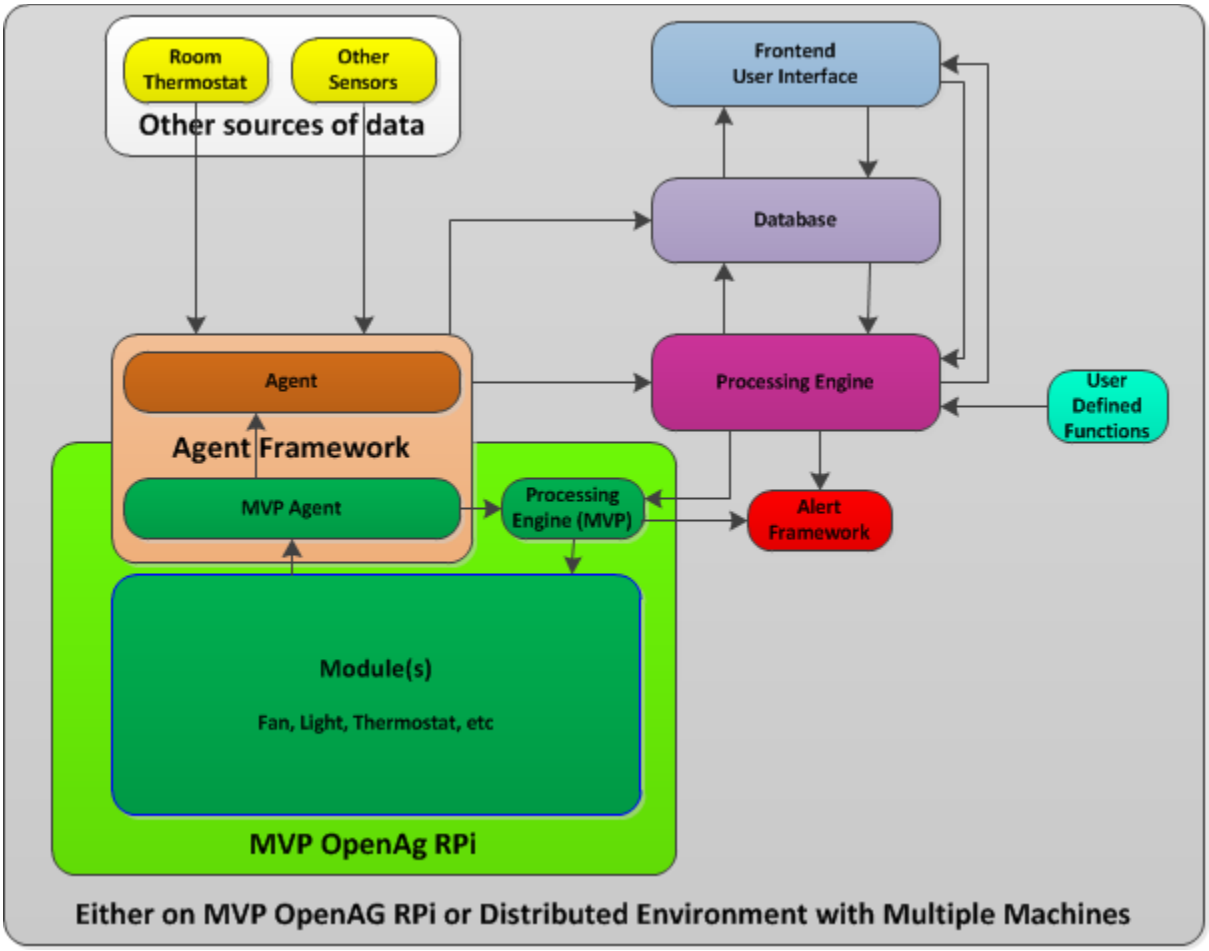
Distributed OpenAG MVP Environment



Distributed OpenAG MVP Environment

The components below can either be contained inside one OpenAG MVP environment or distributed across multiple OpenAg MVPs.

- Frontend User Interface
- Database
- Main Processing Engine
- Alert Framework
- User Defined Functions
- OpenAg MVP
    - Processing Engine
    - Module(s)
    - MVP Agent
- Agent Framework
- Other sources of data

Components of a Distributed OpenAG MVP Environment

Frontend User Interface

- Will be the web frontend for the MVP. It can display data from a single or multiple MVPs
- Inputs
    - Processing Engine (Main) – send information directly to web frontend
    - Database – read data to display on Front-End
- Outputs
    - Processing Engine (Main) – send commands to engine for execution
    - Database – write configuration information for MVP(s)

Database

- Used to store the historical logs from the MVP(s) and provide the configuration and rules for them as well.
- Inputs
    - Frontend User Interface
    - Agent Framework
    - Processing Engine (Main)
- Outputs
    - Processing Engine (Main)
    - Agent Framework

Main Processing Engine

- Provides a set of rules to each of the MVP(s) and controls them based on inputs of data from multiple sources.
- Inputs
    - User Defined Functions
    - Database
    - Frontend User Interface
    - Agent Framework
- Outputs
    - Database
    - Alert Framework
    - Processing Engine (individual MVP)

Alert Framework

- Provides a framework to send Alerts relating to the OpenAG MVP environment to the user.
- Inputs
    - Processing Engine (All)
- Outputs
    - External messaging (Telegram, Email, etc – Modules will provide this servers and be configured through the frontend user interface

User Defined Functions

- Functions be used in the Main Processing Engine
- Inputs
    - None
- Outputs
    - Processing Engine (Main)

MVP

- The individual MVP used to grow plants, and must be able to run independently from the other components. It must be able to cache data going to the database, rules from the main processing engine and alerts.

Processing Engine

- The main brain of the MVP. Controls the modules based on a set of rules and the data from sensors within the MVP. Listens to a heartbeat from the Main Processing Engine and provides one as well.
- Inputs
    - Agent Framework
    - Processing Engine (Main)
- Outputs
    - Alert Framework
    - Modules

Module(s)

- Individual modules to control items within the MVP and provide data from the MVP. Each will have a check_config function and json rules file so that the processing engine can ask it if the configuration is correct. They will also provide a list of items they can do, format of the data and baseline of data. For instance the temperature should be somewhere between 15 and 30 degrees Celsius anything outside of that should be flagged.
- Inputs
    - Processing Engine (local)
- Outputs
    - Agent Framework (local MVP Agent)

MVP Agent

- This will be a MQTT environment (RabbitMQ). There will be two queues Stats and Commands
- Inputs
    - MVP Modules (local)
- Outputs
    - Processing Engine (local)
    - Processing Engine(main)
    - Database

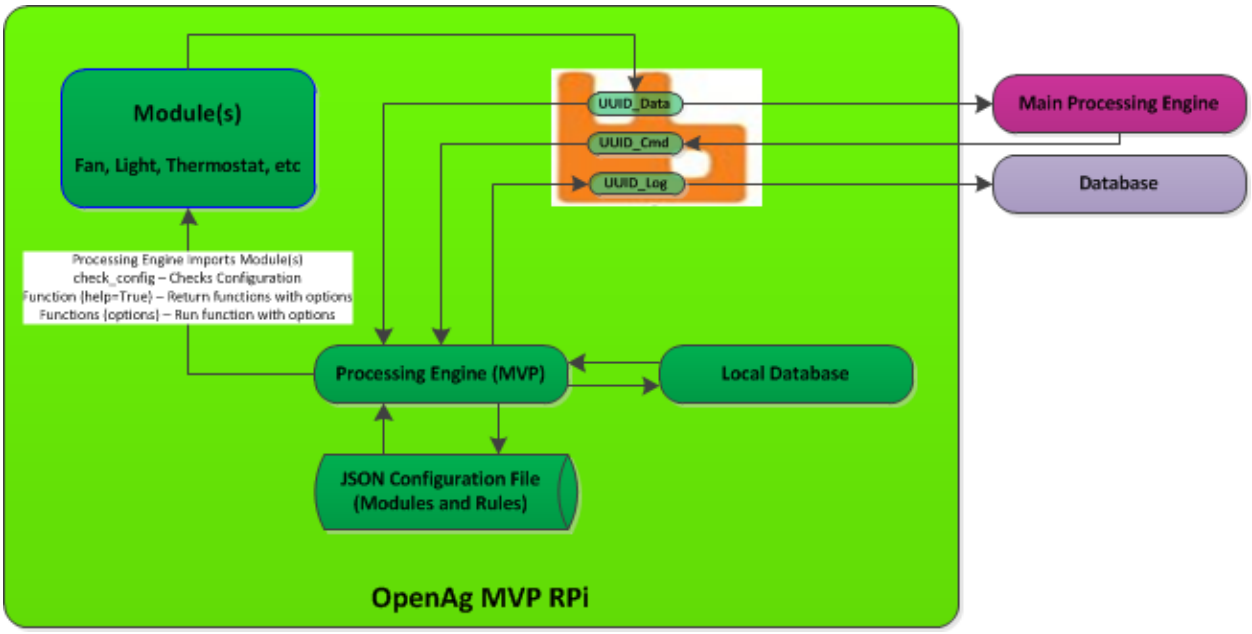
Agent Framework

- The queue will be a MQTT environment (RabbitMQ). There will be two queues Stats and Commands
- Inputs
    - MVP Modules
    - External Modules
- Outputs
    - Processing Engine (Main)
    - Database

Other sources of data

- External sources of data that can help the MVP better function (ie Room Temperature or Thermostat information)
- Inputs
    - None
- Outputs
    - Agent Framework

Modules



Additional Requirements

- module.json (file)
    - A json file named the same as the module name must three main keys
        - Mandatory (mandatory fields)
        - Optional (optional fields)
        - Rest (anything else)
    - Each main key will contain the following keys

        - warn (Boolean) – Whether it should send an warning flag if check fails
        - error (Boolean) – Whether it should send an error flag if check fails
        - check (dict)
    - The check key displays what it should check against the main configuration file and is a dictionary with the following:
        - Key
        - What to Check (but only these conditions)
            - type – what type it should be
                - ie. “Key” : { “type”: “bool”}
            - list – will contain the type check for each item in the list it will make sure it is all the same type as specified. For instance must be all of one type. No string, with integer or float, or vice versa
                - ie . “Key” : { “list”: { “type”: “bool”}}
            - dict – will contain the type check for each key in the dict it will make sure it is all the same type as specified. For instance must be all of one type. No string, with integer or float, or vice versa
                - ie . “Key” : { “dict”: { “type”: “bool”}}
            - dict(list) – will follow the list check
                - ie . “Key” : { “dict”: { “list”: { “type”: “bool”}}}

Required Functions

- check_config – This function uses the module’s json file to check the configuration for each item of that type.
- get_state – This returns the state of the module.
- set_state – This allows you to set the state of the module.