.

MDE Assignment 1: Meta-modeling and instance generation with Refinery

Joeri Exelmans
contact: joeri.exelmans@uantwerpen.be

October 8, 2024

1 Introduction

In this first assignment, you will use Refinery[1], a modern web-based editor and instance generator for domain-specific languages (DSLs).

When creating a DSL, we typically define a meta-model (encoding the *intension of the language*), which describes a set of valid models in our language (= the *extension* of the language). There are several activities that involve (meta-)models, shown in Figure 1.

1. Given a set of example models (possibly sketched with pen and paper), we can (manually) try to extract a meta-model.
2. Given a meta-model, we can automatically generate conforming models.
3. Given a meta-model, we can generate a syntax-directed editor, and use the editor to manually create models.
4. Given a meta-model and a model, we can check if the model conforms to the meta-model.

With Refinery, we can do (2). While the user is defining her meta-model, Refinery can automatically generate models (also called *instances*) that conform to the meta-model. By doing this, the DSL creator can often quickly spot mistakes, such as constraints she forgot to define.

Refinery's approach to meta-modeling is highly *declarative*. The user specifies *what* is allowed or forbidden by means of constraints, rather than *how* the constraints should be checked.

[1]`https://refinery.tools`

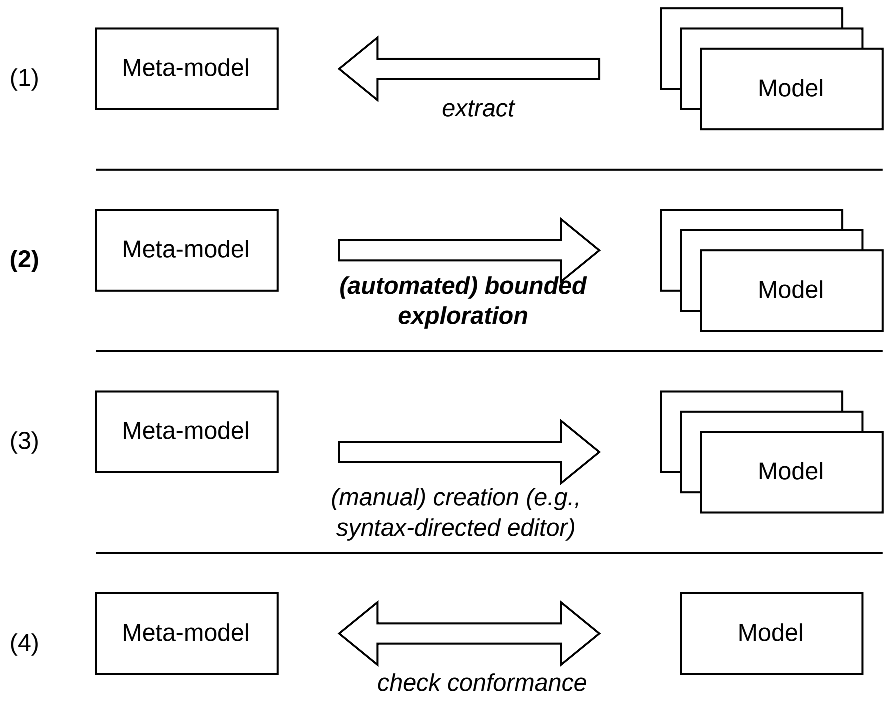


Figure 1: Overview of activities involving (meta-)models.

2 Overview of Assignment

To complete this assignment, you will:

- Open the 'starting point' meta-model, which can be found here: `https://refinery.services/#/1/KLUv_WCOASUKABYRNSNAqaYNeBjojbQlynz9bZOQBo7ROd7XiS1jTLxOkyodXQfIA` `3S5pW2vKzjzPStNwg_NZAO53NFUTLEr-I4gNcBOXgjadwR1hNBQahHBR3K9UHUJ3utas-RSCmeazB262df2gSp4` `jhCuzgp5qh7y9PrKviJJtP7ISkY2eoXjTcv5X1n9g-ypfwCLSAQFMaVbYGS8KOYdq9JJscePxJZXCuPAT84IBgU` `=`
- Extend the starting point with the relations and constraints mentioned in the specification (in section 3)
- Generate one or more instances and observe them. Do they look right to you?
  - If not, repeat the process of modifying the meta-model, and generating instance(s).
- Once you have implemented the specification, think of **one extra constraint** that you think would make sense, and implement it.
- Write a small (PDF) report explaining your solution, by showing code fragments and explaining what they do. Also include your full code in a separate file. Submit the PDF and code in a ZIP file on Blackboard.

Practical stuff:

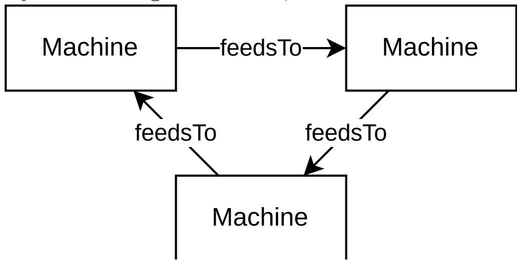
- Students work individually.
- Deadline: ~~8 October~~ 10 October 2024, 23:59.

3 Specification

You will create a meta-model for a DSL in the domain of Production Engineering. More specifically, your language will describe how machines in a factory connect to each other, and how they are operated by staff, that work in shifts.

- A `Factory` contains at least one `Machine`.
- A `Factory` has at least one `Worker` working for it.
- There are exactly three `Shifts`: morning, afternoon, and night.
- `Workers` work one or two `Shifts`. No `Workers` work three `Shifts` (this would be tough for them) and no `Workers` work zero `Shifts`.
- A `Factory` has exactly one `Source` (to receive parts), and one `Sink` (to send out assembled products).
- Connections exist between `Machines` and the `Factory`'s `Source`/`Sink`:

  - Every `Machine` has one or two inputs, that connect to (a) another `Machine`'s output, or (b) the `Factory`'s `Source`.
  - Every `Machine` has one or two outputs, that connect to (a) another `Machine`'s input, or (b) the `Factory`'s `Sink`.
  - Cycles among `Machines`, such as the following, **are** allowed:



    Cycles between machines are common in industry. For instance, think of a cooling circuit.
  - The `Factory`'s `Source` and `Sink` both must be connected to at least one `Machine`, and never more than two `Machines`.
  - A `Source` cannot be connected directly to a `Sink`.

- Every `Machine` is operated by zero or more `Workers`. A `Machine` that is operated by zero `Workers`, is considered autonomous (it can function without an operator).
- For every `Shift`, every non-autonomous `Machine` (i.e., one that *needs* an operator), must have at least one `Worker` operating it during that `Shift`.

Figure 2 shows an example conforming model (please note: for relations that are each other's opposite, links are shown in only one direction - for instance, for every `hasOutput`, there exists a link `hasInput` in the opposite direction, which has been hidden using the view controls).

4 Resources

- **Refinery Documentation:** `https://refinery.tools/learn/`

5 Tips

- The `opposite` keyword can be used to specify that if a link exists in one direction, another link must exist in the opposite direction. In the starting point, `hasOutput` is the opposite of `hasInput`.
- If model generation times out, you can try a bunch of things:
  - Play with the multiplicities in the line that says 'scope node = ...'

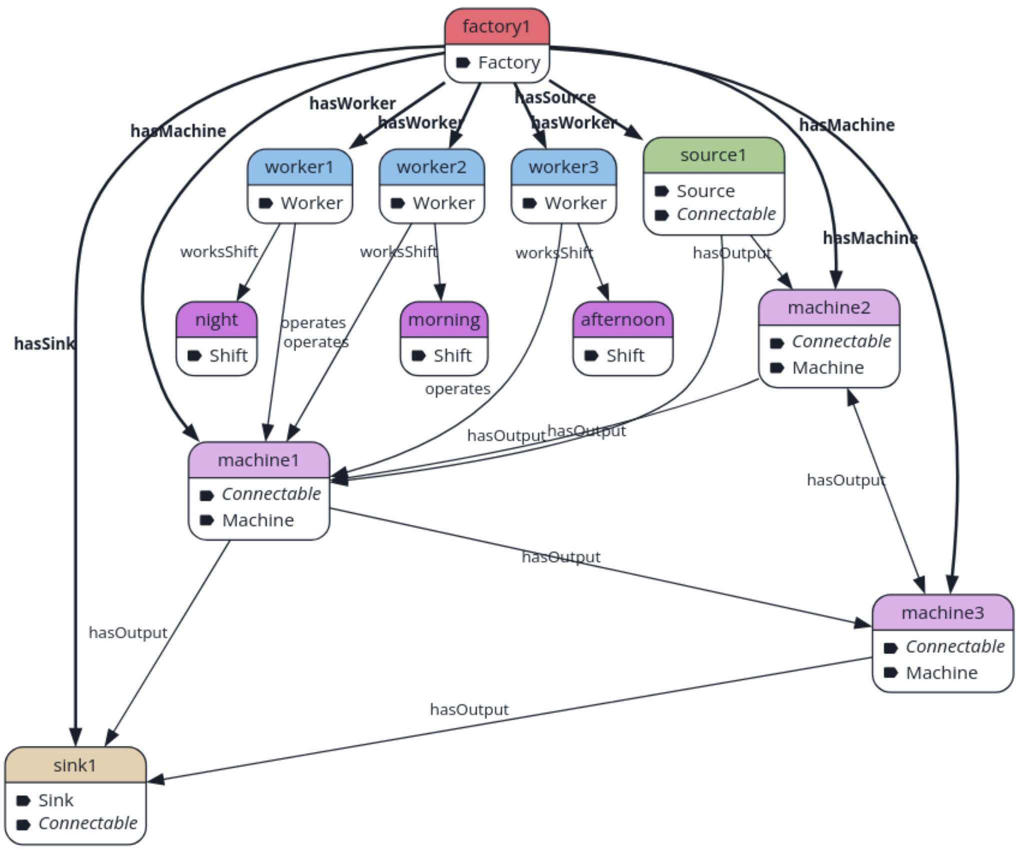


Figure 2: A conforming model (generated from my own solution)

  - Temporarily make the problem space smaller by defining only two Shifts. Note that your final solution must have three Shifts.
  - Most likely, you wrote a constraint that takes too much time to evaluate. See if you can formulate the constraint in a simpler manner.

- In the graph view on the right, you can select to hide certain elements to make the graph more readable. Especially for relations that are each other's opposite, you typically only want to see the relation in one direction.