Assignment 2
Railway Modelling in AToMPM

Simon Van Mierlo
simon.vanmierlo@uantwerpen.be

1 Practical Information

The goal of this assignment is to design a domain-specific modelling language (formalism) and subsequently to model railways in that language in the visual modelling tool AToMPM. The different parts of this assignment are listed below:

1. Implement the abstract syntax of your language in AToMPM (in the /Formalisms/__LanguageSyntax__/SimpleClassDiagram formalism).
2. Enrich the abstract syntax with constraints so that you can check that every model is well-formed.
3. Create a concrete syntax (in the /Formalisms/__LanguageSyntax__/ConcreteSyntax formalism), and generate a modelling environment (by compiling the metamodel and the concrete syntax model). Do this incrementally.
4. Make your railway language visually more appealing by adding actions that move/resize/... model elements and display information on the elements of the model. To implement this, you'll need to make use of the mappers and parsers in the concrete syntax of your formalism.
5. Create some railway models that are representative for all the features in your language. Some should be valid models (check by verifying them), and some should be invalid, to show that your constraints are correct.

Write a report that includes a clear explanation of your complete solution, the modelling choices you made, as well as an explanation of your testing process. Also mention possible difficulties you encountered during the assignment, and how you solved them. You will have to complete this assignment in groups of 2. Submit your assignment (report in pdf, abstract and concrete syntax definition, and example models) on Blackboard before Friday, October 28, 13:00h.

Contact Simon Van Mierlo (simon.vanmierlo@uantwerpen.be) if you have a problem.

2 Requirements

This section lists the requirements of the railway domain-specific language. They are split into two sections: one on abstract syntax, and one on concrete syntax. Make sure to test each requirement with test models!

The abstract syntax of the DSL captures its *syntax* and *static semantics*. The requirements of the railway language are listed below:

- A railway system consists of the infrastructure with the trackwork, the signalling equipment, and the stations, as well as the trains driving on the trackwork and through the stations.
- A railway network consists of a number of interconnected railway *segments*. The types of segments your language needs to support are listed below:
  - **Straights**—the most trivial segment, allows a train to move straight. Has one incoming and one outgoing segment.
  - **Turnouts**—allows a train to go straight, or take a diverging route to another track connected to the segment. Has one incoming segment and two outgoing segments. The direction of a turnout can be changed—it's either in *straight* mode, meaning the train will take the straight route, or in *diverging* mode, meaning the train will take the diverging route.
  - **Junctions**—joins two segments. Has two incoming segments and one outgoing segment. The direction of a junction can be changed—it's either in *straight* or *diverging* mode, selecting the train which can enter the segment.
  - **Stations**—like straights, but can also be at the end or beginning of a track. Has zero or one incoming and zero or one outgoing segment, but is always connected to at least one track. Each station has a unique name, that starts with a single upper case letter followed by zero or more lower case letters, and ends with zero or more numbers.
- Although an undesirable property at runtime, your language should allow for more than one train to be present on a railway segment.
- A railway network does not allow loops (*i.e.*, it should never be possible to start at a station, move in one direction, and end up at that same station).
- The signalling equipment of the railway system consists of *lights*. These lights control the traffic on a segment and can be in two states—*red* means traffic on that segment needs to wait, *green* means the traffic can pass. Segments only allow one-way traffic.

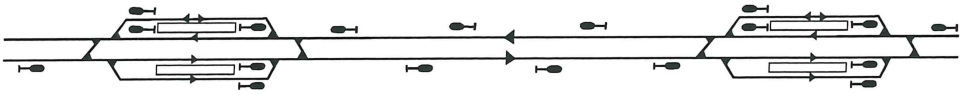
- Trains have a unique identifier, that start with an upper case letter, followed by four numbers. A train can be on at most one segment.

Each train has an associated *schedule*. This schedule is modelled in a second domain-specific language. We do this because we want to load multiple schedules for the same railway network. The requirements for the scheduling language are listed below:

- A schedule is associated to a train by referring to the name of the train.
- The schedule of a train tells it where to go—it contains a sequence of consecutive steps. The 'start' step contains the name of the station where the train will start. It has no incoming steps and one outgoing step. The 'end' step contains the name of the station where the train will stop. It has one incoming step and no outgoing steps. The start and end steps are mandatory. In between, the train has to be instructed to take the diverging or straight route when it encounters a turnout. These 'in-between' steps have exactly one incoming and one outgoing step.
- Each train needs exactly one schedule.
- Each schedule needs exactly one train.

2.1 Concrete Syntax

Notations in railway modelling are standardized. Please base yourself on the notations used in the book "Railway Operation and Control", by Joern Pachl. The following image provides an idea of how a railway network looks like:



Based on this notation, come up with a suitable icon for each element in your language. Further requirements are listed below:

- Model an action that automatically "snaps" a segment when it is connected to another segment.
- Display the direction in which junctions and turnouts are set.
- Display other useful information as you see fit (such as the name of trains and stations).

3 Useful Links

- AToMPM main page: `http://www-ens.iro.umontreal.ca/~syriani/atompm/atompm.htm`
- AToMPM user manual: `https://msdl.uantwerpen.be/documentation/AToMPM/`
- AToMPM git repository: `https://msdl.uantwerpen.be/git/simon/AToMPM` (includes installation instructions)
- Useful tutorials can be found on the 'Tutorials & Demos' page on the main website.