

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.

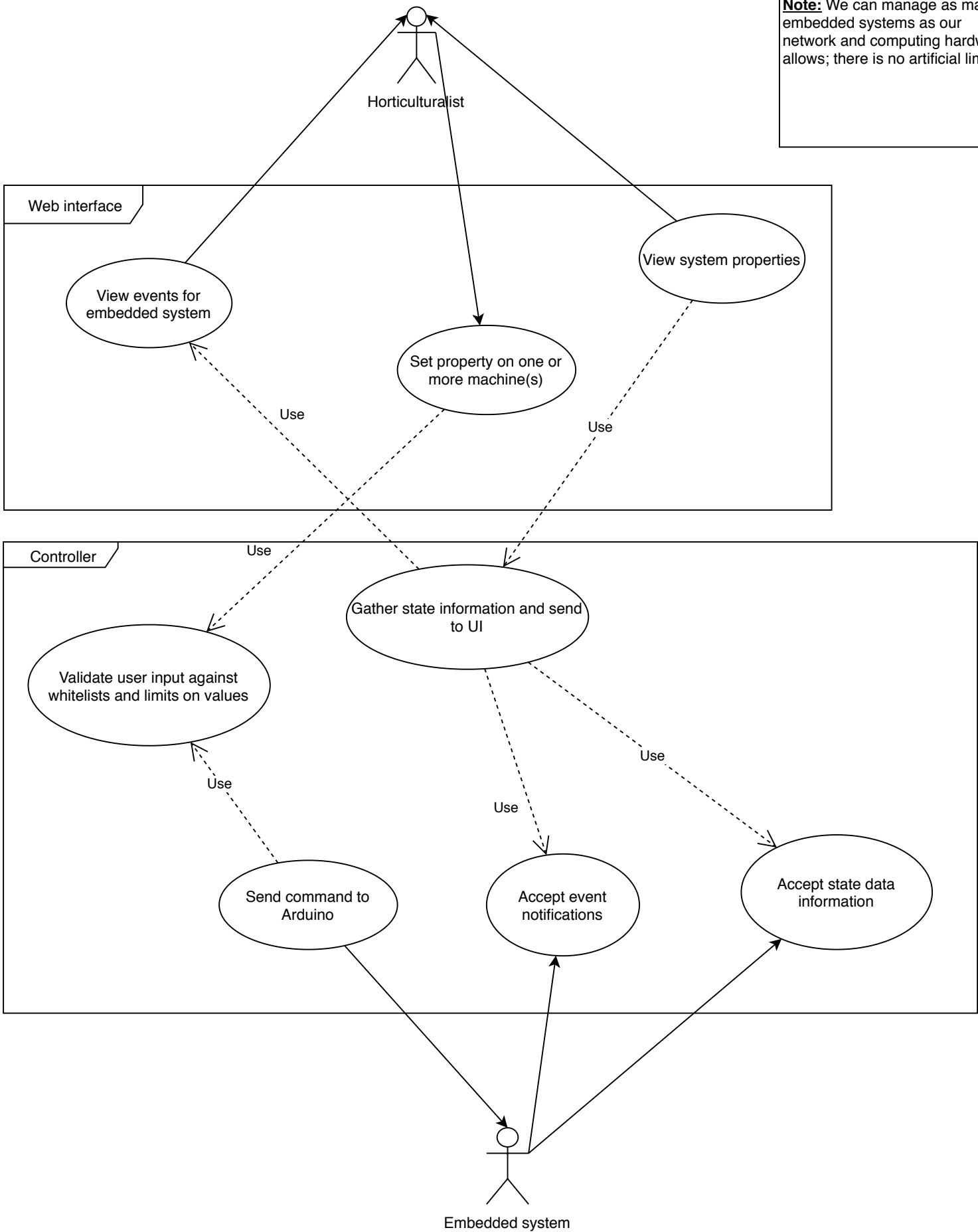
We got ourselves a control panel.

:)

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 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.

Note: We can manage as many embedded systems as our network and computing hardware allows; there is no artificial limit.



Horticulturalist (web browser, supervisory and/or control system)

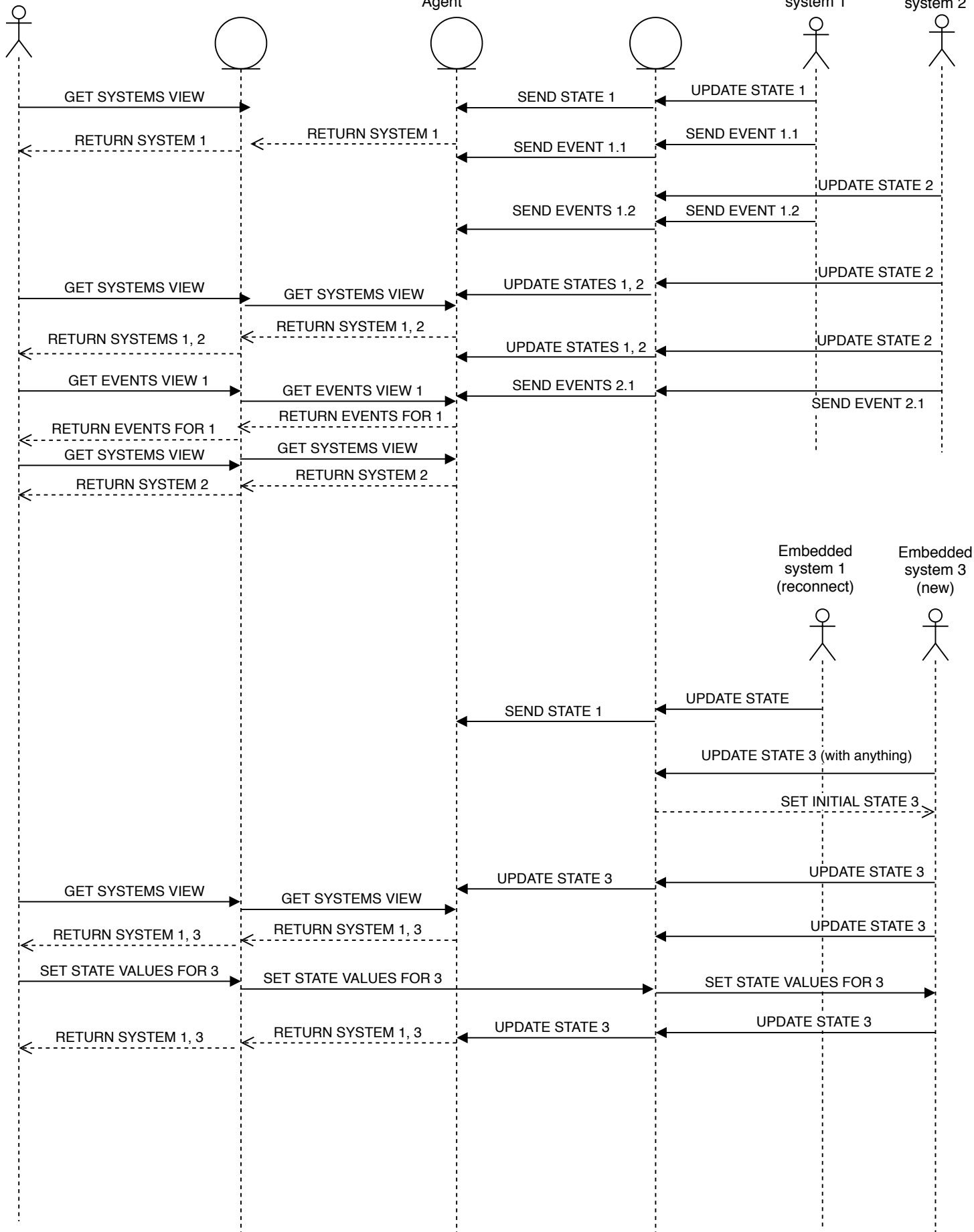
UI (web, ftp, telnet, etc)

UI Transfer Agent

Backend

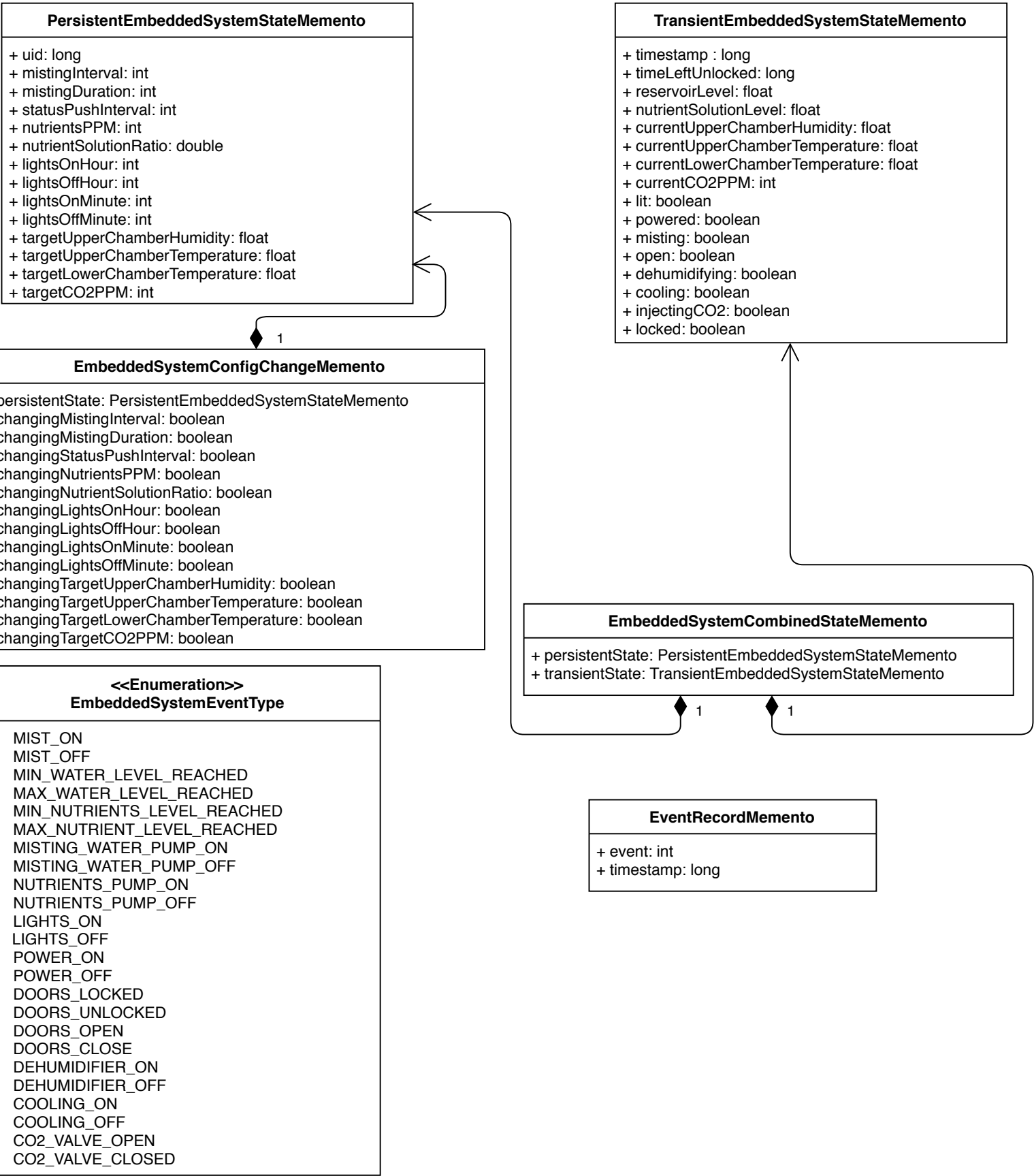
Embedded system 1

Embedded system 2



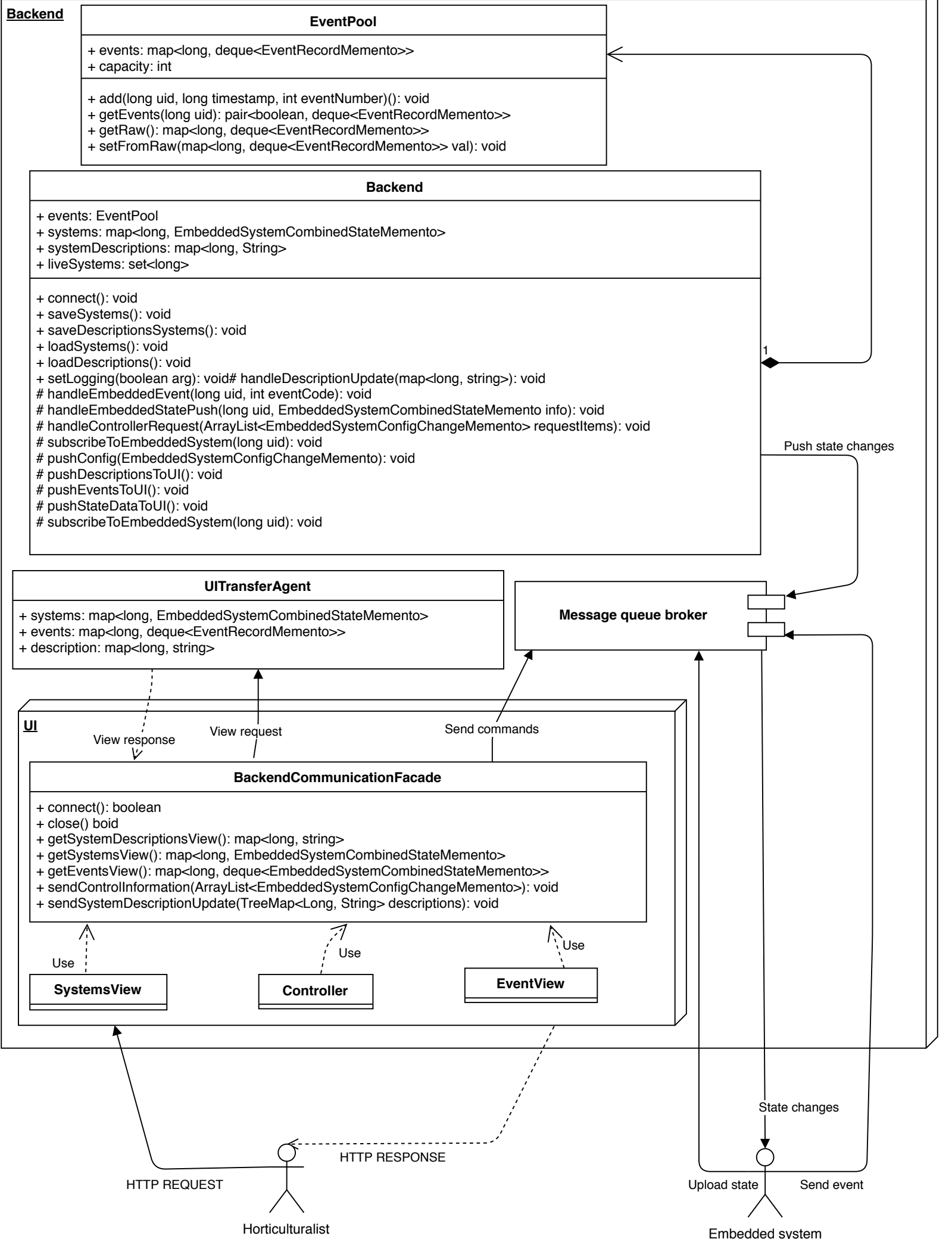
Design patterns:

- 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...



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.



How we store and transmit config changes:

```
public class
PersistentEmbeddedSystemStateMemento {
    private long uid;
    private int mistingInterval;
    private int mistingDuration;
    private int statusPushInterval;
    private int nutrientsPPM;
    private double nutrientSolutionRatio;
    private int lightsOnHour;
    private int lightsOffHour;
    private int lightsOnMinute;
    private int lightsOffMinute;
    private float targetUpperChamberHumidity;
    private float
targetUpperChamberTemperature;
    private float
targetLowerChamberTemperature;
    private int targetCO2PPM;
}
```

How we (temporarily) store and transmit variance in machine status:

```
public class
TransientEmbeddedSystemStateMemento {
    private long timestamp;
    private long timeLeftUnlocked;
    private float reservoirLevel;
    private float nutrientSolutionLevel;
    private float currentUpperChamberHumidity;
    private float
currentUpperChamberTemperature;
    private float
currentLowerChamberTemperature;
    private int currentCO2PPM;
    private boolean lit;
    private boolean powered;
    private boolean misting;
    private boolean open;
    private boolean dehumidifying;
    private boolean cooling;
    private boolean injectingCO2;
    private boolean locked;
}
```

Events and their representation:

```
public class EventRecordMemento {
    private int event;
    private long timestamp;
}

public enum EmbeddedSystemEventType {
    MIST_ON,
    MIST_OFF,
    MIN_WATER_LEVEL_REACHED,
    MAX_WATER_LEVEL_REACHED,
    MIN_NUTRIENTS_LEVEL_REACHED,
    MAX_NUTRIENT_LEVEL_REACHED,
    MISTING_WATER_PUMP_ON,
    MISTING_WATER_PUMP_OFF,
    NUTRIENTS_PUMP_ON,
    NUTRIENTS_PUMP_OFF,
    LIGHTS_ON, LIGHTS_OFF,
    POWER_ON, POWER_OFF,
    DOORS_LOCKED,
    DOORS_UNLOCKED,
    DOORS_OPEN,
    DOORS_CLOSE,
    DEHUMIDIFIER_ON,
    DEHUMIDIFIER_OFF,
    COOLING_ON,
    COOLING_OFF,
    CO2_VALVE_OPEN,
    CO2_VALVE_CLOSED
}
```

How we transmit state changes:

```
public class EmbeddedSystemConfigChangeMemento {
    private PersistentEmbeddedSystemStateMemento
    persistentState = new
    PersistentEmbeddedSystemStateMemento();
    private boolean changingMistingInterval = false;
    private boolean changingMistingDuration = false;
    private boolean changingStatusPushInterval = false;
    private boolean changingNutrientsPPM = false;
    private boolean changingNutrientSolutionRatio = false;
    private boolean changingLightsOnHour = false;
    private boolean changingLightsOffHour = false;
    private boolean changingLightsOnMinute = false;
    private boolean changingLightsOffMinute = false;
    private boolean changingTargetUpperChamberHumidity =
    false;
    private boolean changingTargetUpperChamberTemperature =
    false;
    private boolean changingTargetLowerChamberTemperature =
    false;
    private boolean changingTargetCO2PPM = false;

    public void setNoChanges() {
        setAll(false);
    }

    public void changeAll() {
        setAll(true);
    }
}
```

```

protected void setAll(boolean arg) {
    changingMistingInterval
        = changingMistingDuration
        = changingStatusPushInterval
        = changingNutrientsPPM
        = changingNutrientSolutionRatio
        = changingLightsOnHour
        = changingLightsOffHour
        = changingLightsOnMinute
        = changingLightsOffMinute
        = changingTargetUpperChamberHumidity
        = changingTargetUpperChamberTemperature
        = changingTargetLowerChamberTemperature
        = changingTargetCO2PPM = arg;
}

```

```

public boolean hasChanges() {
    return changingMistingInterval
        || changingMistingDuration
        || changingStatusPushInterval
        || changingNutrientsPPM
        || changingNutrientSolutionRatio
        || changingLightsOnHour
        || changingLightsOffHour
        || changingLightsOnMinute
        || changingLightsOffMinute
        || changingTargetUpperChamberHumidity
        || changingTargetUpperChamberTemperature
        || changingTargetLowerChamberTemperature
        || changingTargetCO2PPM;
}

```

How we validate this thing:

```
public class EmbeddedStateChangeValidator {
    /**
     *
     * @author noob
     * The integer arguments are to be able to use an existing
     lights-on hour to test against.
     * Intended usage is for the case in which the backend is
     told that the user wants to change either only the minute
     or the hour setting.
     * That means it can then pull up the existing value from
     already-known information and calculate,
     */
    public static EmbeddedSystemConfigChangeMemento
    validate(EmbeddedSystemConfigChangeMemento arg, int
    currentLightsOnHour, int currentLightsOnMin, int
    currentLightsOffHour, int currentLightsOffMin) {
        EmbeddedSystemConfigChangeMemento req = arg;
        boolean settingLightsOnHour = false;
        boolean settingLightsOffHour = false;
        boolean settingLightsOnMinute = false;
        boolean settingLightsOffMinute = false;
        final int mistingInterval =
    req.getPersistentState().getMistingInterval();
        final int mistingDuration =
    req.getPersistentState().getMistingDuration();
        final int lightsOnHour =
    req.getPersistentState().getLightsOnHour();
        final int lightsOnMinute =
    req.getPersistentState().getLightsOnMinute();
        final int lightsOffHour =
    req.getPersistentState().getLightsOffHour();
        final int lightsOffMinute =
    req.getPersistentState().getLightsOffMinute();
```



```

// Validating time
if (req.hasChanges()) {
    if (req.isChangingLightsOnHour()) {
        settingLightsOnHour = true;
    }

    if (req.isChangingLightsOnMinute()) {
        settingLightsOnMinute = true;
    }
    if (req.isChangingLightsOffHour()) {
        settingLightsOffHour = true;
    }
    if (req.isChangingLightsOffMinute()) {
        settingLightsOffMinute = true;
    }
    boolean validOnTime = false;
    if (settingLightsOnHour &&
settingLightsOnMinute) {
        validOnTime =
TimeOfDayValidator.validate(lightsOnHour, lightsOnMinute);
    } else if (settingLightsOnHour) {
        validOnTime =
TimeOfDayValidator.validate(lightsOnHour,
currentLightsOnMin);
    } else if (settingLightsOnMinute) {
        validOnTime =
TimeOfDayValidator.validate(currentLightsOnHour,
lightsOnMinute);
    }
    if (!validOnTime) {
        req.setChangingLightsOnHour(false);
        req.setChangingLightsOnMinute(false);
    }
    boolean validOffTime = false;
    if (settingLightsOffHour &&
settingLightsOffMinute) {
        validOffTime =
TimeOfDayValidator.validate(lightsOffHour,
lightsOffMinute);
    } else if (settingLightsOffHour) {

```

```

        validOffTime =
TimeOfDayValidator.validate(lightsOffHour,
currentLightsOffMin);
    } else if (settingLightsOffMinute) {
        validOffTime =
TimeOfDayValidator.validate(currentLightsOffHour,
lightsOffMinute);
    }
    if (!validOffTime) {
        req.setChangingLightsOffHour(false);
        req.setChangingLightsOffMinute(false);
    }
    // Validating misting interval
    if (req.isChangingMistingInterval() &&
(mistingInterval > CommonValues.maxMistingInterval ||
mistingInterval < CommonValues.minMistingInterval)) {
        req.setChangingMistingInterval(false);
    }
    // Validating misting duration
    if (req.isChangingMistingDuration() &&
(mistingDuration > CommonValues.maxMistingDuration ||
mistingDuration < CommonValues.minMistingDuration)) {
        req.setChangingMistingDuration(false);
    }
    // Validating solution ratio of nutrients vs
water
    final double solutionRatio =
req.getPersistentState().getNutrientSolutionRatio();
    if (req.isChangingNutrientSolutionRatio() &&
(solutionRatio > CommonValues.maxNutrientSolutionRatio ||
solutionRatio < CommonValues.minNutrientSolutionRatio)) {
req.setChangingNutrientSolutionRatio(false);
    }

```

```

        // Validating humidity
        final float humidity =
req.getPersistentState().getTargetUpperChamberHumidity();
        if (req.isChangingTargetUpperChamberHumidity()
&& (humidity > CommonValues.maxHumidity || humidity <
CommonValues.minHumidity)) {

req.setChangingTargetUpperChamberHumidity(false);
        }
        // Validating temperature
        final float temperature =
req.getPersistentState().getTargetUpperChamberTemperature()
;

        if
(req.isChangingTargetUpperChamberTemperature() &&
(temperature > CommonValues.maxTargetTemperature ||
temperature < CommonValues.minTargetTemperature)) {

req.setChangingTargetUpperChamberTemperature(false);
        }
        final int ppm =
req.getPersistentState().getTargetCO2PPM();
        // Validating target CO2 levels
        if (req.isChangingTargetCO2PPM()) {
            if (ppm < CommonValues.minCO2PPM || ppm >
CommonValues.maxCO2PPM) {
                req.setChangingTargetCO2PPM(false);
            }
        }
    }
    return req;
}
}

```

(Sorry.)

Minor point of interest:

- We also implicitly use an the object pooling pattern when storing events and system state: We replace state when changing it, using the same memory location.
- Also, the EventPool preallocates a deque and pops the element off the tail prior to inserting at the head.
- Does that count? Probably.
- (I know that didn't *absolutely* need to be discussed, but it was worth talking about.)

Tools used:

- Java 8 was used (OpenJDK)
- NetBeans was grudgingly appreciated.
- Speed tests used the Selenium project, a Python toolkit for web browser testing. (Tip: Setup a VM or dedicate a machine to work with this tool because otherwise using your desktop becomes very difficult.)
- Jackson JSON library, version 3.
- If you ever use that library, ignore all the old tutorials. The new interface is so much simpler; time was definitely wasted fearing the wrath of what turned out to not exist anymore.
- Paho MQTT client library, Java implementation. Protocol version 3.1 kept in order to keep our potential replacement choices as open as possible and.
- Mosquitto MQTT broker, which is a very popular implementation.
- Git and Github were used constantly to ensure we didn't lose any valuable code that we may have stomped on.
- Servlets and custom JSP tags were used to create the web backend.
- Browser speed tests were run on everyone's favourite browser... Firefox!
- They gave us 1/3 second backend processing time, maximum.
- And before we forget, the "Solarized Dark" theme used on our web UI:

<http://thomasf.github.io/solarized-css/solarized-dark.css>

