Assignment 2
Modelling in AToMPM

Joeri Exelmans
joeri.exelmans@uantwerpen.be

1 Practical Information

This assignment will make you familiar with the visual modelling tool **AToMPM**. You will learn to create meta-models and abstract and concrete syntaxes for a domain-specific modelling language (formalism) concerning production systems (factories).

The different parts of this assignment:

1. Implement the abstract syntax of your language in AToMPM.
   - The formalism used will be `/Formalisms/LanguageSyntax/SimpleClassDiagram`
   - You can also quickly create new formalisms with the *create new formalism* button.
2. Enrich the abstract syntax with constraints so that you can check that every model is well-formed.
3. Create a concrete syntax, and generate a modelling environment by compiling the metamodel and the concrete syntax model. Do this incrementally.
   - The formalism for this part will be `/Formalisms/LanguageSyntax/ConcreteSyntax`
4. Create some production system models that are representative for all the features in your language. The requirements for two valid models are specified below, and there should be a third invalid model to show that your constraints detect invalid models.
5. Write a report that includes a clear explanation of your complete solution and the modelling choices you made. Also mention possible difficulties you encountered during the assignment, and how you solved them. Don't forget to mention all team members and their student IDs!

This assignment should be completed in groups of two if possible, otherwise individually is permissible.

Submit your assignment as a zip file (report in pdf + abstract and concrete syntax models) on Blackboard. If you work in a group, only *one* person needs to submit the zip file, while all others *only* submit the report. Contact Joeri Exelmans if you experience any issues.

2 Requirements

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

2.1 Abstract Syntax

There are no modifications in this section from Assignment 1.

The *abstract syntax* of the DSL can be thought of as a set of constraints that instances of your language must conform to. These include multiplicities on types, relations between types, and additional constraints.

2.1.1 Waterway network abstract syntax

The abstract syntax of our waterway network-language consists of the following types:

Watercraft - A manmade "thing" (e.g., boat, ship, drone, ...) that at any time must be located on exactly one Segment or Confluence.

Segment - A "piece" of waterway. A Segment instance can contain at most 1 Watercraft instance. A Segment has one "out"-connection (that connects to another Segment), and one "in"-connection (that also connects to another Segment).

Source - A special segment type that only has an "out". It is a place where new Watercraft is spontaneously generated. A Source has an "out" connection to a Segment, onto which it puts generated Watercraft, as we'll see in the operational semantics. A Source keeps a counter of how many Watercraft instances it has generated so far. A Source has a *rate*, which is a strictly positive integer parameter, indicating the number of steps to wait before another Watercraft is generated (this will be explained in more detail in the section on operational semantics).

Sink - A special segment type that only has an "in". It is a place where Watercraft is destroyed/consumed. A Sink keeps a counter of how many Watercraft instances it has consumed so far.

Note: Sources and Sinks cannot contain any Watercraft.

Confluence - A special segment type with two in-connections (in_0 and in_1) and two out-connections (out_0 and out_1). Just like an ordinary segment, a confluence can contain (at most) 1 Watercraft instance. A Confluence has a *mode*, which is either 0 or 1.

We'll see that, either traffic can flow from in_0 to out_0, or from in_1 to out_1, both not both simultaneously. (Also, traffic can never flow from in_0 to out_1 or from in_1 to out_0.)

Schedule - A total ordering on all the elements (Segments, Sources, Sinks and Confluences) that make up the waterway network. This ordering will be used in the operational semantics. For any waterway network, exactly one Schedule must exist, which defines an ordering on all elements of the network.

For all of these connections, an "in" must connect to an "out" (and vice-versa): If an element A's "out" connects to another element B, then B's "in" must connect to A. Further, a segment's output is not allowed to be connected to the same segment's input.

Hint: Introducing additional abstract type(s), may make your solution cleaner. AToMPM also supports (multiple) inheritance!

2.2 Concrete Syntax

In this assignment, you will have a lot of freedom coming up with your own notation. The only requirements are that:

- Your notation does not need to be beautiful, but it must be clear and understandable. Use intuitive icons, colors, ...
- If you download images from the internet, strictly speaking, you should check the license. For example, `flaticon.com` requires textual attribution which can be placed in your report.

Figure 1 shows an example instance of a possible concrete syntax for the waterway network language.

Further, actions, mappers, and parsers of AToMPM must be used to improve the user experience of creating models in the waterway network language.

- At a minimum, you should display the values of attributes of your instances in your visual concrete syntax.
- Further, you can write an action that "snaps" a watercraft to a segment, when it is connected to that segment.
  - `Formalisms/Pacman` has an example of this in the `Positionable` class.

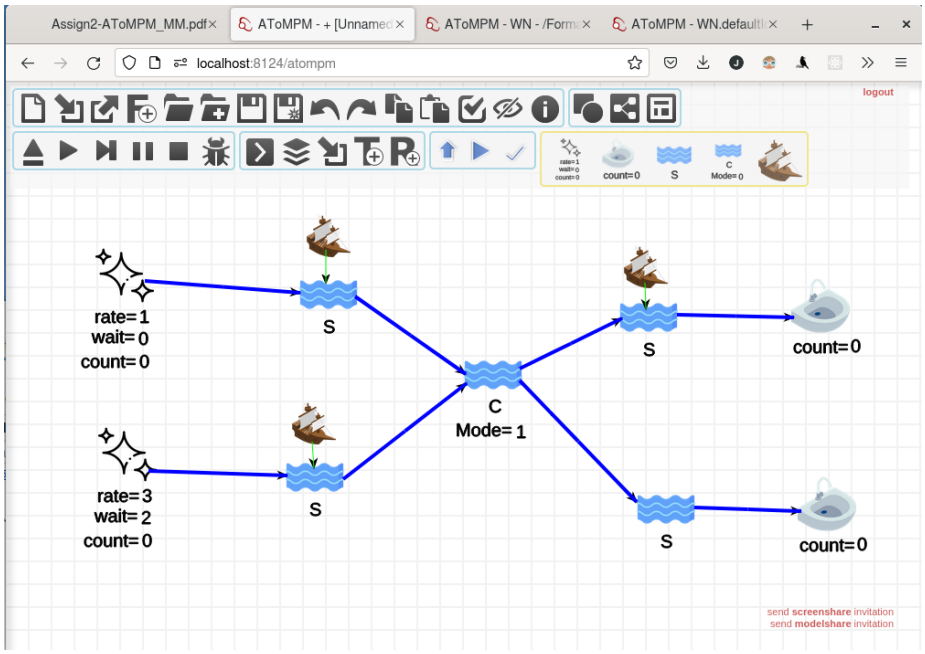

Figure 1: An instance of the concrete syntax of a Waterway Network in AToMPM.

3 For the Next Assignments

The next assignments will continue to use AToMPM, and will build further on your solution for this assignment. Therefore:

- Spend time becoming familiar with AToMPM concepts and interface.
- Report issues, bugs, annoyances, and suggestions to Joeri Exelmans[1].
- Think carefully about your solution, and spend extra time improving the concrete syntax. To prevent issues in future assignments, make sure you keep your abstract syntax as close as possible to your solution for the first assignment.
  - TIP: It must be possible for a Confluence to distinguish between its two inputs and two outputs. If you simply have two incoming/outgoing links of the same type in AToMPM, they will be unordered, and therefore they cannot be distinguished. Try to come up with a simple solution. It should be visible in your concrete syntax which input is 'input0' and which is 'input1'. (This is lacking inthe example in figure 1.)
  - Don't forget to define a schedule! (This is also lacking in the example in figure 1.) Some suggested ways of assigning an ordering on segments:
    - Add a 'priority' integer attribute to every segment.
    - Add a 'higherPriorityThan' association between segment types, effectively allowing you to put all your segments into a linked list that represents the schedule.
- To prepare yourself a bit for the next assignment, look at the AToMPM documentation on how to use transformations, and if possible, begin experimenting.
  - The next assignment will use transformations to implement the operational semantics of the watercraft network.

4 Report

There are a number of requirements for the report. Above all, I must be able to read the report and have a clear understanding of all aspects of the assignment, without having to investigate the model files. I.e., your model files will only be used as a support for your report, not the other way around!

Specifically, the report must contain:

[1] As AToMPM is nearing its end of life, these will not be solved, but rather marked as checks for the next visual meta-modelling tool.

- A brief outline of how the abstract syntax, concrete syntax, and example models meet the requirements of the assignment
  - This may include metamodels, diagrams, (pseudo-)code, etc. as needed to provide the essential details of the assignment.
- A discussion of any interesting decisions and assumptions made.
- A discussion of possible improvements to the abstract/concrete syntax.
- A brief description of the constraints present in your languages.
- Three example waterway network models (you are allowed to show the same models from assignment 1).
  - Two valid, one invalid (which doesn't meet the constraints).
- For each, show:
  - A figure of the waterway network within AToMPM.
  - The results of constraint checking on the invalid waterway network, and which constraint fails.

5 Useful Links and Tips

- AToMPM main page: https://atompm.github.io/
- Download and code: https://github.com/AToMPM/atompm
- Documentation: https://atompm.readthedocs.io/en/latest/

Acknowledgements

Based on an earlier assignment by Randy Paredis.