Veronika Schrenk, Manuel Wallner
HTL Wiener Neustadt, HTL Wiener Neustadt
v.schrenk@gmx.at, manu.wallner@me.com

Sparkling coding - a paradigm shift in teaching coding

Abstract

Creating robots is a very challenging topic for young engineers - either you are restricted by the hardware like the Lego Mindstorms NXT or you have to overcome difficulties like the CBC's C interface.

With our recent work we are introducing a user-friendly software architecture that is based on several easy to use C++ classes. Instead of creating functions by using the low-level motor, sensor and servo functions it works by combining different actions with predefined conditions. A common example would be a simple line-following robot that drives slightly to one direction (action) until the sensor value goes over or under a certain threshold (condition).

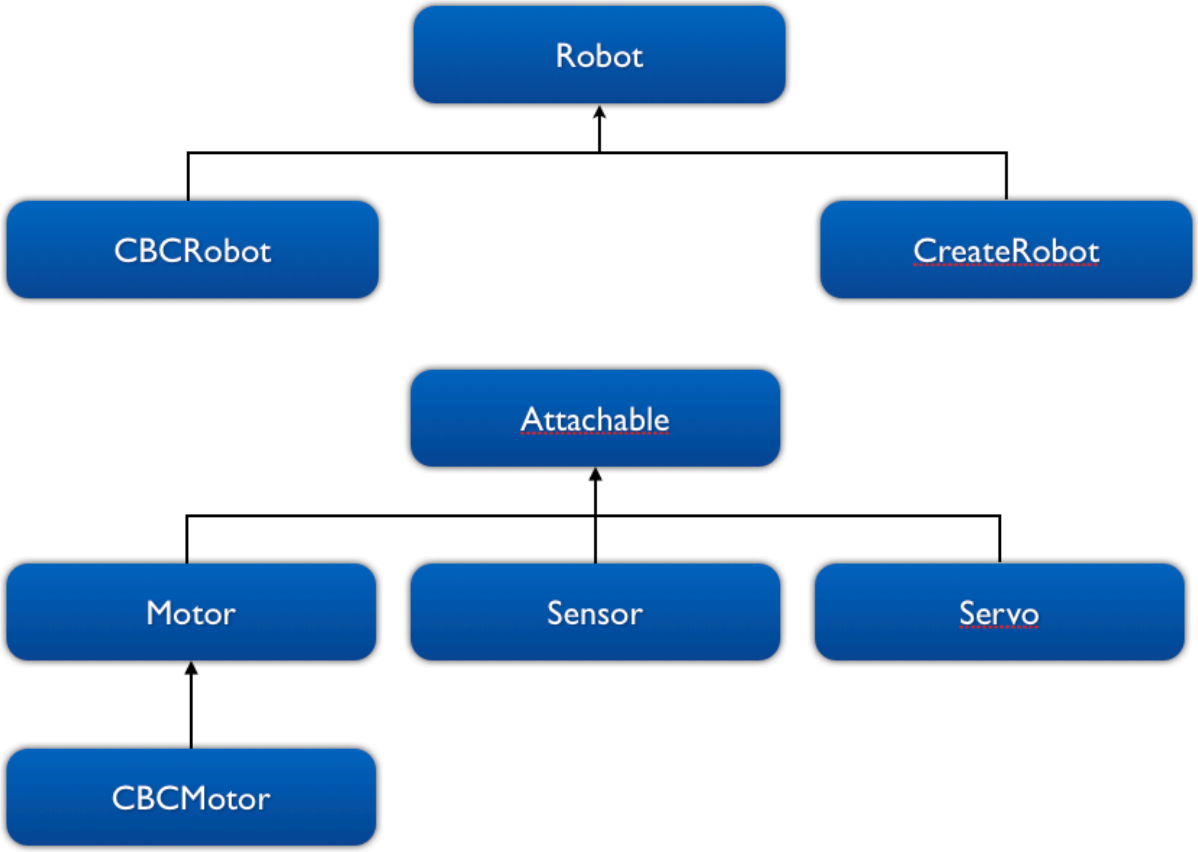
Introduction

First let us answer a simple question. Why did we choose to create a library like this, when we could have simply used the CBC library and save us a lot of time? When we first learned about the CBC library, we soon discovered that it had a few fundamental problems.
One would be that it is hard to learn for beginners. If you look at the function names and their parameters, you realize that some of them, despite having similar tasks, accept different kinds of parameters or return values in a different range. This is something that can introduce many errors into your code, which can also be very hard to locate. For our system, we have used a simple scheme: every value is in a range of 0%-100%. This means that wether you want to drive forward at half speed with the CBC‘s motors or the Create‘s built-in motors, the code will look the same.

Another reason is that we wanted to be able to use Object-Oriented-Programming when programming our robots. This made it possible to define a single interface for the CBC and the Create, so that you can use both interchangeably, which makes it easier to use existing code on both platforms. Of course that also means that the functions for both kinds of robots take the same parameters and return the same values, which, again, makes it easier to use for beginners.

Library

Our library is built around just a few classes. Here you can see an inheritance diagram:



Robot: This is the interface for both the CBCRobot and the CreateRobot. It provides declarations for functions like driving forward or turning.

CBCRobot & CreateRobot: Depending on which base you want to use for your robot to drive, you need to create one of these at the start of the program.

Attachable: This is an interface for everything that can be attached to a port on the CBC

Motor & CBCMotor: This is the main class for every motor. It has functions to set the power of the Motor to a value between -100% and +100%
Sensor: This it the main class for every Sensor. It has a simple method that returns the sensor‘s value between 0% and 100%.
Servo: This is the main class for every Servo. It has methods to rotate the Servo by a specific angle or percentage.

These are the important regular high-level classes that we created for our library. They can be used independently from the rest of the library. The library has a few other classes that were created to implement the before-mentioned actions and conditions:

State: This implements an action. The State can also be created with a Condition, which means that it will run for as long as the Condition returns true. It can also have a successor that runs after the current State.
StateManager: This runs the main application. You add an entry state and the StateManager will run each State until it reaches the last one. It also takes care of stopping the robot after 2:00 minutes and to wait for light at the beginning of the application.
Condition: Just checks for a single thing, like if a sensor value goes over a certain threshold.

Of course you need a few subclasses for the States and the Conditions to be useful. We have created a few simple ones. The most important States are DriveState and ServoState, which can move the robot‘s left and right Motor or a Servo. The most important Condition is SensorCondition, which simply checks if a Sensor‘s value goes over a certain threshold.
The goal of these few classes was to create a system that is easy to use for beginners. It is similar to the Lego NXT Mindstorms Software except that there is no GUI yet for our library. In some cases our approach looks more complicated than regular CBC library code, but we hope that we can fix that in the future. A GUI could also solve this problem.