MDE Assignment 4:
Operational Semantics - Coded in Python

Vanessa Flügel
contact: vanessa.flugel@uantwerpen.be

October 22, 2025

1 Introduction

In previous assignments, we have touched the following topics:

- Meta-modeling
- Instance generation
- Conformance checking
- Concrete and abstract syntax

Of course, a language is not useful without *meaning*, i.e., *semantics*. Depending on the language, the meaning can be pretty much anything. In the case of Class Diagrams, the meaning is a (possibly infinite) set of conforming object diagrams. In the case of executable languages, the meaning is a (possibly infinite) set of execution traces. In this assignment, we will code an *operational semantics* for an executable language.

In model-driven engineering, *operational semantics* is a semantics of the form that evolves an *execution state*, also called *runtime state* or *runtime configuration* from one 'snapshot' to the next. The runtime state may point to parts of the *design model* (the model that was manually created). The design model never changes during execution[1].

2 Overview of Assignment

2.1 Tasks

You will once again work with muMLE. To complete the 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/assignment4.zip`) and place them in your muMLE directory. Open `models.py` and insert your created meta-model. This is now your design model. **Modify your model** to satisfy the new requirements in subsection 3.1. **Add a conforming model**. You can either continue with the conforming model you created in assignment 2 (and add the new elements) or create a new one.

[1] Except in *live modeling*.

2. **Add a runtime meta-model and model**. These should extend your design-model with stateful information. Find the requirements for the runtime-meta-model in subsection 3.2.
3. Open `assignment4.py`. This is where you will **create the actions, preconditions and termination condition**. Some skeleton-code is already provided, modify it to fulfil the requirements in subsection 3.3.
4. Open `runner.py`. **Modify the function** `render_text` to output a small description of the current state (e.g. "[4 lives] You are standing in front of a Door", "[3 lives] You stepped in a Trap!").
5. **Run the simulation** by executing `runner.py`. Include at least one execution trace in your report. *Note: The execution will automatically stop after 10 steps, try to include a trace that shows a termination condition you implemented within that "timeframe".*

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 (the modified files).
- Deadline: 05 November 2025, 23:59.

3 Specification

3.1 Alteration to Meta-Model

The requirements from assignment 2 still remain the same. Now, the model additionally requires:

- There is at most one `Monster` per Level.
- Both `Hero` and `Monsters` are `Creatures` that have a non-negative number of lives and are on a `Tile`.
- Adjacent `Tiles` have an associated direction.

3.2 Runtime Model

In the runtime-model, the following stateful information is added:

- There is exactly one `Clock`, which has an integer attribute `time`.
- There are (abstract) `States`.
- The `WorldState` is the `State` belonging to the `World`. It has an integer attribute `collected_points`.
- The `CreatureState` is the `State` belonging to a `Creature`. It has a boolean attribute `moved`.
- A `Hero` can collect `Items`.
- Globally, no `Creature` may stand on an `Obstacle`.

3.3 Operational Semantics

The game has the following operational semantics:

- 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.
- 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`. (For the `Hero` this action is simply waiting or "Listening for monsters".)
- 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

- The model is cloned between steps. This means the UUIDs change. Always use the name (`od.get_name` to find the name and `od.get` to retrieve the object) when passing something to an action.
- An action can return multiple messages of their effect (see that the return value is a list of strings).
- A precondition should always return a boolean value to determine whether it is fulfilled or not.
- Conformance is automatically checked after every step, so make sure your rules don't lead to your model becoming non-conforming!
- It is advisable to first remove links and then add new ones.
- Use `functools.partial` to pass additional parameters to an action. Example:

```
from functools import partial

def action_do_something(od: ODAPI, necessary_element: str):
    # Implementation...

partial(action_do_something, necessary_element="MyCoolElement")
```

5 API

In the Python functions, whenever you see an object named `od`, it is an instance of the class `ODAPI` ("Object Diagram API"), defined in `api/od.py`. It extends the query-functions of the API from assignment 2 with methods for creating, modifying and deleting:

<table>
<tr><th></th><th colspan="3">Available in Context</th><th></th></tr>
<tr><th></th><th>Local Constraint</th><th>Global Constraint</th><th>ODAPI</th><th>Meaning</th></tr>
<tr><td>this :obj</td><td>✓</td><td></td><td></td><td>Current object or link</td></tr>
<tr><td>get_name(:obj) :str</td><td>✓</td><td>✓</td><td>✓</td><td>Get name of object or link</td></tr>
<tr><td>get(name:str) :obj</td><td>✓</td><td>✓</td><td>✓</td><td>Get object or link by name (inverse of get_name)</td></tr>
<tr><td>get_type(:obj) :obj</td><td>✓</td><td>✓</td><td>✓</td><td>Get type of object or link</td></tr>
<tr><td>get_type_name(:obj) :str</td><td>✓</td><td>✓</td><td>✓</td><td>Same as get_name(get_type(...))</td></tr>
<tr><td>is_instance(:obj, type_name:str
[,include_subtypes:bool=True]) :bool</td><td>✓</td><td>✓</td><td>✓</td><td>Is object instance of given type
(or subtype thereof)?</td></tr>
<tr><td>get_value(:obj) :int|str|bool</td><td>✓</td><td>✓</td><td>✓</td><td>Get value (only works on Integer,
String, Boolean objects)</td></tr>
<tr><td>get_target(:link) :obj</td><td>✓</td><td>✓</td><td>✓</td><td>Get target of link</td></tr>
<tr><td>get_source(:link) :obj</td><td>✓</td><td>✓</td><td>✓</td><td>Get source of link</td></tr>
<tr><td>get_slot(:obj, attr_name:str) :link</td><td>✓</td><td>✓</td><td>✓</td><td>Get slot-link (link connecting object to a value)</td></tr>
<tr><td>get_slot_value(:obj,
attr_name:str) :int|str|bool</td><td>✓</td><td>✓</td><td>✓</td><td>Same as get_value(get_slot(...)))</td></tr>
<tr><td>get_all_instances(type_name:str
[,include_subtypes:bool=True]
) :list<(str, obj)></td><td>✓</td><td>✓</td><td>✓</td><td>Get list of tuples (name, object)
of given type (and its subtypes).</td></tr>
<tr><td>get_outgoing(:obj,
assoc_name:str) :list<link></td><td>✓</td><td>✓</td><td>✓</td><td>Get outgoing links of given type</td></tr>
<tr><td>get_incoming(:obj,
assoc_name:str) :list<link></td><td>✓</td><td>✓</td><td>✓</td><td>Get incoming links of given type</td></tr>
<tr><td>has_slot(:obj, attr_name:str) :bool</td><td>✓</td><td>✓</td><td>✓</td><td>Does object have given slot?</td></tr>
<tr><td>delete(:obj)</td><td></td><td></td><td>✓</td><td>Delete object or link</td></tr>
<tr><td>set_slot_value(:obj, attr_name:str,
val:int|str|bool)</td><td></td><td></td><td>✓</td><td>Set value of slot.
Creates slot if it doesn't exist yet.</td></tr>
<tr><td>create_link(link_name:str|None,
assoc_name:str, src:obj, tgt:obj) :link</td><td></td><td></td><td>✓</td><td>Create link (typed by given association).
If link_name is None, name is auto-generated.</td></tr>
<tr><td>create_object(object_name:str|None,
class_name:str) :obj</td><td></td><td></td><td>✓</td><td>Create object (typed by given class).
If object_name is None, name is auto-generated.</td></tr>
</table>