Assignment 3
Operational Semantics in AToMPM

Cláudio Gomes
claudio.gomes@uantwerp.be

July 29, 2017

1 Practical Information

The goal of this assignment is to implement the operational semantics implemented in the first assignment, but using a graph and rule based model transformation language, in the AToMPM visual metamodelling environment.

1.1 Task Overview

Task 1 *Change the abstract and concrete syntax of Bmod to include simulation state information.*

Task 2 *Implement the operational semantics of Bmod using the MoTiF transformation language.*

Task 3 *Re-create the (non-trivial) model created in the previous semantics, and study its room occupancy.*

Task 4 *Write a report.*

1.2 Deadline and Logistics

Complete this assignment in **groups of 2**.

One, and only one, person in the group must submit the solution on Blackboard before TBA.

Contact Cláudio Gomes (claudio.gomes@uantwerp.be) if you have questions.

2 Requirements

2.1 Task 1

The simulation of a Bmod model requires that a certain run-time state is kept. For example, the current time of the model, and/or the current stage that each person is with respect to his/her schedule, etc...

The necessary changes have to be made to the abstract syntax to be able to represent this information.

Sketch the solution to Task 2 before solving Task 1, to avoid changes after the operational semantics is made (see the advices at the end of this document).

2.2 Task 2

The requirements for this task are the same as the ones for the first assignment, except the task is to be accomplished using the model transformation tool shipped with AToMPM.

2.3 Task 3

The requirements for this task are the same as the ones for the corresponding task in the first assignment.

3 Tips, Tricks, Pitfalls, and Issues

Below is a list of known obstacles, and advice, about the current assignment:

Advice. To solve Task 2, it is advised to attach extra derived information to the model (e.g., information about the shortest path from each room to any other room). This information needs to be taken into account when solving Task 1, and can be computed as a pre-processing step of the simulation.

Advice. Any change to the abstract or concrete syntax element x automatically renders any existing transformation rule that uses x obsolete. To remedy this, x must be re-added to the rule, using the most recent toolbar.

It is easier to prevent this, by having a sketch of the solution to Task 3, before starting creating any transformation rule.

Advice. In solving Task 2, it is advised to have a pre-processing stage where the run-time information of the simulation is reset to an initial state. This will make your life easier as you will be running the transformation many times (e.g., to test each new transformation rule).

Advice. When a transformation rule fails and you don't know exactly why, relax the LHS and try the transformation again, to see if it matches. Then increase the LHS incrementally, to see where the mistake is.

Bug. When an element is created outside a LHS or RHS, its attributes contain generic code. After this element is moved into a LHS or RHS, these attributes acquire default code that is consistent with its use.

This convenient behavior is not implemented for links, since where are not technically moved into the RHS or LHS, but rather created there.

The behavior is also not implemented for groups of elements.

As a consequence, make sure the attributes of elements in LHS and RHS are correctly set. Failing to check this for elements in the RHS will cause the matches to fail, with no errors being declared.

Advise. When using the default values for attributes, make sure the default value is of the correct type. That is, if an attribute is declared as an “int”, then the default value should be 0, and not “0” (a string). This also applied when using the setAttr function in your transformations.

Failing to abide by this rule will make AToMPM store the expression as is, and will thus cause mistakes to be made later. For example, suppose the integer attributes ’watch’ and ’time2Move’ are wrongly set to “0” and “1”, respectively. Then, the following code will just concatenate two strings, and check if they are the same as getAttr(’after’,’3’):

```
result = getAttr('watch','0') + getAttr('time2Move','0') == getAttr('after','3')
```

Bug. Due to bugs storing the JSON, avoid using tabs, double quotes in the code. Represent strings with single quotes and use spaces instead of tabs.

Advice. When using the print statement to print an object, its id is printed. This the same id that you see when editing that object’s attributes. This is particularly useful when using the `getNeighbours` function.

Advice. You can duplicate a rule by locating its `R_*.model` file and duplicating it. Then you can open the new rule and make the necessary changes.

Bug. If some type *X* was removed from the abstract syntax, don’t try to remove the instances of type *X* from any existing model. AToMPM will crash if you do that.

Instead, create a new model, and abandon the old one. You can remove the old one by just removing the corresponding .model file.

Advice. A possible workflow while creating the operational semantics in AToMPM is as follows:

1. Sketch some transformation rules.
2. Create those transformation rules.
3. Reference them in the transformation schedule.
4. Create (or use an existing) simple model to test the transformation.
5. Run the transformation.
6. If everything went well, backup and go to step 1.