MDE Assignment 5:
Operational Semantics - Rule-Based Model Transformation

Vanessa Flügel
contact: vanessa.flugel@uantwerpen.be

November 5, 2025

You are discouraged but allowed to make use of any GenAI tool in solving the assignment or writing the report (for example, to correct grammatical mistakes) as long as you adhere to the UA Guidelines for students on responsible use of GenAI.

You are required to cite any use of GenAI and describe what portions of the assignment you used it for. Note that a significant part of demonstrating that the learning goals have been achieved, includes being able to explain the relevant concepts in the assignment, explain the design choices in your implementation, and critically discuss your solution.

This will be evaluated in a short ($\sim$ 15 minutes) oral evaluation, which will be conducted in the weeks following the deadline of every assignment.

1 Introduction

In this assignment, you will implement the same operational semantics of the 'RPG'-DSL from the previous assignment, but this time, using rule-based model transformations. During simulation, there is still a *design model* (that doesn't change), and a *runtime model* (that changes after every execution step). The only difference is how the execution steps are defined: namely as a set of *transformation rules*, as opposed to Python code.

A transformation rule classically contains three parts:

Left-Hand Side (LHS) A pattern that must occur, for the rule to produce a match.

Negative Application Condition (NAC) A pattern that must not occur, for the rule to produce a match.

Right-Hand Side (RHS) Describes what the matched pattern should look like *after* executing the rule.

LHS and NAC together define when a rule can fire. First, the LHS-pattern is searched for in the model that is being transformed (this model is sometimes called the *host graph*). For every LHS-match, the matcher attempts to *grow* the match with the elements defined in the NAC. Only if the latter does *not* succeed, does the entire rule match.

LHS and RHS together define what happens when the rule fires: Any element that occurs in the LHS, but not in the RHS, is deleted. Any element that does not occur in the LHS, but occurs in the RHS, is

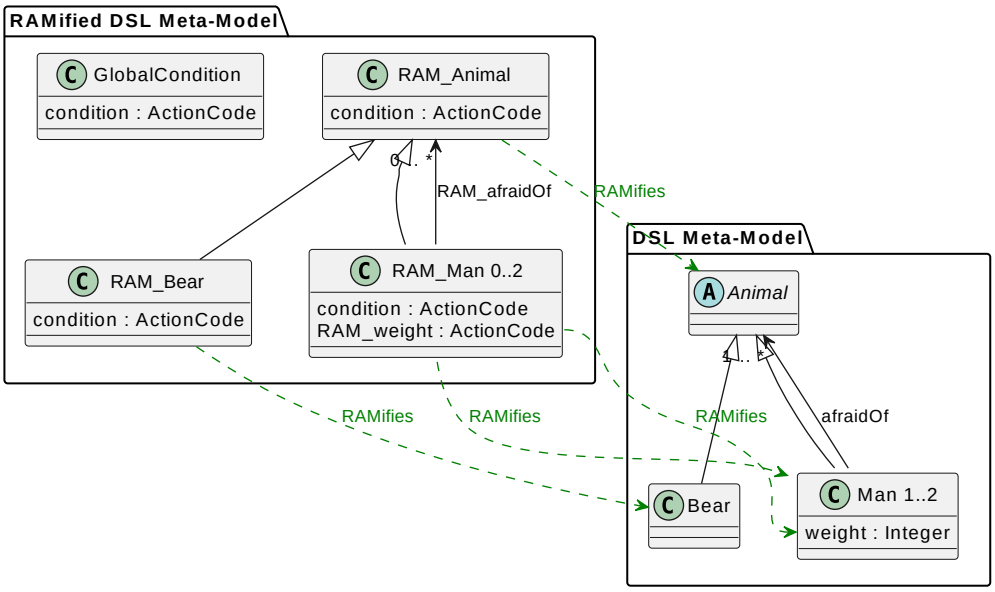

Figure 1: RAMification of our 'woods'-metamodel.

created. Any element that occurs in both LHS and RHS is left untouched (with the exception of attributes, which can be updated, as we will see).

The LHS-, NAC- and RHS-patterns that a transformation rule consists of, are models too: they are instances of the *RAMified runtime model*. RAMification turns the meta-model of the host graph into a new, slightly different meta-model. For instance:

- the lower-cardinalities for all types become zero
- abstract classes become concrete
- the types of attributes are turned into ActionCode (LHS/NAC: boolean expression indicating whether to match, RHS: expression of new value)
- every class gets an additional `condition`-attribute
- a `GlobalCondition`-class is added

RAMification can be done automatically. Figure 1 shows an auto-generated RAMified meta-model of a small example-formalism.

2 Overview of Assignment

2.1 Tasks

To complete this assignment, you will:

1. Download the ZIP-archive containing the assignment files from the website (`http://msdl.uantwerpen.be/people/hv/teaching/MSBDesign/assignments/files/assignment5.zip`) and place them in your muMLE directory. Some of the files from the previous assignment will be reused (`models.py` and `simulator.py`), so place the new files in the same location. It should not be necessary to modify your (meta-)models. However, if you do, describe any changes you made in your report.
2. **Recreate the operational semantics using rules.** Refer to section 3 for the specification and notes. Place all rules you create in the `rules` directory. Every rule consists of at least two files: the LHS and RHS patterns. Name these files `r_<name of your rule>_<lhs | rhs>.od`. Additionally, it is possible to add optional NAC patterns. The first NAC file is named `r_<name of your rule>_nac.od` and all additional NACs `r_<name of your rule>_nac2.od`, `r_<name of your rule>_nac3.od`, etc.
3. Open the file `assignment5.py` and **implement the action generator and termination condition**.
4. Either reuse your textual renderer function from the previous assignment or adapt it for this new assignment.
5. **Run the simulation** by executing `rule_runner.py`. Include at least one execution trace in your report. *Note: Remember that the execution automatically ends after 10 steps.*

Write a short PDF report explaining your solution by showing code fragments and your thought process behind everything. Include both team members' names on the report.

2.2 Practical

- Students work in pairs.
- One team member submits a ZIP file containing your report and code.
- Deadline: 19 November 2025, 23:59.

3 Specification

The role-paying game has the following operational semantics (mostly unchanged from assignment 4):

- Every time-step starts with a move from the `Hero`. As long as they are alive, they can choose an adjacent `Tile` to move to. Depending on the type, they also:
  - `Trap`: Lose a life.
  - `StandardTile` with `Item`: Collect the `Item`. The `Item` is removed from the `Tile`. In case of a `Objective`, the points are added to the total collected points.
  - **Note: You can implement these additional actions as separate rules.**

- If the `Hero` is currently located on a `Door` and has the matching `Key`, they may also choose to use the `Door`.
- Next, if there is a `Monster` in the same `Level` as the `Hero` and it is alive, it moves to a random adjacent `Tile`. **Note: Instead of a completely random choice, you can implement the monster moving so that you pick the direction.**
- After this, if both the `Monster` and `Hero` are on the same `Tile`, they fight. The one with more lives wins the fight and the loser loses one life.
- Lastly, if all possible actions have been carried out, the time on the `Clock` moves on one step.
- This loop continues until
  - the `Hero` has no more live or
  - all `Objectives` have been collected.

4 Tips

- To update a slot, the slot must occur in both LHS and RHS. If the slot only occurs in RHS, it will be created (possibly leading to an object having two attributes with the same name).
- Items in NAC/LHS/RHS are matched by their *name* in the pattern. Therefore, it is best to not use anonymous objects/links in patterns.
- Multiple correct solutions exist. The RHS of every rule can contain arbitrary code (in a `GlobalCondition`). Code is needed sometimes because model transformation rules by themselves are less expressive than code. However, try to use not too much code! Solutions that are easy to explain / understand are favoured.
- Beware that an empty NAC (i.e., a NAC that exists as a file, but contains no elements) will *always* match, therefore causing the entire rule to *never* match.

5 API

The same API that we are already familiar with, is available in the attributes of objects in LHS / NAC / RHS patterns:

	Availability in Context					
	Meta-Model Constraint		Model Transformation Rule			
	Local	**Global**	**NAC LHS**	**RHS**	**OD-API**	**Meaning**
Querying						
`this :obj`	✓		✓	✓		Current object or link
`get_name(:obj) :str`	✓	✓	✓	✓	✓	Get name of object or link
`get(name:str) :obj`	✓	✓	✓	✓	✓	Get object or link by name (inverse of `get_name`)
`get_type(:obj) :obj`	✓	✓	✓	✓	✓	Get type of object or link
`get_type_name(:obj) :str`	✓	✓	✓	✓	✓	Same as `get_name(get_type(...))`
`is_instance(:obj, type_name:str [,include_subtypes:bool=True]) :bool`	✓	✓	✓	✓	✓	Is object instance of given type (or subtype thereof)?
`get_value(:obj) :int|str|bool`	✓	✓	✓	✓	✓	Get value (only works on Integer, String, Boolean objects)
`get_target(:link) :obj`	✓	✓	✓	✓	✓	Get target of link
`get_source(:link) :obj`	✓	✓	✓	✓	✓	Get source of link
`get_slot(:obj, attr_name:str) :link`	✓	✓	✓	✓	✓	Get slot-link (link connecting object to a value)
`get_slot_value(:obj, attr_name:str) :int|str|bool`	✓	✓	✓	✓	✓	Same as `get_value(get_slot(...))`
`get_all_instances(type_name:str [,include_subtypes:bool=True]) :list<(str, obj)>`	✓	✓	✓	✓	✓	Get list of tuples (name, object) of given type (and its subtypes).
`get_outgoing(:obj, assoc_name:str) :list<link>`	✓	✓	✓	✓	✓	Get outgoing links of given type
`get_incoming(:obj, assoc_name:str) :list<link>`	✓	✓	✓	✓	✓	Get incoming links of given type
`has_slot(:obj, attr_name:str) :bool`	✓	✓	✓	✓	✓	Does object have given slot?
`matched(label:str) :obj`			✓	✓		Get matched object by its label (the name of the object in the pattern)
Modifying						
`delete(:obj)`				✓	✓	Delete object or link
`set_slot_value(:obj, attr_name:str, val:int|str|bool)`				✓	✓	Set value of slot. Creates slot if it doesn't exist yet.
`create_link(link_name:str|None, assoc_name:str, src:obj, tgt:obj) :link`				✓	✓	Create link (typed by given association). If `link_name` is None, name is auto-generated.
`create_object(object_name:str|None, class_name:str) :obj`				✓	✓	Create object (typed by given class). If `object_name` is None, name is auto-generated.