|  |  |
| --- | --- |
| School of Innovation, Design and Engineering |  |
| DVA436 – Model-Driven Engineering  25 May 2016 |
|  |  |
| Jean Malm (jmm11001@student.mdh.se)  Andreas Granholm (agm11002@student.mdh.se)  Petr Fejfar (pfr16001@student.mdh.se)  Sharvathul Hasan (san15014@student.mdh.se) |

**Project Assignment: Model-to-Text transformation**

**Table of Contents**

[**Introduction**](#h.ucqpdppa1hcx)

[**Meta-models description**](#h.l59hr8m84xfn)

[**Structural model**](#h.j2kbgfvjkq9b)

[**Behavioural model**](#h.bz4shj5om90e)

[**Transformation description**](#h.hdx9s19d2sc2)

[**Case Study**](#h.k4aj0bcuqdlv)

[**Instruction Manual**](#h.bygx5dzf73rq)

[**Limitations**](#h.g08zvkiz5vk4)

[**Future Improvements**](#h.usa9h6w8wn5c)

[**References**](#h.1pxo4p4dkaye)

# Introduction

A model is an abstract representation of knowledge and activities that govern a particular domain. Meta-models are used to formalize and capture concepts and relationships among them in a domain. A meta-model usually define the abstract syntax and static semantics of a modelling language. Model transformations are the manipulation of models which defines a mapping between source and target models.

In this report, we describe meta-models for defining an object-oriented programming language as well as a model-to-code transformation based on those models.

# Meta-models description

The definition of the object-oriented program was divided into two different meta-models, in order to separate the structure of the program from its behaviour.

## Structural model

The structural meta-model defines the structure of the program in terms of its classes and their data.

Figure 1 shows the meta-model for defining the structure of the program. It is defined as a set of classes with their corresponding methods and variables. One method in the structure is also designated as the entry point of the program execution, i.e. its main function.

The structural model also contains a set of valid data types that our language supports, as well as valid access modifiers.

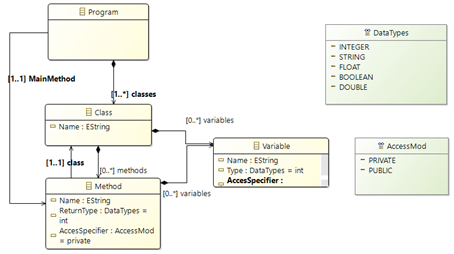


Figure 1. Structural meta-model for object-oriented programs

## Behavioural model

The behaviour of a program is defined as a set of function bodies that are then mapped to existing class methods. Our metamodel supports basic functionality in the form of a set of statements:

* Variable instantiation
* Assignment
* Loop (while)
* Branch (If-else)
* Function call
* Return
* Exception handling (try-catch, throw)

Aside from Exceptions, all current statements require some sort of expression to work. Expressions can be everything from constant values to variables or even function calls. The full list of supported expressions can be seen in Figure 2.

In addition to the standard statements found in normal OOP languages, to make modelling easier some standardized functionality nodes were added to the model. In our case we have user-IO in the form of readline and writeline nodes, saving some time in modelling IO-heavy applications.

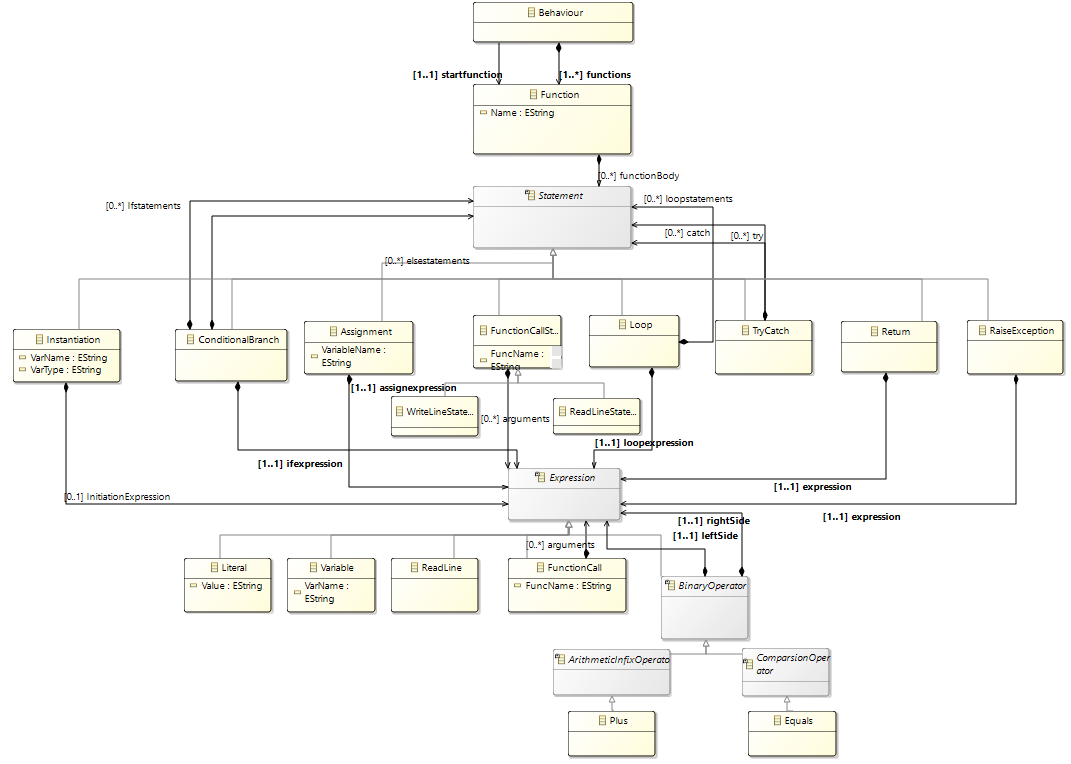


Figure 2. Behaviour meta-model for the program.

# Transformation description

As the behaviour and structural models were designed to be combined from the start, the decision to skip the intermediate model-to-model transformation was taken, as this would essentially only have been adding the function bodies to its corresponding class method.

During the code generation the structural model is iterated in order to generate each class. For each class, its variables are generated by reading the different attributes of each variable, i.e. name and type.

The methods are generated by first taking the method signature, i.e. access specification, returntype, name and input variables. For the body, the corresponding function body is found in the behaviour model instance, by doing a name comparison (for instance, the method Foo in class A would look for a function with the name A.Foo).

Finally, the name of the output file is determined by the name of the class containing the method designated as the program entrypoint.

# Case Study

The case study involved in the project assignment is to design a modelling language for object-oriented platforms. The language addresses the principle of separation of concerns by two different meta-models. One meta-model is to define the structure of the program and the other one is to define the behaviour of the program. The structural model contains classes, their attributes and methods. The behaviour model consists of assignments, conditions, iterations,etc. Finally, a model-to-text transformation is defined using Xtend which takes structural and behavioural model instances as input and generates the code in an object-oriented programming language.

# Instruction Manual

1. Run *CodeGenRunner* plugin project as *eclipse application*.
2. Press the *Generate Code* button on the toolbar.
3. 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. Choose target language and press finish.
4. Generated code files should be in the chosen output directory.

# Limitations

Currently, the structural meta-model does not support class hierarchies, which is necessary to be able to express the necessary concept of inheritance.

Furthermore, the modeller has no ability to specify required dependencies that the program might have, for instance in the form of including standard library files.

Additionally, the behaviour model is somewhat limited in terms of the amount of supported datatypes and operators standard to many programming languages.

Finally, there is no real validity checking during the transformation, so the code is generated exactly as it has been specified in the model. As the modeller is still rather free to do what he/she wants, there is a risk of faulty code caused by user error during the behaviour modelling.

# Future Improvements

The model and its transformations should be improved in order to overcome the above limitations:

* Structural meta-model can be improved in such a way that it should support class hierarchies.
* The modeller should have the ability to include standard library files that is required for the program.
* Models can be extended to support wide variety of data types and operators present in other programming languages.
* Some validation checking for the correctness of the model has to be performed during model transformations.

The model can be extended so that it can include other functionalities such as casting of data types, etc.

The model currently supports Java and C++. In order to enhance the usefulness of the meta-models, support for other programming languages should be added as well.

# References

https://eclipse.org/xtend/documentation/index.html