Assignment 1
Building evacuation modelling in metaDepth

Cláudio Gomes
claudio.gomes@uantwerp.be

November 21, 2019

1 Practical Information

The goal of this assignment is to design a domain-specific modelling language (formalism) for representing the movement of people in a building floor, in case of an emergency.

The language is called *Bmod*.

This section describes the tasks that you need to complete, and the next section gives the requirements about each individual task.

The solution to each tasks depends on the way you solved the other tasks. For example, Task 4 (below) is strongly related to Task 1. So *read carefully the whole document* and make a solution sketch of each task, before implementing the solution.

The tasks are purposefully ambiguous. Use common sense or ask Cláudio if you are unsure about what a task means.

Tooling. The tools used in this assignment are research prototypes. Therefore, *expect bugs and usability issues*. Feel free to report them to Cláudio, or to the tool's developers. Make sure to solve each task incrementally, and either make regular backups of the input files, or use version control software. These tools can crash and corrupt the input files.

Section 4 describes known bugs and general advice.

1.1 Task Overview

Task 1 *Describe the abstract syntax of the Bmod language in the metaDepth tool.*

Task 2 *Describe validity constraints that any model must obey in order to conform to Bmod.*

Task 3 *Describe the operational semantics of Bmod.*

Task 4 *Use Bmod to model a building floor and emergency routes.*

Task 5 *Write a report.*

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 Bmod language is to be used by anyone (that is, non computer experts) that wants: a) to design a new floor evacuation plan, or b) to study an existing plan. To accomplish this, it should be possible to use Bmod to model:

1. floor plans (rooms, doors),
2. emergency signs (fluorescent arrows indicating the path to an emergency exit),
3. the cause(s) of the emergency (a fire),
4. the behavior of the occupants, and
5. dangerous conditions that should not happen during the evacuation (e.g., a room becoming too crowded).

2.1.1 Floor plans

To model the floor plans, it must be possible to represent rooms and door, and how they are connected. Furthermore, each room is comprised of multiple cells. Each person can be in one cell at a time, and move to an adjacent cell. No two occupants can be in the same cell.

2.1.2 Emergency Signs

To model the emergency signs, it is enough that each sign represents a directional relationship between two doors (indicating the next door after one door is crossed). There can be bifurcations in the emergency path.

2.1.3 Cause of Emergency

To model the cause of the emergency, it is sufficient to represent in which cell the cause is. The only cause of emergency we consider is a fire.

2.1.4 Behavior of Occupants

The behavior of each occupant has two phases: perception and action. The perception models how the occupant interprets the cues in its environment to conclude that there is a fire. The action models what the occupant does after the perception phase.

For the perception, the following list shows the possible perception levels (see Task 3 for their meaning):

- Nervous;
- Listener;
- Smeller;
- Observer.

For the action, the occupant can have one of the following profiles (see Task 3 for their meaning):

- Newcomer;
- Experienced;
- Your own action profile (which you will have to implement in Task 3).

Each occupant should therefore be configured with a pair (*Perception*, *Action*), where *Perception* and *Action* take one of the above values.

2.1.5 Dangerous Conditions

During the simulation of the building evacuation, it is important that the user is warned whenever specific conditions happen. The user of Bmod should be able to model generic conditions regarding room occupancy:

- A specific room has an occupancy greater than a given number.

The user can create as many conditions as he/she wishes.

2.1.6 Other requirements

Your metamodel should include the appropriate multiplicities for each association, and at least one example of inheritance.

2.2 Task 2

Since Bmod is made to model a single building floor, you must make sure that any valid Bmod model obeys the following constraints:

1. It should be possible for a person to move from any room to any other room;
2. Your first own constraint;
3. Your second own constraint.

To showcase your properties, include the following (simple) models, for each constraint:

- a model violating that, and only that, constraint; and
- a model satisfying that, and all the other, constraints.

These models do not have to be realistic.

Tooling. To implement this task, use the validation functionalities provided by metaDepth. Since the tool is a prototype, there can be bugs in the validation. If you suspect that a bug is causing your validation code to fail (or crash), take note of the validation code, remove it, and implement the validation in a `checkValidity.eol` script, that throws an exception if the model is not valid. Finally, mention this incident in the report, and include the validation code that did not work.

2.3 Task 3

Bmod users will run simulations with the intent is studying multiple evacuation scenarios.

In this task, you must implement the operational semantics of a given Bmod model, in a `operationalSemantics.eol` script.

This script will simulate the movements of people, and fire, within the building floor. At each simulation step, the movements of the people, and fire, will be computed. A person can only move once, as well as the fire. Once every movement has been performed, the dangerous conditions are checked, warnings are issued, and the simulation proceeds to the next step, where everything is repeated again.

The computation of the occupants' movement is done before that of the fire. That is, only when every occupant has had its chance to move, will the fire be propagated.

The output produced should be both readable by a person, and easy to parse (because that will be useful in Assignment 5).

2.3.1 Occupant Movement

There are two phases to compute a person's movement:

1. Decide to which cell to move;
2. Move to that cell.

The first phase is dictated by the perception and action profiles of the person. The perception controls when a person concludes that there is a fire. The profile controls which cell the person will move to.

The perception levels, and their meaning, are:

- Nervous — If the fire alarm rings, he/she concludes that there is a fire.
- Listener — Only when a nearby person says that there is a fire, will he/she conclude that there is fire.
- Smeller — Only when he/she smells smoke, will he/she conclude that there is fire.
- Observer — Only when he/she sees fire, will he/she conclude that there is fire.

The fire alarm rings everywhere in the building floor, when there is a fire. The moment a person concludes that there is a fire, he/she will always tell any person in the same room that there is a fire. A person smells smoke when there is a fire in the same room as the person. A person sees fire when it is in the adjacent cell.

After a person concludes that there is a fire, he/she will move according to his/her action profile:

- Newcomer — Picks the adjacent cell that brings her/him closer to the nearest person in the room, or chooses no cell;
- Experienced — Picks the adjacent cell that brings her/him closer to the emergency exit pointed out by the emergency signs. If there is a bifurcation, he/she picks one emergency path and sticks to it.
- Your own action profile — Moves to an adjacent cell according to your implementation.

The adjacent cells to a person are in four possible directions (North, South, East, West).

After deciding which adjacent cell to move to, the person will move to it if there is no obstruction (either a fire, or a person in that cell). If there is an obstruction, the person will try to move to a free cell that is adjacent to the obstruction, without moving over it.

For example, if a person has chosen to move North, but the North cell is obstructed, then the person will try to move to either to the East or West cells of the obstruction. In this example, the East and West cells of the obstruction are the only cells available to move to.

Another example: if a person has chosen to move East and that cell is obstructed, then the person will try the cells that are North and South of the obstruction.

If the person did not choose any cell to move to (e.g., when he/she decides that there is no fire), or there are no cells available to move to, then the person does not move.

2.3.2 Fire Movement

The fire propagates to every adjacent cell where there is no fire. Once in a cell, the fire remains in that cell until the end of the simulation. If the fire propagates to a cell where a person is, that person dies and cannot move again.

2.4 Task 4

Pick a non trivial floor of the current building, and model it, modelling at least one dangerous condition. Perform multiple experiments, each showing one of the following scenarios:

- The dangerous conditions is satisfied at least once; and
- A contention point (i.e., a point where people accumulate for long periods of time).

A non trivial floor should include at least two emergency exits.

2.5 Task 5

The grading process depends on a well written, and concise, report.

A good report should:

1. Explain the general approach to each task, and in case of ambiguities, describe your decisions and rationale.

2. Explain the structure of your submission, and the purpose of each individual file.

3. Describe any issues encountered with the tools, and how they were overcome.

4. Give an estimate of how many hours in total the group spent to solve the assignment. This measure is not to evaluate the efficiency of the group, but rather to have an estimate of the relative difficulty of the assignment, and provide a fairer grade.

A script (bash, or powershell, or cmd, etc...) should be included that runs (or shows how to run) each of the simulations.

Optionally, a screencast showcasing the language and the simulations can be included.

3 Tools and Documentation

- MetaDepth main page, documentation, examples, and download: `http://metadepth.org/`

4 Tips, Tricks, Pitfalls, and Issues

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

Advice. Build your metamodel (and models) incrementally, checking whether they work at each point, and make use a version control system.

Advice. Do not use reserved words for variable/constraint/class names. Example reserved words are: `in`, `for`, `Node`, etc... In case you mistakenly used a reserved word, the error message that MetaDepth gives might not be helpful in tracing the error to the use of the reserved word!

Bug. Avoid having nested models. They are not parsed correctly by metaDepth.

Bug. Avoid using derived attributes. If you need derived attributes, these can be computed in the .eol script, using assignments of the form

```
element.~derivedAttr = expression;
```

Bug. Any instruction after the first operation declaration in EOL is ignored. So, code instructions first, and then operation declarations.

Bug. If a model element has attributes that do not correspond to the corresponding metamodel element, metaDepth will not throw an error message, and will most likely not parse the other attributes of the element correctly. For example, if the metamodel declares a person to have a name and an age attribute, and the model instantiates a person with `pname = John` and `age = 20`, it is likely that John's attributes will not be parsed.