The Bmod DSL in MPS by JetBrains

Mathias Ooms

University of Antwerp, Middelheimlaan 1, Antwerp

mathias.ooms@student.uantwerpen.be[1,]

Abstract

MPS (MetaProgrammingSystem) is a tool developed by JetBrains, which allow users to design their own domain-specific languages (DSL). This environment gives the possibility to define and create a suitable workbench and IDE (Integrated Development Environment) for a DSL. In this project, I've implemented a DSL called "Bmod", an evacuation simulation tool of building floors, to show the abilities of MPS by JetBrains.

Keywords: DSL, MPS, JetBrains, Bmod, projectional editing, MDE

1. Introduction

Context. The DSL "Bmod" is a modeling language that allows users to easily instantiate a building floor that consists of rooms, cells, occupants,... that can simulate the movement of people in case of an emergency. To create this DSL, it's important to understand how we model a DSL in MPS in the first place. It's necessary to familiarize yourself with the structure that MPS provides, to understand the working and implementation of "Bmod" in MPS.

Overview. Section 2 contains some references to related work and useful sources. Section 3 explains the working of MPS. Section 4 gives additional information on how "Bmod" is implemented. Section 5 concludes and compares MPS to other MDE (Model Driven Engineering) tools.

Email address: `mathias.ooms@student.uantwerpen.be` (Mathias Ooms)

2. Related work

For the DSL "Bmod" I like to refer to the assignment given by Clàudio Gomes [1]. This assignment contains the functionality provided in "Bmod". Concerning the working of MPS, there are some very helpful tutorial video's online [2] (playlist), which explains every aspect and illustrates the abilities of MPS. As a hint: some video's are outdated use the textual variant as well! An example of a good first hands-on project: [3], these are more up-to-date.

3. The workings of MPS

Programming in MPS demands a certain file structure. At the root, we notice two main directories: a solution (sandbox) directory and a language directory. The language directory contains the complete definition of your DSL, whereas the solution is a directory that contains instances (models) of this DSL.

The language directory is build up by six main components:

Structure. Here the different kind of Concepts (conceptually a class) are defined. They specify which properties, children and references a Concept may have. These Concepts can be initiated by the user in their models. The entire structure can be seen as the metamodel (abstract syntax).

Editor. Editors allows the designer to create user interfaces to edit their concepts (concrete syntax). MPS can offer this by making use of a projectional editor. This editor allows the user to edit the AST representation of code directly, while the user sees plain text and edits this, behind the scenes they actually edit the AST itself.

Constraints. By making use of Constraints, the designer can limit values of properties, relationships between Concepts. These are extension of rules the structure needs to obligate.

Behavior. A way to add functionality to Concepts, by implementing methods (like classes) that can be invoked.

Typesystem. A mechanism to check type-system rules. These rules are evaluated on-the-fly and will report errors in the model instance (much like a good IDE would do).

Generator. A DSL gets ultimately translated into code. The rules for translating concepts is defined in the Generator.

Additional. MPS is a very advanced MDE tool and has much more into it, but this being a first hands-on experience, these are the essentials that are necessary to start.

4. Bmod in MPS

I've implemented Bmod as a textual-language rather than a visual-language which would've been easier to conceive for the average user. But as this is a first experience with the tool and the aim is to understand the mechanisms of MPS rather than actually using Bmod, this choice seems appropriate.

Structure. To define Bmod in MPS, we start off by defining the structure. To model a building floor, it's convenient to create the Concept Floor. This is our main concept and is declared in MPS as Root concept (as this node will have no parent in the AST). To mimic an alarm in this floor, we simply add a property "alarm" as a boolean. Floor consists of rooms, which is a concept of its own, so we must define this as one of it's children (not a property) with the appropriate cardinality. When we want to refer to other concepts we declare these under references (like the relationship "neighbors" between Cells). I continue this logical process for: Rooms, Cells, Occupants,... and add two enumerations "perception" and "action" to easily set the profiles in an Occupant.

Editors. To easily create a model instance, preserve a certain order and add some visual change to get a easy-to-use DSL, we use editors. When we don't use editors, you would be able to create a valid model, but there wouldn't be any order, very difficult to use, display of variables that may not be important at all,... That's why we use them, this way the simulation comes to life.
A good example of this is the Occupant-editor, where there is a lot of information necessary, but we don't want to see it per se all the time. That's where inspected layouts come in handy, here you declare which variables of the concept you want to see when opening the inspected view. This way only information is displayed when you want it and thus keeping a clear view of the simulation.
I've even used some text-background colors to change when the condition alters. Also here I've applied this same process for the other concepts.

Behavior. Here are all the operational semantics defined respectively to their concepts. These are just Java implemented methods, defining the asked behavior of Bmod in [1]. It's sufficient to mention that the function "simulate(iterations)" of the concept Floor, starts the simulation with a given amount of steps.

Constraints. When a user needs to model a room, he needs to define all the relations between cells. So normally when a cell 1 is the west relation of cell 2, the user also needs to define the "logical" east relation. Fortunately we can define constraints for this being: every time a new value (a cell) gets assigned to the direction-relation, this new cell its opposite direction-relation is immediately assigned. This creates a very easy way to create relations between cells.

Typesystem. A good problem to solve with a checking rule is the fact that multiple occupants can't be allowed in one cell except if that cell is outside. By defining this checking rule multiple occupants on one cell (that's not outside) will be marked indicating that this is not allowed (like an IDE would do).

Generator. I've made an attempt to visualize Bmod with actual GUI, but after I couldn't even reproduce the basic shapes out of a tutorial [3], I left it out as the main aspects where pointed out. The most important part to remember is that generators can map your DSL to actual code.

Solution. As last I've made a small example of a possible model called "Middelheim". When calling the Floor it's simulate method, you can actually see the occupants moving, the fire spreading,...

5. Conclusions and comparisons

To conclude I think it's safe to say that there aren't actually any limitations on what's possible in MPS, but I honestly find it having quite a steep learning curve, especially if you want to create a visual language. This is than again a very strong aspect of AToMPM, which makes as much use of the visual aspect of designers and so being very intuitive. Both these tools can handle big projects with lot's of complexity, which is than again a weak point for MetaDepth. MetaDepth is limited in options (which takes away complexity) and can serve as a good starting point for learning to design DSL's, but is very unstable in its use.
The ultimate conclusion that I take, MPS is an advanced tool that has no real boundaries but it's more adequate for the advanced designers.

References

[1] C. Gomes, Buiding evacuation modelling in metadepth, MDE (2019) 1–5. URL http://msdl.cs.mcgill.ca/people/hv/teaching/MSBDesign/assignments/Assign1-metaDepth.pdf

[2] JetBrains, Mps, JetBrainsTV. URL https://www.youtube.com/watch?v=1yQ5kj6svRM&list=PLQ176FUIyIUYWfVn1BulLIJXILHUI8t9_

[3] JetBrains, Shapes - an introductory mps tutorial, JetBrains. URL https://confluence.jetbrains.com/display/MPSD34/Shapes+-+an+introductory+MPS+tutorial