LeWIS – Let's Write Intricate Simulators!

Michael Hart, Michael Wedel, Owen Arnold

Introduction

Lewis is a Python framework for rapidly developing detailed device simulators. It is distributed via PyPI, GitHub and DockerHub.

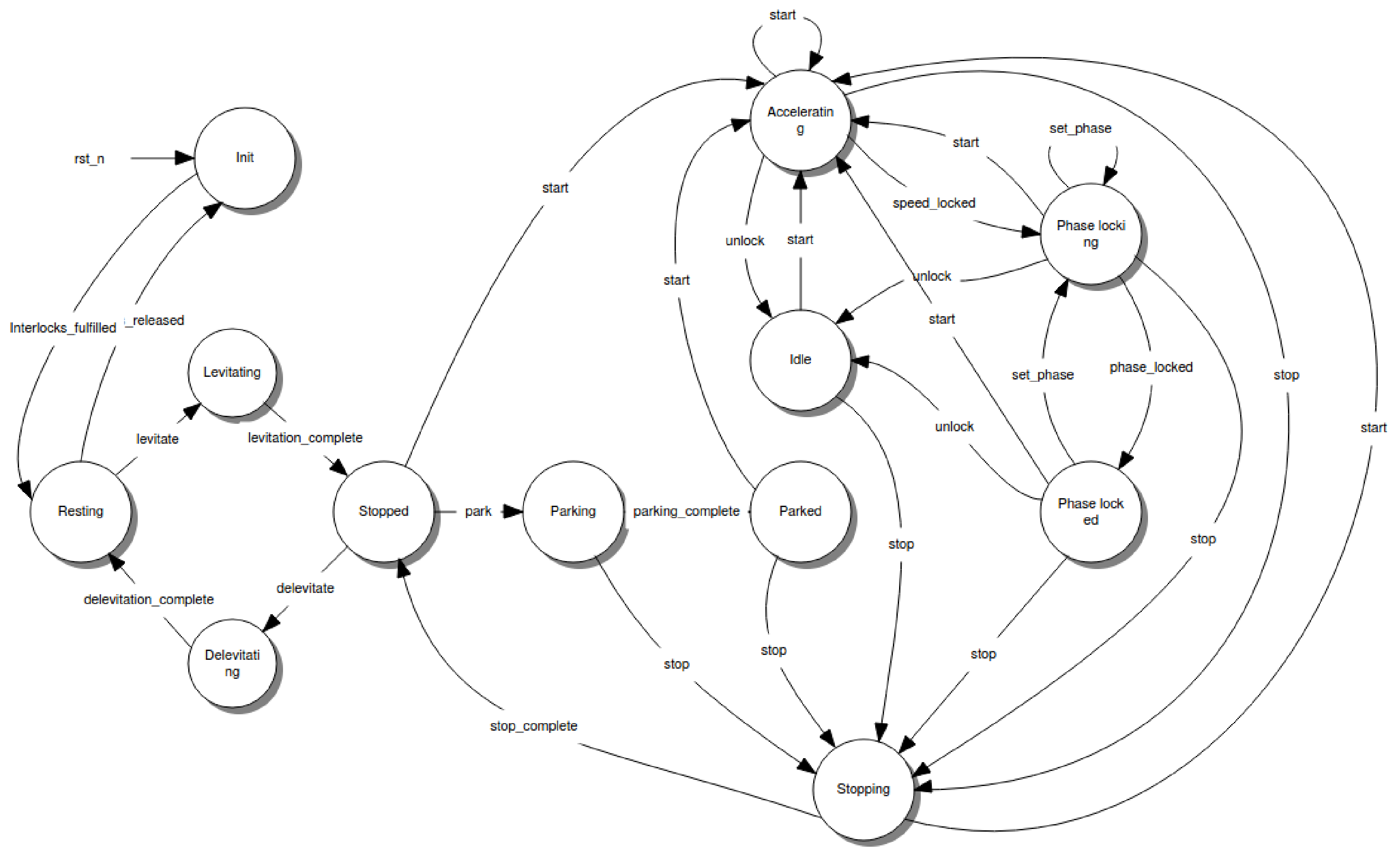
Device simulators are valuable for a variety of purposes in the context of developing software that interacts with hardware. Some example use cases are:

- Substituting for physical devices when they are unavailable
- Testing failure conditions without risk to real devices
- Automated system and unit tests of device communication
- Perform "dry runs" of user scripts to check validity

Simple simulators which only echo back static values are not enough for many of these use cases. Especially in the context of the ESS, where we have to develop control software while hardware devices are still unavailable, we needed to have a convenient and uniform way of creating and running device simulators that capture rich device behaviour in as much detail as possible.

State Machine

Many devices have complex behaviour that is best modelled using a state machine. For example, this is the state diagram of a chopper that will be used at the ESS:



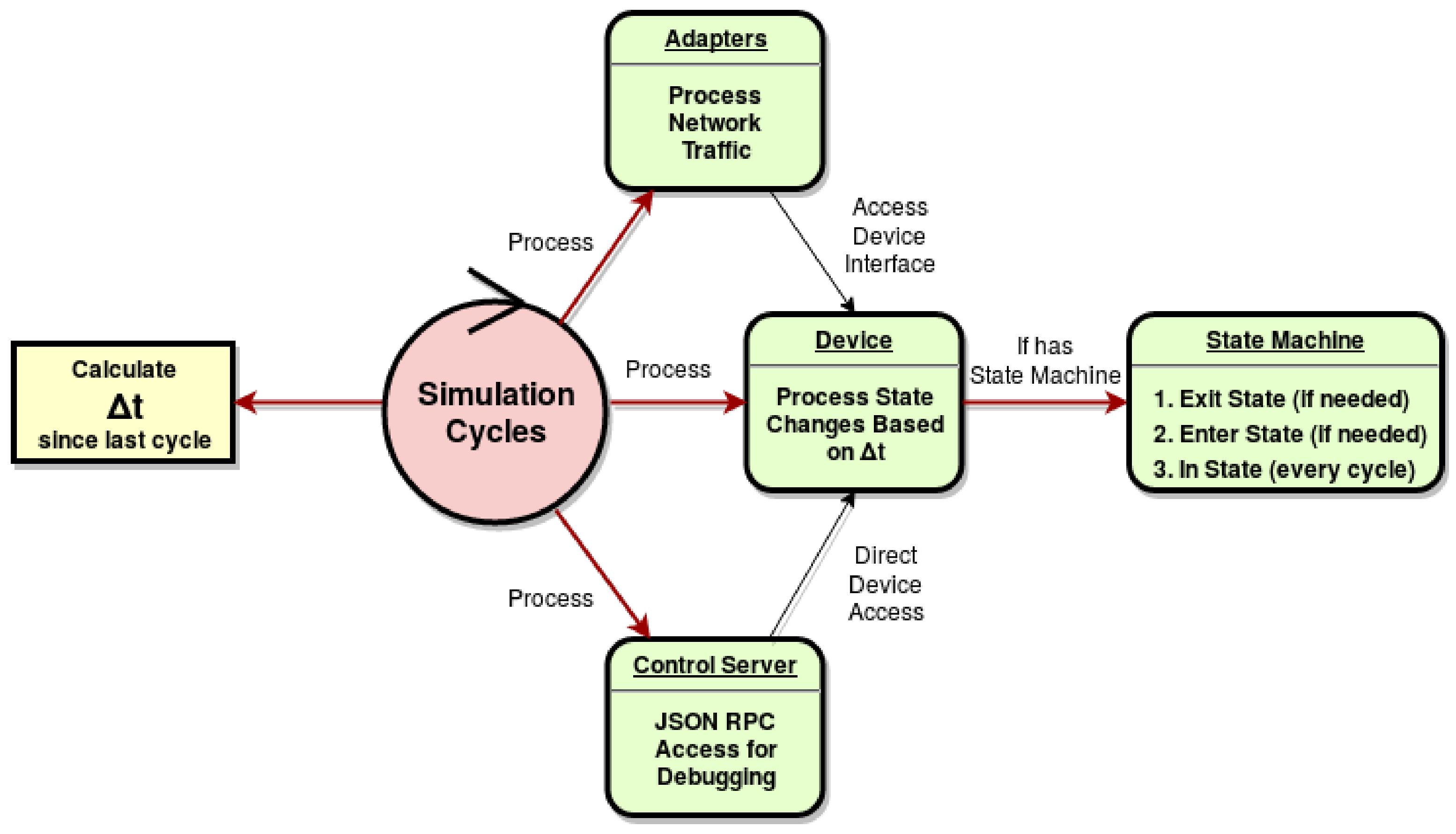
Lewis provides a base class that enables modelling a state machine by specifying the available states and the conditions for transitioning between them. Transitions are handled automatically by Lewis, and events are raised when they occur, so that the simulator developer only needs to implement how the device should react.

Lewis supports several common device protocols out-of-the box.

A device simulator can be written once, in a protocol-agnostic way, and then exposed via any protocol using a thin Interface layer that connects properties, attributes and functions of the Device to PVs, registers or commands, depending on the protocol.

Cycle-driven

It is important for a simulation to behave in a predictable and reproducible manner. To ensure this, Lewis operates based on cycles.



The Simulation Engine issues cycles periodically, processing network traffic and device simulation. For every cycle, a Δt value – the time since the last cycle – is passed through the simulation. Any device behaviour that is time-dependent (temperature changes, acceleration, movement, …) should be modified by this Δt value.

By modifying the Δt and the cycle rate, we can also speed up and slow down the simulation, as well as modifying simulation fidelity, without any changes to the device implementation.

Very Simple Example:

```
class VerySimpleDevice(Device):
    value = 10


class SimpleTCPInterface(StreamAdapter):
    commands = [
        Var('value', read_pattern='^V$', write_pattern='^V=(.+)$')
    ]

    in_terminator = '\r\n'
    out_terminator = '\r\n'
```

Running and Interacting with Simulation:

```
$ lewis VerySimpleDevice -p "stream: {port: 2020}"
$ telnet localhost 2020
V
10
V=20
V
20
```

EPICS Interface for Same Device:

```
class SimpleEPICSInterface(EPICSAdapter):
    pvs = {
        'Value': PV('value', type='int')
    }
```

```
$ lewis VerySimpleDevice -p "epics: {prefix: 'SIM:'}"
$ caget SIM:Value
SIM:Value                      10
$ caput SIM:Value 20
Old : SIM:Value                      10
New : SIM:Value                      20
```

Control Client on Command-line:

```
$ lewis-control device value
10
$ lewis-control device value 20
$ lewis-control device value
20
```

Control Client in Python:

```
from time import sleep
from lewis.core.control_client import ControlClient

client = ControlClient(host='127.0.0.1', port='10000')
chopper = client.get_object('device')

chopper.target_speed = 100
chopper.initialize()

while chopper.state != 'stopped':
    sleep(0.1)

chopper.start()
```

StateMachineDevice Example:

```
from lewis.devices import StateMachineDevice

from lewis.core.statemachine import State
from lewis.core import approaches


class DefaultMovingState(State):
    def in_state(self, dt):
        old_position = self._context.position
        self._context.position = approaches.linear(old_position,
                                     self._context.target,
                                     self._context.speed, dt)
        self.log.info('Moved position (%s -> %s), target=%s,
                      speed=%s', old_position,
                      self._context.position,
                      self._context.target,
                      self._context.speed)


class SimulatedExampleMotor(StateMachineDevice):
    def _initialize_data(self):
        self.position = 0.0
        self._target = 0.0
        self.speed = 2.0

    def _get_state_handlers(self):
        return {
            'idle': State(),
            'moving': DefaultMovingState()
        }

    def _get_initial_state(self):
        return 'idle'

    def _get_transition_handlers(self):
        return {
            (('idle', 'moving'), lambda: self.position !=
                                         self.target),
            (('moving', 'idle'), lambda: self.position ==
                                         self.target)
        }

    @property
    def target(self):
        return self._target

    @target.setter
    def target(self, new_target):
        if self.state == 'moving':
            raise RuntimeError('Cannot change target, moving.')

        if not (0 <= new_target <= 250):
            raise ValueError('Target is out of range [0, 250]')

        self._target = new_target

    def stop(self):
        self._target = self.position

        self.log.info('Stopping movement after user request.')

        return self.target, self.position
```

Where to find Lewis

GitHub: https://github.com/DMSC-Instrument-Data/lewis
DockerHub: https://hub.docker.com/r/dmscid/lewis/
ReadTheDocs: http://lewis.readthedocs.io/

```
$ pip install lewis

$ docker pull dmscid/lewis
```