Project report: *Modelling language for simplified Object-Oriented target platforms*

INTRODUCTION

We have been proposed the modelling of a language for simplified object-oriented platforms. Faced with this problem, we have to separate two concepts within a programming language:

- The structural level
- The behavioural level

So, our objective will be, on the one hand, to design the models of these areas so that, on the other hand, we can develop a *model2text* transformation (M2T) that will allow us to generate the code corresponding to the union of both models that corresponds to the objective functionality of the problem.

METAMODELS DESCRIPTIONS

As mentioned previously, we have two areas of study within the proposed problem:

Structural model

The structural model, as its name suggests, refers to the structural aspects of any program by referring to classes, variables, methods...
Attributes that every program must have in order to develop its functionality correctly. The "skeleton" of the programme would be defined by this model.

Behavioural model

We would be talking about the behaviour that any object-oriented language would have. We would have in this model all those functionalities which a programmer manages to give shape to the program. Finding elements like *while* or *for* loops, *if* conditions, mathematical operations, logic, function calls...
That is, everything that makes up the code itself. The body of every program.

DESCRIPTION OF THE TRANSFORMATION

We decided to create only two models and directly transform each one of them into text to later join them in the same file, but without generating an intermediate model, that is to say, without carrying out an M2M transformation in the middle of the process. We did, but we

needed to insert 2 inputs to generate the code, which Acceleo does not allow, so we had to do an M2M conversion and then do an M2T transformation.

To generate the code of each model, each *.mtl* file will have as input the corresponding model so that, from this file, the attributes of each class can be accessed and the code can be formed correctly according to the "Java standards".

For each class, its global variables and then its methods are read, forming them, as has been said, respecting the standards.

- To set the name of the output file, it is determined by the name of the class containing the method designated as the program entry point.
- For global variables, the access mode, its type and name are considered
- For the methods, the body of the methods is first considered, that is, the accessibility of the method, what they return, the name of the method and the parameters (with the type and name of each one, separated by commas) obtained by iterating.
    - Later, as we already know, in OOP languages there can only be operational code within a function, so that within each function the operations available are analysed so that they can be transformed into text. Such operations may include:
        - Assignments
        - Matematical operations
        - Logical operations

CASE STUDY

The case study consists of designing a modelling language for object-oriented platforms. The problem is addressed, as already mentioned in the introduction, with the separation of the problem into 2 parts. The structural part and the behavioural part, so 2 metamodels will be defined. One metamodel will define the structure of the program and the other will define the behaviour of the program. The structural model contains classes, their attributes and methods. The behavioural model consists of assignments, operations, conditions, loops... (We can see how were designed in the Images 1 and 2)

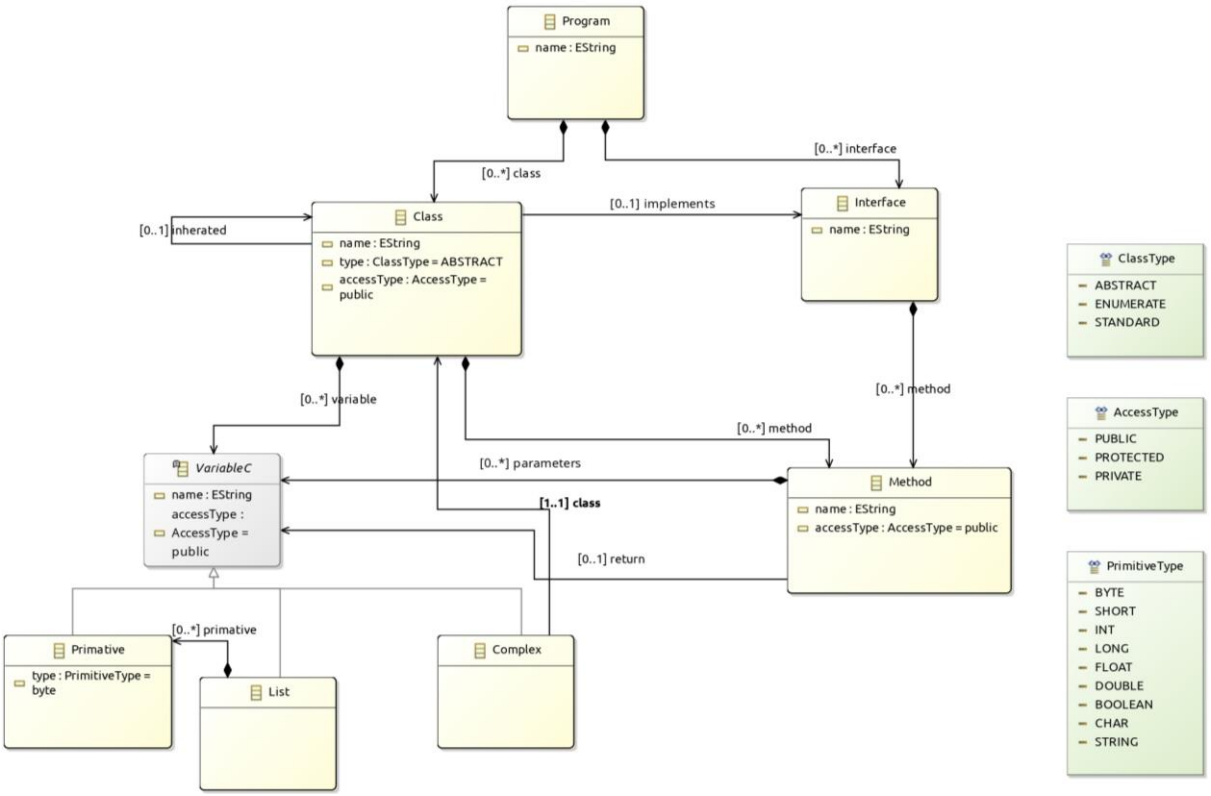

Image 1: Structural metamodel

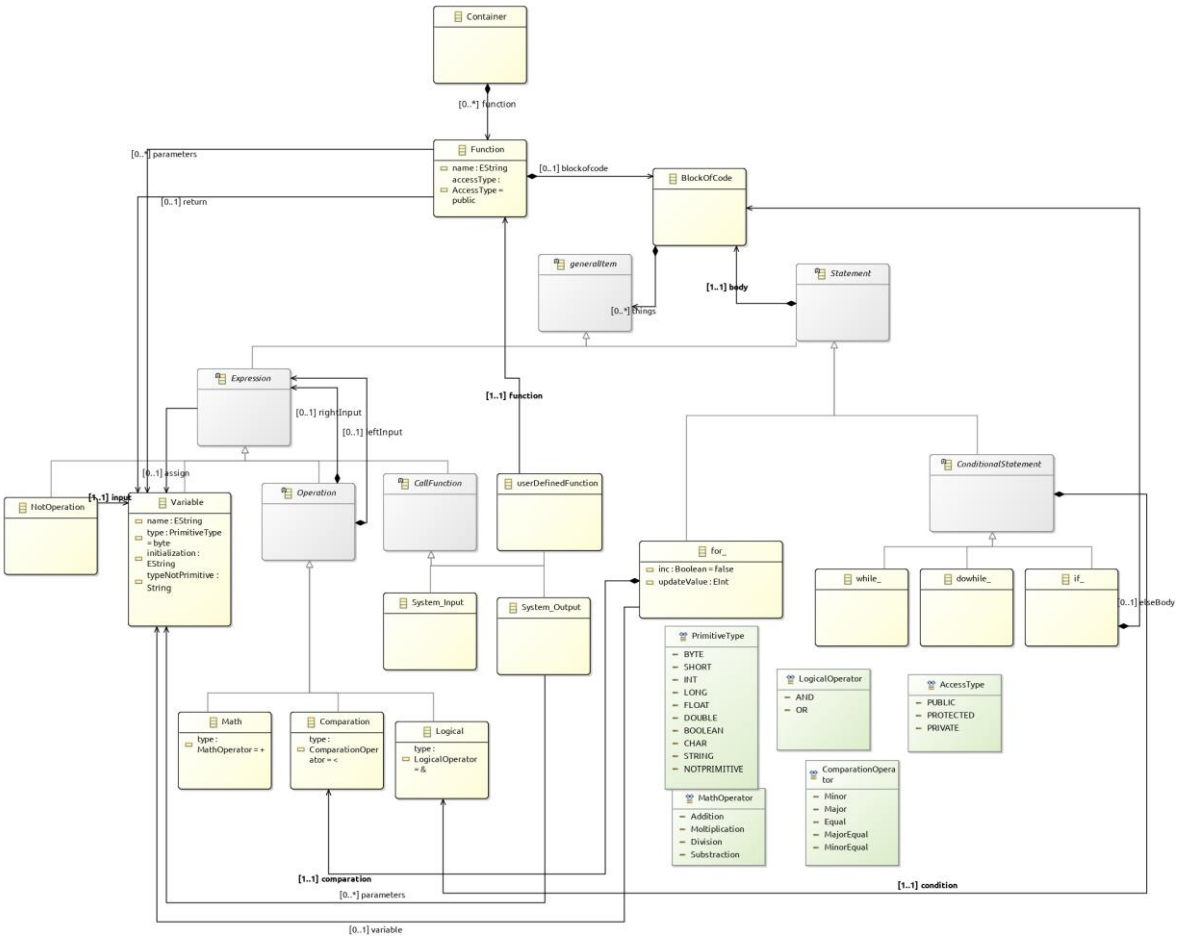

Image 2: Behavioural metamodel

An M2M transformation will be performed with these models in order to later perform the M2T transformation (as we can see on Image 3), since the tool we use (*Acceleo*) does not allow us to insert 2 models as input.

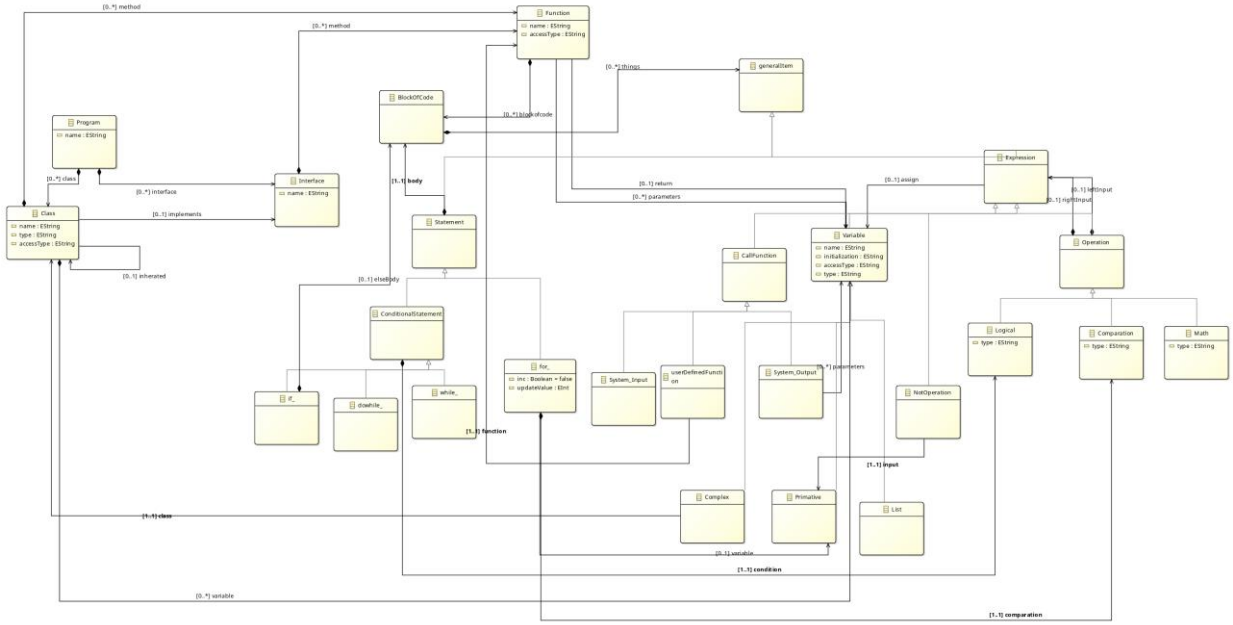

Image 3: Model To Model (M2M) transformation

Finally, a model-to-text transformation is defined using *Acceleo* that takes the models described as input and generates the code in an object-oriented programming language such as Java.

HOW TO RUN THE PROJECT

1. Run the project plugin project as *eclipse application*.

2. Fill in necessary values in the wizard such as instance of structural and behavioural model, destination folder onto which the file has to be saved. (Image 4 shows the configuration wizard of the M2M transformation and the Image 5 shows the configuration wizard of the M2T transformation)

3. Generated code files should be in the chosen output directory.

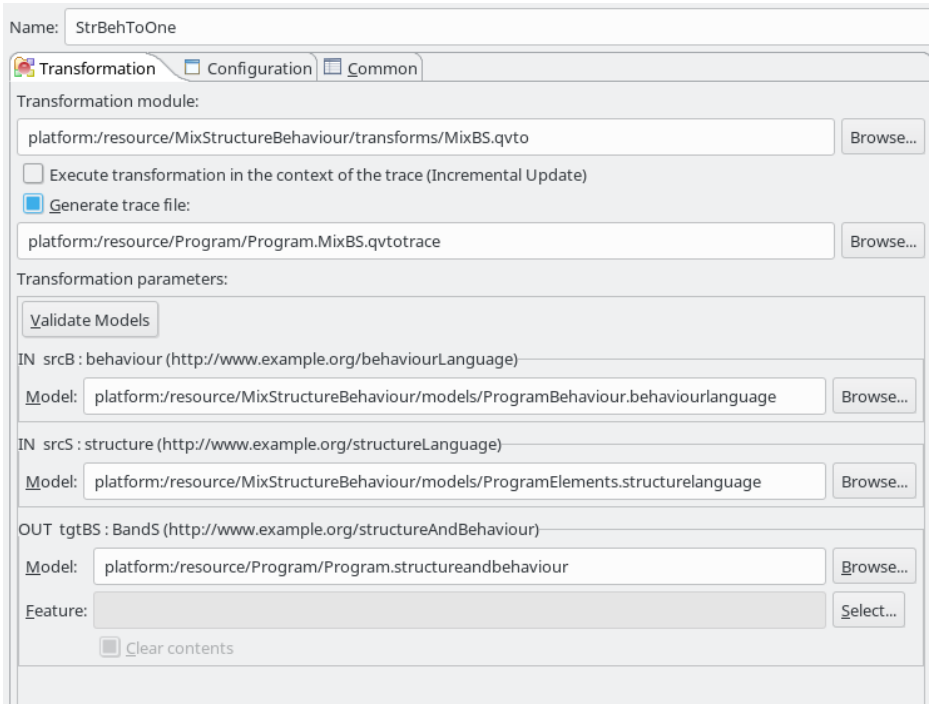

Image 4: M2M transformation configuration

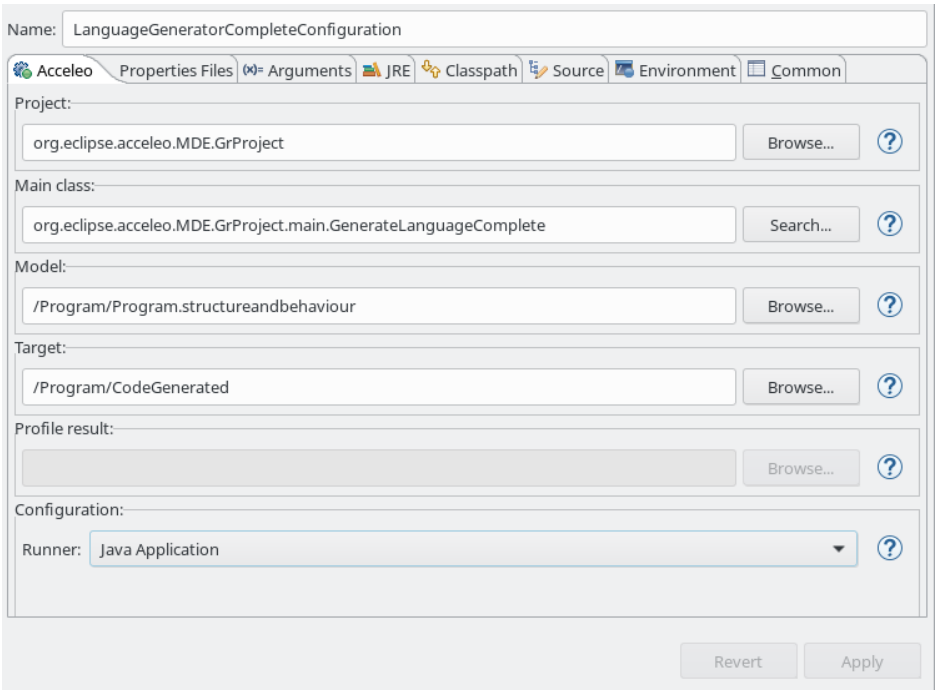

Image 5: M2T transformation configuration

LIMITATIONS AND POSSIBLE FUTURE EXTENSIONS

Limitations

The program encounters several limitations such as the absence of the possibility to handle exceptions as well as to generate created exceptions.
Nor do we find the possibility of adding libraries that give us additional functionalities.
Types are not considered in operations (this is trusted to the user), i.e. you can perform an operation with different types without the appearance of an error message.
We also find the impossibility to work with individual elements of a list, if you want to perform an operation in relation to a list created by the user, you must perform this operation to the whole list.

Future Extensions

The first long-term objective would be to address the constraints described above:

- Creation of exceptions and the possibility to handle them
- Possibility of adding libraries
- Detect possible errors in operations checking the types of the operators
- Access to the individuals from a list in order to operate with them separately

Other objectives would be the self-documentation of the code and support of type casting. As well as dedicated support for other programming languages.