IoT Software Foundation for a Smart City

Team Cache Money

Connecting Smart Devices via websocket-servers

Using python's websocket library we created a local server that smart city sensors and actuators can connect to and communicate with each other via this server

The rules that guide our devices can be loaded into this environment can be shaped according to the city's needs

Some device types established, many more to come

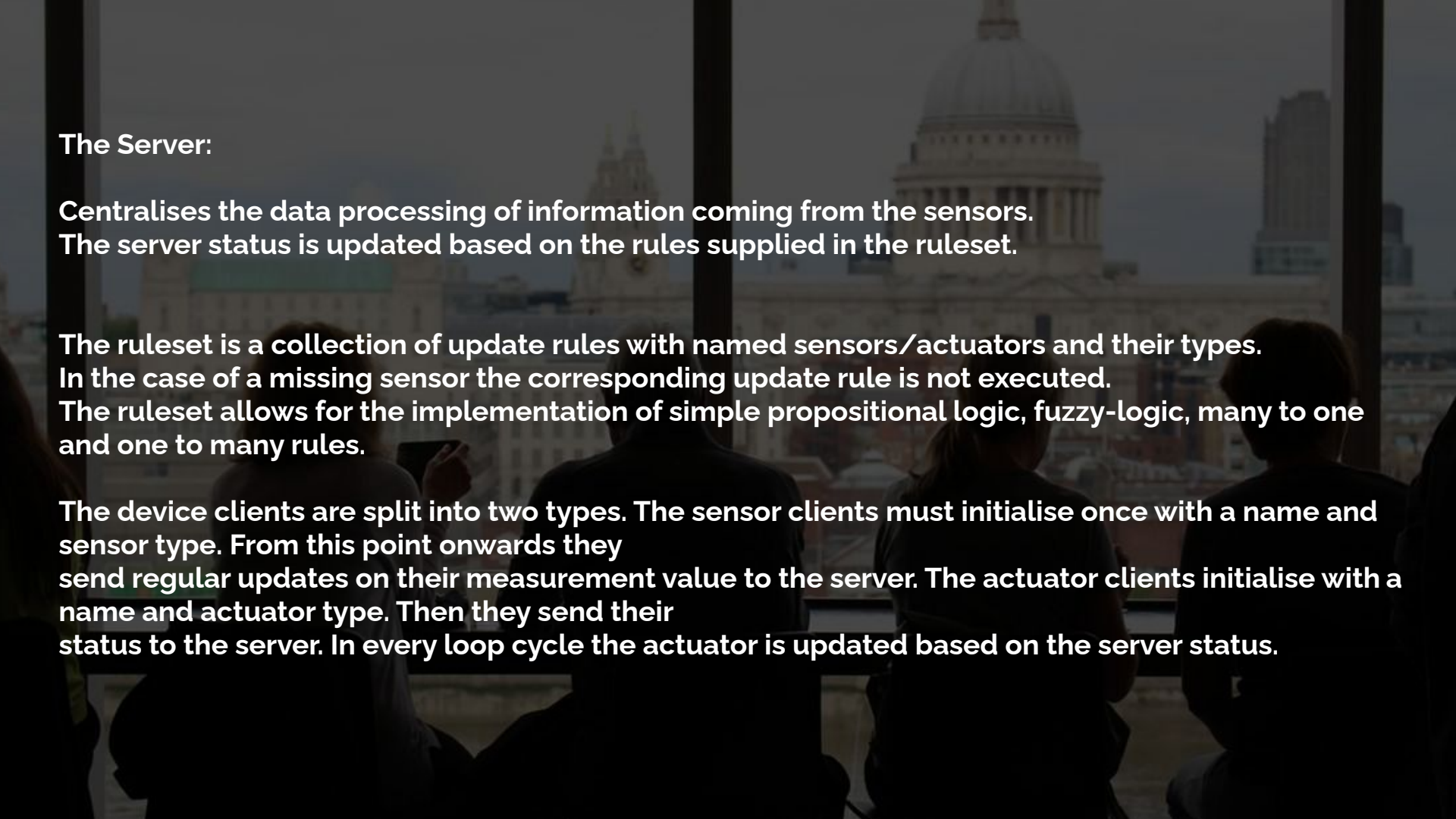
For simulation purposes mock data is generated on a simulation city grid that agents can walk through

What does the system's API service setup look like and how can the devices communicate with each other?

1. System Architecture

Device communication securely via central API service

- **API service: python websocket server**
  Gathers device data, sends out received data, regulates device signin/signout and device communication
- **Devices: python websocket clients**
  Different classes for different device types, differentiation between sensors and actuators: sensors send measurements to server, actuators receive measured data from server to update their status



The Server:

Centralises the data processing of information coming from the sensors.
The server status is updated based on the rules supplied in the ruleset.

The ruleset is a collection of update rules with named sensors/actuators and their types.
In the case of a missing sensor the corresponding update rule is not executed.
The ruleset allows for the implementation of simple propositional logic, fuzzy-logic, many to one and one to many rules.

The device clients are split into two types. The sensor clients must initialise once with a name and sensor type. From this point onwards they
send regular updates on their measurement value to the server. The actuator clients initialise with a name and actuator type. Then they send their
status to the server. In every loop cycle the actuator is updated based on the server status.

2. Device Management

- **Easily add new sensors and actuators**
  New sensors and actuators can easily be added during server runtime. Just run client_server.py/client_actuator.py enter some details and you are good to go!

- **Add new rules anytime!**
  You can add a new rule anytime, by adding it to the config.py file. The server will keep updating its rules.

Smart Sensors

To measure the CITY's environment.

Different types: Motion Sensor, Proximity Sensor,...

Smart Actuators

Governed by YOUR rules.

Can receive multiple inputs

Different types: street lights, dimmable lights, doors, ...

...easy to add more

Device communication solely through a **SECURE CONNECTION** via the websocket server

3. Rule Management and Rule Engine

Rule Management via config.py file → configure the system to **your city's needs** and implement **your own smart rules** for the devices

➔ **Different devices are initialized by python classes**

Class definitions can be seen in ruleset.py file and simple python code is needed to set up all your devices and the corresponding rules

➔ **Connect devices to server**

Run the client_actuator.py file in the terminal to connect an actuator to the server and client_sensor.py to connect a sensor to the server. You will be guided through the initialization (choose device type and name) and you can read off all relevant information for this device in the corresponding terminal

3. Rule Management and Rule Engine

Rule Management via config.py file → configure the system to **your city's needs** and implement **your own smart rules** for the devices

➔ **Implement your rules**
Using arbitrary python functions and your devices' data in the updates section of the config.py file. This way, also complicated logic can be integrated into the rules

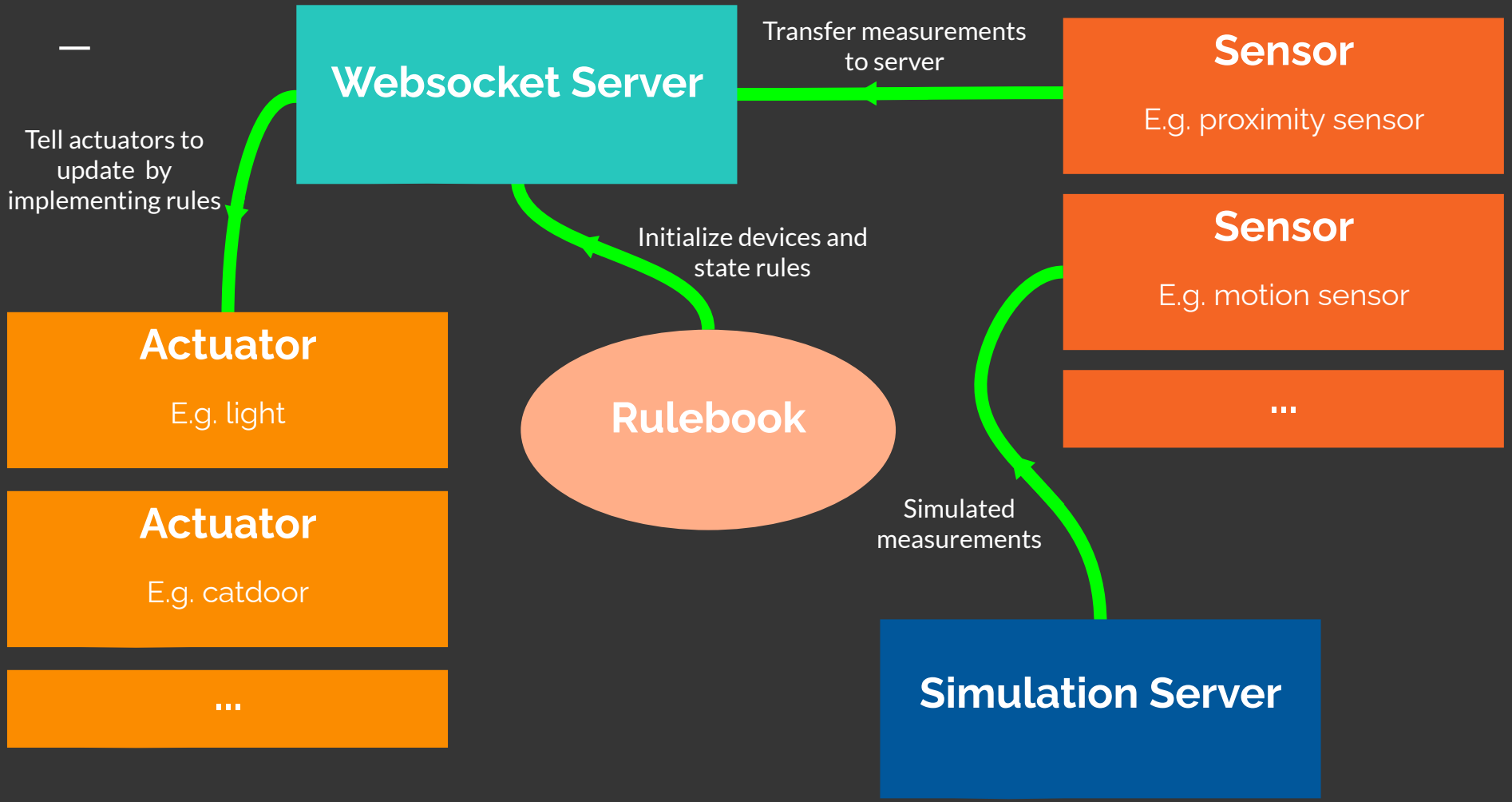
➔ **Server will run your rules and update the devices**
And you can add more devices to the server at any time

➔ **Sample config.py file shows a possible configuration for the rulebook**

4. Simulation Data

Separate websocket server to simulate devices and agents in a city

- ➔ **Simple quadratic grid with adjustable size as city environment**
  Agents and devices are places into this grid
- ➔ **Randomly walking agents generate sensor data**
  Random walk of adjustable number of agents, distance and other measurements can be performed in this grid environment and this simulated data can be used by the sensors
- ➔ **Devices can communicate with this simulation server**
  To receive measurement data or to update their status for the simulation
- ➔ **Soon: Visualization of simulated city on city_server in browser**


Websocket Server
Transfer measurements to server
Sensor
E.g. proximity sensor
Tell actuators to update by implementing rules
Initialize devices and state rules
Sensor
E.g. motion sensor
Actuator
E.g. light
Rulebook
...
Actuator
E.g. catdoor
Simulated measurements
...
Simulation Server