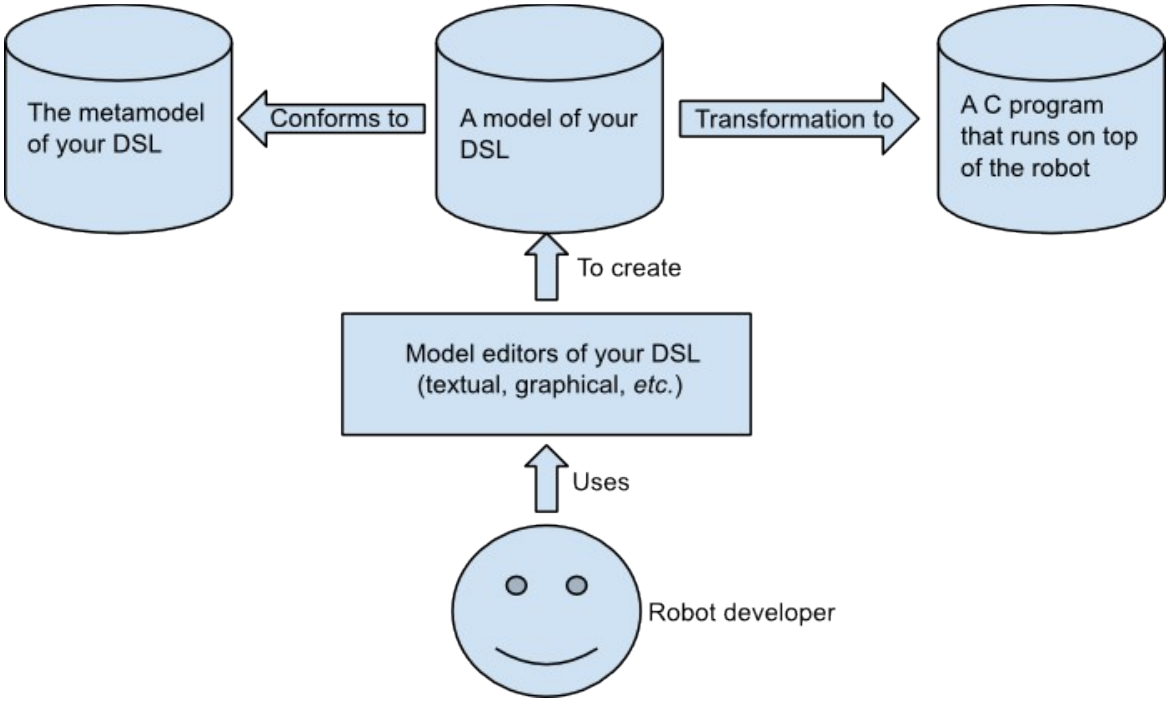
Model-Driven Engineering

TP – Creating a domain-specific language (DSL) with EMF and Kermeta

Goal

The goal of these three practical sessions is to create a language for programming the behaviour of a Drawing Lego robot based on the Logo Language. This robot can detect physical objects (thanks to a sonar), move forward, rotate, emit a sound, and display some texts on its screen.



Schema describing the use of a DSL for programming a Lego robot

The development toolkit provided with the robot relies on C language. The main drawback is that C is a general-purpose language (GPL) and is, therefore, not dedicated to the development of Lego robots. Using C to develop robots will polluted the developed code with C-specific routines and hacks. The use of a language dedicated to robots will alleviate their development.

As explained in the three classes devoted to MDE, the development of such a dedicated programming language consists of three steps:

1. The creation of the metamodel (the abstract syntax) of the new programming language (Here we will Mimic Logo). The metamodel defines and organises, through a UML-like class diagram, the concepts of the programming language (instructions, conditional statements, *etc.*).
2. The creation of a textual editor to ease the coding of programs using the newly created language. This step involves the creation of a grammar (the concrete syntax) for the textual language. The concrete syntax is provided in the next Section.
3. A simulator to test the program
4. A compiler that transforms (compiles) a program of your language into C code that runs on top of the robot (we cannot bypass C everywhere since the robot needs C programs to run).

Not eXactly C (NXC)

NXC is a simplification of C to run on top of Lego robots (LEGO NXT MINDSTORMS). This document describes concepts of this language required for the pratical sessions.

http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_Guide.pdf

1.1.Program structure

An NXC program looks like a C one. It is mainly composed of fonctions. NXC introduces the conception of Task that corresponds to a thread:

```
task name(){
// the task's code is placed here
}
```

A NXC program must have a least one task called « main » that will be used as the entry point of the program.

1.2.API NXC (excerpt)

Wait(time)

Function

Make a task sleep for specified amount of time (in 1000ths of a second). The time argument may be an expression or a constant.

```
Wait(1000); // wait 1 second
Wait(Random(1000)); // wait random time up to 1 second
```

Stop(bvalue)

Function

Stop the running program if bvalue is true. This will halt the program completely, so any code following this command will be ignored.

```
Stop(x == 24); // stop the program if x==24
```

Random(n)

Value

Return an unsigned 16-bit random number between 0 and n (exclusive). n can be a constant or a variable.

```
x = Random(10); // return a value of 0..9
```

Random()

Value

Return a signed 16-bit random number.

```
x = Random();
```

SetSensorLowspeed(port)

Function

Configure the sensor on the specified port as an I2C digital sensor (9V powered). The port may be specified using a constant (*e.g.*, S1, S2, S3, or S4) or a variable.

```
SetSensorLowspeed(S1);
```

SensorUS(n)

Value

Return the processed sensor reading for an ultrasonic sensor on port *n*, where *n* is 0, 1, 2, or 3 (or a sensor port name constant). Since an ultrasonic sensor is an I2C digital sensor its value cannot be read using the standard Sensor(n) value. A variable whose value is the desired sensor port may also be used.

```
x = SensorUS(S4); // read sensor 4
```

Figure : Output ports constants

Off(outputs)

Function

Turn the specified outputs off (with braking). Outputs can be a constant or a variable containing the desired output ports. Predefined output port constants are defined in Figure 1.

```
Off(OUT_A); // turn off output A
```

OnFwdSync(outputs, pwr, turnpct)

Function

Run the specified outputs forward with regulated synchronization using the specified turn ratio. Outputs can be a constant or a variable containing the desired output ports. Predefined output port constants are defined in Figure 1.

```
OnFwdSync(OUT_AB, 75, -100); // spin right
```

OnRevSync(outputs, pwr, turnpct)

Function

Run the specified outputs in reverse with regulated synchronization using the specified turn ratio. Outputs can be a constant or a variable containing the desired output ports. Predefined output port constants are defined in Figure 1.

```
OnRevSync(OUT_AB, 75, -100); // spin left
```

RotateMotorEx(outputs,pwr,angle,turnpct,sync,stop)

Function

Run the specified outputs forward for the specified number of degrees. Outputs can be a constant or a variable containing the desired output ports. Predefined output port constants are defined in Figure 1. If a non-zero turn percent is specified then sync must be set to true or no turning will occur. Specify whether the motor(s) should brake at the end of the rotation using the stop parameter.

```
RotateMotorEx(OUT_AB, 75, 360, 50, true, true);
```

PlayToneEx(frequency, duration, volume, bLoop)

Function

Play a single tone of the specified frequency, duration, and volume. The frequency is in Hz. The duration is in 1000ths of a second. Volume should be a number from 0 (silent) to 4 (loudest). All parameters may be any valid expression.

```
PlayToneEx(440, 500, 2, false);
```

NumOut(x, y, value, clear = false)

Function

Draw a numeric value on the screen at the specified x and y location. Optionally clear the screen first depending on the boolean value of the optional "clear" argument. If this argument is not specified it defaults to false.

```
NumOut(0, LCD_LINE1, x);
```

TextOut(x, y, msg, clear = false)

Function

Draw a text value on the screen at the specified x and y location. Optionally clear the screen first depending on the boolean value of the optional "clear" argument. If this argument is not specified it defaults to false.

```
TextOut(0, LCD_LINE3, "Hello World!");
```

Button Constants	Value
BTN1, BTNEXIT	0
BTN2, BTNRIGHT	1
BTN3, BTNLEFT	2
BTN4, BTNCENTER	3
NO_OF_BTNS	4

Figure : Button constants

ButtonPressed(btn, reset)

Value

Return whether the specified button is pressed. Optionally clear the press count. Valid values for the btn argument are listed in Figure 2.

```
value = ButtonPressed(BTN1, true);
```