Fogget-about-it grow management system

Preparing the groundwork for the next agricultural revolution.

Problem definition:

- Over the last few years, there has been research into the cultivation technique known as “fogponics”, a high-intensity form of agriculture which benefits more from automation than any other.

- This project provides a management and control system for a fleet of fogponics units that can be extended to other use-cases.

- This is in continuation of a project I have been running off and for half a year. The physical prototype is (mostly) built but lacks a control panel.

We got ourselves a control panel.

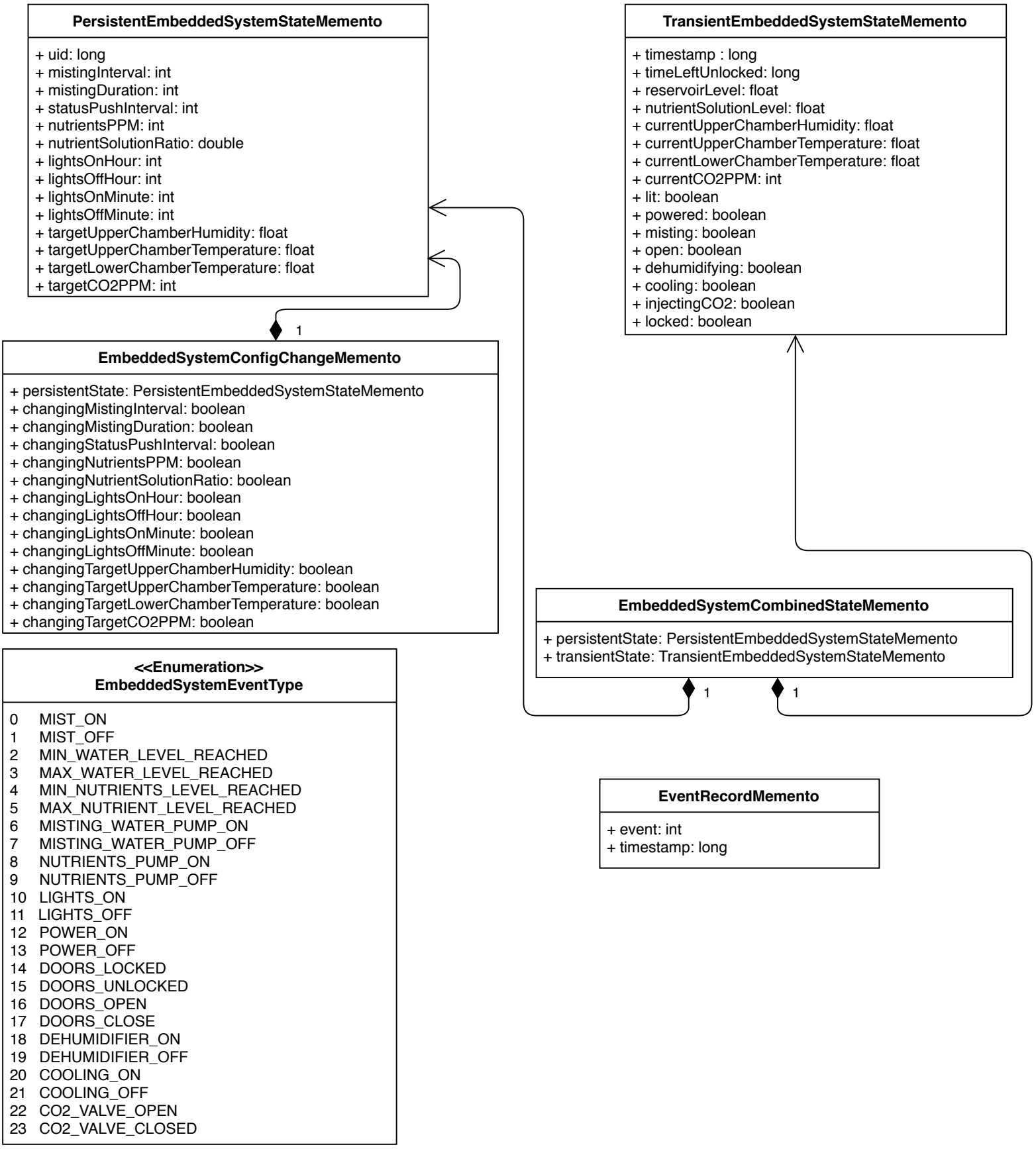
:)

Functional requirements:

- We're building a basic control system for an IoT use-case.
- The control system gathers and updates the current state information of the embedded systems growing plants.
- It must also be able to maintain a list of recent events for perusal by the end-user.
- New systems are to be automatically given sane defaults and added to the system pool. The only user responsibility must be sourcing the embedded system with a unique 64-bit integer ID.
- The end-user must be able to change settings.
- These settings must be validated in order to correct against nonsense inputs.
- "Computer, set the rooting chamber to -273 degrees C…"
- The embedded system running the grow environments should do their its own validation as good engineering, but layers of safety protect against screwups downstream.
- In effect, it acts as an application firewall.
- That's basically it as far as *functional* requirements go.

Design patterns:

- The main design pattern in use here is the memento pattern.
- Memento allows you to use state to represent logical items for recall when convenient.
- It is also really good for indicating your next intentions.
- Literally stands for “memory aid.”
- Also used in the web UI are builder patterns, but their use is perfunctory...
- Reusing code to write HTML tags to a page more neatly.
- Aside: I really wanted to keep it unix by using a simple command-line app, but we went that way instead and deep down I’m kinda glad I stepped out of my bubble.
- We also use a static factory method or two.
- However, they’re nothing too interesting as far as our use-case is concerned. They’re also completely perfunctory and don’t really warrant too much discussion.
- They’d be far more central for our purposes and worth talking about if we were writing a serializer, for example...


PersistentEmbeddedSystemStateMemento
+ uid: long
+ mistingInterval: int
+ mistingDuration: int
+ statusPushInterval: int
+ nutrientsPPM: int
+ nutrientSolutionRatio: double
+ lightsOnHour: int
+ lightsOffHour: int
+ lightsOnMinute: int
+ lightsOffMinute: int
+ targetUpperChamberHumidity: float
+ targetUpperChamberTemperature: float
+ targetLowerChamberTemperature: float
+ targetCO2PPM: int
TransientEmbeddedSystemStateMemento
+ timestamp : long
+ timeLeftUnlocked: long
+ reservoirLevel: float
+ nutrientSolutionLevel: float
+ currentUpperChamberHumidity: float
+ currentUpperChamberTemperature: float
+ currentLowerChamberTemperature: float
+ currentCO2PPM: int
+ lit: boolean
+ powered: boolean
+ misting: boolean
+ open: boolean
+ dehumidifying: boolean
+ cooling: boolean
+ injectingCO2: boolean
+ locked: boolean
1
EmbeddedSystemConfigChangeMemento
+ persistentState: PersistentEmbeddedSystemStateMemento
+ changingMistingInterval: boolean
+ changingMistingDuration: boolean
+ changingStatusPushInterval: boolean
+ changingNutrientsPPM: boolean
+ changingNutrientSolutionRatio: boolean
+ changingLightsOnHour: boolean
+ changingLightsOffHour: boolean
+ changingLightsOnMinute: boolean
+ changingLightsOffMinute: boolean
+ changingTargetUpperChamberHumidity: boolean
+ changingTargetUpperChamberTemperature: boolean
+ changingTargetLowerChamberTemperature: boolean
+ changingTargetCO2PPM: boolean
EmbeddedSystemCombinedStateMemento
+ persistentState: PersistentEmbeddedSystemStateMemento
+ transientState: TransientEmbeddedSystemStateMemento
1
1
<<Enumeration>>
EmbeddedSystemEventType
0 MIST_ON
1 MIST_OFF
2 MIN_WATER_LEVEL_REACHED
3 MAX_WATER_LEVEL_REACHED
4 MIN_NUTRIENTS_LEVEL_REACHED
5 MAX_NUTRIENT_LEVEL_REACHED
6 MISTING_WATER_PUMP_ON
7 MISTING_WATER_PUMP_OFF
8 NUTRIENTS_PUMP_ON
9 NUTRIENTS_PUMP_OFF
10 LIGHTS_ON
11 LIGHTS_OFF
12 POWER_ON
13 POWER_OFF
14 DOORS_LOCKED
15 DOORS_UNLOCKED
16 DOORS_OPEN
17 DOORS_CLOSE
18 DEHUMIDIFIER_ON
19 DEHUMIDIFIER_OFF
20 COOLING_ON
21 COOLING_OFF
22 CO2_VALVE_OPEN
23 CO2_VALVE_CLOSED
EventRecordMemento
+ event: int
+ timestamp: long

Model-Controller-View:

- Views are sent out the backend and into a separate utility that decouples data being sent out from the rest of the backend program; views talk to that program and don’t even talk about our backend.
- Commands and state changes are sent via MQTT. These represent the controls and the in-flight representation of the model, respectively.
- The web UI uses a custom Servlet as a controller that kicks up the MQTT client the very first time it runs.
- The views are gathered and formatted by custom tag classes invoked by an otherwise logic-free JSP page.
- This represents proper modern practices.