A Domain-Specific Language for Lego Vehicles

Luc Engelen, Yaping Luo*, Ana-Maria Sutii**

*y.luo2@tue.nl, **a.m.farcasi@tue.nl

October 14, 2014

1 Introduction

The exercise for this week consists of two assignments. The first assignment involves producing NXC code from the platooning DSL, and Java code from the bounding box DSL. The second assignment requires creating a transformation from the platooning DSL to the bounding box DSL. This exercise is a continuation of the last week's exercise. That means that you can reuse the metamodels and grammars defined last week. If you change your metamodels and/or the grammar for this exercise, please resend them too.

You can use the Epsilon tool suite (EGL and ETL), QVTo, or a combination of the two to solve the assignments.

Your submission should contain all the transformation files that you prepared, including:

- Platooning to NXC
- BoundingBox to Java
- Platooning to BoundingBox

To compile the NXC code you can use Bricx Command Center (`http://bricxcc.sourceforge.net/`). A tutorial on programming LEGO NXT robots using NXC that contains all the commands that you need can be found at `http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_tutorial.pdf`.

If you have any further questions on the exercise, please send the questions to either Yaping Luo or Ana-Maria Sutii.

2 First Assignment

Platoon & NXC

The first assignment requires you to implement a template that generates NXC code from a Platoon model. The generated code is supposed to make your platoon of Lego vehicles follow the defined route. You can make the vehicles communicate via bluetooth. In the bluetooth protocol, you can consider that the *front_runner* vehicle is the master and the following vehicle is the slave. The leading vehicle is the only one that contains the route commands and it will transmit commands to its following vehicle as he executes them (the following vehicle then transmits the commands to its following vehicle etc.). Furthermore, the generated code should be generic for all following vehicles (i.e. you should not generate code for each car in the platoon separately).

Because the application in NXC can be quite complex taking synchronization issues into account, we only ask that your generated NXC code compiles. You do not need to run the code on the Lego vehicles to prove that it behaves correctly. However, we want to see at least the following snippets of code in your generated code:

1. Code that makes the bluetooth connection between the vehicles (in the order in which they are defined in the Platoon DSL). We provide functionality that allows a master vehicle to connect to its slave vehicle ("Autoconnect.nxc"). Just import this file into your NXC application.

2. Code that uses the distance sensors on the Lego vehicles to ensure that the minimum and maximum headway between the vehicles is respected.

3. Code that executes the commands (forward, turn etc.) and transmits them to the following vehicle one by one recursively.

BoundigBox & Java

Additionally, you have to generate Java code from a BoundingBox model. The generated code is supposed to process the movements of the vehicle, and compute a bounding box for said movements. By definition, a bounding box is a rectangle with the minimum area that contains all the points reached by the vehicle while executing the moves. Your resulting Java program should compute and show two bounding box coordinates (the lower left corner and the upper right corner of the box). For example, the model in Figure 1 should generate a Java program that yields the bounding box defined by coordinates $(-50, 0)$ and $(0, 30)$ as output. For this purpose, assume the vehicle starts in $(0, 0)$. Summarizing: when an "up 10" is performed, the vehicle moves to $(0, 10)$. A successive "$left$ 50", results in $(-50, 10)$ and so on.

```
Box:
    up 10
    left 50
    down 10
    right 20
    up 30
```

Figure 1: A BoundingBox model example in Xtext editor: The platoon will move inside the Box. You can assume that in the beginning the robot is facing north (the up direction). The up direction corresponds to the north, the left and right directions correspond to the west and the east respectively and the down direction corresponds to the south.

3 Second Assignment

The second assignment requires you to create a model transformation platoon2boundingbox.

The platoon2boundingbox transformation takes, as input, a model written in the Platoon language and transforms it to a valid model written in the BoundingBox language. The resulting list of movements (in the BoundingBox language) should describe the same route as that in the Platoon model.

For example, the Platoon model in Figure 2 is transformed into the BoundingBox model in Figure 3. We assumed that the vehicles are in the $(0, 0)$ point initially and that they are facing the upward (north) direction.

```
platoon:
    LV Lego0 route R1
    FV Lego1 front runner Lego0
    FV Lego2 front runner Lego1

route R1:
    forward(10)
    turn Left
    forward(50)
    turn Right

constraints:
    20 <= headway <= 30
```

Figure 2: A Platoon model example.

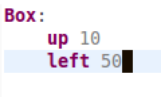

Figure 3: The resulting BoundingBox model after transforming the Platoon model in Figure 2.