uPyEasy
Architecture Guidelines

1 Executive Summary

MicroPython is a software implementation of the Python 3 programming language, written in C, that is optimized to run on a microcontroller. MicroPython is a full Python compiler and runtime that runs on the micro-controller hardware. The user is presented with an interactive prompt (the REPL) to execute supported commands immediately. Included are a selection of core Python libraries, MicroPython includes modules which give the programmer access to low-level hardware.

MicroPython supports a number of ARM based architectures and has since been run on Arduino, ESP8266, ESP32, and Internet of things hardware.

uPyEasy (microPyEasy) is a free and open source MCU firmware for the Internet of things (IoT) and originally developed by the Lisa Esselink. It runs on any MicroPython platform. The name "uPyEasy" by default refers to the firmware rather than the hardware which runs. At a low-level the uPyEasy firmware works the same as the NodeMCU firmware and also provides a very simple operating system on the micropython platform. The main difference between uPyEasy firmware and NodeMCU firmware is that the former is designed as a high-level toolbox that just works out-of-the-box for a pre-defined set of sensors and actuators. Users simply hook-up and read/control over simple web requests without having to write any code at all themselves, including firmware upgrades using OTA (Over The Air) updates.

The uPyEasy firmware can be used to turn any micropython platform into simple multifunction sensor and actuator devices for home automation platforms. Once the firmware is loaded on the hardware, configuration of uPyEasy is entirely web interface based.uPyEasy firmware is primarily used on micropython modules/hardware as a wireless WiFi sensor device with added sensors for temperature, humidity, barometric pressure, LUX, etc. The uPyEasy firmware also offers some low-level actuator functions to control relays.

The firmware is built on the micropython core which in turn uses many open source projects. Getting started with uPyEasy takes a few basic steps. In most cases micropython modules come with AT or NodeMCU LUA firmware, and you need to replace the existing firmware with the uPyEasy firmware by flashing the hardware with a python based flash tool to use it.

uPyEasy is initially re-using the webinterface from ESPEasy, but not the code since ESPEasy is C-based and uPyEasy is python based. Many of the concepts used in uPyEasy, like asynchronicity and a database to store information, are different from ESPEasy.

uPyEasy
Architecture Guidelines

2 Version History

Date	Version	Comments
15-01-2018	0.1	Initial draft
10-04-2018	0.2	Added plugin queues

3 Content

4 Purpose

The uPyEasy firmware can be used to turn the micropython module into an easy multifunction sensor device for Home Automation solutions like Domoticz. Configuration of uPyEasy is entirely web based, so once you've got the firmware loaded, you don't need any other tool besides a common web browser.

Where ESPEasy is designed for the ESP8266 SOC (System On a Chip), is uPyEasy designed for the micropython platform. Currently the micropython platform is supporting dozens of SOC's, and so does uPyEasy.

Due to it's rapid Agile development cycles, is uPyEasy primarily developed on the unix platform, while the STM32F405RGT, Pyboard, ESP32, ESP8266 and Pi platforms are heavily used for testing purposes.

These uPyEasy Architecture Guidelines are written in accordance to the Togaf 9.1 ADM.

5 Scope

The scope of uPyEasy:

- MicroPython based
- Multiple platforms (basically all platforms supported by micropython)
- Lan/WLan
- Support for GPIO, I2C, SPI, UART, I2S and CAN interface protocol
- Support for a SD card
- Support of psRam
- OTA firmware updates

6 Goals, Objectives, and Constraints

6.1 Goals

- Written in MicroPython
- Multiple platforms (basically all platforms supported by micropython)
- Basic testing/development done on:
    - Linux
    - STM32F405RGT
    - PyBoard
    - ESP32
    - ESP8266
- Async webserver
- Threads when possible
- Template based webpages
- Lan/WLan transparency
- Support for GPIO, I2C, SPI, UART, I2S and CAN interface protocol
- SSL support both for webserver and client
- Maximum reuse of libraries
- Rules in python
- Override in plugins (custom plugins overrule standard plugins)
- Plugin drag 'n drop (limited in smaller soc's due to memory constraints)
- Support of ANY library that is supported in micropython
- Support of viper speed
- Dynamic support of SD card
- Support of psRam
- Use of FileDB to store config and other values
- Bootloader supporting OTA firmware updates (Yaota8266, 247KB), meaning 1MB esp's will have 750KB free at most when using OTA.
- i18n support

6.2 Constraints

To prevent that uPyEasy is spread thin over to many platforms for to many purposes, leading to uncessary low quality and development times, it's necessary to limit the purpose using this scope.

The scope is:

- Only platforms supported by micropython are supported by uPyEasy
- Only the Python derative MicroPython is used
- A micropython platform must have support for the uPyEasy supported interface protocol like I2C
- A micropython platform must have tcp/ip support, either WiFi or Ethernet.

7 Architecture Principles

Architecture principles are used to steer the development of uPyEasy so that the development doesn't get make solutions, or get lost in solving problems, that are not needed.

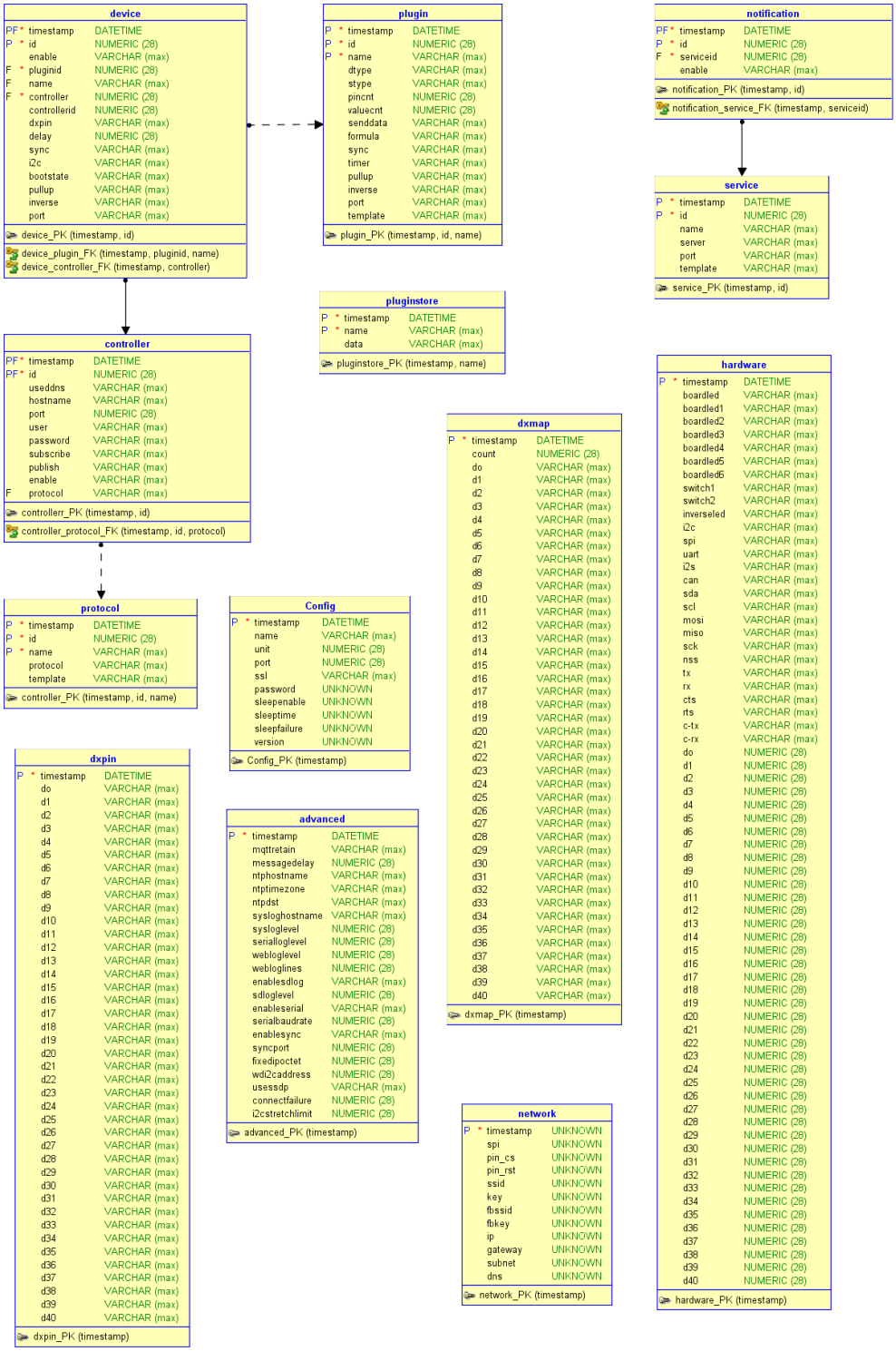
The principles are:

- All major processes are implemented in async processes.
- There are no more threads as there are cores in a platform
- When a process is thread based, it will use a separate async task scheduler for each thread
- When having multiple threads, one of the threads is used solely for the Plugin handler process.
- Communication between major processes are done using async queues
- All persistent values are stored in a database
- All regular modules are 'frozen', frozen modules allow you to "bake in" code into the firmware image you flash onto the device.

8 Baseline Architecture

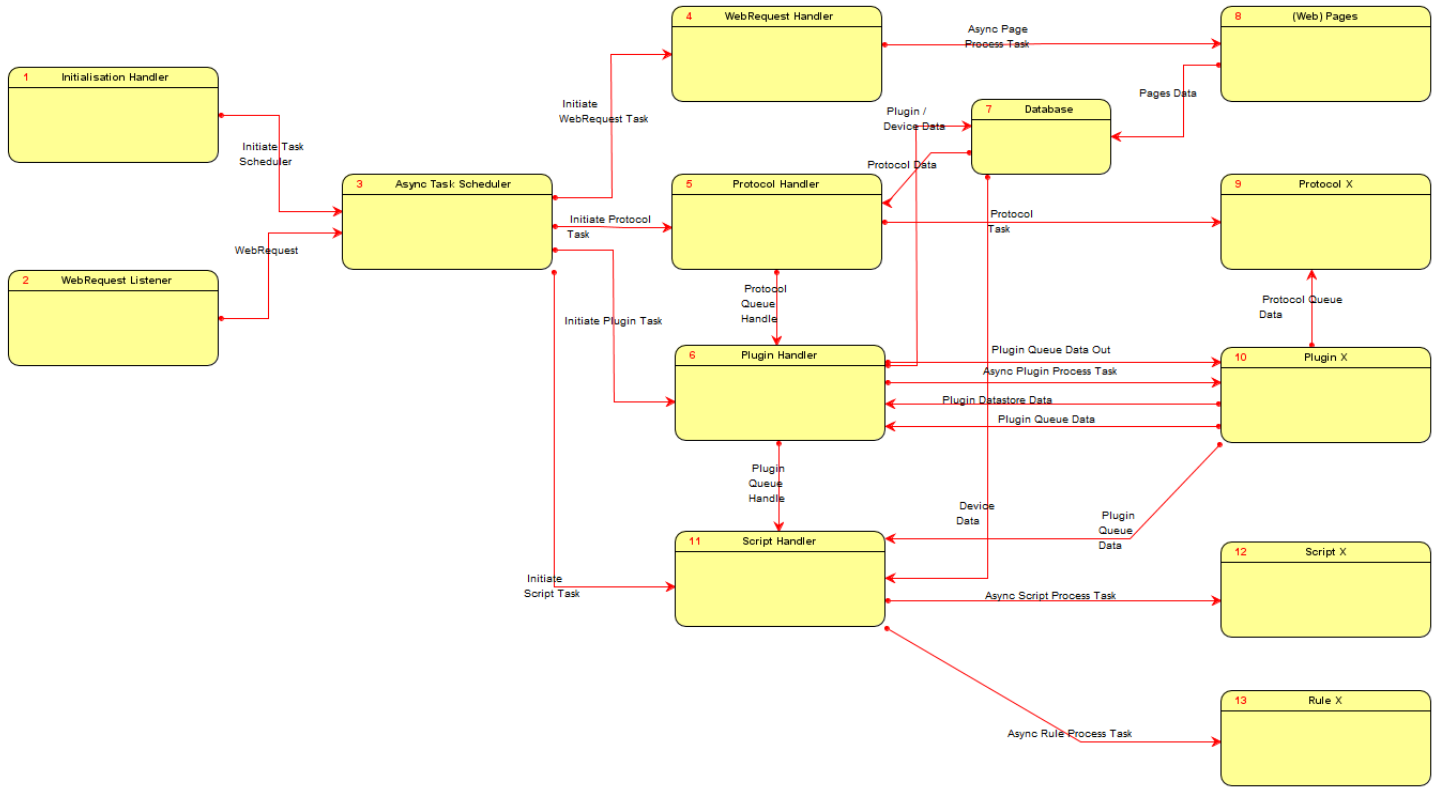
8.1 Data Architecture Model

The data entities and attributes and their DRD.



8.2 Application Architecture Models

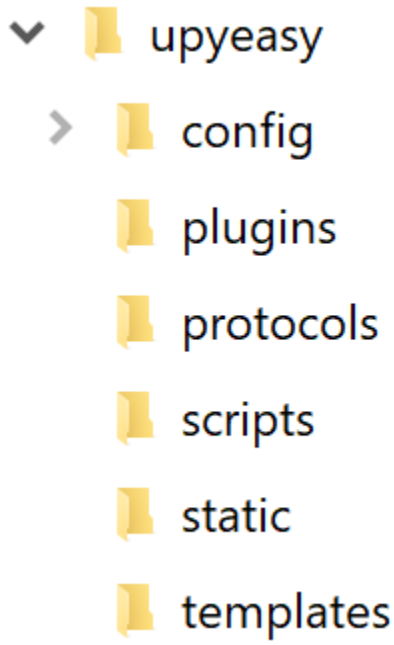
All (major) processes are connected in a data flow diagram:



8.3 Technology Architecture Model

8.3.1 Frozen uPyEasy modules

uPyEasy is using the following (frozen) directory structure to store it's frozen modules (python files):



In the main upyeasy directory are all main processes, python files, stored.

The config directory is where the, filedb based, database record files are stored. Each tables is a subdirectory of config, where each record is a file in the subdirectory.

In the plugins directory are all plugins stored, each plugin can be switch on or off in the _init_.py file before it's frozen.

In the protocols directory are all protocols stored, each protocol can be switch on or off in the _init_.py file before it's frozen.

In the scripts directory are all scripts stored, each scripts can be switch on or off in the _init_.py file before it's frozen.

The static directory is used to store static files needed for the webserver like the stylesheet of javascript files.

The templates directory is used to store all compiled html templates, only compiled html templates (which have the .py extension) are frozen.

8.3.2 Flash uPyEasy modules

In the main upyeasy directory the SSL cert and key files are stored in this directory.

In the plugins directory are plugins stored, these plugins can be accessed from the files explorer. Since it's flash, dynamic change of the plugin is possible.

In the protocols directory are protocols stored, these protocols can be accessed from the files explorer. Since it's flash, dynamic change of the protocol is possible.

In the scripts directory are all scripts stored, these scripts can be accessed from the files explorer. Since it's flash, dynamic change of the script is possible.

