Fogget-About-It Grow System, v.1

A management system for your next-gen fleet of fogponic growrooms

Colin Gilbert #200 254 322

In fulfillment of the requirements for CS 476: Software Development Project

University of Regina, Canada

November 21, 2019

Foreword and dedication:

When I use the singular “we”, it is not merely an effort to write academically for a capstone project. I consider this a continuation of my previous class project for Maher Elshankankiri’s Internet of Things course last semester, which was one of the most inspiring classes I’ve ever had the pleasure of attending.

Without my loving family and its funding, the financial burdens imposed by building such a complex project would have made all this a non-starter. We still warehouse a vast quantity of gear that I thought we’d need but won’t, and this particular inconvenience to them hadn’t yet been committed to writing until now. Their status as stakeholders and occasional team members in this particular endeavor has already been noted in my previous report last semester, but is very much worth repeating.

1. Motivation and problem Statement:

Fogponics horticulture is the use of ultrasonic misting devices to provide a nutrient-rich mist of about 5-30 microns into the roots of plants, with the aim of affecting increased growth rates. It is a form of aeroponics, in which plants are grown without a substrate. We aim to bring fogponics into the mainstream by making it easier to conduct, as it suffers from the disadvantage of requiring fine-tuning (too much fog stop nutrient absorption and kills the plants) and the deposition of fertilizer salts upon the ceramic discs of the fogging units. We aim to alleviate these problems by having the machine periodically flush out between feedings.

The author is in possession of one such tool, but found that it necessary to observe its status and give it schedules in an elegant manner. This project is an opportunity to fulfill these requirements and to share these results to the public. It was also a result of questioning how best to help others with our knowledge of computers – computer science graduates form an elite among employees, which generates class resentment: We have a social obligation to offer something back if we wish to avoid such destructive ill-will, at the very least until our societies smarten up and offer opportunity and dignity to everybody.

2. Feasibility Study:

Fogponics was chosen due to low water and nutrient consumption, hygiene, nutrient control, and the small, spherical, fuzzy roots established by fogponics-grown plants offering low use of space. The water consumption issue is likely the most important - the world is heading towards an electricity-cheap and freshwater-scarce economy. Fogponics combined with water recapture offer a system that is extremely water-efficient. There is ample reason for such technology to be used on a wider scale, possibly replacing other forms of horticulture in certain markets as producers are forced to use less and less water: Our system will allow them to adapt their operations to an extent that is impossible with traditional methods. With fogponics, nutrient absorption is far more efficient and as such the use of expensive fertilizers is greatly reduced; we suffer zero runoff.

Among other benefits, shared by all enclosed grow-rooms, is the ability to create a specific environment to favour certain aspects of plant growth: This allow for greater repeatability and makes quality control much easier. In a crowded marketplace, a brand can benefit greatly from such advantages. Nutrient profiles can also be calculated and dosed traditionally as with hydroponics and soil-based methods, but fogponics takes this to another level – changes in plants become apparent extremely soon after dosage changes. This can allow a horticulturalist to tweak parameters and force certain traits in a plant far more easily than when she relied upon a substrate containing plenty of nutrient leftovers from previous feeds – a crop can be raised in a manner that is optimal for plant growth right before changing growth stages or adding specialist nutrient profiles for the consumer; unneeded nutrients can harm produce quality or result in less crop.

There exists prior effort towards automated plant growing: MIT's OpenAg initiative (1) is already on its third iterations of its open-source *Personal Food Computer.* On the other side of the Atlantic we have Becker, Caddell, and Gutierrez from Worcester Polytechnic Institute (2) : They have created a working fogponics system for growing vegetable greens which they call the *Integrated Farming System*. However, the systems above seem to lack any real integration for a fleet of such machines run by a central control system, which is necessary to meet the needs of even a single urban family: Space constraints make it very difficult for city-dwellers to dedicate vast, superlatively well-lit space for crop production. As such, greater quantities of smaller machines are likely to be their main mode of use. We propose software that allows the user to coordinate the schedules of multiple fogponics system, spread out across the home, in order to fulfill the need for fresh greens, high-nutrition berries, and medical crops year-round.

The author believes that the practices surrounding the use of fogponics are almost ready for prime-time, and posits that once a workable open-sourced version is refined it will soon find use in both home and industry. The scope of this project is to integrate with an existing physical prototype currently in advanced development and to provide a management platform for future devices, using standardized messaging format to accommodate other modes of production and future upgrades.

(1) https://www.media.mit.edu/groups/open-agriculture-openag/overview/

(2) https://web.wpi.edu/Pubs/E-project/Available/E-project-042612-154246/unrestricted/MQP_-_IFS_report.pdf

A reader not yet introduced to this work should be aware that this report only covers the distributed control system – the embedded systems programming, the physical construction of growrooms, choice and connection of actuators and sensors, and any topics related directly to plant growth are all outside the scope of this particular document. Also, prior to be ready for serious use, our current system must be secured as part of an overall network architecture – this means setting HTTPS certificates, DNSSec, TLS on the message queue broker, firewalling, and monitoring. These tasks are the responsibility of a system integrator but in the future we intend on providing a secure-by-default implementation for home users. However, as stated prior this lies outside the scope of the immediate project.

3.a – Functional requirements

Our system must:

- Auto-configure upon assembly by providing sane defaults for newly-installed devices.
- Provide meaningful status messages.
- Modify lighting/nutrient schedules and save them for persistent use.
- Turn growroom systems on or off.
- Lock or unlock chamber doors.
- Notify the user if they must take action, such as refilling liquids or performing repairs.
- Provide an elegant and workable interface to perform the above tasks.
- Disallow invalid inputs.

3.b – Actors and use-cases

The first actor is the horticulturalist. This can either be a human being, or the machines delegated to administer on that person's behalf. The chief responsibilities for this role, regardless of silicon content, is to provide oversight of affairs and respond to any operational needs that may arise.

The second actor is our embedded system. The embedded system administers water, nutrients, and CO^2 to the plants based on a provided schedule. It also keeps temperature and humidity within acceptable margins and regulates power to the lights, if we have a system that uses artificial lighting. It also maintains control of any door locks present. Furthermore, it also signals state change for consumption by decision-makers.

3.c – Software qualities

Correctness:

- The system must return the most up-to-date values in order to assist decision-making.
- The backend must send to the embedded system the actual values given by the user, provided they pass quality checks.
- The web interface does not hold any state; it is updated every time with fresh information from the backend

Robustness:

- The system must not allow for unusually high or low values to be processed. Allowing unusual inputs could create hazards to both plant and person, or a mess on the floor.
- The system also must not crash if unexpected or illegal inputs are entered. This is important because an unexpected system shutdown may affect the growing process.

Timeliness:

- The system must display the webpage within two seconds of action being taken, as an unresponsive user interface makes adoption highly unlikely.

User Friendliness

- It must be clear to the user what units are being used by the application. If the user does not know what measurements are being used, they may unintentionally put in incorrect inputs or become unnecessarily confused.
- The interface should be easy to understand at a glance. As our potential users may not have a lot of experience with computers, the interface needs to be similar to other applications for familiarity. If the user is unable to understand the interface they will either not use our product, or make mistakes harmful to their interests.

4.a – Software Architecture:

The model-control-view architecture is ubiquitous to the point of seeming like a natural tendency of software, and once again our project fits within it very neatly: There are three programs in to our software system: The backend, a small program that assists in the one-way transfer of view information from the backend, and the web UI which consumes that information. We also provide a program of a simulated embedded system for operational testing. Here we clearly encapsulate within the model-controller-view architecture the various parts of our control system:

Model

- The state of each embedded system is maintained as a memento object and kept in-memory by the backend. Each time the state is received from an embedded system, it overwrites the previous in-memory memento object with a new one.
- Each memento object is composed of a transient state and a persistent. Transient state is collected by the backend from our embedded systems and is only ever destined to the horticulturalist but never offered to the embedded systems; as the name suggests, it is not intended for long-term storage. By contrast, persistent state can be modified is by the user and is both sent by and pushed to the embedded systems – it is also persisted to file on our backend. The direction of data transfer is a prime difference between these two sets of information.
- The backend system maintains a list of all recent events in addition to the memento objects representing the state of our embedded systems.

Controller

- The backend performs validation of the persistent data sent by the horticulturalist prior to consumption by the embedded system. It does not, however, perform validation on the transient state going in the other direction, as the user needs that data unmodified.
- The backend drives a loop pushing fresh state information and events to the one-way transfer agent.

View

- A web-based UI (3) is provided that currently runs on the same machine as our backend; it connects to our one-way transfer agent which receives state updates over a plain socket. The backend sends to that program all updates to state. It completely decouples the presentation from the application itself and provides a nearly-constant response time for all user queries.
- The helper program does not handle commands: The direction of data flow is one-way only. Everything that can effect a state change is done via our pub-sub protocol: This provides a clean line of separation between controls and views; commands are done over our pub-sub protocol.

Note: We would like to emphasize that although we provide a web interface, it is one of many potential interfaces that an operator can use. If required, we can harness the same messaging scheme in order to send information to any alternative consumer. For example we can implement, in increasing order or difficulty a command-line control application, periodic reports, live backups, federated management, and even online analytics that can assist in detecting, predicting, and preventing failure.

4.b Design patterns

As mention above, the main pattern in use within this is *memento*. State is preserved in these object and used to perform both updates and views on data. We have a five main forms of memento objects present: These include the previously-discussed *TransientEmbeddedSystemStateMemento* (1) and *PersistentEmbeddedSystemStateMemento* (2) and employs composition to create the *EmbeddedSystemCombinedStateMemento* (3) which is a composition of *(1)* and (2) and is used as the representation for in-memory storage in our backend. We also have the *EmbeddedSystemConfigChangeMemento* (4), a composition of a persistent state memento with a set of booleans that are used to configure on one end and verify on the other which variables to update. At last, we have an *EventRecordMemento* (5) which is stored in a double-ended queue upon receiving an event notification from any of the embedded systems. Here are the details of these constructs:

```
TransientEmbeddedSystemStateMemento:                            (1)
     long timestamp;
     long timeLeftUnlocked;
     float reservoirLevel;
     float nutrientSolutionLevel;
     float currentUpperChamberHumidity;
     float currentUpperChamberTemperature;
     float currentLowerChamberTemperature;
     int currentCO2PPM;
     boolean lit;
     boolean powered;
     boolean misting;
     boolean open;
     boolean dehumidifying;
     boolean cooling;
     boolean injectingCO2;
     boolean locked;
```

```
PersistentEmbeddedSystemStateMemento:                              (2)
     long uid;
     int mistingInterval;
     int mistingDuration;
     int statusPushInterval;
     int nutrientsPPM;
     double nutrientSolutionRatio;
     int lightsOnHour;
     int lightsOffHour;
     int lightsOnMinute;
     int lightsOffMinute;
     float targetUpperChamberHumidity;
     float targetUpperChamberTemperature;
     float targetLowerChamberTemperature;
     int targetCO2PPM;


EmbeddedSystemCombinedStateMemento:                                (3)
      PersistentEmbeddedSystemStateMemento persistentState;
      TransientEmbeddedSystemStateMemento transientState;


EmbeddedSystemConfigChangeMemento:                                 (4)
     PersistentEmbeddedSystemState persistentState;
     boolean 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;


EventRecordMemento:                                                (5)
     int event;
     long timestamp;
```

Each of these memento objects has a corresponding JSON representation for serialization, which is used as message and gets sent around the network.

Algorithms used:

The topics used to filter for status updates, event notifications, and commands are used as an integral part of the algorithm, so they necessitate mention prior to further discussion. These messages are sent over a pub-sub channel provided by an implementation of the MQTT protocol, an open-specification pub/sub protocol described at http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. In brief, every device has a client ID that represents it uniquely within its world, and it subscribes to the topics that it wishes to received. The broker allows new topics to be created on-the-fly for messages that need to sent to one system in specific. These are the message types we use here:

- `pushEventFromEmbedded`: This signals to the backend that an event has taken place inside an embedded system. These events will be discussed in further detailed later.
- `pushStatusToBackend`: This is sent to the backend to inform it of the current state of one of the embedded systems. These are sent periodically, at a rate determined by the network's capacity.
- `pushConfigToEmbedded`: These are sent to a specific embedded system to make it change settings.
- `updateDescriptionRequest`: These are sent by the user interface to update the description of a given system. These descriptions are stored in the backend and provide a friendly name for the user's convenience.
- `updateEmbeddedStateRequest`: These are sent by the user interface to the backend, prior to the backend sending a configuration push to an embedded system (or not, if the request has no valid values.)

Here we discuss the the most interesting part of the application, the backend: Collects and stores all the relevant information from the network of embedded systems and provides a line of defense from improper configuration from the user. Here is its structure:

```
Backend:

EventPool events // Will be discussed in the next few pages
map<long, EmbeddedSystemCombinedStateMemento> systems
map<long, String> systemDescriptions
set<long> liveSystems

// These are public
boolean connect()
void saveSystems()
void saveDescriptionsSystems()
void loadSystems()
void loadDescriptions()

// These are internal
handleEmbeddedStatePush(long uid, EmbeddedSystemCombinedStateMemento
info)
handleDescriptionUpdate(map<long, string>)
handleEmbeddedEvent(long uid, int eventCode)


handleControllerRequest(list<EmbeddedSystemConfigChangeMemento>
requestItems)
subscribeToEmbeddedSystem(long uid)
pushConfig(EmbeddedSystemConfigChangeMemento)
pushDescriptionsToUI()
pushEventsToUI()
pushStateDataToUI()
setLogging(boolean)
```

Over the next few pages we describe these methods in detail. The syntax used is that of a pseudo-language notation, intended as general enough to not betray any implementation details while communicating clearly the intent of the design.

`boolean connect()`

This method connects to the MQTT broker, and subscribes to `updateDescriptionRequest`, `configEmbeddedRequest`, and `pushStatusToBackend`. The return value is used by the main loop driver program to determine whether or not the connection was a success.

`void saveSystems()`

This method saves to persistent storage file containing a mapping of UIDs to `PersistentEmbeddedSystemState` objects, named `SYSTEMS.SAVE`.

`void saveDescriptions()`

This method saves to persistent storage a file containing a mapping UIDs to user-provided descriptions. The file is named `DESCRIPTIONS.SAVE`.

`void loadSystems()`

This method loads from `SYSTEMS.SAVE`, if present, a mapping of UIDs to `PersistentEmbeddedSystemState` objects.

`void loadDescriptions()`

This method loads from `DESCRIPTIONS.SAVE` a mapping of UIDs to descriptions, if it is available. We note that `SYSTEMS.SAVE` and `DESCRIPTIONS.SAVE` are serialized as JSON strings for additional safety over an implicit serialization format, which may cause data integrity or memory access errors in case of improper or malicious programming.

`void`
`handleEmbeddedStatePush(long uid, EmbeddedSystemCombinedStateMemento info)`

When we have a `pushStatusToBackend` message, we replace the relevant `EmbeddedSystemCombinedStateMemento` in `systems.` If it is not present, we add it to systemsMap, and subscribe the MQTT topic `pushEventFromEmbedded/<uid-of-embedded-system>`; the corresponding embedded system will uses the same topic string to send information to the backend. Then, we send a `pushConfigToEmbedded/<uid-of-embedded-system>` containing an `EmbeddedSystemConfigChangeMemento` set to sane defaults by a static factory method, onward to the embedded system over MQTT. We then also add its uid in `liveSystems`, if it isn't present already.

`void handleDescriptionUpdate(map<long, string>)`

When a message is determined to be of type `updateDescriptionRequest`, we deserialize a long-to-string map and update the descriptions of the UIDs represented by the long integer in our `systems` mapping. If they are not present, we ignore them.

`void handleEmbeddedEvent(long uid, int eventCode)`

If the incoming message is of type `pushEventFromEmbedded,` we verify the uid's presence in `liveSystems,` and if it is present we add the event to our events object via its `add()` method (more on that later.)

`void handleControllerRequest(list<EmbeddedSystemConfigChangeMemento> requestItems)`

Each of the systems in the list provided by the argument gets checked against the `liveSystems` set, and if present it gets validated by a function described as follows: When any of the booleans indicating a change is set to true, we check each of the corresponding values against a set of acceptable values: For example, the nutrient solution ratio can never be more than 0.01, as too much fertilizer can kill plants. We also validate the hours and minutes of the day at which lights must turn off and on.

`subscribeToEmbeddedSystem(long uid)`

This is a helper method that subscribes the backend to a topic string consisting of `embeddedSystemStatePush/<uid-of-embedded-system>`. It is used by the `handleEmbeddedStatePush()` method when receiving a message from a previously-unkown embedded system.

`pushConfig(EmbeddedSystemConfigChangeMemento)`

This method serializes the argument and sends it over MQTT using the topic `pushConfigToEmbedded/<uid-of-embedded-system>`, with the uid found in the argument's object data.

`pushDescriptionsToUI()`

This opens a tcp connections to localhost and sends the `systemDescriptions` map to our UI transfer agent, for eventual use by the user interface.

`pushEventsToUI()`

This pushes all the `EventRecord` objects present in our `EventPool` structure to the UI transfer agent on localhost.

`pushStateDataToUI()`

This pushes the `systems` mapping to the localhost UI transfer agent.

`setLogging(boolean arg)`

This sets and unsets logging for our backend system.

A helper class for our backend service is our very own `EventPool`. It allows you to record an event from a given system with a simple function call, does some book-keeping, and allows you to extract a time-sorted deque of `EventRecordMemento` items.

```
EventPool
 // Data members
map<long, deque<EventRecordMemento>> events;
int capacity;
// Constructor
EventPool(int bufferSize)

// Methods
void add(long uid, long timestamp, int event)
pair<boolean, deque<EventRecordMemento>> getEvents(long uid)
map<long, deque<EventRecordMemento>> getRaw()
```

`EventPool(int bufferSize)`

Initializes the `events` map, and sets the `capacity` member to the argument length.

`void add(long uid, long timestamp, int event)`

If there is in the events map a key with the value of `uid`, we verify the number of elements in its corresponding entry consisting of a queue. If there are too many elements (as indicated by the member `capacity`), we pop the element at the end of the queue and add an `EventRecord` memento to its head. If `uid` is not present, we create a new deque, pre-allocate it to capacity elements, and recursively calls itself to add the relevant event.

`map<long, deque<EventRecordMemento>> getRaw()`

Returns the underlying data structure; used to send all the events to the UI transfer agent.

The user interface is rather implementation-specific and will be covered in section 5 of this report. Furthermore, our data transfer agent is such a simple little utility that it does not even warrant great detailing: All it does is act as a TCP server and run in an endless loop, collecting information from the backend and offering it to the UI when requested. We will also not cover the backend simulator, as will not exist in a real deployment.

4.c: Class diagram

The following page shows a cleaned-up, simplified class diagram that shows the basic logical structure of our program. Please note that a number of details have been deliberately omitted for ease of reading.

5.a: Source code organization

We have the source code spread out across four repositories: First we have the commonly used code that is split out for reuse. We then have our backend controller in its own project, along with a main loop driver. Then we have our UI itself in its own repository. There are two views, one for examining all embedded sytems and the other to examine the events on a specific embedded system. We then have a controller to collect any form data to send to the user. All UI classes use a backend communication facade to perform serialization, marshalling, and data transfer.

5.b: Deployment diagram

Our backend and UI are both intended to run on the end-user’s system, with the browser connecting to a localhost-based Servlet runtime. We also require an MQTT broker on the host, and run the UI transfer agent simultaneously.

5.c: Data storage specification

We do not employ SQL, or even a NoSQL database; for such a projects our needs were completely met by simply having the backend save JSON files representing the map objects and their contents to persistent storage.

Here is an example `SYSTEMS.SAVE`:

```
{"1222412280320808180":
{"uid":1222412280320808180,"mistingInterval":15000,"mistingDuration":
2000,"statusPushInterval":1000,"nutrientsPPM":0,"nutrientSolutionRati
o":0.01,"lightsOnHour":8,"lightsOffHour":22,"lightsOnMinute":0,"light
sOffMinute":0,"targetUpperChamberHumidity":70.0,"targetUpperChamberTe
mperature":25.0,"targetLowerChamberTemperature":18.0,"targetCO2PPM":1
2000},"2029434316172924358":
{"uid":2029434316172924358,"mistingInterval":15000,"mistingDuration":
2000,"statusPushInterval":1000,"nutrientsPPM":0,"nutrientSolutionRati
o":0.01,"lightsOnHour":8,"lightsOffHour":22,"lightsOnMinute":0,"light
sOffMinute":0,"targetUpperChamberHumidity":70.0,"targetUpperChamberTe
mperature":25.0,"targetLowerChamberTemperature":18.0,"targetCO2PPM":1
2000},"6585073459379416509":
{"uid":6585073459379416509,"mistingInterval":15000,"mistingDuration":
2000,"statusPushInterval":1000,"nutrientsPPM":0,"nutrientSolutionRati
o":0.01,"lightsOnHour":8,"lightsOffHour":22,"lightsOnMinute":0,"light
sOffMinute":0,"targetUpperChamberHumidity":70.0,"targetUpperChamberTe
mperature":25.0,"targetLowerChamberTemperature":18.0,"targetCO2PPM":1
2000}}
```

Here is an example `DESCRIPTIONS.SAVE`:

```
{"1222412280320808180":"Alpine
strawberries","4219980363452217641":"Roses \"Baron Giraud de
l'Ain\"","6585073459379416509":"Basil and lettuces"}
```

5.d: Project link

Our project does not run on a public-facing website, as it is a solution for dealing with a network of physically-linked devices. However, the source code can be found at https://www.github.com/colingilbert/fogget-grow-system.

6.a: Programming languages

We used Java 8.0 for the entire project. It has been developed using the OpenJDK environment, in order to keep the system as free and open-source as possible.

6.b: External libraries

The Eclipse Paho MQTT client library, Java

Jackson 3.0, a JSON serialization library for Java

The CSS theme for the web UI was provided by http://thomasf.github.io/solarized-css/solarized-dark.css

6.c: Software tools and environments used

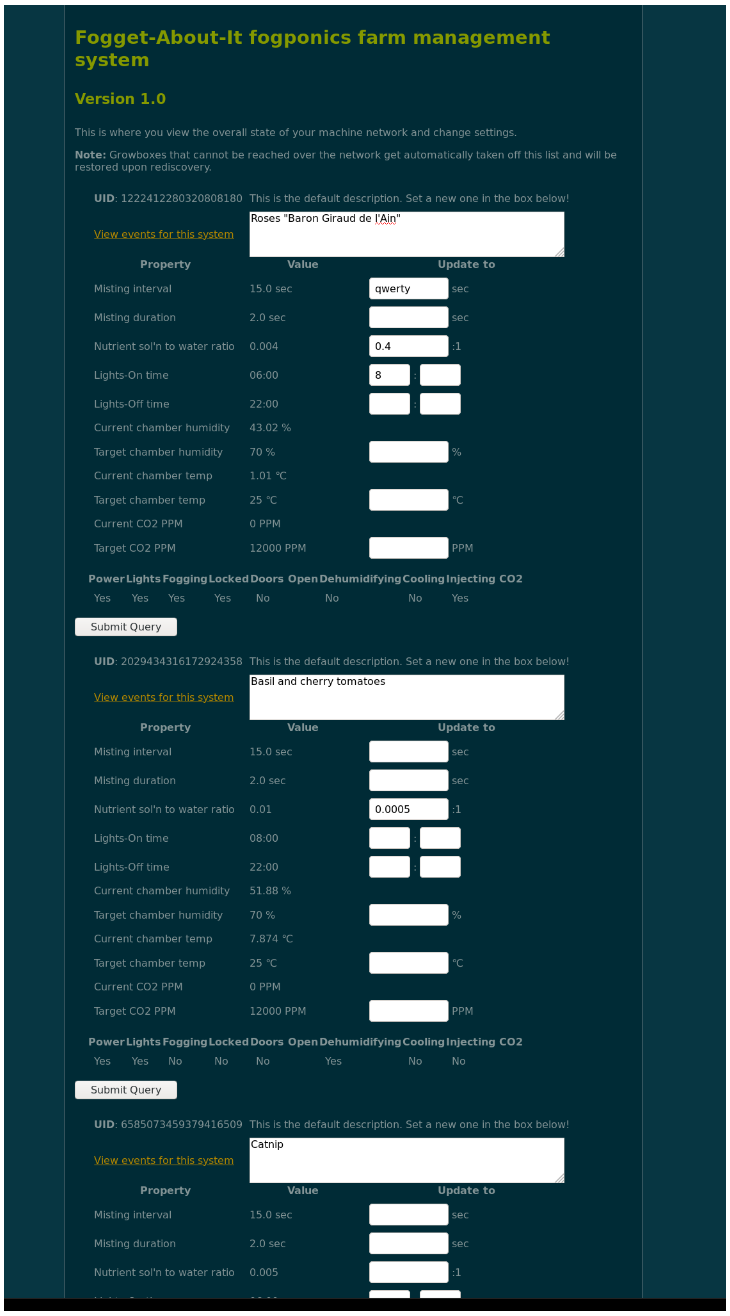
We used Netbeans 8.2, with the default Glassfish server 5.0

The UI is programmed in JSP, using custom tags to separate the Java code from the JSP itself; we use no embedded scriplets.

The Mosquitto MQTT broker is one of the most popular, with the version of MQTT in use for our project currently capped at 3.1.

7.a: Functional testing

For this, we ensure that our system can update its values. Note here how we only hve default descriptions; we also have a few nonsense values in one of the property textboxes.



Fogget-About-It fogponics farm management system

Version 1.0

This is where you view the overall state of your machine network and change settings.

Note: Growboxes that cannot be reached over the network get automatically taken off this list and will be restored upon rediscovery.

UID: 1222412280320808180 This is the default description. Set a new one in the box below!

Roses "Baron Giraud de l'Ain"

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	qwerty sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.004	0.4 :1
Lights-On time	06:00	8 :
Lights-Off time	22:00	:
Current chamber humidity	43.02 %	
Target chamber humidity	70 %	%
Current chamber temp	1.01 °C	
Target chamber temp	25 °C	°C
Current CO2 PPM	0 PPM	
Target CO2 PPM	12000 PPM	PPM

Power	Lights	Fogging	Locked	Doors Open	Dehumidifying	Cooling	Injecting CO2
Yes	Yes	Yes	Yes	No	No	No	Yes

Submit Query

UID: 2029434316172924358 This is the default description. Set a new one in the box below!

Basil and cherry tomatoes

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.01	0.0005 :1
Lights-On time	08:00	:
Lights-Off time	22:00	:
Current chamber humidity	51.88 %	
Target chamber humidity	70 %	%
Current chamber temp	7.874 °C	
Target chamber temp	25 °C	°C
Current CO2 PPM	0 PPM	
Target CO2 PPM	12000 PPM	PPM

Power	Lights	Fogging	Locked	Doors Open	Dehumidifying	Cooling	Injecting CO2
Yes	Yes	No	No	No	Yes	No	No

Submit Query

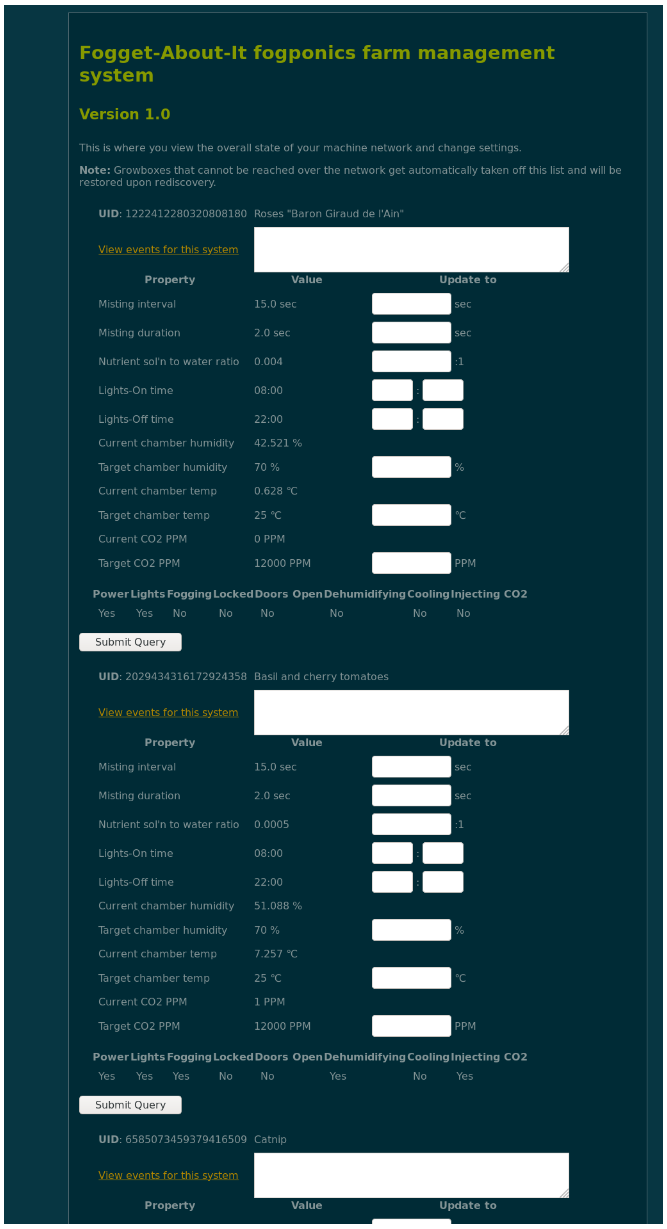
UID: 6585073459379416509 This is the default description. Set a new one in the box below!

Catnip

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.005	:1

Let’s now see what happens:



Fogget-About-It fogponics farm management system

Version 1.0

This is where you view the overall state of your machine network and change settings.

Note: Growboxes that cannot be reached over the network get automatically taken off this list and will be restored upon rediscovery.

UID: 1222412280320808180 Roses "Baron Giraud de l'Ain"

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.004	:1
Lights-On time	08:00	:
Lights-Off time	22:00	:
Current chamber humidity	42.521 %	
Target chamber humidity	70 %	%
Current chamber temp	0.628 ℃	
Target chamber temp	25 ℃	℃
Current CO2 PPM	0 PPM	
Target CO2 PPM	12000 PPM	PPM

Power	Lights	Fogging	Locked	Doors Open	Dehumidifying	Cooling	Injecting CO2
Yes	Yes	No	No	No	No	No	No

Submit Query

UID: 2029434316172924358 Basil and cherry tomatoes

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.0005	:1
Lights-On time	08:00	:
Lights-Off time	22:00	:
Current chamber humidity	51.088 %	
Target chamber humidity	70 %	%
Current chamber temp	7.257 ℃	
Target chamber temp	25 ℃	℃
Current CO2 PPM	1 PPM	
Target CO2 PPM	12000 PPM	PPM

Power	Lights	Fogging	Locked	Doors Open	Dehumidifying	Cooling	Injecting CO2
Yes	Yes	Yes	No	No	Yes	No	Yes

Submit Query

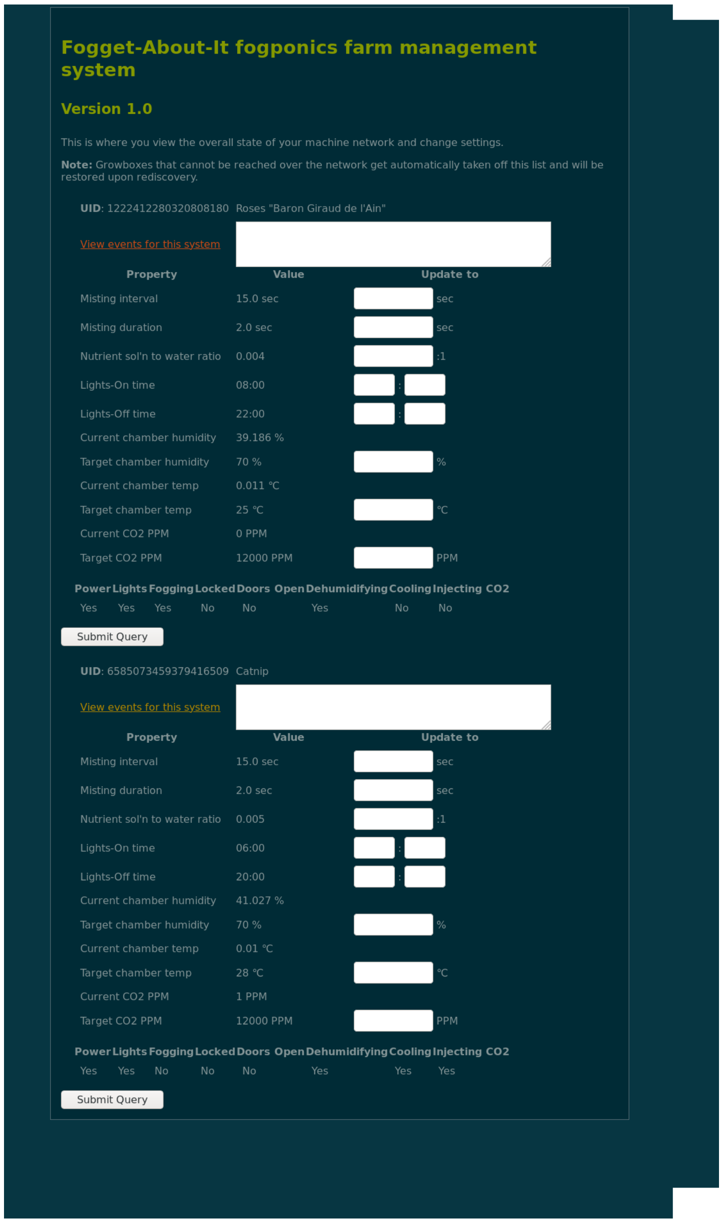
UID: 6585073459379416509 Catnip

View events for this system

Property	Value	Update to

All allowable values have been modified, and our descriptions now make sense. Note that we were able to set everything for many systems in one go.

Here is what happens when we kill the process in which one of our system simulators was running:



Fogget-About-It fogponics farm management system

Version 1.0

This is where you view the overall state of your machine network and change settings.

Note: Growboxes that cannot be reached over the network get automatically taken off this list and will be restored upon rediscovery.

UID: 1222412280320808180 Roses "Baron Giraud de l'Ain"

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.004	:1
Lights-On time	08:00	:
Lights-Off time	22:00	:
Current chamber humidity	39.186 %	
Target chamber humidity	70 %	%
Current chamber temp	0.011 ℃	
Target chamber temp	25 ℃	℃
Current CO2 PPM	0 PPM	
Target CO2 PPM	12000 PPM	PPM

Power	Lights	Fogging	Locked	Doors Open	Dehumidifying	Cooling	Injecting CO2
Yes	Yes	Yes	No	No	Yes	No	No

Submit Query

UID: 6585073459379416509 Catnip

View events for this system

Property	Value	Update to
Misting interval	15.0 sec	sec
Misting duration	2.0 sec	sec
Nutrient sol'n to water ratio	0.005	:1
Lights-On time	06:00	:
Lights-Off time	20:00	:
Current chamber humidity	41.027 %	
Target chamber humidity	70 %	%
Current chamber temp	0.01 ℃	
Target chamber temp	28 ℃	℃
Current CO2 PPM	1 PPM	
Target CO2 PPM	12000 PPM	PPM

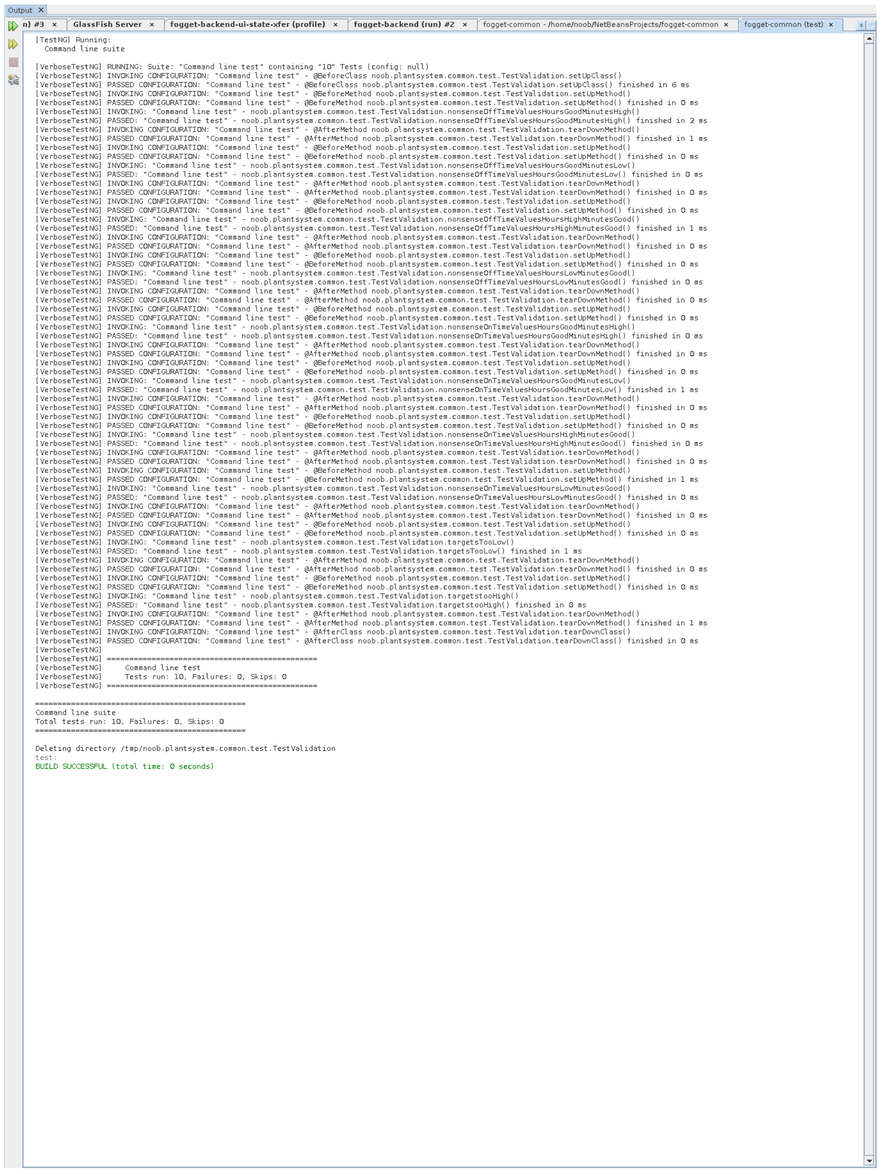
Power	Lights	Fogging	Locked	Doors Open	Dehumidifying	Cooling	Injecting CO2
Yes	Yes	No	No	No	Yes	Yes	Yes

Submit Query

We are now certain that if we make settings changes on a given device, it’ll be on a live one.

7.b: Robustness testing

While our previous test showed us that the validating functionality works, we need to know if it works for all possible state changes. Here we have the output of a program that does just that:

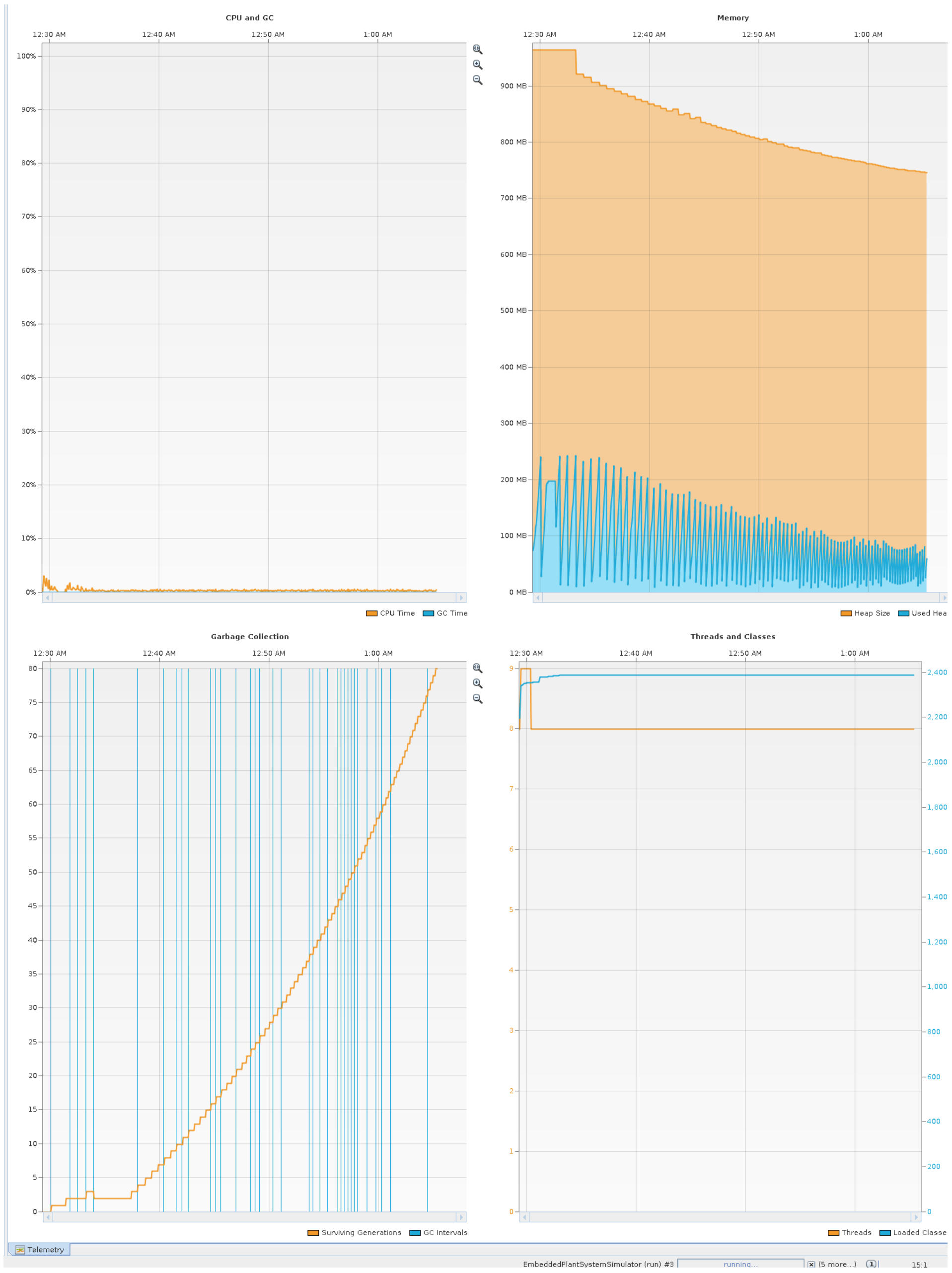

Output
n) #3
GlassFish Server
fogget-backend-ui-state-xfer (profile)
fogget-backend (run) #2
fogget-common - /home/noob/NetBeansProjects/fogget-common
fogget-common (test)

```
[TestNG] Running:
  Command line suite

[VerboseTestNG] RUNNING: Suite: "Command line test" containing "10" Tests (config: null)
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeClass noob.plantsystem.common.test.TestValidation.setUpClass()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeClass noob.plantsystem.common.test.TestValidation.setUpClass() finished in 6 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursGoodMinutesHigh()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursGoodMinutesHigh() finished in 2 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 1 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursGoodMinutesLow()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursGoodMinutesLow() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursHighMinutesGood()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursHighMinutesGood() finished in 1 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursLowMinutesGood()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOffTimeValuesHoursLowMinutesGood() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursGoodMinutesHigh()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursGoodMinutesHigh() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursGoodMinutesLow()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursGoodMinutesLow() finished in 1 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursHighMinutesGood()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursHighMinutesGood() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 1 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursLowMinutesGood()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.nonsenseOnTimeValuesHoursLowMinutesGood() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.targetsTooLow()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.targetsTooLow() finished in 1 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod noob.plantsystem.common.test.TestValidation.setUpMethod() finished in 0 ms
[VerboseTestNG] INVOKING: "Command line test" - noob.plantsystem.common.test.TestValidation.targetstooHigh()
[VerboseTestNG] PASSED: "Command line test" - noob.plantsystem.common.test.TestValidation.targetstooHigh() finished in 0 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod noob.plantsystem.common.test.TestValidation.tearDownMethod() finished in 1 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterClass noob.plantsystem.common.test.TestValidation.tearDownClass()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterClass noob.plantsystem.common.test.TestValidation.tearDownClass() finished in 0 ms
[VerboseTestNG]
[VerboseTestNG] ===============================================
[VerboseTestNG]     Command line test
[VerboseTestNG]     Tests run: 10, Failures: 0, Skips: 0
[VerboseTestNG] ===============================================

===============================================
Command line suite
Total tests run: 10, Failures: 0, Skips: 0
===============================================

Deleting directory /tmp/noob.plantsystem.common.test.TestValidation
test:
BUILD SUCCESSFUL (total time: 0 seconds)
```

7.c: Time-efficiency testing

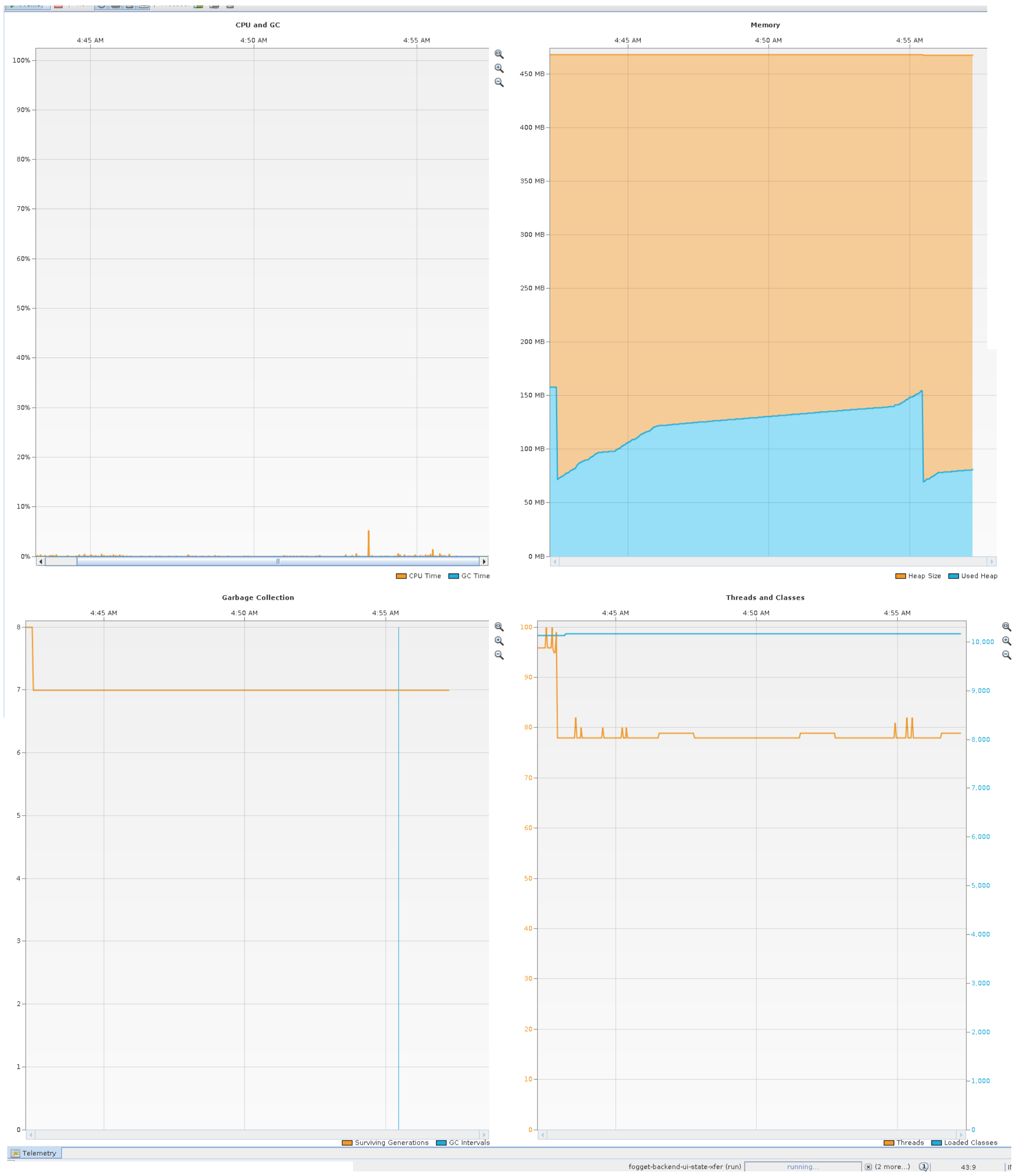
Here we profile the memory consumption and thread count over time in order to determine whether or not our app will slow down over time.

Backend



The increased generation count is due to holding a number of in a collection for a long time. Note that the number of threads remains constant and that the memory use settles down over time.

The servlet for our UI



After initial setup, the number of threads spawned goes down and only spikes once per web request.

These little spikes you can see are caused by automatic browser tests, and the threads successfully join.

These are excellect results.

Web request speed tests

We setup a script in Selenium, a python-based web testing environment. Here are the timing results, I milliseconds, of multiple iterations of accessing index.jsp (systems view):

322, 319, 316, 323, 317, 314, 319, 322, 311, 319, 311, 311, 311, 312, 311, 312, 311, 330, 316, 318, 341, 335, 317, 319, 329, 317, 335, 316, 325, 328, 318, 331, 315, 316, 318, 317, 314, 315, 328, 322, 323, 318, 329, 311, 323, 315, 347, 313, 383, 318, 316, 314, 315, 316, 336, 318, 314, 327, 314, 325

A third of a second. Not bad!

Here are the results for accessing the event.jsp page for a given system:

18, 24, 17, 17, 19, 29, 61, 35, 19, 19, 21, 36, 29, 17, 17, 19, 18, 17, 19, 21, 25, 35

The only way to get better results would be to use a language not using garbage collection such as C C++, or Rust. We believe that the only real variance is added by the garbage collection. However, the user doesn't notice or care at this point, which means that we're achieving good results.