

MDE Assignment 6:

Translational Semantics - Rule-Based Model Transformation

Vnaessa Flügel
contact: vanessa.flugel@uantwerpen.be

November 19, 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 (~ 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 rules that transform models of the role-playing game (RPG) formalism to Petri nets. The benefit of transforming to another formalism, is that the entire language toolchain (in the case of Petri Nets: interactive simulation and highly optimized analysis/model checking) becomes available to us.

The translation will be done by means of rule-based model transformation, which you are already familiar with.

Model transformation in muMLE always happens within the same formalism. To transform *across* formalisms, we perform a simple trick of merging the meta-models of the input and output formalisms. In our case, this means merging the meta-models of the ‘RPG’-DSL and the ‘Petri Net’-language.



When merging (meta-)models, naming collisions can occur. In this assignment, the types of the Petri net formalism have been named specifically to avoid naming collisions.

Once we have a merged meta-model, we modify it in one more way: We add an abstract class named **Top**, which becomes the superclass of all classes¹. We also add an association named **generic.link**, whose source and target is **Top**. This effectively allows **generic.link** to connect anything with anything. Not that the

¹In type theory, the ‘top’-type is the supertype of all types. See https://en.wikipedia.org/wiki/Top_type

merged meta-model still is a valid meta-model for the existing RPG-language: it merely contains additional types. In the general case, adding new types to a meta-model does not break compatibility with existing models.

Next, the merged meta-model is RAMified, and we can finally create model transformation rules that transform ‘RPG’-models into ‘Petri Net’-models. We will do this not in a single rule, but with a multiple rules, which build the Petri Net model incrementally. We will also not erase the ‘RPG’-model (or any of its elements), but rather “append” the Petri Net model to it, and we will create traceability-links from the created ‘Petri Net’-elements back to their respective ‘RPG’-elements.

2 Overview of Assignment

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/assignment6.zip>) and place them in your muMLE directory. Just like in the last assignment, files are reused (mainly `models.py` and `simulator.py`).
2. **Create a merged meta-model.** The assignment files include the meta-model for the ‘Petri net’-formalism as well as a script to automatically merge the models. Execute `generate_merged.py` and you should see the file `merged_mm.od` appear in the same directory.
3. **Create rules to transform ‘RPG’-models to ‘Petri net’-models.** The rules are created in the same way as in the last assignment, but using the (ramified) new merged meta-model. Place them in a directory called `translation`.
4. Open the file `translate_runner.py` and **add your rules** to the list `rule_names`. Note that they will be executed in the order in which you place them in the list. Order them accordingly.
5. **Transform your model** by executing `translate_runner.py`. The script will execute each rule as long as it finds matches and then move on to the next. **This can take a while!** After each rule has finished, a snapshot is created so you can start from an intermediate step if you change something about a later rule (the script expects a directory called “snapshots” for this). Include the **final** result of your translation in your submission.

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.

2.1 Practical

- Students work in pairs.
- One team member submits a ZIP file containing your report and code.
- Deadline: 03 December 2025, 23:59.

3 A Note on Specification

The specification is largely the same as in the previous assignments. There are three major changes:

- You do not have to implement the fight mechanic - In this version the **Hero** and **Monster(s)** only move, with all side-effects such as collecting **Items**, stepping in **Traps** or using **Doors** for the **Hero**
- You do not have to keep track of the elapsed timesteps (but the order of actions should still be preserved!)
- You do not have to implement a termination condition

A meta-model for Petri nets is already provided for you. You will need to use the following types:

- **PNPlace**: A place in the Petri net, to store tokens every **PNPlace** has a **PNPlaceState** in the runtime meta-model, with an integer attribute **numTokens**. The two are connected with an association of type **pn_of** (**PNPlaceState** -> **PNPlace**).
- **PNTransition**: A transition in the Petri net. It can be connected to places with associations of type **arc** for regular arcs or **inh_arc** for inhibitor arcs. Note that arcs are used both to connect places to transitions (= consume tokens) and transitions to places (= produce tokens).

4 Tips

- Work incrementally! Large rules will take long to execute, so if you notice that a rule takes too long to produce matches (5+ minutes), break it up into smaller parts. You can make use of **generic_links** to find elements you have created in previous rules (i.e. add them in the RHS of rule A, then later use them in the LHS of rule B to find the element).
- When creating a new transformation rule, create the LHS first, and to see if your LHS produces the kind of match you are expecting, you can immediately execute it (without RHS or NAC). Pay attention to the match-set printed to the terminal.
- When creating an object or link in the RHS of a rule, you can specify its name with the **name**-attribute. By making the names of Petri Net elements refer to the names of the Port-elements to which they relate, you can greatly improve the understandability of your generated Petri Net.



One (dirty, but useful!) example of exploiting the **name**-attribute is the following: Suppose I am creating two places for the same element of the original model, tracking to different things, they can be distinguished by their names. In a later rule's LHS, I can even write a condition that checks the name (e.g., using the Python function `.startswith("some_prefix_")`) if I want to match with a specific kind of place. This also has the benefit of speeding up the matching!

- The assignment files also include a file called **translation_executer.py**. This file applies the operational semantics of Petri nets to the generated model: Finding activated transitions and giving you the choice to fire them. You can pass a snapshot to this script as an argument (`python translation_executer.py snapshots/snapshot_xx.someRule.od`) as the model for execution. This can be helpful to determine whether your Petri net behaves as expected.

5 API

Here is the API, once more:

	Availability in Context					Meaning
	Meta-Model Constraint		Model Transformation Rule			
	Local	Global	NAC LHS	RHS	OD-API	
<i>Querying</i>						
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 <code>get_name</code>)
get_type(:obj) :obj	✓	✓	✓	✓	✓	Get type of object or link
get_type_name(:obj) :str	✓	✓	✓	✓	✓	Same as <code>get_name(get_type(...))</code>
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 <code>get_value(get_slot(...))</code>
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)
<i>Modifying</i>						
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 <code>link_name</code> is None, name is auto-generated.
create_object(object_name:str None, class_name:str) :obj				✓	✓	Create object (typed by given class). If <code>object_name</code> is None, name is auto-generated.