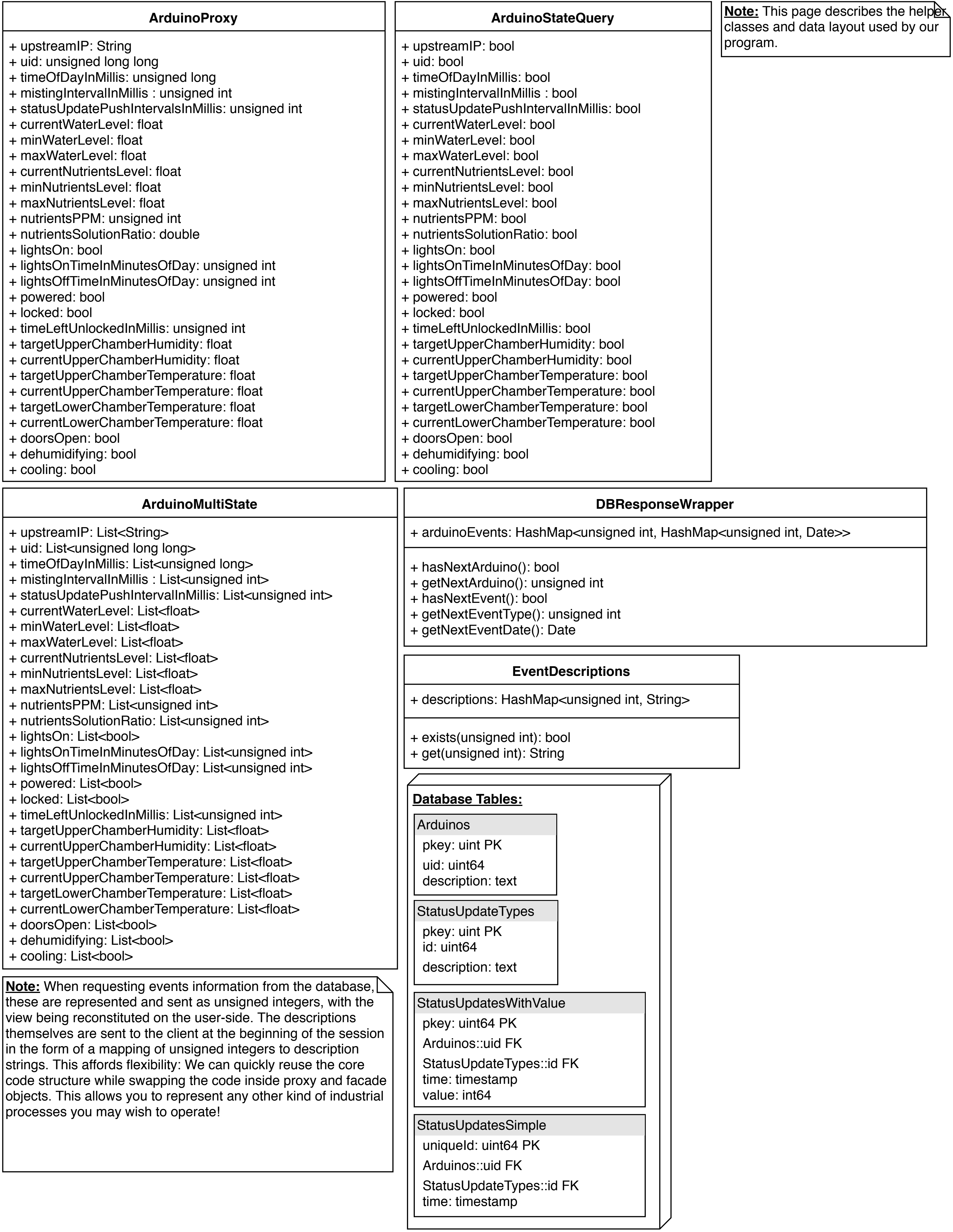


ArduinoProxy

+ upstreamIP: String
+ uid: unsigned long long
+ timeOfDayInMillis: unsigned long
+ mistingIntervalInMillis : unsigned int
+ statusUpdatePushIntervalsInMillis: unsigned int
+ currentWaterLevel: float
+ minWaterLevel: float
+ maxWaterLevel: float
+ currentNutrientsLevel: float
+ minNutrientsLevel: float
+ maxNutrientsLevel: float
+ nutrientsPPM: unsigned int
+ nutrientsSolutionRatio: double
+ lightsOn: bool
+ lightsOnTimeInMinutesOfDay: unsigned int
+ lightsOffTimeInMinutesOfDay: unsigned int
+ powered: bool
+ locked: bool
+ timeLeftUnlockedInMillis: unsigned int
+ targetUpperChamberHumidity: float
+ currentUpperChamberHumidity: float
+ targetUpperChamberTemperature: float
+ currentUpperChamberTemperature: float
+ targetLowerChamberTemperature: float
+ currentLowerChamberTemperature: float
+ doorsOpen: bool
+ dehumidifying: bool
+ cooling: bool

ArduinoStateQuery

+ upstreamIP: bool
+ uid: bool
+ timeOfDayInMillis: bool
+ mistingIntervalInMillis : bool
+ statusUpdatePushIntervalInMillis: bool
+ currentWaterLevel: bool
+ minWaterLevel: bool
+ maxWaterLevel: bool
+ currentNutrientsLevel: bool
+ minNutrientsLevel: bool
+ maxNutrientsLevel: bool
+ nutrientsPPM: bool
+ nutrientsSolutionRatio: bool
+ lightsOn: bool
+ lightsOnTimeInMinutesOfDay: bool
+ lightsOffTimeInMinutesOfDay: bool
+ powered: bool
+ locked: bool
+ timeLeftUnlockedInMillis: bool
+ targetUpperChamberHumidity: bool
+ currentUpperChamberHumidity: bool
+ targetUpperChamberTemperature: bool
+ currentUpperChamberTemperature: bool
+ targetLowerChamberTemperature: bool
+ currentLowerChamberTemperature: bool
+ doorsOpen: bool
+ dehumidifying: bool
+ cooling: bool

Note: This page describes the helper classes and data layout used by our program.

ArduinoMultiState

+ upstreamIP: List<String>
+ uid: List<unsigned long long>
+ timeOfDayInMillis: List<unsigned long>
+ mistingIntervalInMillis : List<unsigned int>
+ statusUpdatePushIntervalInMillis: List<unsigned int>
+ currentWaterLevel: List<float>
+ minWaterLevel: List<float>
+ maxWaterLevel: List<float>
+ currentNutrientsLevel: List<float>
+ minNutrientsLevel: List<float>
+ maxNutrientsLevel: List<float>
+ nutrientsPPM: List<unsigned int>
+ nutrientsSolutionRatio: List<unsigned int>
+ lightsOn: List<bool>
+ lightsOnTimeInMinutesOfDay: List<unsigned int>
+ lightsOffTimeInMinutesOfDay: List<unsigned int>
+ powered: List<bool>
+ locked: List<bool>
+ timeLeftUnlockedInMillis: List<unsigned int>
+ targetUpperChamberHumidity: List<float>
+ currentUpperChamberHumidity: List<float>
+ targetUpperChamberTemperature: List<float>
+ currentUpperChamberTemperature: List<float>
+ targetLowerChamberTemperature: List<float>
+ currentLowerChamberTemperature: List<float>
+ doorsOpen: List<bool>
+ dehumidifying: List<bool>
+ cooling: List<bool>

Note: When requesting events information from the database, these are represented and sent as unsigned integers, with the view being reconstituted on the user-side. The descriptions themselves are sent to the client at the beginning of the session in the form of a mapping of unsigned integers to description strings. This affords flexibility: We can quickly reuse the core code structure while swapping the code inside proxy and facade objects. This allows you to represent any other kind of industrial processes you may wish to operate!

DBResponseWrapper

+ arduinoEvents: HashMap<unsigned int, HashMap<unsigned int, Date>>

+ hasNextArduino(): bool
+ getNextArduino(): unsigned int
+ hasNextEvent(): bool
+ getNextEventType(): unsigned int
+ getNextEventDate(): Date

EventDescriptions

+ descriptions: HashMap<unsigned int, String>

+ exists(unsigned int): bool
+ get(unsigned int): String

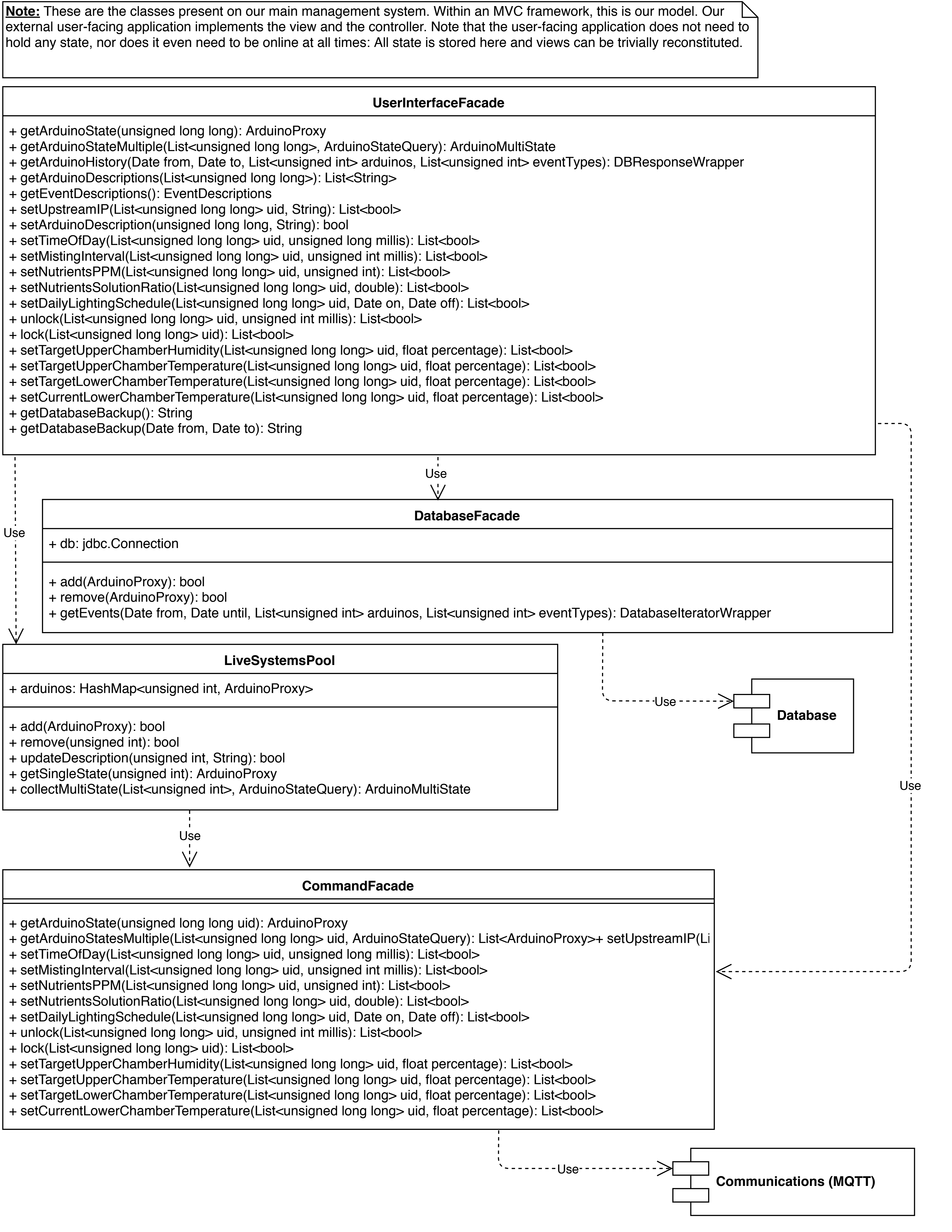
Database Tables:

Arduinos
pkey: uint PK
uid: uint64
description: text

StatusUpdateTypes
pkey: uint PK
id: uint64
description: text

StatusUpdatesWithValue
pkey: uint64 PK
Arduinos::uid FK
StatusUpdateTypes::id FK
time: timestamp
value: int64

StatusUpdatesSimple
uniqueId: uint64 PK
Arduinos::uid FK
StatusUpdateTypes::id FK
time: timestamp



Note: These are the classes present on our main management system. Within an MVC framework, this is our model. Our external user-facing application implements the view and the controller. Note that the user-facing application does not need to hold any state, nor does it even need to be online at all times: All state is stored here and views can be trivially reconstituted.

UserInterfaceFacade

+ getArduinoState(unsigned long long): ArduinoProxy
+ getArduinoStateMultiple(List<unsigned long long>, ArduinoStateQuery): ArduinoMultiState
+ getArduinoHistory(Date from, Date to, List<unsigned int> arduinos, List<unsigned int> eventTypes): DBResponseWrapper
+ getArduinoDescriptions(List<unsigned long long>): List<String>
+ getEventDescriptions(): EventDescriptions
+ setUpstreamIP(List<unsigned long long> uid, String): List<bool>
+ setArduinoDescription(unsigned long long, String): bool
+ setTimeOfDay(List<unsigned long long> uid, unsigned long millis): List<bool>
+ setMistingInterval(List<unsigned long long> uid, unsigned int millis): List<bool>
+ setNutrientsPPM(List<unsigned long long> uid, unsigned int): List<bool>
+ setNutrientsSolutionRatio(List<unsigned long long> uid, double): List<bool>
+ setDailyLightingSchedule(List<unsigned long long> uid, Date on, Date off): List<bool>
+ unlock(List<unsigned long long> uid, unsigned int millis): List<bool>
+ lock(List<unsigned long long> uid): List<bool>
+ setTargetUpperChamberHumidity(List<unsigned long long> uid, float percentage): List<bool>
+ setTargetUpperChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ setTargetLowerChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ setCurrentLowerChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ getDatabaseBackup(): String
+ getDatabaseBackup(Date from, Date to): String

DatabaseFacade

+ db: jdbc.Connection

+ add(ArduinoProxy): bool
+ remove(ArduinoProxy): bool
+ getEvents(Date from, Date until, List<unsigned int> arduinos, List<unsigned int> eventTypes): DatabaseIteratorWrapper

LiveSystemsPool

+ arduinos: HashMap<unsigned int, ArduinoProxy>

+ add(ArduinoProxy): bool
+ remove(unsigned int): bool
+ updateDescription(unsigned int, String): bool
+ getSingleState(unsigned int): ArduinoProxy
+ collectMultiState(List<unsigned int>, ArduinoStateQuery): ArduinoMultiState

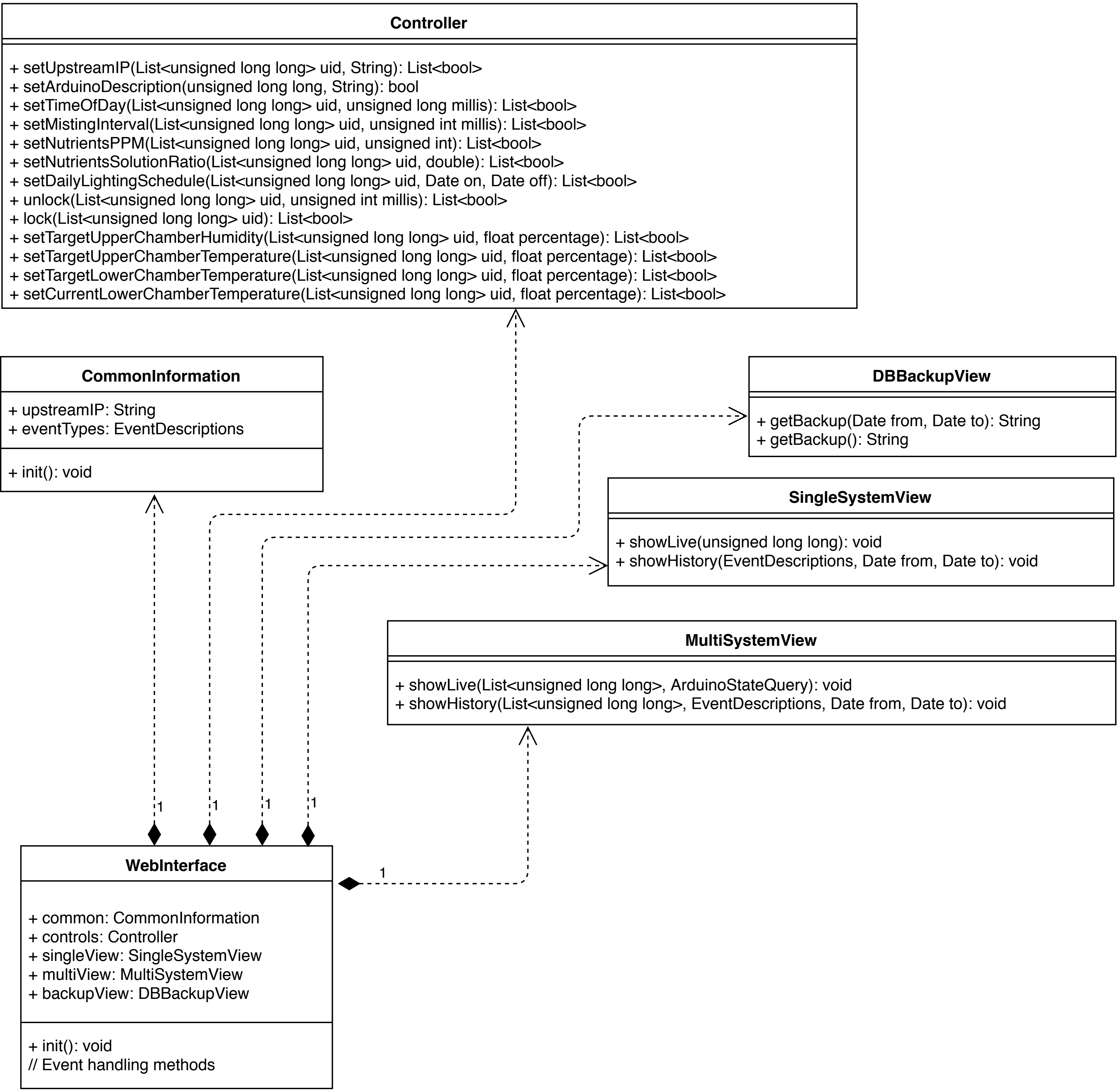
CommandFacade

+ getArduinoState(unsigned long long uid): ArduinoProxy
+ getArduinoStatesMultiple(List<unsigned long long> uid, ArduinoStateQuery): List<ArduinoProxy>+ setUpstreamIP(Li
+ setTimeOfDay(List<unsigned long long> uid, unsigned long millis): List<bool>
+ setMistingInterval(List<unsigned long long> uid, unsigned int millis): List<bool>
+ setNutrientsPPM(List<unsigned long long> uid, unsigned int): List<bool>
+ setNutrientsSolutionRatio(List<unsigned long long> uid, double): List<bool>
+ setDailyLightingSchedule(List<unsigned long long> uid, Date on, Date off): List<bool>
+ unlock(List<unsigned long long> uid, unsigned int millis): List<bool>
+ lock(List<unsigned long long> uid): List<bool>
+ setTargetUpperChamberHumidity(List<unsigned long long> uid, float percentage): List<bool>
+ setTargetUpperChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ setTargetLowerChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ setCurrentLowerChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>

Database

Communications (MQTT)

Note: This is the web interface class interaction diagram. It polls the main management system and can reconstitute itself by using its API. It holds no application state by default: This is done for manageability and to ensure fresh state information at all times.



Controller

+ setUpstreamIP(List<unsigned long long> uid, String): List<bool>
+ setArduinoDescription(unsigned long long, String): bool
+ setTimeOfDay(List<unsigned long long> uid, unsigned long millis): List<bool>
+ setMistingInterval(List<unsigned long long> uid, unsigned int millis): List<bool>
+ setNutrientsPPM(List<unsigned long long> uid, unsigned int): List<bool>
+ setNutrientsSolutionRatio(List<unsigned long long> uid, double): List<bool>
+ setDailyLightingSchedule(List<unsigned long long> uid, Date on, Date off): List<bool>
+ unlock(List<unsigned long long> uid, unsigned int millis): List<bool>
+ lock(List<unsigned long long> uid): List<bool>
+ setTargetUpperChamberHumidity(List<unsigned long long> uid, float percentage): List<bool>
+ setTargetUpperChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ setTargetLowerChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>
+ setCurrentLowerChamberTemperature(List<unsigned long long> uid, float percentage): List<bool>

CommonInformation

+ upstreamIP: String
+ eventTypes: EventDescriptions

+ init(): void

DBBackupView

+ getBackup(Date from, Date to): String
+ getBackup(): String

SingleSystemView

+ showLive(unsigned long long): void
+ showHistory(EventDescriptions, Date from, Date to): void

MultiSystemView

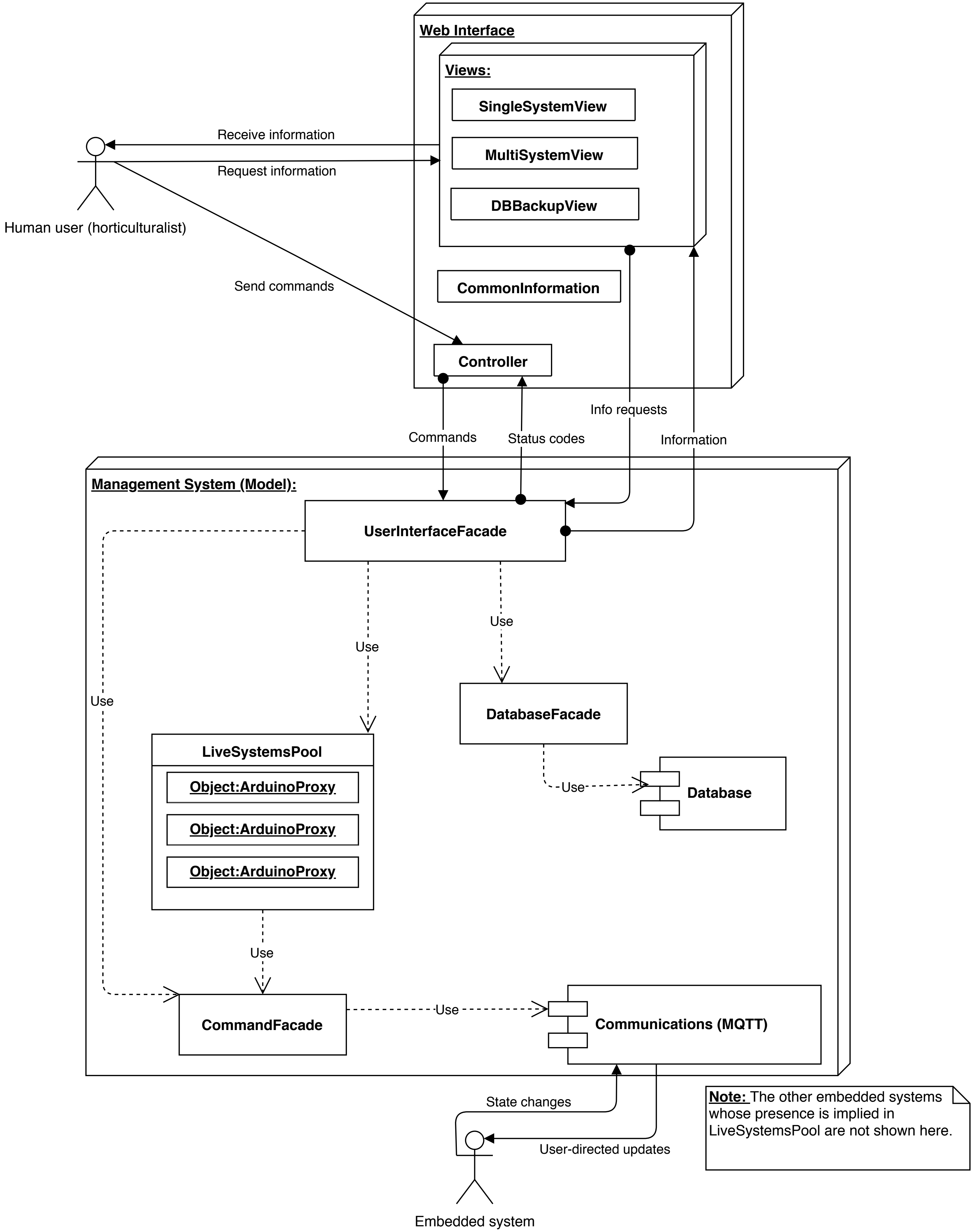
+ showLive(List<unsigned long long>, ArduinoStateQuery): void
+ showHistory(List<unsigned long long>, EventDescriptions, Date from, Date to): void

WebInterface

+ common: CommonInformation
+ controls: Controller
+ singleView: SingleSystemView
+ multiView: MultiSystemView
+ backupView: DBBackupView

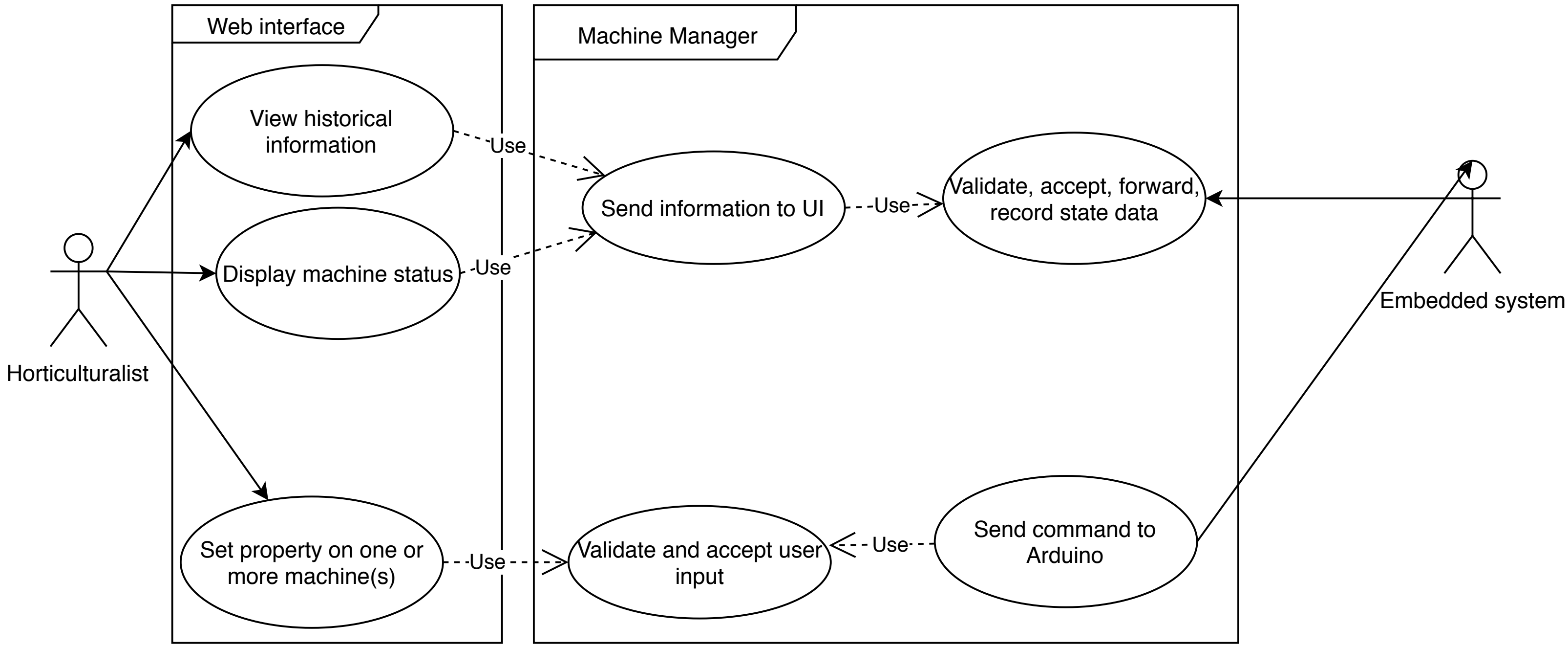
+ init(): void
// Event handling methods

Note: Here is the architectural diagram. We note that the separation between the web interface and management system can be physical (ie: two different machines) or logical (both software running on a single machine.) However, flexibility is ensured by keeping the same communications protocols - we simply setup the web interface to use a different upstream IP address!



Note: The other embedded systems whose presence is implied in LiveSystemsPool are not shown here.

Note: We are only limited in the number of embedded systems by the controller's hardware and the network's capacity.


Web interface
Machine Manager
View historical information
Display machine status
Set property on one or more machine(s)
Send information to UI
Validate, accept, forward, record state data
Validate and accept user input
Send command to Arduino
Use
Use
Use
Use
Use
Horticulturalist
Embedded system