# SystemC - Minimal example

https://github.com/thisDotLucas/DATSimulatorC/tree/main

SystemC creates a discrete simulation during which created modules execute their defined functionality. Modules use ports from the SystemC library for communication between each other during simulation. In the minimal example we have two modules: a motor and motor controller. In the first iteration the motor will continuously through out the simulation report it's angle to the controller.

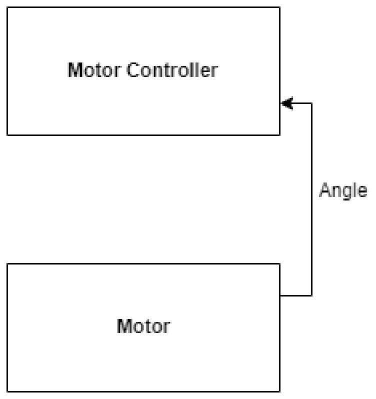

Figure 1: Diagram of the simulation model.

In the diagram the components are defined by rectangles with the name of the component in the center. The arrow defines a signal going between two ports, an in port exists in the module where the arrow head is pointing and an out port exists in the module that the arrow starts from. The text next to the arrow describes the data being sent between the ports. In the program the motor module has a data member that keeps track of it's angle and at every discrete time step it writes that data to the motor controller using the ports. The motor controller prints these values to the console in

the following format:
@ <Current time in simulation> - Angle: <Angle>
For now the angle data member is static, there is no way of changing it during simulation.

To extend on this model I added another line of communication and added another data member for the motor: torque.

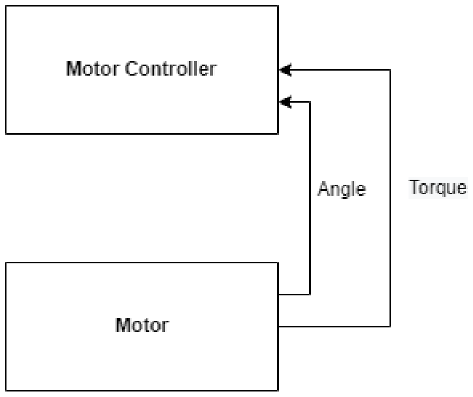

Figure 2: Diagram of the extended simulation model.

The behavior is exactly the same, I added two ports and connected them with a signal and the motor controller uses the data in the same way as the angle data, it prints it using the same format to the console.

To further extend this I created two more communication lines, this time from the motor controller to the motor. These communication lines will be used to write over the data members in the motor. To change the values mid simulation I created a JSON format to define values and times for when values change and a corresponding parser to access the values. The JSON file gets deserialized into a JSON configuration object that needs to be passed into the modules constructors.

```json
{
    "SimulationTime": 200,
    "StepSize": 100,
    "Angles": [
        { "Value": 90, "Time": 0 },
        { "Value": 60, "Time": 5 },
        { "Value": 30, "Time": 15.5 }
    ],
    "Torques": [
        { "Value": 20, "Time": 0 },
        { "Value": 40, "Time": 5 }
    ]
}
```

The first JSON key *SimulationTime* defines the length of the simulation in seconds, note that it's in simulation time and not wall-clock time. The JSON key *StepTime* defines the simulated time in milliseconds between each discrete time step in the simulation. The JSON keys *Angles* and *Torques* contains arrays of JSON objects, these JSON objects have two keys: *Value* and *Time*. The key *Value* defines an value and the key *Time* defines at what time the corresponding data member should be set to the given value.

In the program the JSON configuration object has an interface which takes a time value *t1* as an argument and then returns the value so that the returned value's time *t2* satisifes *t1* >= *t2* while there exists no other *t* so that *t* > *t2* and *t* =< *t1*. As an example with the torques from the JSON example above for 0 =< time < 5 the returned value is 10 and for 5 =< time $< \infty$ the returned value is 40.

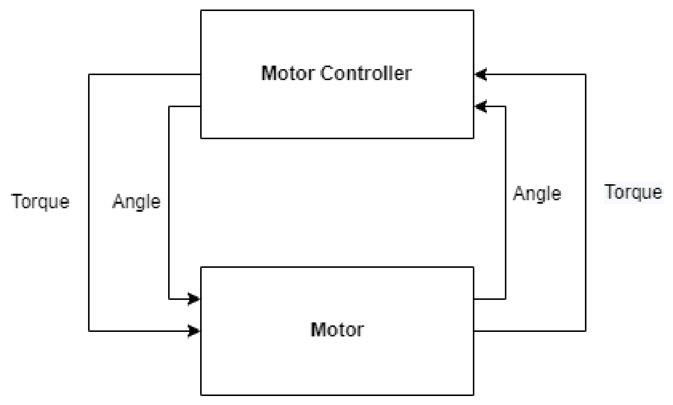

Figure 3: Diagram of the simulation model with possibility to write to the motor from the controller.

SystemC seems like a promising library for the project, even though it's main use has been in the computing hardware industry the interface provided seems generic and customizable to our needs. The project for this minimal example can be found from the Github url at the start of the document, it uses Cmake to build cross-platform.