CSE – 621
LEJOS PROJECT – PHASE II
Java Code Generation from Statechart Diagram
Instructor: Dr. Eric Rapos

Bhisma Adhikari, Nick DeGennaro

March 2020

1 Description of Source and Target Meta-Models

This primary goal of this phase of the project was to generate compilable java code from a statechart diagram. In other words, the task was to transform given textual representation of statechart diagram into a valid java file.

1.1 Source Meta-Model

The source file for the transformation is any valid state-machine/statechart diagram [5] file generated using the StarUML software. State-machine diagrams are UML standard diagrams used for behavioral modeling of systems. The textual representation of a startchart diagram generated by StarUML is a json [1] file, saved with an extension '.mdj', for example SampleStatechart.mdj. Json is a file format (or a string format) used commonly for data-exchange. In this format, data or objects are stored as key-value pairs, where the key is always a string, while the value can either be a string or a nested json object.

While generating an equivalent java code from the startchart diagram, we are concerned only with the information regarding States (Vertices, in StarUML's terminology) and Transitions. Both of these information are located in the "region" section of the statechart diagram. Transitions contain information about triggers and guards, if any.

1.2 Target Meta-Model

The target file for the transformation is a single valid java file (CSE621.java), that can be compiled and run on the LeJOS EV3 hardware. The file must represent the same information as the given input statechart diagram file.

2 Transformation Rules and Notes

1. First of all, the input json file (startchart diagram) is parsed to get the “vertices” and “transitions” as java Map objects, using java libraries jackson-core [3], jackson-databind [4], and jackson-annotations [2]. The corresponding jar files are included in the zipped folder for submission.
2. States in the startchart diagram are transformed to enumeration in the java program. Since the possible states were known in advance (IDLE, FORWARD, BACKWARD, ROTATE_LEFT, ROTATE_RIGHT), we decided to transform them in the java code using enum.
3. Transitions in the statechart diagram are transformed to equivalent switch-case statements in the java program. This functionality is implemented with the method transitionToStmt() in uml2lejos.java. This is where the main part of the transformation takes place.
4. When the sign “==” appears in distance or timer triggers, the equivalent generated code contains “<” and “>” respectively rather than “==”. This is done on purpose to enhance reliability of the LeJOS robot.
5. Sections of the code which are required in the generated java file but are not directly generated from the given statechart diagram are hard-coded. These include the import and package statements and the definitions of class Robot, enum Color, and enum State. These sections of code are hard-coded in the text files – code-imports.txt and code-other-classes.txt, and read by uml2lejos.java during the transformation. Both of these files are included in the zipped folder for submission.
6. The output CSE621.java file is not only a valid java file but also is well-formatted by the use of tabs and newlines as appropriate. Thus, the generated code is readable to a human programmer.

References

[1] Introducing json. `https://www.json.org/json-en.html`. Accessed: 03-25-2020.

[2] Mvn repository: Jackson annotations. `https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations`. Accessed: 03-25-2020.

[3] Mvn repository: Jackson core. `https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core/2.8.4`. Accessed: 03-25-2020.

[4] Mvn repository: Jackson databind. `https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind/2.8.4`. Accessed: 03-25-2020.

[5] State machine diagrams. `https://www.uml-diagrams.org/state-machine-diagrams.html`. Accessed: 03-25-2020.