Assignment 2
Building modelling in AToMPM

Cláudio Gomes
claudio.gomes@uantwerp.be

November 21, 2019

1 Practical Information

The goal of this assignment is to repeat the modelling task of the previous assignment, in the AToMPM visual metamodelling environment (see below for installation instructions).

The assignment structure is the same as the previous assignment, and the advice with respect to tooling applies to this assignment as well.

1.1 Task Overview

Task 1 *Implement the abstract syntax of Bmod in AToMPM using the* `/Formalisms/__LanguageSyntax__/SimpleClassDiagram` *formalism.*

Task 2 *Describe the concrete visual syntax of Bmod.*

Task 3 *Enhance the usability of Bmod by adding actions to automatically move/snap/resize model elements.*

Task 4 *Use Bmod to model a non trivial building floor of your choice, and include at least one dangerous situation. The requirements for this task are the same as the ones for the corresponding task in the first assignment.*

Task 5 *The requirements for this task are the same as the ones for the corresponding task in the first assignment.*

1.2 Deadline, Logistics, and Plagiarism

Complete this assignment in **groups of 2**.

One, and only one, person in the group must submit the solution on blackboard before the deadline announced on the course web page: `http://msdl.cs.mcgill.ca/people/hv/teaching/MSBDesign/`.

Discussion on the following topics with your classmates *is encouraged*:

- Interpretation of the assignment requirements;
- Technical difficulties with the tools;

Discussion on the main concepts of the domain that may lead to similar solutions is *discouraged*. Creating a domain model is a highly creative process, and therefore it is unlikely that two groups will have a similar solution.

Contact Cláudio Gomes (claudio.gomes@uantwerp.be) if you have questions and/or need some help.

2 Requirements

2.1 Task 1

The requirements for this task are the same as in assignment 1, except the tool used is AToMPM.

The formalism should be called Bmod. The abstract syntax model file should be called BmodMM.model. The metamodel (a.k.a. compiled abstract syntax) should be called Bmod.metamodel.

At the end of this task, the following folder should exist in atompm installation directory:

`atompm/users/yourName/Formalisms/Bmod/`

And it should include every file that you used to solve this assignment.

2.2 Task 2

A suitable concrete syntax should clearly show the relevant information to anyone reading a Bmod plan model. This includes at least:

1. The name of each room;
2. Each person's perception and action profile, and in which room the person is in;

The concrete syntax file should be called

`Bmod.defaultIcons.model`

And the compiled concrete syntax should be called

`Bmod.defaultIcons.metamodel`

2.3 Task 3

Enhance the usability of Bmod by:

1. adding an action that automatically positions a person roughly in the middle of cell, when it is placed in the cell.
2. adding an action that snaps cells together when these are connected (see existing Pacman example on how to do this).
3. adding an action that color codes a person according to its state (alive, detected fire, dead, etc. . .)

2.4 Task 4

Pick a non trivial floor of the current building, and model it, modelling at least one dangerous condition.

2.5 Task 5

The same requirements as Task 5 in assignment 1 apply.

Additionally:

1. The metamodel should be included;
2. Which actions where implemented (you don't have to include the implementation of each action);
3. The floor modelled in Task 4, and an explanation of the dangerous condition(s) that you included (what is the condition, and why did you include it).

3 Tools and Documentation

The recommended and simplest way to run AToMPM is to use the docker Virtual Machine (VM) image made available online: `atompmvm.tar.gz`

Instructions on how to use the image to create a virtual machine are provided in the `README.md` file, available online: `README.md`

Your assignment will be graded on a VM created from the same image.

- AToMPM git repository (do not use the release, as it is outdated): `https://msdl.uantwerpen.be/git/simon/AToMPM`.
- AToMPM documentation, accessible also from the UI of AToMPM, is in: `https://msdl.uantwerpen.be/documentation/AToMPM/index.html`

4 Tips, Tricks, Pitfalls, and Issues

Below is a list of known obstacles, and advice, while working with AToMPM:

Advice. The installation of AToMPM in windows requires specific versions of python, igraph, and visual studio. Make sure the installation will point to the correct versions.

Advice. There are plenty of examples shipped with AToMPM. Of particular relevance is the PacMan language, which includes action code for snapping elements together, and other features that you may need.

Advice. While working in AToMPM, keep an eye in the browser console, as warnings are shown there. For example, when an element is dropped inside some other element, but no containment relationship was created, a warning is issued. This is important since sometimes the containment relationships fail to be created.

Advice. For containment relationships, it is advised that these are not invisible links, but instead barely visible ones. Containment relationships between an element A and an element B are only created if the element A is moved into element B. If element A is pasted or created directly on top of element B, the relationship is not created, resulting in hard to debug problems. The containment links are a visual cue that the containment relationship was created.

Bug. After performing a window switch with ALT+TAB, if a right arrow key is pressed while editing a text area, a tab is inserted, instead of just moving the arrow to the right. This is indented to facilitate indentation of code.

Advice. Backup the formalisms folder, or keep it under source control. This is because AToMPM can corrupt the model files if it crashes.

Advice. Keep in mind that any change to the abstract or concrete syntax automatically invalidates any Bmod model that you have made. Therefore, make sure you made very small models to test the abstract and concrete syntax, before solving Task 4.

Advice. A possible workflow while creating a language in AToMPM is as follows:

1. Create a single abstract syntax element and its attributes.
2. Save and compile the abstract syntax.
3. Add the icon or link description of the newly created element to the concrete syntax.
4. Save and compile the concrete syntax.
5. Test the concrete syntax by creating a simple model with the newly created element.
6. Backup, and go to step 1.