Constraints in "Traffic" formalism

Akos Nagy[*]

UA,2011

Abstract

A potential extension to "Traffic" formalism could be to meta-model classes and constraints and take them to translate to Alloy representation. It enables the analysis about models created in the "Traffic" language. In this paper a possible solution is presented in AToM^3. Explicit meta-model of a class, an attribute and a constraint is given and the relation to "Traffic" formalism is described. Then transformation of a segment of a vehicle traffic network to Alloy is detailed by Graph Grammar rules. The translation written in Python code is also shown.

Keywords: AToM^3, Alloy, constraints, transformation, Traffic formalism

[*] *Email address:* `AkosGergoe.Nagy@student.ua.ac.be` (Akos Nagy)

June 29, 2011

1. Introduction

A useful contribution to the Traffic formalism is to enable analysis about models (constructed in the Traffic formalism) in Alloy[1]. Traffic is introduced in (Vangheluwe and de Lara, 2004) and the syntax is meta-modelled in the E-R Diagrams formalism, but in this paper we focus on Traffic which is meta-modelled (Kühne, 2006) with the help of Class Diagrams. Thus one may write OCL constraints to the classes (Bottoni et al., 2000) of the meta-model of Traffic. However, we could explicitly model a constraint in the Traffic formalism itself when building a vehicle network model. For example: a "Sink" class has two attributes. Instead of writing constraints for them in OCL, we can generate Alloy code and reason about different properties with Alloy. As OCL is a declarative language on the top of UML classes, OCL could be replaced by Alloy. In this paper we present explicitly meta-modelled classes, attributes and constraints, graph grammar rules to transform the class, its attributes and the constraints to Alloy. All of the work is done in AToM^3, *A Tool for Multi-formalism and Meta-Modelling*[2]. Section 2 gives an overview of related work. Section 3 presents the design of our new architecture and gives details about transformation to Alloy. Section 4 shows how the problem is linked to the "Traffic" language. Section 5 describes an experiment about the usage of the given application. Section 6 compares the work shown in this paper to two related works. Section 7 concludes and gives ideas for future development.

2. Related work

(Vangheluwe and de Lara, 2004) introduced a formalism for Traffic and mapping it to Petri Nets. Syntax, semantics and semantic mapping is shown. "Traffic" abstract syntax is meta-modelled with the help of E-R Diagrams formalism. In the paper of (Shah et al., 2009) a solution is given to transform "From UML to Alloy and Back Again". They use OCL constraints together with UML classes. Their solution outlines the transformation from UML to Alloy on different layers of the MOF hierarchy and based on the origial UML2Alloy transformation. Similar and more detailed transformation is explained in (Anastasakis et al., 2008), also based on the use of

[1]http://alloy.mit.edu/community/
[2]http://atom3.cs.mcgill.ca/

UML2Alloy. They automatically translated their UML model to Alloy by using MOF2Text mapping. (Georg et al., 2001) details a comparison of UML/OCL and Alloy in their case-study after specifying a problem of a given distributed system.

3. Class and constraint

As a first step I model explicitly a class, class attributes and a constraint. They are simplified and have limited capacities. The "Constraint_" class serves as an abstract class. Three children classes represent building blocks to construct constraints. The parent class contains a string with the names of the attributes to be constrained. All the children classes contain options to select to construct a constraint chain. The responsibility of the "Class_" named class is to provide the attribute values that are connected to it. This class has now only 1 attribute to allow to name it. However, an attribute is meta-modelled explicitly by the class "Attribute_". An other class called "Alloy" can be found in the meta-model, it simply represents the generated Alloy code later on. Each entity has a simple graphical appearance as well. After using the "Gen" button the formalism is available to create a model. This work is done by AToM^3.

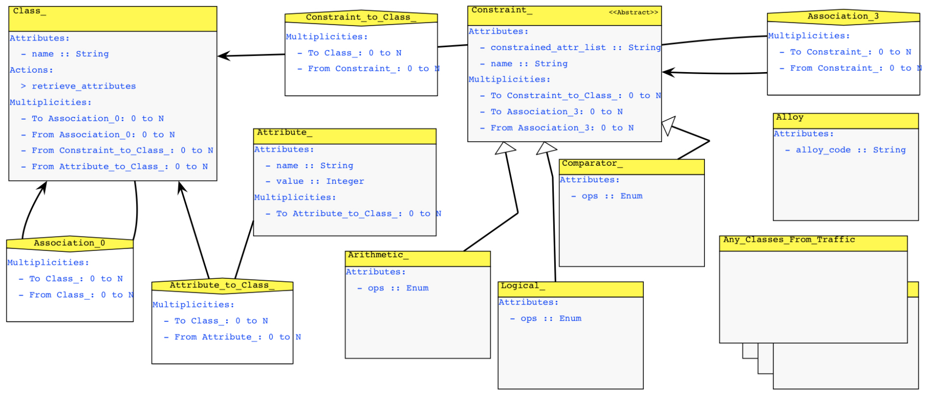

Figure 1: Constraint modelling combined with "Traffic"

A simple example model consists of an instance of a "Class_" class, instances of "Constraint_" classes to give the actual constraint, and an instance of an "Alloy" class. Moreover, the "Class_" object has attributes (if any), and a particle of a traffic model. The constraint object "chain" is associated to the object of "Class_" typed object. It means that the corresponding attributes in the list of a constrain object are constrained by the chosen restriction type. For example: one object has two attributes: max_capacity and car_counter. These attribute names are accepted in the constraint object and the binary operator is applied to those attributes. Precedence is the order of the names of the attributes, thus if the >= operator is chosen the meaning of the constraint is: max_capacity >= car_counter. Different types of constraints enable to create more detailed constraints. The "Class_" instantiation retrieves the attribute values as well, that are connected from the traffic model.

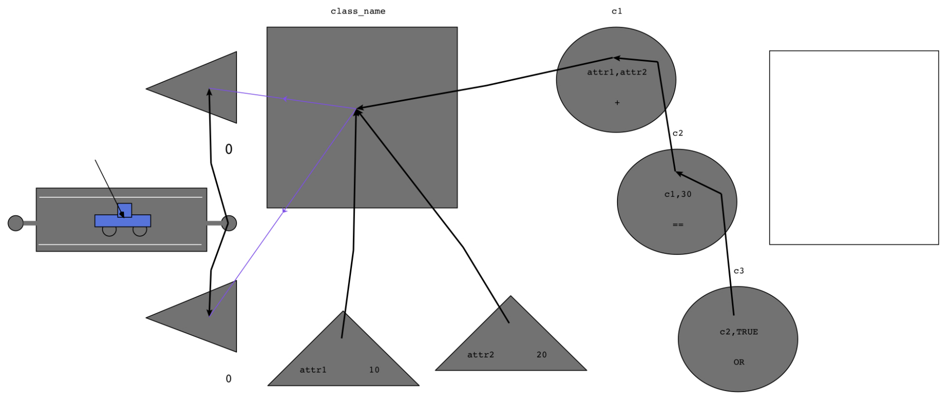

Figure 2: "Traffic" particle with a constraint on the "Sinks"

Now the generation of textual Alloy annotation is done by execution of Graph Grammar rules. It is not the most comfortable way to emit Alloy code, but we transform from a visual notation to a textual notation. An other approach could be the "action" on possible "Generate" button, when Python code generates Alloy code by looping through the "ASGroot.listNodes" list. It requires lot of programming but in this simple case it is still applicable. After running the only rule that matches this simple model, the appropriate Alloy code is filled into the object of "Alloy" class. On the LHS and the RHS there are identical objects of a "Class_" and "Alloy". At the matching, an action is written that gives value to the "Alloy" object. The attribute values are retrieved by "Class_" and the constraint is formulated after parsing the whole "constraint chain". Afterwards the generated code is suitable to run bounded exploration in Alloy.

```
class_node = self.getMatched(graphID, self.LHS.nodeWithLabel(2))
class_node_name = class_node.name.getValue()

constraint_node_name = "" #in Alloy a fact does not need a name
attr_names = ["attr1","attr2"] #TODO:traverse connected attributes of class

constraint_choice = " + " #TODO:traverse constraint chain for operators

new_code = "sig " +class_node_name+ "{"
new_code += attr_names[0]+":Int,"
new_code += attr_names[1]+":Int"
new_code += "}\n"
new_code += "fact " +constraint_node_name+ "{"
new_code += "all c:"+class_node_name+" | c."+attr_names[0]+
" "+constraint_choice+" c."+attr_names[1] #TODO:use every constraint
new_code += "}\n"
new_code += "pred show{}\n"
new_code += "run show for 3\n"

alloy_node = self.getMatched(graphID, self.LHS.nodeWithLabel(1))
alloy_node.alloy_code.setValue(new_code)
alloy_node.graphObject_.ModifyAttribute("alloy_code",new_code)

print "rule run"
print new_code
pass
```

Figure 3: "Python code snippet to produce Alloy representation"

4. Link to "Traffic" formalism

Since both "Traffic" and the previously introduced formalism are meta-modelled in Class Diagram Formalism they can be opened at the same time in ATom^3 to provide the extended "Traffic" formalism. After creating a fragment part of a vehicle traffic network, for example placing only one "Sink" onto the working area of AToM^3, we must instantiate our "Constraint_" classes and "Class_" class. Then associate "Class_" object to the "Sink" object, associate "Constraint_" objects to "Class_" object (and to each other if necessary), fill in the required information into "Constraint_" object and finally run the Alloy code generator Graph Grammar rules. Afterwards the Alloy code is available for analysis about the "Sink" entity. It is necessary to instantiate both "Class_" and (obviously) "Constraint_" because "Class_" contains the methods to retrieve the necessary attributes from the associated vehicle network entity and "Constraint_" stores the possible options of the constraints.

5. Experiment

I created a small part of a possible vehicle traffic network, 2 sinks are placed in AToM^3 and are connected to one road segment. Cars can drive into one sink (for example a parking lot) and if it is full, max_capacity is reached, then the cars need to drive to the other sink. A "Class_" is also instantiated and the connected to the sinks. A "Constraint_" is also placed and associated to "Class_". The attribute names are taken (inserted to constraint lists) and after executing the Graph Grammar rule, Alloy code is produced below:

```
sig cl_name{max_capacity:Int,car_counter:Int}
fact con_name{all c:cl_name | c.max_capacity >= c.car_counter}
pred show{}
run show for 1
```

In Alloy then we can create traces and examine the behaviour of the sinks by adding new cars to the sinks.

6. Comparison

In (Shah et al., 2009) a more sophisticated solution is given to transform UML model with OCL constraint to Alloy. Transformations are carried out on different levels of MOF hierarchy and UML2Alloy is used to produce Alloy model. In my work traffic network entities and constraints are about to analyse in Alloy and Graph Grammar rules are used in AToM^3. Similarly, in (Anastasakis et al., 2008) transformation from UML to Alloy is introduced. Moreover, they offer a mapping of class diagrams to Alloy and a mapping of OCL to Alloy with the explanation that a number of OCL constraints cannot be expressed in Alloy. In my work OCL constraints are replaced by the possibility of Alloy exploration. In the papers above Alloy 3 is used as the target syntax, in my work the transformation is done for Alloy 4, however the differences are subtle.

7. Conclusion and future work

A simple solution is provided to the initial problem, explicitly modelled constraints, classes and attributes are translated to Alloy to enable some kind of analysis about models in Traffic formalism. It is possible to choose attributes and apply constraints on them. However, the mapping between traffic network entities and corresponding Alloy representation is hard coded and not efficient because much programming is needed even though Graph

Grammar rules are used to prepare the mapping. In the future better solution would be to realize graph-to-graph rules instead of graph-to-text. Also the extension of optional constraints is needed, instead of only binary mathematical operators or logical expression. Moreover, the methods to retrieve given attributes should be implemented as well later on.

References

Anastasakis, K., Bordbar, B., Georg, G., Ray, I., 2008. On challenges of model transformation from uml to alloy. Software and Systems Modeling.

Bottoni, P., Koch, M., Parisi-Presicce, F., Taentzer, G., 2000. Consistency checking and visualization of ocl constraints. Lecture Notes in Computer Science.

Georg, G., Bieman, J., France, R., 2001. Using alloy and uml/ocl to specify run-time configuration management: A case study. Workshop of the pUML-Group.

Kühne, T., 2006. Matters of (meta-) modelling. Software and Systems Modeling.

Shah, S. M., Anastasakis, K., Bordbar, B., 2009. From uml to alloy and back again. ACM International Conference Proceeding Series.

Vangheluwe, H., de Lara, J., 2004. Computer automated multi-paradigm modelling for analysis and design of traffic networks. Winter Simulation Conference.