Capa

SimpleMio is a domain-specific language (DSL) designed to simplify the programming of robotic behaviors. This documentation provides an in-depth look at the components of SimpleMio, including the meta model, grammar, type checking, code generation, quick fixes, and auto-completion features ADICIONAR MAIS QUANDO FIZERMOS MAIS. The aim is to provide to high-school students an intuitive syntax so they can set some instruction to the thymio robot without needing to know how to program.

# Metamodel

This is the meta model that defines the structure of the SimpleMio language.

(PRINT DO META MODEL)

**It includes the following key entities:**

* Model: The root entity that contains the entire program;
* Program: Represents a collection of events;
* Event: Defines a trigger and associated actions;
* Action: Represents an action to be performed;
  + Contains:
    - actionName: EString
    - actionSpecifier: EString
    - strength: Eint (SQUE PRECISA SER ALTERADO DPS)
* ConditionalSensor: Represents a sensor condition;
* Sensor: Represents a sensor;
  + Contains:
    - sensorName: EString
    - sensorSpecifier: EString
    - strength: Eint (SQUE PRECISA SER ALTERADO DPS)
* Or: used to set precedence when building a condition;
* Not: used to set precedence when building a condition;
* And: used to set precedence when building a condition.

**As you can see by the image, these are the entity relationships:**

* A “Model” contains a “Program”;
* A “Program” contains multiple “Event”;
* An “Event” have a “ConditionalSensor” and multiple “Action”;
* A “ConditionalSensor” contains an “Or”, an “And, a “Not” and a “Sensor”;
* An “Or” contains 2 “ConditionalSensor”, the left part and the right part of the “Or”;
* An “And” contains 2 “ConditionalSensor”, the left part and the right part of the “And”;
* A “Not” contains a “ConditionalSensor”;
* The “Sensor”, the “Or”, the “And” and the “Not” are all super types of “ConditionalSensor”.

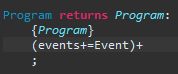
# Grammar

The grammar that defines the syntax for writing SimpleMio programs. Below are the key grammar rules:

For the Model:



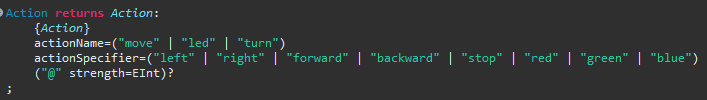
For the Program:



For the Event:



For the Action:



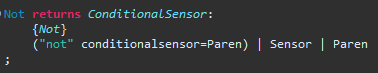
For the Or:



For the And:



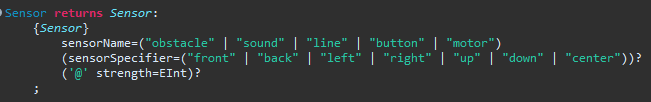
For the Not:



For the Parenthesis:



For the Sensor:



For the Single Line Comment:



**Key Components**:

* **Model Structure:** The grammar comprises a hierarchical structure with a root Model containing a single Program, which, in turn, consists of multiple Events.
* **Event Definition:** Each Event combines a conditional sensor expression (Or) with a sequence of actions, allowing for complex behavior triggers.
* **Actions and Sensors:** **Actions** such as **move, led, and turn** are defined alongside **sensors** like **obstacle, sound, line, button and motor**, enabling diverse event triggers based on environmental stimuli.

As you can see, a **sensor** is composed by a **sensorName** (obstacle, sound, line, button or motor), a **sensorSpecifier** (front, back, left, right, up, down, or center) and optionally the user can set the strength of the sensor by adding a value after the “@” symbol.

An **action** is composed by an **actionName** (move, led or turn), an **actionSpecifier** (left, right, forward, backward, stop, red, green or blue) and optionally the user can set the strength of the action by adding a value after the “@” symbol.

The user of the DSL can also add comments to the program by adding a “#” before the comment.

An event is composed by a conditionalSensor followed by a “->” and a at least 1 action, each action is separated by a comma “,”, i.e, conditionalSensor -> action (","action)\*

SimpleMio's grammar provides a robust foundation for defining robotic behaviors, offering users a flexible and intuitive language for programming diverse actions.

# Validator

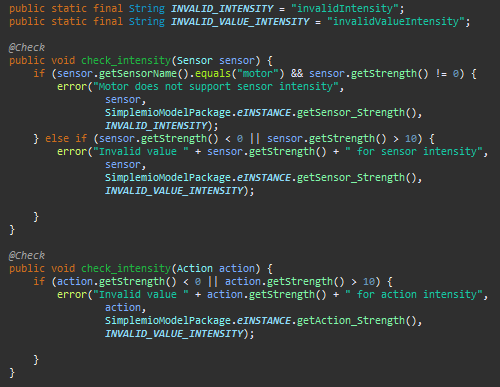
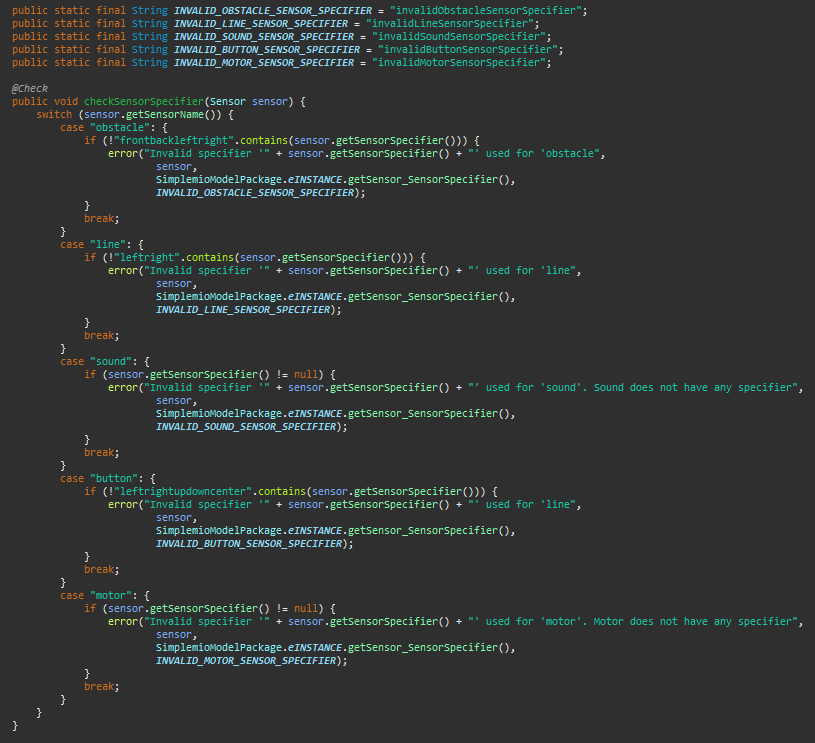
The type checking ensures that the SimpleMio code adheres to defined constraints. Key checks include:

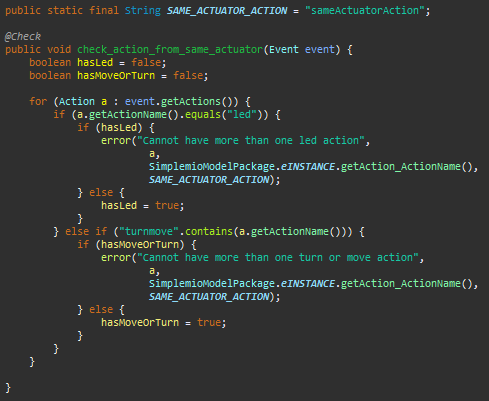
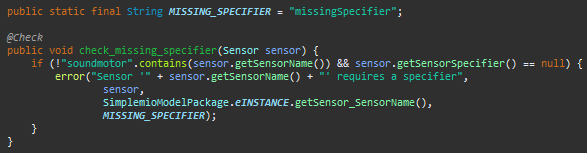
* Action Specifiers: Validates that actions have correct specifiers.
* Sensor Specifiers: Ensures sensors have valid specifiers.
* Intensity Values: Checks that intensity values are within the allowed range (0-10).

Validator Implementation

The `SimpleMioValidator` class contains these custom validation rules:







To sum up, these are what each action or sensor is allowed to be followed by:

**move** can be followed by forwards, backwards or stop.

**led** can be followed by red, blue or green.

**turn** can be followed by right or left.

**obstacle** can be followed by front, back, left or right.

**line** can be followed by left or right.

**button** can be followed by left, right, up, down and center.

**sound** and **motor** don’t have any specifier.

The sensor **motor** doesn’t support intensity.

For all other **sensors** and **actions**, which support intensity, the intensity must be a number between 0 and 10.

**The validator also doesn’t allow overlapping actions** like, more than 1 action led or more than 1 action turn or move in total.

Each error is also identified with a different tag so further it is possible to implement the quickfix.

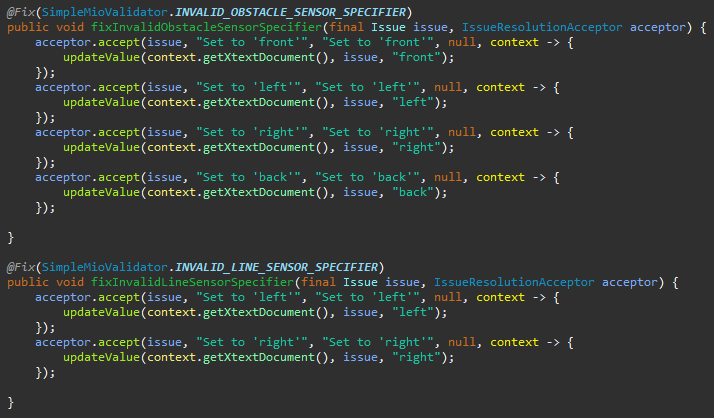
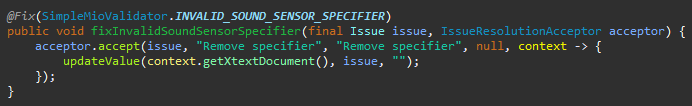
# Code Generation

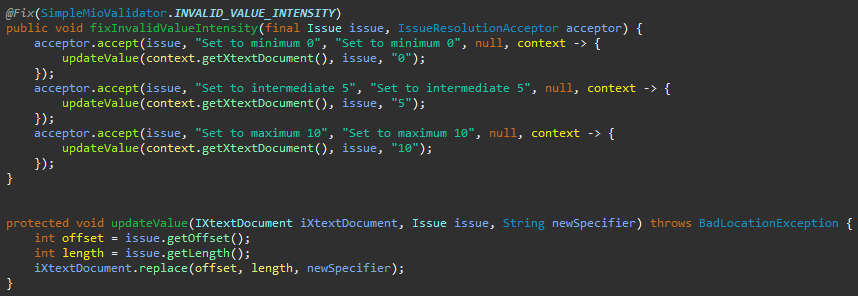
# IDE Services

The IDE services that were implemented in the DSL were: quickfix and auto-completion

## Quickfix

As mentioned before, for the quickfix this was the implementation to correct the errors thrown:

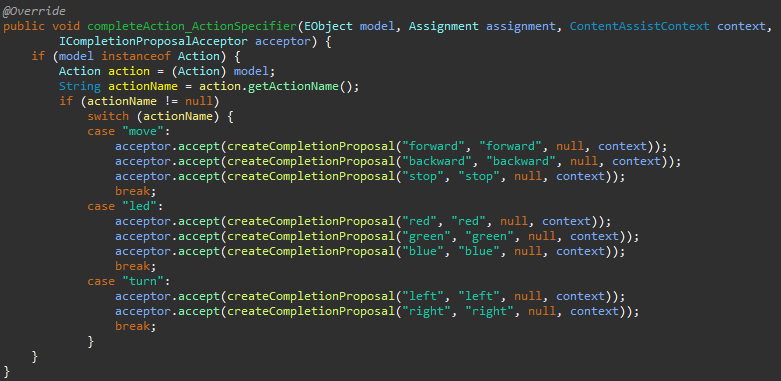
  

Basically if any error is detected the IDE will provide a valid solution to correct the program. When an intensity is wrongly inputed the IDE will provide 3 solutions, a minimum value (0), a medium value (5) and a maximum value (10)

## Auto-completion

For the auto-completion, we implemented the auto-completion for sensors and actions and this is what we implemented:

Basically for each sensor or action that needs a specifier the IDE will suggest the specifier.

# Tests