step is to describe the structural design, ergo, the static modelling of the protocol. A high-level view of the model can be seen in Figure 1.

3.1 System Modelling

There are 2 main entities: Track and Train.

Tracks are composed of another entity: VSS. Each track has a reference to composing VSSs as well as to the first and last VSS in the *begin* and *end* relations. VSS are sub-classed by Begin and End (the first and last VSS of the track, respectively).

Trains follow the same pattern as a Track: they have cars (wagons), and the first and last Car are kept as relations, to facilitate access (head and tail, respectively). Additionally, we created two fields, unknowns, a set os VSS, and brokenCar, a lone Car, to keep some data which facilitated our modelling. This was done in order to circumvent certain problems that appeared at the very end of the development process.

Cars have a *succ* relation, allowing for recursively defining trains, and Head and Tail sub-classes, which play the same part as Begin and End.

3.2 Properties Specification and Verification

Several constraints were added to enforce the correctness of the model.

The fact *Multiplicities* describes how connections between the different signatures of the system work. We used a many-to-one relationship between VSS and Tracks, ensuring each VSS is mapped to only one track. There is a one-to-one relationship between each Track and both the Begin and End VSSs.

We opted to consider only linear, acyclic tracks, with no junctions or intersections. This was made to facilitate the implementation of our model, as well as focus on the core cases - junctions and intersections could easily justify having their own model, due to the increased complexity and the necessity of having some form of traffic control and scheduling. The fact *linearTrack* ensures that each track is linear and non-circular.

The same logic was applied to the Train signature: *Multiplicities* maps Cars to Trains in a many-to-one fashion, as well as both Head and Tail in a one-to-one fashion; *linearTrain* ensures the integrity of the train.

Another important aspect assumed to be true for any Train is the fact *noTrainSelfCollision* that assures that there is never a Car with a VSS assigned to it that is the VSS of the car in front of it or any of this VSS's successors.

3.3 Model Validation

To validate all properties of the structural design, we created the following run commands:

- **Run Commands**

  (a) *trackLinearity*: run command to validate Track linearity;

  (b) *trainLinearity*: run command to validate Train linearity;

  (c) *trainPositions*: run command to validate the position of 4 Trains in one track.

- **Check Commands**

  (a) *noTrackCycle*: check command to verify if it's possible to go from any VSS different from the begin VSS back to the start of the track;

  (b) *noEndSucc*: check command to verify if there are any successor VSSs of the end of the track;

  (c) *trainInOnlyOneTrack*: check command to verify if each train only occupies one track

4 Behavioural Design

4.1 Configuration

The initial state of the model can be configured in the *Init* fact, which ensures our system initialises with the desired properties. The configuration used for the development of the work considered that all Trains would start Fully Operational and connected. We also ensured that there was no more than one can in a given VSS, to facilitate modelling and development.

Moreover, we state that all trains have at least one Car between the Head and the Tail.

Lastly, we define the initial states of the VSSs, ensuring that any VSS with a Car is in the state Occupied, that the remainder of the VSSs are in the state Free, and that no VSSs are in the state Unknown.

4.2 Behavioural Modelling

A VSS can be in one of three states: Free, Occupied, and Unknown. These are defined as subsets of VSS rather than extensions.

In the same way, Trains can be in one of three states:

- Incomplete Train: Train that lost a Car and, therefore suffers loss of integrity;

- Offline Train: Train that lost its communications with the Movement Authority;

- Fully Functional: a Train that has full integrity and is connected. Assumed as the default state, for ease of implementation.

Cars have a *position* relation, which relates a given Car to the VSS it occupies.

4.3 Event Modelling

In terms of modelling the actual behaviour of the system, we considered the following possibilities of events:

(a) A train moves (*move[t: Train]* predicate)

Given a Train t:

- t only moves if it has not reached the end of the track and the successor of the head's VSS is Free;
- All cars of t move one position forward;
- Changes to the states of the VSS:
  - If t is Fully Functional:
    - The new Free VSSs are all those that are not unknown and not occupied by Fully Functional or Incomplete trains;
    - The new Occupied VSS are all those occupied by Online or Incomplete trains except those VSS known to be unknown;
    - The new Unknown VSS are the same as in the previous state.
  - If t is Offline:
    - There are no changes in the States as far as the Movement Authority is concerned because even though the train moved, it is not able to communicate its new positions to the MA.
  - If t is Incomplete:
    - The new Free VSSs are all those that are not unknown and not occupied by Fully Functional or Incomplete trains;
    - The new Occupied VSS are all those occupied by Online or Incomplete trains except those VSS known to be unknown;
    - The positions occupied by the lost fragment of the train are added to Unknown.

(b) A train loses communication with the Movement Authority (*loseConnection[t: Train]* predicate)

Given a Train t not Offline:

- t is added to the Offline subset;
- All train positions remain the same;
- Changes to the states of the VSS:
  - Free VSSs are untouched;
  - The VSS occupied by the cars of the train are moved from the Occupied subset and added to the Unknown subset;
  - The VSS occupied by the cars of the train are added to the *unknowns* Train relation.

(c) A train regain communication with the Movement Authority (*loseConnection[t: Train]* predicate)

Given an Offline train t:

- t is removed from the Offline subset;

- All train positions remain the same;
- Changes to the states of the VSS:
  - The new Free VSSs are all those that are not unknown and not occupied by Fully Functional or Incomplete trains;
  - The new Occupied VSSs are all those occupied by Online or Incomplete trains except those VSSs known to be unknown;
  - The VSSs in the train's *unknown* relation are subtracted from the Unknown State subset.

(d) A train loses integrity (*loseCar[t: Train, c: Car]* predicate)

Given a Train t and a Car c such that *c in t.cars*:

- t is added to the Incomplete set of trains;
- All VSSs occupied by the cars behind c will be added to Unknown;
- c is added to the *brokenCar* relation.

(e) System Stuttering (*nop[]* predicate)

All VSS and Train states remain unchanged.

As a way of controlling all these possible events, we use the fact *Traces* that forces that only one of these events occurs at a given state.

4.4 Behavioural Properties Specification and Verification

The first property to define is under the fact *onlyOneState*. As the VSS's states and the Train states are defined as subsets, the disjointedness is not inherently guaranteed. We enforce them with an *always* temporal operator and ensure there are always no intersections between any subset combinations and that any of the mentioned signatures always belong to some state.

4.5 Behavioural Validation

To validate all properties of the behavioural design, we create the following run and check commands:

- **Run Commands**

  (a) *trainMovement*: run command to validate the movement of a train from the beginning to the end of the track;

  (b) *trainConnections*: run command to validate both losing and regaining connections given 2 trains *t1, t2* such that:
  - t1 starts at the beginning of the track;
  - No train starts at the End of the track;
  - "eventually gainConnection[t2]" ensures that t2 will both lose connection and later regain it;
  - Immediately before regaining connection, t2 must move;
  - Eventually there will be no free VSS between trains.

  (c) *runLoseCar*: run command to validate both losing and regaining connections *t, c* such that:
  - t starts at the beginning of the track;
  - No train starts at the End of the track;
  - "eventually loseCar[t, c]" ensures that t will lose c;

  - After losing a car, t must move;
  - Eventually there will be no free VSS between trains.

- **Check Commands**

  (a) *fullSafety*: Check command to verify if there are ever any train collisions between any 2 trains *t1* and *t2*

  (b) *noTrainMovementCollision*: Check command to verify if there are no train collisions given 2 trains *t1, t2* such that:
  - t1 starts at the beginning of the track;
  - t2 does not start at the end of the track;
  - Eventually one of the trains reaches the end of the track;
  - Eventually there will be no free VSS between trains.

  (c) *noTrainMovementCollision2*: Check command to verify if there are no train collisions given 2 trains *t1, t2* such that:
  - t1 starts at the beginning of the track;
  - t2 does not start at the end of the track;
  - Eventually one of the trains reaches the end of the track;
  - Eventually there will be no free VSS between trains;
  - One of the trains loses and regains connection with the Movement Authority.

5 Conclusion

One of the central takeaways from this work was that modelling decisions have a powerful impact on a system's capability and complexity. We encountered certain pain points, like, for example, updating the VSS state after a Train gets reconnected, which could've been avoided through different modelling decisions. The fact we were already quite advanced in the work made it difficult to roll back and change the structural model.

Another key consideration is related to the way complexity seemed to creep up as the project developed. As we followed along and resolved certain cases, new challenges kept presenting themselves. We feel this was an important lesson in scoping, at least as far as model validation is concerned.

All in all, Alloy is a powerful tool, provided one can abstract the meaningless details and focus on the key aspects and target use cases of the system.

A Annexes

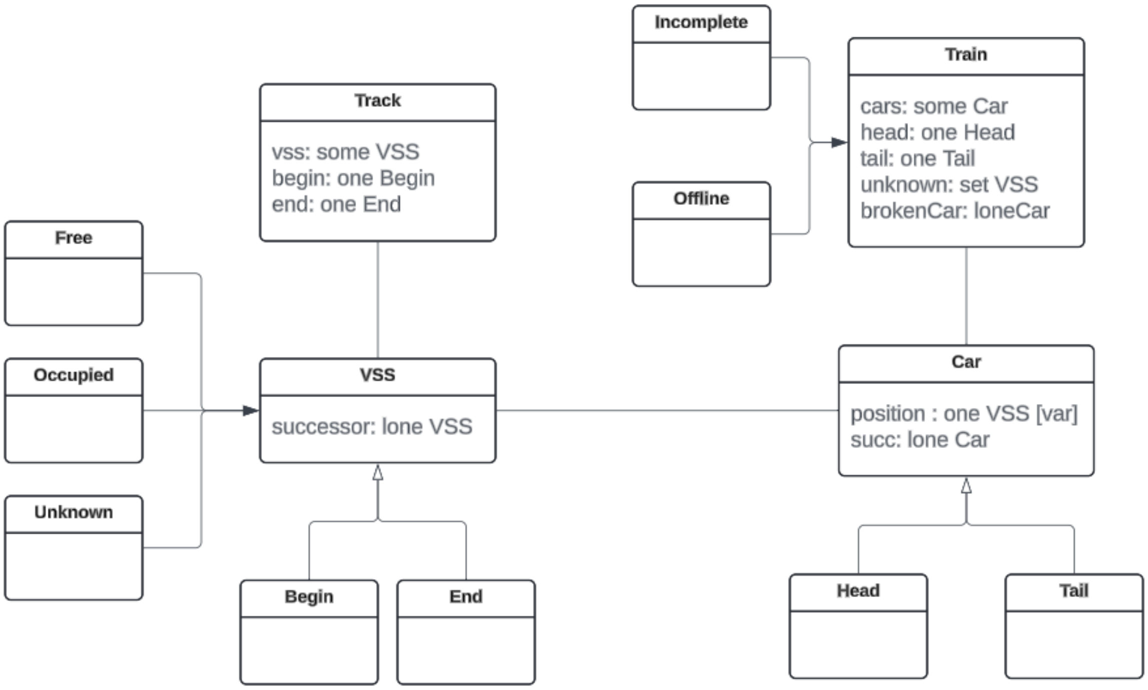

Figure 1: Conceptual Model