

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

Vanessa Flügel

Contact: vanessa.flugel@uantwerpen.be

October 1, 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 minute) oral evaluation, which will be conducted in the weeks following the deadline of every assignment.

1 Introduction

In this first assignment, you will use Refinery¹, a modern web-based editor and instance generator for domain-specific languages (DSLs).

The meta-model of a DSL, typically a class-diagram and set of constraints over instances, can be used in two distinct ways. The first is to check whether a given model ‘conforms to’/‘is typed by’ the meta-model. This can, for example, be used in ‘syntax-directed editors’ to prevent modellers from building non-conforming models. The second way is to generate conforming instances. This can be interesting when designing a language.

The process is shown in Figure 1: The language engineer creates a meta-model and generates (*syntactically* correct by construction) instances from it. These instances are then checked by a human, typically a domain expert, for *semantic* errors. In other words, an expert determines whether the model makes

¹<https://refinery.tools>

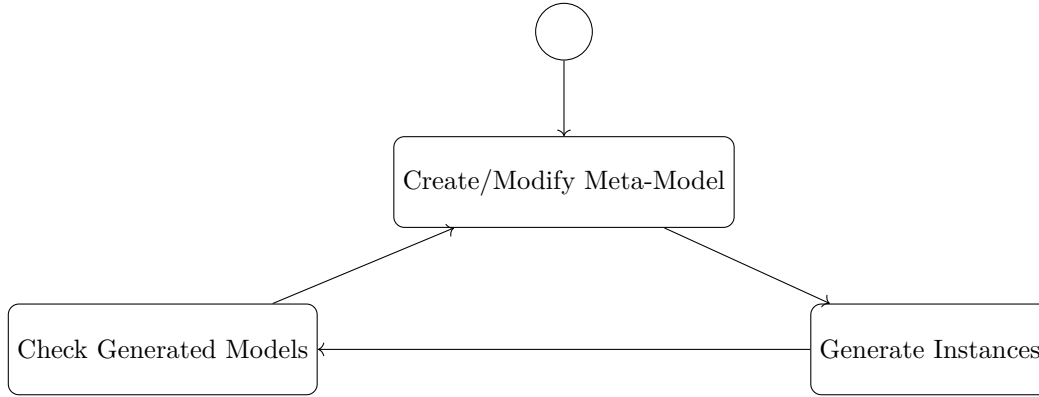


Figure 1: Process of refining a meta-model using model generation

sense for the domain. If a problem is found, the constraints are typically not restrictive enough. In this case, the language engineer refactors the meta-model and the process is repeated until the generated instances are correct.

With Refinery, we can do exactly this. While the user is defining her meta-model, Refinery can automatically generate models (also called instances) that conform to the meta-model. These are then used to identify problems.

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.

2 Overview of Assignment

2.1 Tasks

To complete this assignment, you will:

1. Open the ‘starting point’ meta-model: https://deemz.org/refinery/#/1/KLUv_WBiATUKAGaSOCQgragD3P4ISXt_Us76LXH7J8DMOALPQuYdhcY8-YonkAgUFggtADIALQDIOGWGhW810t4ugTXb2nTsbD2frQRGdcsrcbzbxbxgi79wKrrroohVpfcZ00PzAQpQB2BAzsgmj7x9IxfKDF4oEghSOHahyNhh2qgSCFwV79qvKcrmljlxAfhrunOKZ4CdT6u9rm36epSXZrpVSnrI1SMGPGJwtBku2tSv11uHNNKo-XrbcaWYgPJGCneQ8ziDg5n-VazeaHd8v93LR7iBk_iYQjp-EZRQ0-1ErybyEX1M2WzfpIRfOmICBCghikB5ZblxuoUXCsBEp2DbScSK5NXJmDfiVQLDxSViYGFjE9wq3cBGfSblmi2QwKLXGdEVMULBcHbSTBGIF1EvwOWFcEsX09AAW-nhssepsgukZ1cPdLGuZALlcG. **Extend the starting point** with the relations mentioned in the specification in section 3. Do not add constraints yet.

2. **Generate** instances (1 - 2) and identify problems. Problems specifically means *semantic* problems, so identify things that do not make sense in the domain. **Add constraints** to the meta-model and generate new instances. Show that the identified problem(s) are not present in the new instances. **Repeat the process** to identify more problems. In subsection 3.2 ideas for such constraints are listed, but you are encouraged to identify and implement own constraints.

Write a short PDF report explaining your solution by showing code fragments, your generated models, and your thought process behind them. 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: 8 October 2025, 23:59.

3 Specification

The DSL you will create describes characters in a role-playing game (RPG). The language will describe the attributes of the characters and how they are related. You can see examples for conforming models in Figure 2 and 3.

3.1 Classes and Attributes

The language must contain the following:

- There is exactly one **Hero**, who has a level and a name.
- Every **Hero** has exactly one **Race**.
- A **Race** can be for example **Human**, **Elf**, **Orc** or **Dwarf**.
- Every **Hero** has exactly one **Role**.
- A **Role** can be for example **Wizard**, **Archer**, **Thief** or **Knight**.
- Every **Hero** has **Attributes**, which have a value.
- An **Attribute** can be for example **Strength**, **Intelligence**, **Agility** or **Perception**.
- Every **Hero** has **Skills**, which have a skill name, a minimum level of the hero required, and can have **attributes**, which the hero must have.
- A **Skill** can be for example **Archery**, **Magic** or **Stealth**.

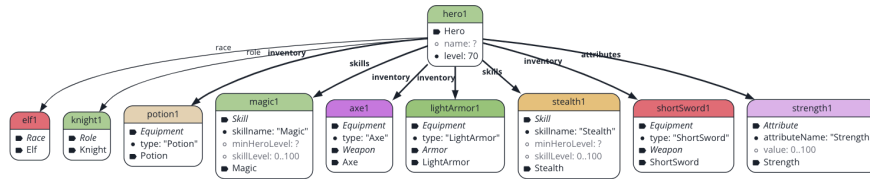


Figure 2: A conforming model (generated from my own solution)

- A Hero has an inventory, which can contain Equipment.
- The available types of Equipment are Weapon, Armour and Potion.
- A Weapon can be for example a Sword, Bow, Axe or Staff.
- An Armour can be HeavyArmour or LightArmour.

3.2 Constraints

Possible ideas for constraints are:

- A Hero cannot have only Attributes with value 0.
- A Hero cannot have multiple Races/Roles.
- A Hero cannot have the same Attribute/Skill more than once.
- The inventory cannot contain too much Equipment.
- A Hero cannot have more than one type of Armour.
- Only a Wizard can wield a Staff.
- A Thief cannot wear HeavyArmour.
- A Wizard must have Magic.
- A Knight must have Strength at a value higher or equal to 10.
- Magic requires Intelligence.
- An Archer must have a Bow.
- If a Hero has a Bow, he must have Archery.
- A Dwarf cannot be a Wizard.

4 Resources

- Refinery Documentation: <https://refinery.tools/learn/>

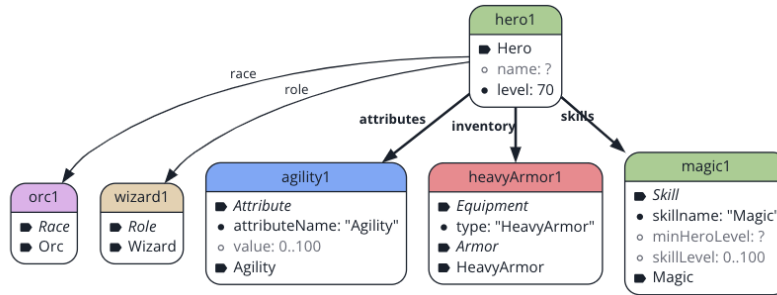


Figure 3: Another conforming model (generated from my own solution)

5 Tips

- Attributes are not assigned values by default when generating a model. It is possible to define a fixed set of values that can be assigned by using **decision rules** and **concretization rules**. In the starting point, you can see how to assign one of three possible values to the **Hero**'s level.
- To limit how many of a specific class are allowed in a model, use **scope**.
- To compare subclasses in a constraint, a small workaround is necessary. You can add a string attribute representing an identifier for the subclass and assign it to every instance. The starting point shows an example for this.
- 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 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.