BMOD Implementation Using Sirius

Sylvain Elias

University of Antwerp, Middelheimlaan 1, Antwerp

Sylvain.Elias@student.uantwerpen.be

Abstract

Sirius is a technology that allows users to create custom graphical modeling workbenches that are composed of a set of Eclipse editors (diagrams, tables and trees) therefore it allows the users to create, edit and visualize EMF models that are based on a structured data model. BMOD is a Domain Specific Language implemented using Sirius, it is used to simulate building evacuation during an emergency.

Keywords: Domain Specific Language, Model Driven engineering, Model, Metamodel

1. Introduction

Sirius is particularly adapted for users that have defined a DSL and need graphical representations to better elaborate and analyze a system. The Eclipse editors created with Sirius are described by a model, also known as metamodel, which defines the complete structure of the modeling workbench, its behavior and all the edition and navigation tools. This description of a Sirius modeling workbench is dynamically interpreted by a runtime within the Eclipse IDE.[1] In this report we'll discuss the implementation of BMOD using Sirius and the limitations of this tool. We'll start by defining BMOD in in 2 then we'll talk about its implementation using Sirius in 3, we'll compare it with other tools in 4 and then we'll conclude in 5

2. BMOD

BMOD should allow the user to design a floor that verifies the rules set by the metamodel, the user should be able to add rooms, divide them into

cells which can hold occupants, connect the rooms together using doors, place emergency exit signs, set the occupants' perception (when they get alarmed by the existence of an emergency) and action (how they react after being alarmed) and check if a dangerous condition occurs (a room has more occupants than a preset number). The emergency we'll be working on in this specific project is the existence of a fire in the floor. This DSL should be animated to show the user the propagation of the fire in the cells and how occupants react to it. The purpose of this language is to aid specialists in designing buildings that can be easily navigated during an emergency by the help of rightly placed emergency exit signs.

3. Implementation

3.1. Metamodel

Sirius uses Ecore Tools which is a graphical editor for Ecore models to create metamodels for the DSL. Classes are represented as a Class Classifier in the palette and can be easily created by dropping them in the provided diagram, adding attributes to them and linking them to each other using different relations. The main relation between different classes is the Composition relation which indicates that the class is composed of different classes and this is crucial for the visual syntax design explained in 3.2

- I started designing the BMOD metamodel by adding the *Floor* class which has no attributes, it is composed of *Room*, *Outside* and *Door* classes.
- *Outside* is an empty class used to represent the safety area where occupants are safe from the fire
- *Room* is a class that has a name and condition as attributes, condition is used by the user to set the dangerous condition that needs to be checked. This class is composed of 1 *EmergencySign* and 1 or many *Cell* classes.
- *Door* is an empty class that is connected to 2 *Target* classes
- *EmergencySign* is an empty class that is connected to 1 source *Door* and 1 or 2 target *Door*, the source is used to specify which door this emergency sign is linked to which is important especially since a room can have multiple doors

- *Cell* can be connected to 0 to 4 different *Cell* (left, right, top and bottom connections) and is composed of 0 or 1 *Content* class
- *Content* is an abstract class, it is used as a super type for both *Fire* and *Occupant* classes. This abstract class is used to make connections between the cell and its content easier
- *Fire* is an empty class, it's only purpose is to be contained in a cell to indicate that it's on fire
- *Occupant* has three attributes: isAlarmed to indicate if this occupant is currently alarmed or not, perception which is linked to an *perceptions* enumeration class that enumerates the list of perceptions available and action which is also linked to a *actions* enumeration class. I didn't add a isDead attribute since the cell can only have one content so we can't add fire to a cell that has a dead occupant
- *Target* is an abstract class that is the super type of both *Cell* and *Outside* class since they're both targets to *Door*

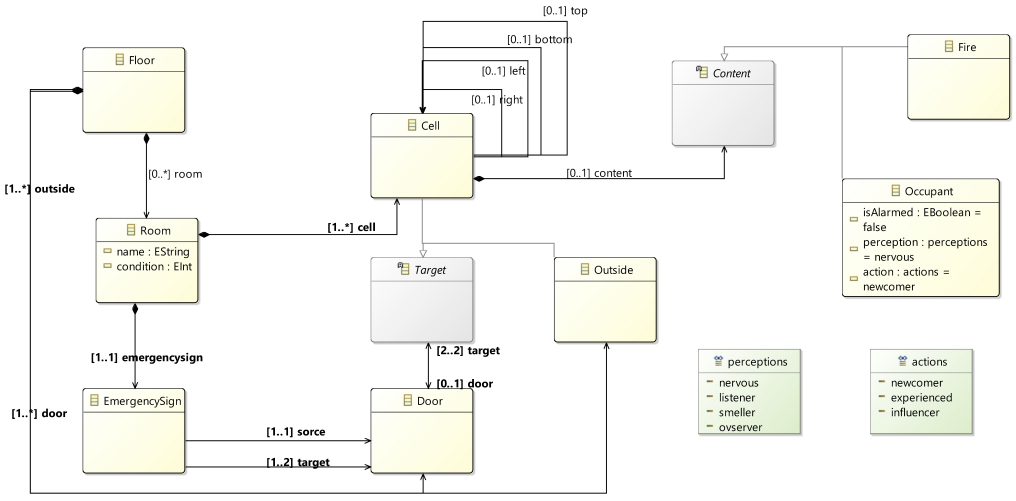

Figure 1: Image showing the final metamodel

3.2. Visual syntax

The first step in designing a visual syntax in sirius is setting a viewpoint; a viewpoint provides a set of representations (diagrams, tables or trees) that the end user will be able to instantiate. For BMOD a diagram is added as a representation for this viewpoint which has the *Floor* class as its domain class so we can interpret and view all the classes that it is composed off. This diagram is made of multiple nodes representing the classes in the metamodel and the edges connecting them. If a class doesn't have a node associated to it in a diagram it won't be represented in the visual syntax

1. Classes with no attributes or containment are simply represented by a node holding a workspace image like *Fire*, *Door* and *EmergencySign*.
2. *Occupant* is represented by a node containing a workspace image but it also has a label which displays the perception and action attribute next to the image
3. *Room* isn't visualized using a normal node since it is a class that contains other classes, therefore I used a container node which, not only holds a workspace image that also displays the label, but it also holds all the classes that it is composed of. The main class *Floor* can only contain nodes that it is directly composed of so adding another container node helps in displaying all the classes.
4. *Cell* is the same as *Room*, it is also a container node that holds *Fire* and *Occupant*
5. Edges also have to be represented in the model so I added relation based edges in the viewpoint to every edge available in the metamodel that should be displayed in the model and set their source and target mappings depending on each one of them.

In Sirius, adding the ability to edit the available model by adding new elements has to be done separately so I added node creation tools (or container creation tools for the *Cell* and *Room* classes) to the viewpoint which create new instances of the element created and adds them to the model; the same thing is done for edge creation but one of the features that sirius has is the ability to set different actions to happen when a node is created so I used this feature to make double connections easy between cells (when a cell 1 is connected to the left to cell 2, cell 2 is automatically connected to the right to cell 1). Normal nodes can be easily deleted in the visual syntax model but edges need to have a delete element tool added to the viewpoint so I created

them and when an edge gets deleted the nodes connected to them have to be noticed so an unset method is called to remove this selected edge from the class attached to it. I also added a double click function for the classes that have attributes to open a new dialog to allow for a better and easier editing of these attributes. Since the *Occupant* has two states: either they are alarmed or they aren't, I added a style customization which changes the image used to display the occupant so that their current state can be easily viewed by the user.

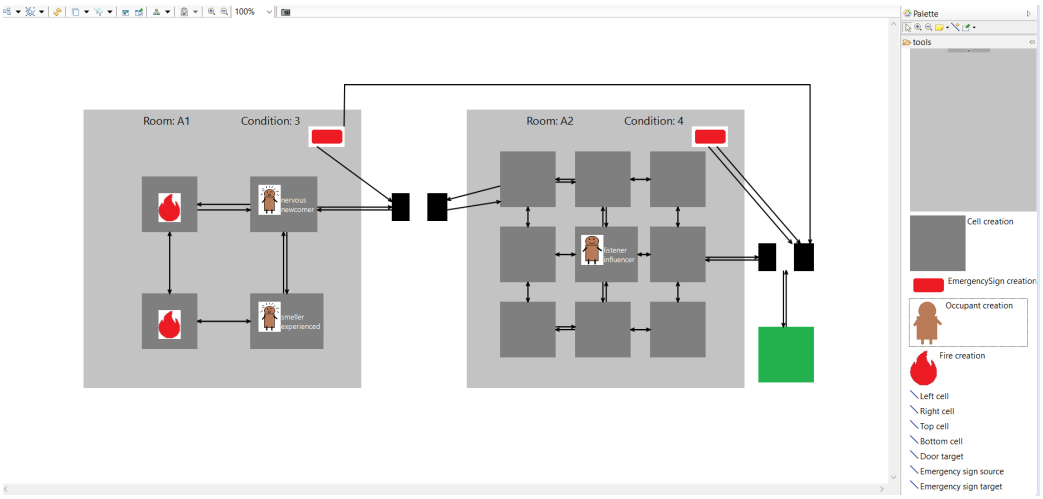

Figure 2: Image showing a sample model created using the visual syntax

3.3. Constraints validation

Sirius automatically shows when the metamodel constraints aren't met when validating the model created (e.g. when a room that is supposed to have 1 or more cells doesn't have any) but extra constraints have to be added so I added validation tools to the viewpoint to verify that the model has no errors that aren't mentioned in the metamodel (when all rooms aren't connected together, when one door is used to connect two cells of the same room instead of two cells from different rooms, or if one of the emergency sign target door doesn't have *Outside* between one of its targets). These constraints were implemented to call on java service functions which return a Boolean value to check if the constraint was validated or not. If not validated, Sirius shows a sign next the the element in the model with a message to make it clear for the user where the error came from and for what reason.

3.4. Operational semantics

Sirius as a tool by itself doesn't present a way to add operational semantics to a model so the fire propagation and occupant movements couldn't be modelled. It should be noted though that this functionality could be added by using external tools like GEMOC Studio which is an Eclipse product (on top of Sirius) but I didn't implement it since it is out of the scope of this project Gemoc [2].

4. Comparison with other tools

4.1. AToMPM

Sirius and AToMPM share the same metamodel creation technique but Sirius has the composition relation which bases the model creation on it while AToMPM doesn't need this complexity. When it comes to visual syntax AToMPM allows model visualization and model creation and editing by simply creating a visual syntax for the classes presented in the metamodel while in Sirius it is more complex since different presentations have to be created for different kinds of classes and edges and node/edge creation and deletion have to be set up manually one by one. But on the other side Sirius also adds the functionality to add actions when new elements are added to the model which isn't available in AToMPM. Also AToMPM makes it easy to snap elements together after being created in the model while Sirius has a feature called ELK which allows the user to edit the default arrange all function for the created model but it is still in its experimental phase and it caused my metamodel to crash when added to my project so I couldn't test out its functionality. AToMPM offers the user the option to directly create operational semantics in an easy and visual way while you have to use external tools to implement these functionalities in Sirius.

4.2. Metadepth

Sirius is completely different than Metadepth since it has a visual presentation for everything which makes it easier to create the metamodel and the model itself while Metadepth takes a different approach since it only has a textual syntax and everything should be coded instead of using drag and drop techniques and visual representation of the elements you're editing

5. Conclusion

In conclusion, Sirius is a really strong tool and it has an easy learning curve, and with the addition of extra tools that it supports it could be really easy to build big projects using it. In general Sirius was easier to use than AToMPM since it is more intuitive than the latter but both these tools were simpler and felt more natural to use than Metadepth since they both have visual representations of the DSL you're developing while having the same functionalities and even more.

References

[1] Eclipse, Sirius overview, `https://www.eclipse.org/sirius/overview.html`, 2019.

[2] Gemoc, Gemoc, `http://gemoc.org/`, 2019.