Advanced Software Engineering Final Report

TABLE OF CONTENTS

1 FINAL DESIGN

1.1 DESIGN APPROACH AND OVERVIEW

1.1.1 ASSUMPTIONS

Based on the research of the domain and its main stakeholders, we defined following assumptions for the project:

- The considered application area is **smart** farming: i.e., utilization of information and communication technologies (ICTs) in the rural domain, with a primary focus on agriculture. In particular, we aim at a **web-based software application that allows managing stakeholder-specific aspects** of the farming domain depending on their role.
- Regarding the scale of the area for conducting the farming, **micro** farming is considered: "Micro farming is small-scale, high-yield, sustainably minded farming, generally conducted by hand in urban or suburban areas."[1] More concretely, we target growing areas of 4.5m^2 (3m x 1.5m) e.g., within a garden or a greenhouse.
- The dimensions of the growing area are based on the reach and capabilities of a **multi-purpose farming device** similar to *FarmBot Genesis*[2]:

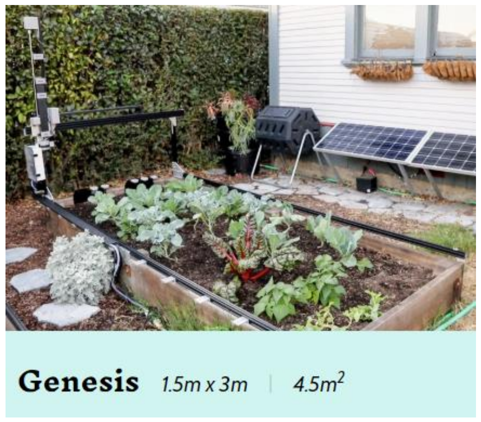


Figure 1 FarmBot Genesis hardware dimensions

- A **farming site** is the bounded growing area with the previously noted dimensions (4.5m^2).
- **Positions** are points/locations for planting crops within a farming site, and are placed at the intersections of a fixed 2D square grid:

[1] https://hellohomestead.com/what-is-micro-farming/

[2] https://farm.bot/products/farmbot-genesis-v1-5

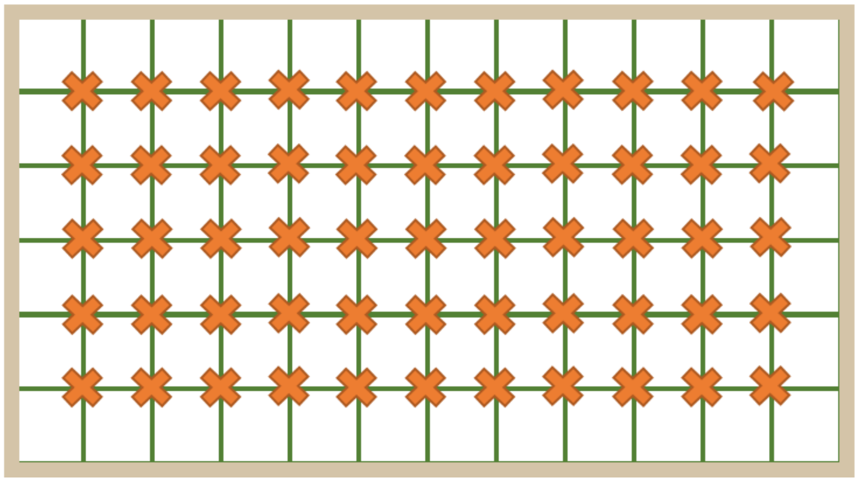


Figure 2 2D grid positioning system. Each position is identified by its discrete [X,Y] coordinates within the grid, ranging from (1,1) (lower left corner) to (11,5) (upper right corner).

- The positions are located 25cm (0.25m) away from each other and the boundaries of the farming site.
- Based on the dimensions and the 25cm spacing rule, 5 plants can fit along the shorter edge, and 11 along the longer edge of the site, which comprises a total of 55 crops in terms of capacity per site.

- **Farming devices** supported by our application are multi-purpose bots and have all necessary capabilities (sensors and actuators) for treating different plant species within a farming site: sensors for soil humidity, sensors for detecting plant growth, actuators for seeding, watering, weeding, and cutting, as well as for treating the plants with different substances such as fertilizers or pesticides.
  - **Weeding** (i.e., removing weeds) is done by a "weed suppressor using a camera to identify weeds by comparing all plants in the area to the locations of the planted seed"[3]
- We assume that **plants** have 5 distinguishable life-cycle stages:

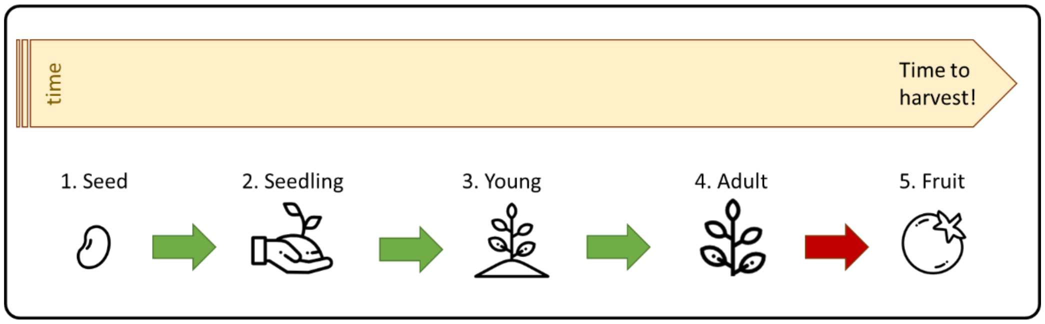


Figure 3 Life-cycle stages of a plant

[3] https://en.wikipedia.org/wiki/FarmBot

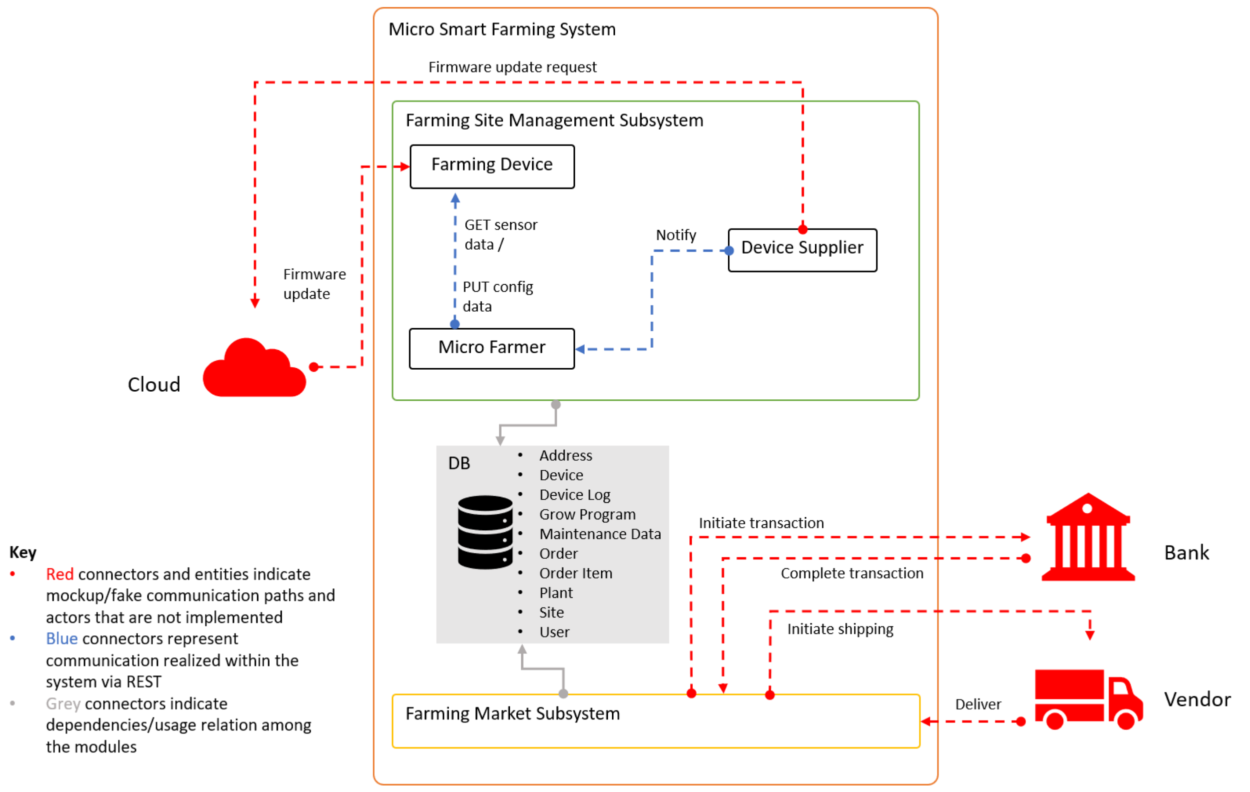
The treating for specific plant species can be defined over the plant's life-cycle stages in grow-programs. We assume that **the farming devices can detect** (via e.g., AI-assisted cameras) that the plant has developed a **fruit** which can be harvested.
We assume that farming devices can detect the growth stages from stage 2 onward.

- The resulting micro smart farming application will use a **predefined** set of (fictional, exemplary) farming **devices** and their corresponding **suppliers** which means a micro farmer can register a new device by simply finding its model in the list provided in the application, and its general data will be automatically assigned to the new device.
- We assume that the micro farmers already have their devices in possession; i.e. the system **will not provide means** to buy a new device
- The mockup devices will **communicate** with the micro farmer's subsystem via REST to exchange sensor and actuator data.

1.1.2 THOUGHT PROCESS AND MAJOR DESIGN DECISIONS

In this section we discuss our major design decisions and how we got to them.

As a starting point, we created a sketch of the involved parties and interactions in the system:


Micro Smart Farming System
Firmware update request
Farming Site Management Subsystem
Farming Device
GET sensor data / PUT config data
Notify
Device Supplier
Firmware update
Micro Farmer
Cloud
DB
Address
Device
Device Log
Grow Program
Maintenance Data
Order
Order Item
Plant
Site
User
Initiate transaction
Complete transaction
Bank
Initiate shipping
Deliver
Vendor
Farming Market Subsystem
Key
Red connectors and entities indicate mockup/fake communication paths and actors that are not implemented
Blue connectors represent communication realized within the system via REST
Grey connectors indicate dependencies/usage relation among the modules

Each design decision starts with a design issue or problem that is to be tackled.

The major design points we have identified at the beginning of this project are:

1. Application structure
2. Involved subsystems
3. User management and access control
4. Device capabilities
5. Responsibility for device capabilities
6. Device operational state mediation
7. Notification support

Design decisions (DDs):

1. As per the assignment sheet, a layered architecture is required to enable modularization and separation of concerns. We have chosen the Model-View-Controller (MVC) pattern as the general architecture pattern. Furthermore, the intercommunication among the relevant actors of the system (farmer <-> device, device supplier <-> farmer) will be realized in a client-server fashion.

2. On a higher abstraction level, we defined 2 major subsystems: (1) the Farming Site Management Subsystem, and (2) the Farming Market Subsystem. This distinction allows for a more purpose-oriented organization of the components: actions such as buying and selling are part of a different context than e.g., growing a crop or assigning a device to a farming site. On a lower level, of course use cases like device maintenance are also realized by a task-specific subsystem, which is in our case a mockup of external behavior and is not explicitly modelled.

3. A major point is also the access control and thus managing different user roles. We decided that every user of the system has to be registered with the system. A registered user can either be a micro farmer, a seed supplier, a device supplier or a consumer. According to the chosen account (user) type, they get roles assigned to them: e.g., a user that registers as a micro farmer, gets the roles (farmer, consumer, seed supplier) assigned, ultimately defining what actions they can perform with the system (see Table 1).

Table 1 User roles and corresponding access rights

ROLE	RIGHTS
FARMER	own and manage farming sites, own and manage farming devices, create and adapt grow programs
SEED SUPPLIER	sell plants (seeds, saplings, yield) and grow programs
DEVICE SUPPLIER	sell devices, maintain sold devices
CONSUMER	buy plants (seeds, saplings, yield), grow programs and devices

4. We have decided that are system should support multi-purpose farming devices, i.e. farm bots, such as the FarmBot Genesis[4]. These devices have all necessary capabilities (sensors and actuators) for treating different plant species within a farming site: sensors for soil humidity, sensors for detecting plant growth, actuators for seeding, watering, weeding, and cutting, as well as for treating the plants with different substances such as fertilizers or pesticides.

5. The capabilities described in DD-4 are defined by the supplier and stored on the devices themselves. Basic functioning and maintenance are the supplier's

[4] https://en.wikipedia.org/wiki/FarmBot

responsibility and therefore the device supplier has access rights for the devices' maintenance data to be able to tackle issues or ship replacements when necessary.

6. Since the device supplier is required to notify a micro farmer about e.g., imminent failures of their device, the device supplier, logically, first needs to be informed about the occurrence of such an event from a device. To support this, we intend to apply the *observer pattern* so that the device is the *observable* and the device supplier is the *observer*. When a significant (unexpected or extreme) change such as a failure event occurs on the device, the adequate device supplier should be notified about the event.

7. Notifications are to be supported by the system in a way that enables a device supplier to inform a certain micro farmer of their device's state in case of e.g., imminent failures. The device supplier gathers this information from the device itself (see DD-6). Since the desired communication is asynchronous and one-way communication, i.e. the micro farmer does not "respond" to a received notification, we intend to apply a publish/subscribe based protocol to support the notification service. In particular, the device supplier will be the publisher, sending out messages on a topic such as, e.g., "device state" or "device firmware update", and the micro farmer will be the subscriber, receiving on their end these particular messages on the subscribed topics.

1.1.3 ARCHITECTURAL DESIGN DECISIONS (ADD)

1.1.3.1 ADD 1: "ROLE-BASED ACCESS CONTROL"

In the context of managing different user roles as a requirement, facing that, by considering and implementing only the authentication process, the accessing to the resources within the application is not tailored based on the type of the user, we decided to implement a role-based access control (RBAC) by using the spring security to distinguish the type of access to the resources based on the specific user types to ensure FR04 (Sustainable Finance and the Role of Securities Regulators and IOSCO Final Report The Board OF THE, n.d.) and meet the precise security needs, accepting the sophistication of managing accesses as an administrative task.

1.1.3.2 ADD 2: "DEVICE MULTI-CAPABILITY SUPPORT"

In the context of device multi-capability support, facing that the system must support the combination of at least four different types of single-purpose farming devices or at least four different capabilities, we decided that our system should only support Farmbots, to achieve all necessary capabilities (sensors and actuators) for treating different plant species, accepting that our system is limited only to the capabilities provided by these Farmbots.

1.1.4 DESIGN OVERVIEW

In this section, we provide an overview of the final solution of our project.

As described in the previous section, we have decided to build our main application in an MVC fashion. The MVC pattern is complemented very well by the domain-driven design (DDD) approach, since the MVC itself does not define how the Model should be structured. Furthermore, we realized a separate subsystem as a RESTful application, which should demonstrate the behavior (e.g., setting actions, logging, communicating with the main application) of the farming devices. Both of these applications use the model code generated with the Eclipse Modeling Framework and XMI (XML Metadata Interchange) as the persistence mechanism. The Model is used in both applications as a plugin.

An overview of the final architecture of the system can be seen in Figure 4.

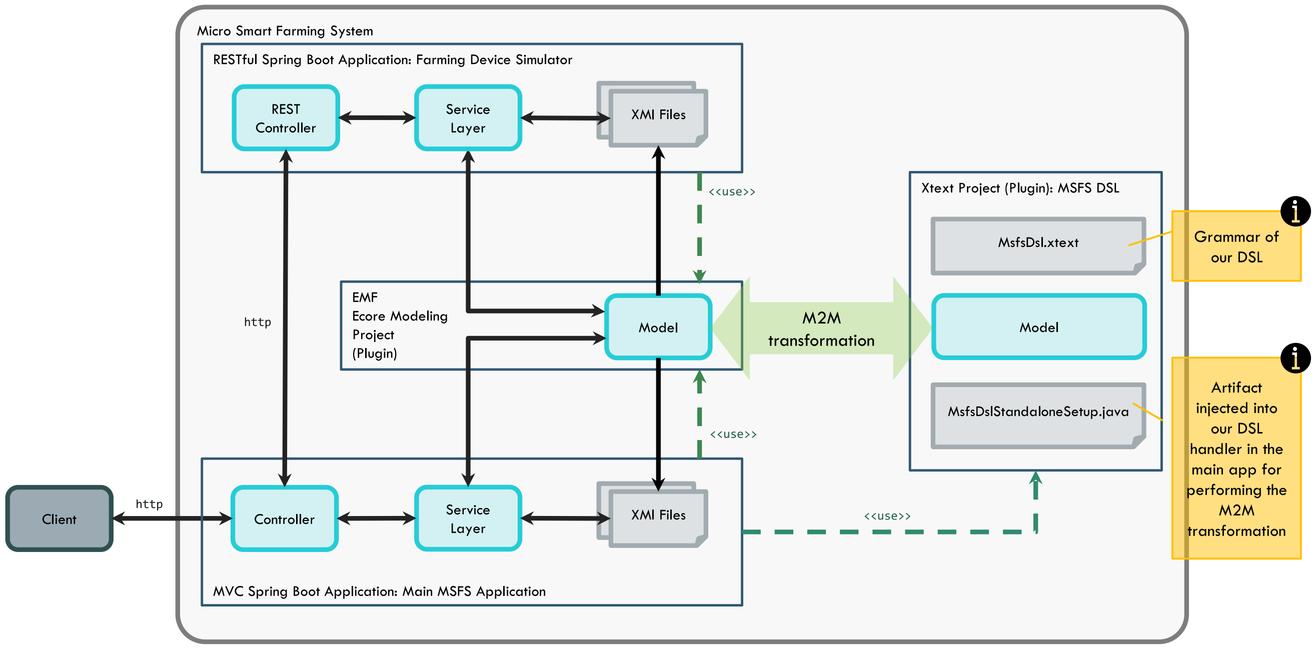

Micro Smart Farming System
RESTful Spring Boot Application: Farming Device Simulator
REST Controller
Service Layer
XMI Files
<<use>>
Xtext Project (Plugin): MSFS DSL
MsfsDsl.xtext
Grammar of our DSL
http
EMF Ecore Modeling Project (Plugin)
Model
M2M transformation
Model
MsfsDslStandaloneSetup.java
Artifact injected into our DSL handler in the main app for performing the M2M transformation
<<use>>
Client
http
Controller
Service Layer
XMI Files
<<use>>
MVC Spring Boot Application: Main MSFS Application

Figure 4 Architecture of the MSFS

The overall layered architecture of the system may be best described by our component diagram (i.e. Development View), since it (among other things) represents the organization of software modules (see Figure 5).

The status-quo of our system's main data model is depicted by the EMF ecore model, shown in Figure 6.

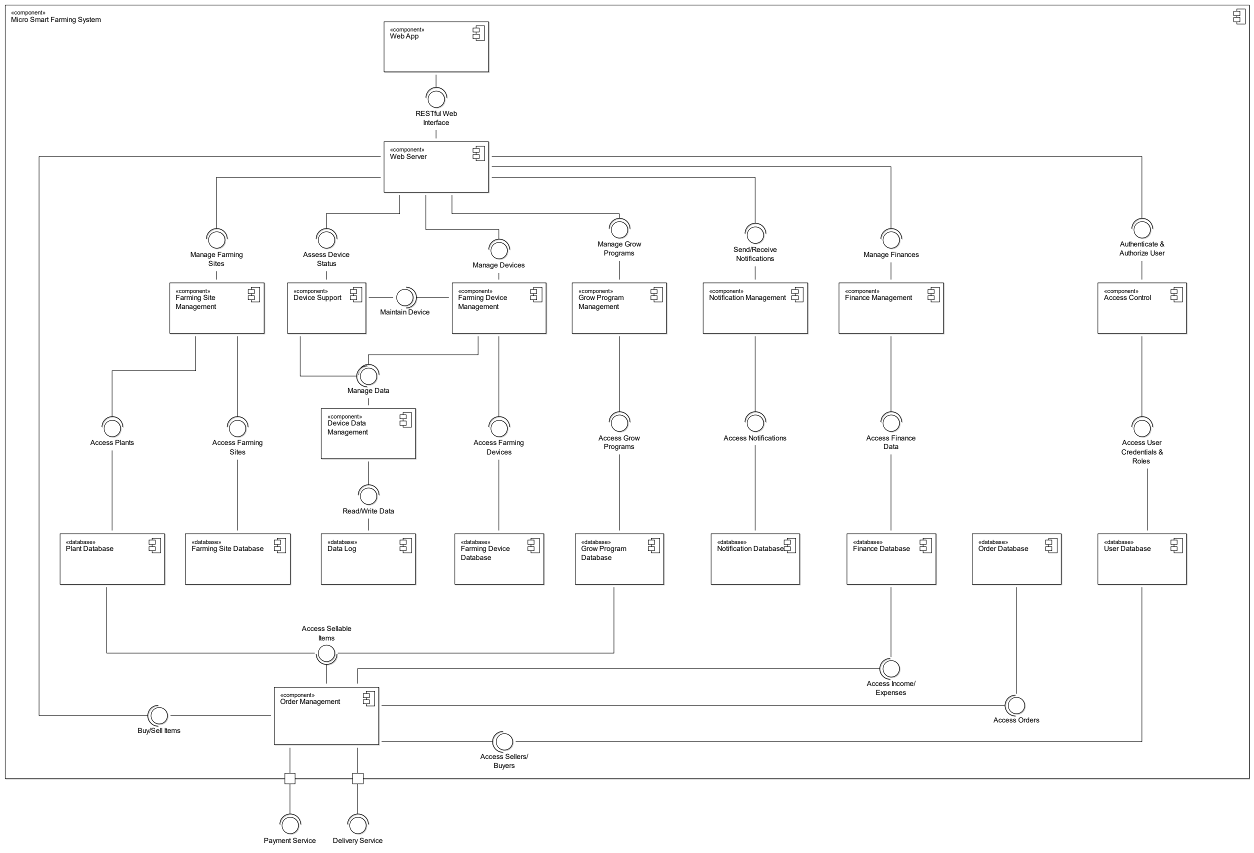

«component» Micro Smart Farming System
«component» Web App
RESTful Web Interface
«component» Web Server
Manage Farming Sites
Assess Device Status
Manage Devices
Manage Grow Programs
Send/Receive Notifications
Manage Finances
Authenticate & Authorize User
«component» Farming Site Management
«component» Device Support
Maintain Device
«component» Farming Device Management
«component» Grow Program Management
«component» Notification Management
«component» Finance Management
«component» Access Control
Manage Data
«component» Device Data Management
Access Plants
Access Farming Sites
Access Farming Devices
Access Grow Programs
Access Notifications
Access Finance Data
Access User Credentials & Roles
Read/Write Data
«database» Plant Database
«database» Farming Site Database
«database» Data Log
«database» Farming Device Database
«database» Grow Program Database
«database» Notification Database
«database» Finance Database
«database» Order Database
«database» User Database
Access Sellable Items
Access Income/ Expenses
«component» Order Management
Access Orders
Buy/Sell Items
Access Sellers/ Buyers
Payment Service
Delivery Service

Figure 5 Component diagram

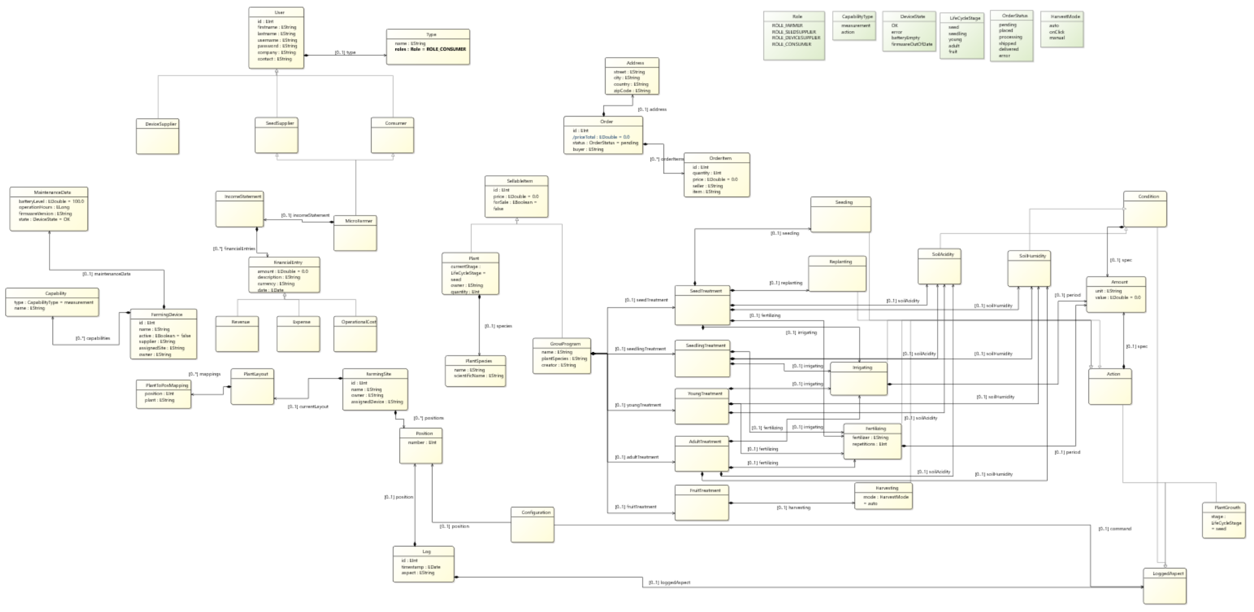


Figure 6 UML representation of the EMF ecore model in Eclipse

1.2 DEVELOPMENT STACK AND TECHNOLOGY STACK

1.2.1 DEVELOPMENT STACK

Under development stack, we understand a collection of frameworks and tools for managing dependencies, building and versioning our application.

We have considered following alternatives:

- Apache Ant
- Apache Maven
- Gradle

We decided to use **Apache Maven**, because our team has some previous experience with this tool, and it is relatively easy to use.

1.2.2 TECHNOLOGY STACK

- **Eclipse Modeling Tools 2020-03** (downloaded via installer from https://www.eclipse.org/downloads/packages/) as IDE
- **Spring Framework** to realize the MVC structure of our application
- **Spring-Boot 2.3.0.RELEASE** (https://spring.io/projects/spring-boot#overview) to build and run our application in a simple way
- **HTML5**, with **CSS** and **Javascript** for realizing the web-based frontend
- **Eclipse Paho Java client** (https://www.eclipse.org/paho/clients/java/) for realizing the notification system
- ~~**Chart.js v2.9.3** **Error! Bookmark not defined.** (https://www.chartjs.org/docs/latest/) library for v isualizing graphs/charts~~
- **Canvas.js** (https://canvasjs.com/) library for visualizing graphs/charts

  [required dependency: **Gson** (https://github.com/google/gson), version: 2.8.5]
- ~~**MySQL DB** & **JDBC driver** for realizing persistence~~
- **XMI**-based persistence, which is native to EMF models, using EResources.
- **Spring Security** (**version**: see Spring-Boot version) for realizing access control

1.3 MAJOR CHANGES COMPARED TO SUPD

- There where no changes to the *software* compared to SUPD, because at that point we did not have any implementation in place but have rather focused on the design of the system.
- Thus, there were some changes in the design artifacts (based on the feedback of our supervisor) which preceded the implementation:
    - Fixed overall architecture model (see 1.1.4 Design Overview)
    - Scenarios
        - Added note of required access to maintenance data by the device supplier explicitly to UC08 and UC09
    - Logical view
        - Added other layers to this view (namely: presentation, application, and data-access layers), @SUPD only the domain layer was represented
        - Fixed relationships among Entities to better represent a realistic setting (e.g., *Micro Farmer* does NOT contain a *Farming Site* as was depicted in @SUPD version, and the relations between *User - Type - Role* are not simple associations anymore, but are represented with aggregation, incorrect usage of dependency relations are fixed)
        - Included *Operational Cost* into the model
        - Included members with different visibility & scope according to their usage and semantics
    - Process View
        - Fixed and extended models to depict the system's business-relevant and domain-specific processes/tasks, with explicit focus on:
            - Registration mechanism for Farming Devices
            - Assignment mechanism between Farming Sites and Farming Devices
            - Setting actions to a Farming Device (and execution of those accordingly)
            - Grow Program execution
        - The models are now represented in more detail, with concrete class instances (anonymous objects denoted with :Class), and if there is a physical Actor (such as different users of our systems: Micro Farmer, Device Supplier, etc.), the Actor is depicted using an UML actor symbol.
    - Development View
        - Merged access control and authentication into access control
    - Physical View
        - Adapted model to represent the final deployment better, with the supervisor's feedback in mind regarding the deployed artifacts
- Technology stack
    - Slight changes in used libraries and persistence mechanism (see 1.2.2 Technology Stack)
    - Updated version numbers and dependencies

2 UPDATED SYSTEM REQUIREMENTS

The requirements have not changed with respect to the previous (SUPD) submission. What is added is the *Verification status* (see last column in Table 2), showing whether the requirement has been met (OK) or not (NOT OK).

We are proud to report that all of the identified requirements for the target system have been met!

Table 2 Requirements

ID	F/N[5]	Priority	Description	Affected UC	Affected roles	Verification method	Verification status
1	F	high	The system must support growing of multiple different plant species within a farming site at the same time.	-	-	Testing/code review	OK
2	F	High	Multi-device-capability support. The system must support the combination of at least 4 different types of single-purpose farming devices, or at least 4 different capabilities (e.g., irrigation, branch-cutting, soil weeding) of a multi-purpose farming device (farmbot)	UC01, UC10, UC11	Farming Device	Testing/code review	OK
3	F	High	Grow-program execution. The system must support internal, autonomous processes (e.g.,	UC03	-	Testing/code review	OK

[5] F = functional requirement, N = non-functional requirement

			realized as Java threads) to "execute" grow-programs				
4	F	High	Role-based access control. The system must support appropriate access regulations to adhere to the GDPR.	UC01-UC09	Micro Farmer, Seed Supplier, Device Supplier, Consumer	Testing/code review	OK
5	F	High	Farming Device management. The system must support registering new devices, reassigning registered, and removing them.	UC01	Micro Farmer	Testing/code review	OK
6	F	High	Grow-program management. The system must support the creation, adaptation, and deletion of grow programs.	UC03	Micro Farmer	Testing/code review	OK
7	F	High	Plant-layout management. The system should support the creation, adaptation and deletion of plant layouts for farming sites.	UC02	Micro Farmer	Testing/code review	OK
8	F	High	Order management. The system should support buying and selling of plants (at different life-cycle stages) and grow-programs.	UC05, UC06	Seed Supplier, Consumer	Testing/code review	OK

9	F	High	Notification service. The system should support means for the Device Supplier to notify a Micro Farmer about e.g., severe device issues.	UC08	Device Supplier, Micro Farmer	Testing/code review	OK
10	F	Medium	Device firmware updates. The system should support mockup functionality of updating device firmware.	UC07	Device Supplier, Farming Device	Testing/code review	OK
11	F	Medium	Initiating item shipment. The system should support mockup functionality of initiating shipment of items (in cases of orders, as well as replacement items for devices)	UC05, UC06, UC09	Device Supplier, Seed Supplier	Testing/code review	OK
12	F	Medium	Finances visualization. The system should support chart-like visualization of income, expenses, and profit of a Micro Farmer.	UC04	Micro Farmer	Testing/code review	OK
13	F	Medium	Device logs management. The system should support mockup functionality of logging a device's actions and measurements.	UC10, UC11	Farming Device	Testing/code review	OK
14	N	High	Data integrity. The system				OK

			should protect against unauthorized manipulation of data.		
15	N	High	Interoperability. The system should support interactivity among its internal components and external entities (e.g., farming devices).		OK
16	N	High	Modularity. The system should be built in a modular fashion to enable separation of concerns and make the system easier to maintain.		OK
17	N	High	Security. The system should secure concerns related to manipulating the farming devices, since their operation affects the farming sites, and in turn, a Micro Farmer's crops and finances.		OK
18	N	High	Usability. The system should be easily understandable and intuitive to use.		OK
19	N	Medium	Scalability. The system should be able to handle changes		OK

			of workload and number of users.		
20	N	Medium	Availability. The system's functionality should always be available to the users, and not be severely affected by partial system failures.		OK

3 UPDATED 4+1 VIEWS MODEL

3.1 SCENARIOS / USE CASE VIEW

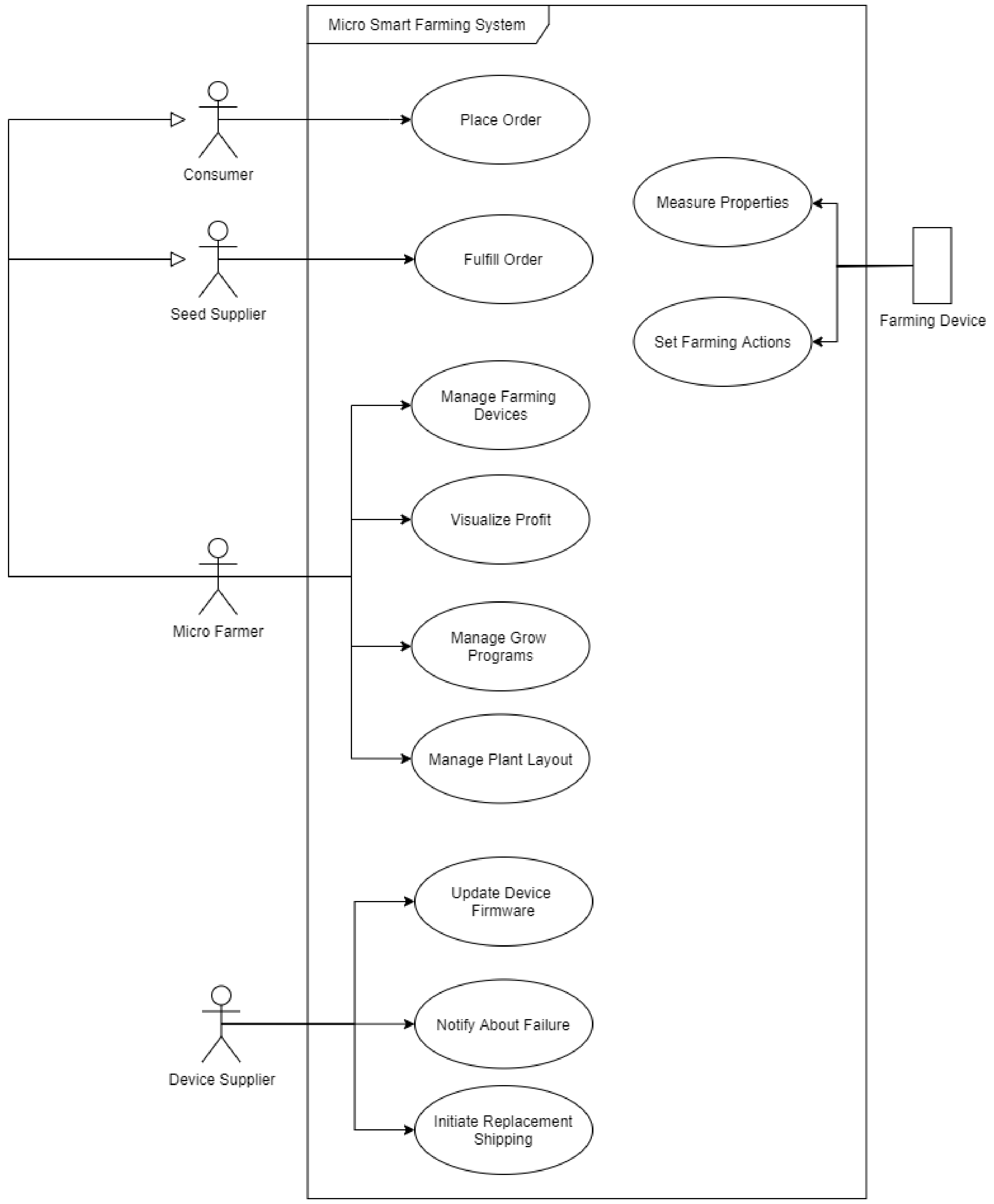


Figure 7 Scenarios / Use case diagram

Figure 7 shows the scenarios/use cases we have identified as the most relevant to be supported by our system, and the corresponding actors involved in those use cases. Complex scenarios, i.e., the ones that may encompass more related use cases, are represented as one use case (see e.g., "Manage Grow Programs"). Concretely, under "managing" we understand an umbrella term for actions such as *viewing*, *editing*/*updating*, *deleting*, *creating*, etc. The available main courses of action and alternative actions are described in detail in the use case descriptions below. This reduction of the amount of detail is done for the sake of simplicity and a more comprehensive, compact view of the supported functionality.

When it comes to the actors of the system, the actors we defined as *primary* and that are to be interacting with the frontend, are placed on the left-hand side of the system, while the *secondary* ones (in this case only the Farming Device) interacting with the software backend, are placed on the right-hand side.

Furthermore, it is important to note that we have concretized the "buy" and "sell" use cases to "place order" and "fulfill order", respectively, and have explicitly modelled the Consumer and Seed Supplier actors to emphasize the distinction among the supported use cases for those roles, as well as to be able to depict the fact that a Micro Farmer is both Consumer and Seed Supplier, being able to both buy and sell items.

In the following, for each of the modelled use cases, a detailed use case description is given.

Use Case Descriptions

Title	Manage Farming Devices
ID	UC01
Description	A micro farmer can manage their farming devices by registering new devices and assigning them to a farming site, removing registered devices, or reassigning devices to another farming site
Actors	Micro Farmer
Assumptions/Constraints	The Micro Farmer must have registered with the system
Postcondition upon success	The Micro Farmer has successfully performed an adjustment/action to their farming device(s)
Postcondition upon failure	System notifies user about the error
Trigger	The Micro Farmer selects "Manage farming devices" in the menu
Basic course of action	1. View registered devices 2. Select a device 3. View device info
Extensions	3.a Edit device (reassign) 3.b Remove device
Alternative actions	1.a Register new device 2.a Enter device data 3.a Assign device to a farming site

Title	Manage Plant Layout
ID	UC02
Description	A micro farmer can manage their plant layout by changing/adapting the current plant layout or defining a new one from scratch
Actors	Micro Farmer
Assumptions/Constraints	The Micro Farmer must have registered with the system
Postcondition upon success	The Micro Farmer has successfully performed an adjustment/action to their plant layout
Postcondition upon failure	System notifies user about the error
Trigger	The Micro Farmer selects "Manage plant layout" in the menu
Basic course of action	1. View current plant layout 2. Change current plant layout 3. Save changes
Extensions	3.a Discard changes
Alternative actions	2.a Define new plant layout 3.a Save changes

Title	Manage Grow Programs
ID	UC03
Description	A micro farmer can manage their grow programs by adapting existing ones or creating new ones
Actors	Micro Farmer
Assumptions/Constraints	The Micro Farmer must have registered with the system
Postcondition upon success	The Micro Farmer has successfully performed an adjustment/action regarding grow programs
Postcondition upon failure	System notifies user about the error
Trigger	The Micro Farmer selects "Manage grow programs" in the menu
Basic course of action	1. View grow programs 2. Select grow program 3. Adapt grow program 4. Save changes
Extensions	3.a Remove grow program 3.b Mark grow program as sellable
Alternative actions	2.b Create new grow program 3.b Save new program

Title	Visualize Profit
ID	UC04
Description	A micro farmer can visualize their profit by tracking operational costs, expenses from buying agricultural items and income from selling agricultural items
Actors	Micro Farmer
Assumptions/Constraints	The Micro Farmer must have registered with the system
Postcondition upon success	The Micro Farmer has successfully visualized their profit information
Postcondition upon failure	System notifies user about the error
Trigger	The Micro Farmer selects "Visualize profit" in the menu
Basic course of action	1. View income statement (contains revenue, and different expenses) 2. Generate visualization (i.e. chart) of profit over time
Extensions	-
Alternative actions	-

Title	Place Order
ID	UC05
Description	A Consumer can buy items such as seeds, plant samplings, grow programs and yield
Actors	Consumer (Micro Farmer/Company/Individual),
Assumptions/Constraints	The Consumer must have registered with the system, The inventory needs to have available/sellable items, The Consumer has selected items for purchase
Postcondition upon success	The Consumer has successfully placed an order
Postcondition upon failure	System notifies user about the error
Trigger	A Consumer selects "Check out" in the menu
Basic course of action	1. Enter delivery address 2. Enter payment data 3. Select "Place order"
Extensions	-
Alternative actions	-

Title	Fulfill Order
ID	UC06
Description	A micro farmer or an agricultural company can sell agricultural items such as seeds, plant samplings, grow programs and yield
Actors	Seed Supplier (Micro Farmer/Company)
Assumptions/Constraints	The Seed Supplier must have registered with the system, The inventory needs to have available/for-sale items
Postcondition upon success	Order has been successfully fulfilled and forwarded to the order management
Postcondition upon failure	System notifies user about the error
Trigger	A Consumer has placed an order (see UC05)
Basic course of action	1. View order 2. Initiate shipping of item(s) in order
Extensions	-
Alternative actions	-

Title	Update Device Firmware
ID	UC07
Description	A Device Supplier can access the maintenance data of farming devices and perform firmware updates
Actors	Device Supplier
Assumptions/Constraints	The Device Supplier must have registered with the system
Postcondition upon success	The Device Supplier has successfully performed a firmware update
Postcondition upon failure	System notifies user about the error
Trigger	The Device Supplier selects the "Manage Device Data" option
Basic course of action	1. View list of devices (their maintenance data) 2. Select a device with outdated firmware 3. Update device firmware remotely
Extensions	-
Alternative actions	-

Title	Notify About Failure
ID	UC08
Description	A Device Supplier can send a notification to a Micro Farmer regarding an imminent failure of their farming device, upon viewing the maintenance data and inspecting the error notification sent automatically by the device.
Actors	Device Supplier
Assumptions/Constraints	The Device Supplier must have registered with the system A severe error has occurred on a device
Postcondition upon success	The Device Supplier has successfully sent a notification to the owner of the damaged farming device
Postcondition upon failure	System notifies user about the error
Trigger	Device Supplier gets notification from the device in question and clicks on "Inspect"
Basic course of action	1. View list of devices (their maintenance data) 2. Latest notification message from the device will appear under "Message" 3. Select "Notify owner"
Extensions	-
Alternative actions	-

Title	Initiate Replacement Shipping
ID	UC09
Description	A Device Supplier can initiate shipping of replacement devices or batteries
Actors	Device Supplier
Assumptions/Constraints	The Device Supplier must have registered with the system, A device or its battery is damaged
Postcondition upon success	The Device Supplier has successfully initiated the shipping of a replacement item
Postcondition upon failure	System notifies user about the error
Trigger	Device Supplier gets notification from the device in question and clicks on “Inspect”
Basic course of action	1. View list of devices (their maintenance data) 2. Latest notification message from the device will appear under “Message” 3. Select “Ship replacement device”
Extensions	-
Alternative actions	2.a Select “Ship replacement battery”

Title	Measure Properties
ID	UC10
Description	A Farming Device can measure properties at positions within a farming site
Actors	Farming Device
Assumptions/Constraints	The Farming Device is registered and assigned to a farming site
Postcondition upon success	The Farming Device has successfully logged the measurements into the system
Postcondition upon failure	Server sends error details as response to the Farming Device
Trigger	The Farming Device has performed a measurement at a position within the assigned farming site
Basic course of action	1. Initiate upload of current measurement with timestamp to system server 2. Receive "OK" status from system server indicating successful upload
Extensions	-
Alternative actions	-

Title	Set Farming Actions
ID	UC11
Description	A Farming Device can set farming actions at positions within a farming site
Actors	Farming Device
Assumptions/Constraints	The Farming Device is registered and assigned to a farming site
Postcondition upon success	The Farming Device has successfully logged the performed farming action to the system
Postcondition upon failure	System sends error details as response to the Farming Device
Trigger	The Farming Device has performed an action at a position within the assigned farming site
Basic course of action	1. Initiate upload of performed action data with timestamp to system server 2. Receive “OK” status from system server indicating successful upload
Extensions	-
Alternative actions	-

3.2 LOGICAL VIEW

The logical view of the system focuses on realizing the application's functionality, depicting the structural elements, key abstractions and mechanisms, as well as the separation of concerns and distribution of responsibilities among them.

With the defined desired functionality (see Scenarios / Use Case View) and identified requirements (see Updated System Requirements) in mind, we have created a class diagram on a relatively high level of abstraction, encompassing the crucial elements involved in the domain and using the ubiquitous language when naming them to reflect the domain properly, but also to provide stable grounds for the subsequent implementation.

Figure 8 shows the final version of the logical view of the Micro Smart Farming System (MSFS) for the DEAD milestone, depicting the domain layer with its most important components, as well as the interconnections among them.

Building blocks of domain driven design (DDD) are denoted with according UML annotations. As can be seen in the diagram, we have decided to employ a role-based system for access control; each user of the MSFS has to register and choose an account type. According to the chosen account type, the respective roles are assigned to that account, in turn defining access rights for this user. This mechanism was explained in Section 1.1.2.

Elements that posses an identity and a thread of continuity, such as User and its derived classes, Farming Device, Farming Site, etc., are Entities in our system. As opposed to this, objects such as Maintenance Data or a device's Capability, are Value Objects, and provide more information on, i.e. *describe* the Entities.

Furthermore, for abstracting concepts like creation logic, we incorporated the Factory pattern, particularly using a Grow Program Factory to create Grow Programs. For executing Grow Programs, we envisioned having a Grow Program Execution Service (this will be covered further in the Process View). As for collections of objects that can be used to aggregate, access and store e.g., entities, we utilized repositories as well; Device Log Repository containing Log Items logged from a Farming Device after an Action or a Measurement was performed, and Plant Repository holding Plants.

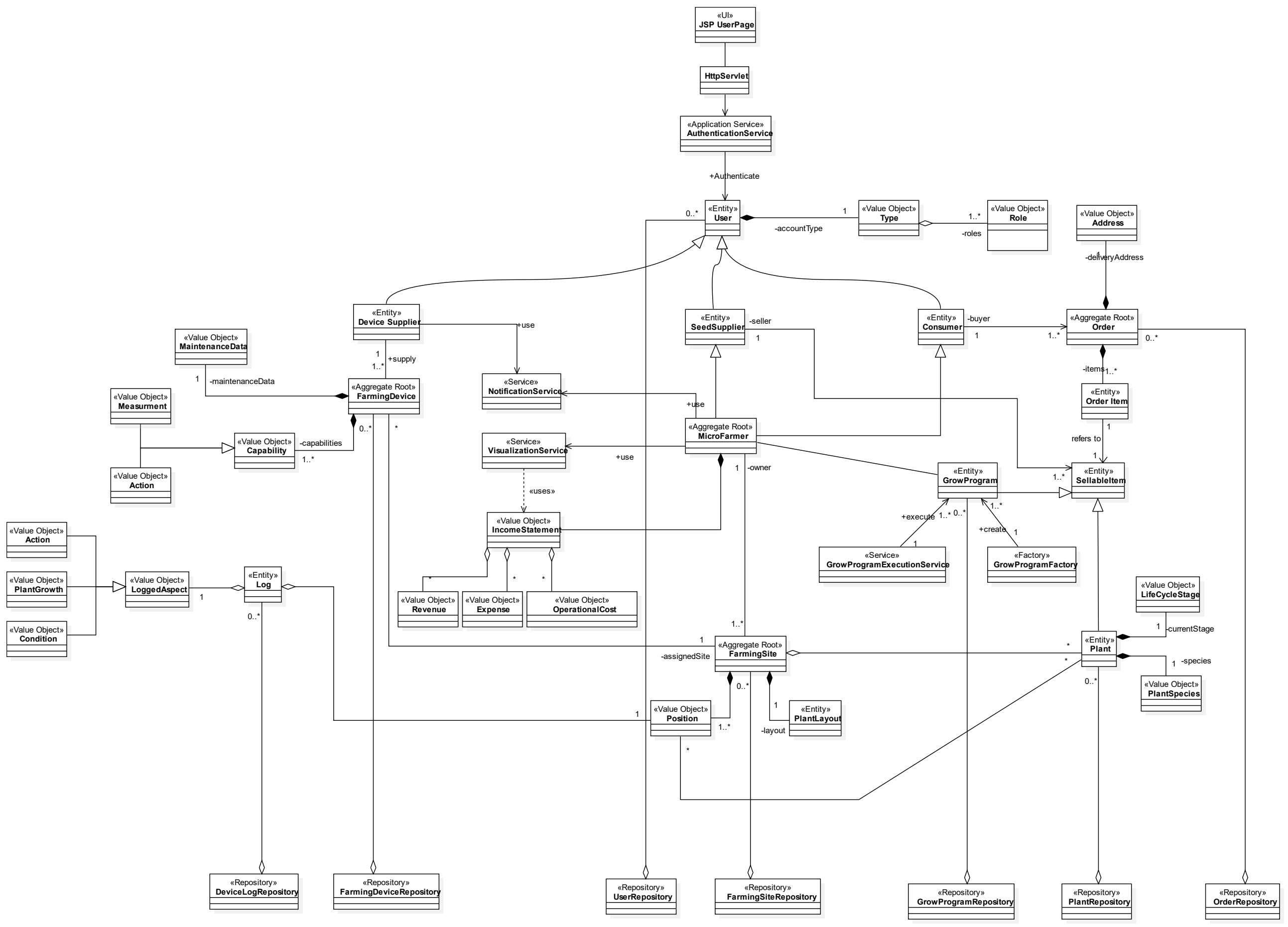

«UI» JSP UserPage
HttpServlet
«Application Service» AuthenticationService
+Authenticate
«Entity» User
0..*
-accountType
1
«Value Object» Type
1..*
-roles
«Value Object» Role
«Value Object» Address
-deliveryAddress
«Entity» Device Supplier
+use
1
+supply
1..*
«Value Object» MaintenanceData
1
-maintenanceData
«Aggregate Root» FarmingDevice
«Service» NotificationService
«Entity» SeedSupplier
-seller
1
«Entity» Consumer
-buyer
1
1..*
«Aggregate Root» Order
0..*
-items
1..*
«Entity» Order Item
1
refers to
1
«Value Object» Measurment
«Value Object» Capability
-capabilities
1..*
0..*
«Value Object» Action
«Service» VisualizationService
+use
«Aggregate Root» MicroFarmer
+use
1
-owner
«Entity» GrowProgram
1..*
«Entity» SellableItem
«uses»
«Value Object» IncomeStatement
+execute
1..*
0..*
1..*
+create
1
«Service» GrowProgramExecutionService
«Factory» GrowProgramFactory
«Value Object» Revenue
«Value Object» Expense
«Value Object» OperationalCost
«Value Object» Action
«Value Object» PlantGrowth
«Value Object» Condition
«Value Object» LoggedAspect
1
«Entity» Log
0..*
«Value Object» LifeCycleStage
1
-currentStage
1..*
1
«Aggregate Root» FarmingSite
-assignedSite
«Entity» Plant
1
-species
«Value Object» PlantSpecies
0..*
«Value Object» Position
1
1..*
1
-layout
«Entity» PlantLayout
«Repository» DeviceLogRepository
«Repository» FarmingDeviceRepository
«Repository» UserRepository
«Repository» FarmingSiteRepository
«Repository» GrowProgramRepository
«Repository» PlantRepository
«Repository» OrderRepository

Figure 8 Logical View

3.3 PROCESS VIEW

A view of the Registration and Login activities (i.e., how we designed/envisioned them) are shown on a high abstraction level in the activity diagram below.

Before a user can access any aspect of the system specific to his account type, they have to be successfully authenticated and authorized within the system (see Login). The realization shown in the diagram with a limitation of 5 login attempts helps prevent a legitimate user's credentials to be brute forced by an attacker.

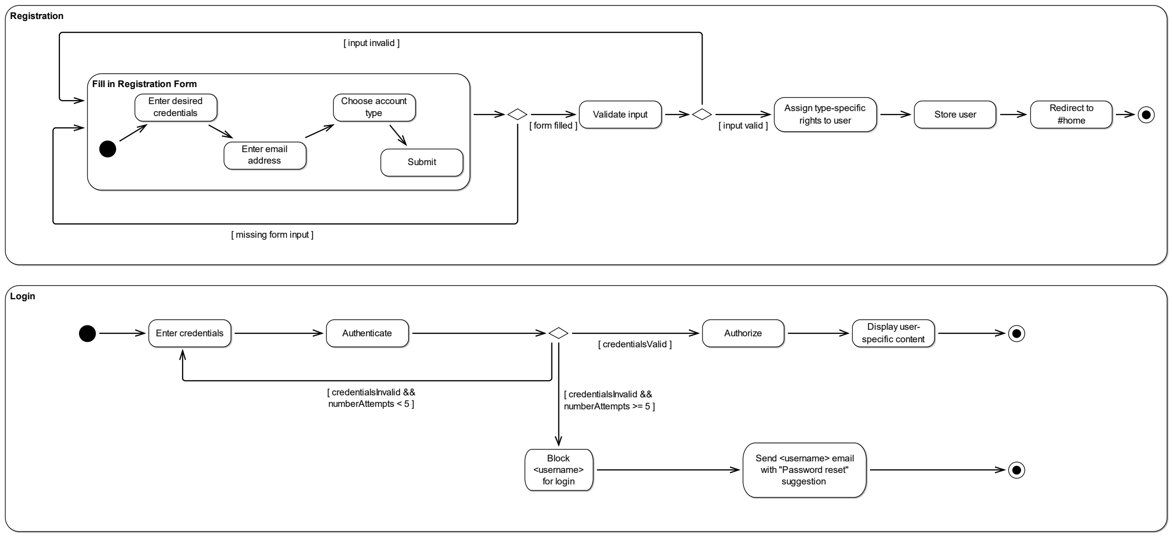

Registration
[input invalid]
Fill in Registration Form
Enter desired credentials
Enter email address
Choose account type
Submit
[form filled]
Validate input
[input valid]
Assign type-specific rights to user
Store user
Redirect to #home
[missing form input]
Login
Enter credentials
Authenticate
[credentialsValid]
Authorize
Display user-specific content
[credentialsInvalid && numberAttempts < 5]
[credentialsInvalid && numberAttempts >= 5]
Block <username> for login
Send <username> email with "Password reset" suggestion

Figure 9 Registration and Login activities

A consumer, who could be a Micro Farmer, a Company or an Individual can buy items in the market. A consumer selects an item to buy and completes the procedure by filling his shipping address and then payment data in the order form. Then the order is saved, and the order confirmation will be displayed to the consumer. Then it will be forwarded to the seller for processing. The seller then initiates shipping and updates his stock and marks the order as shipped.

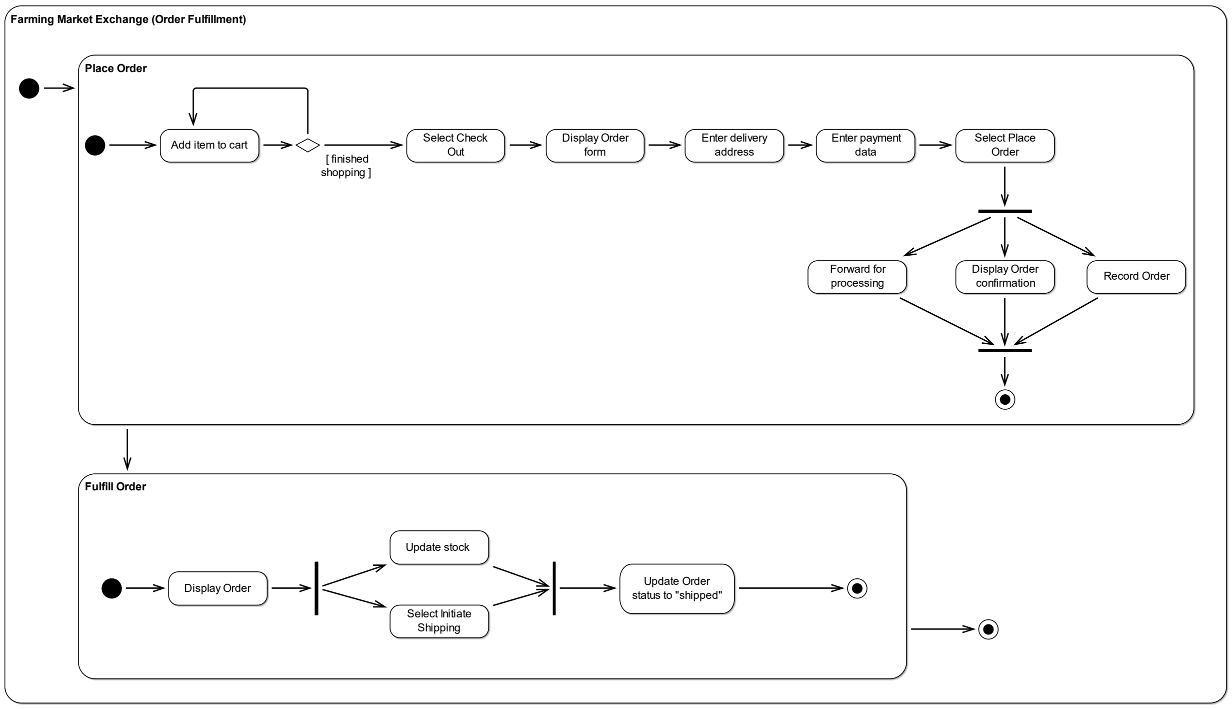


Figure 10 High-level process view showing actions from placing an order to fulfilling it

The Micro Farmer can register new devices to the system. The standard procedure for this use case is shown below.

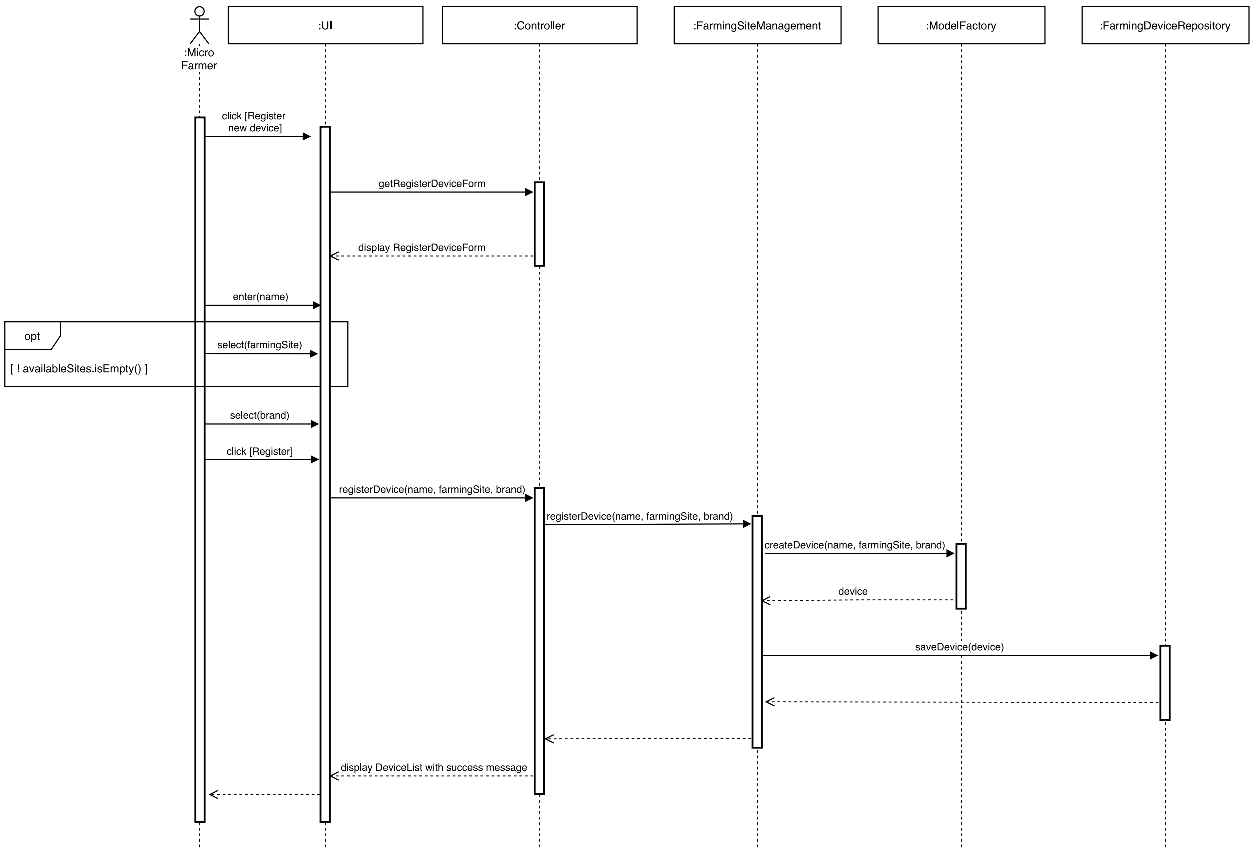

:Micro Farmer
:UI
:Controller
:FarmingSiteManagement
:ModelFactory
:FarmingDeviceRepository
click [Register new device]
getRegisterDeviceForm
display RegisterDeviceForm
enter(name)
opt
select(farmingSite)
[! availableSites.isEmpty()]
select(brand)
click [Register]
registerDevice(name, farmingSite, brand)
registerDevice(name, farmingSite, brand)
createDevice(name, farmingSite, brand)
device
saveDevice(device)
display DeviceList with success message

Figure 11 Registering a new farming device

A micro farming specific use case is the changing the registration settings, i.e., the assignment of the devices to farming sites. This use case is described by the diagram below. The device's assignment to a site can either be (1) changed (if there are available, unassigned sites), or (2) removed (if the device is already assigned to a site):

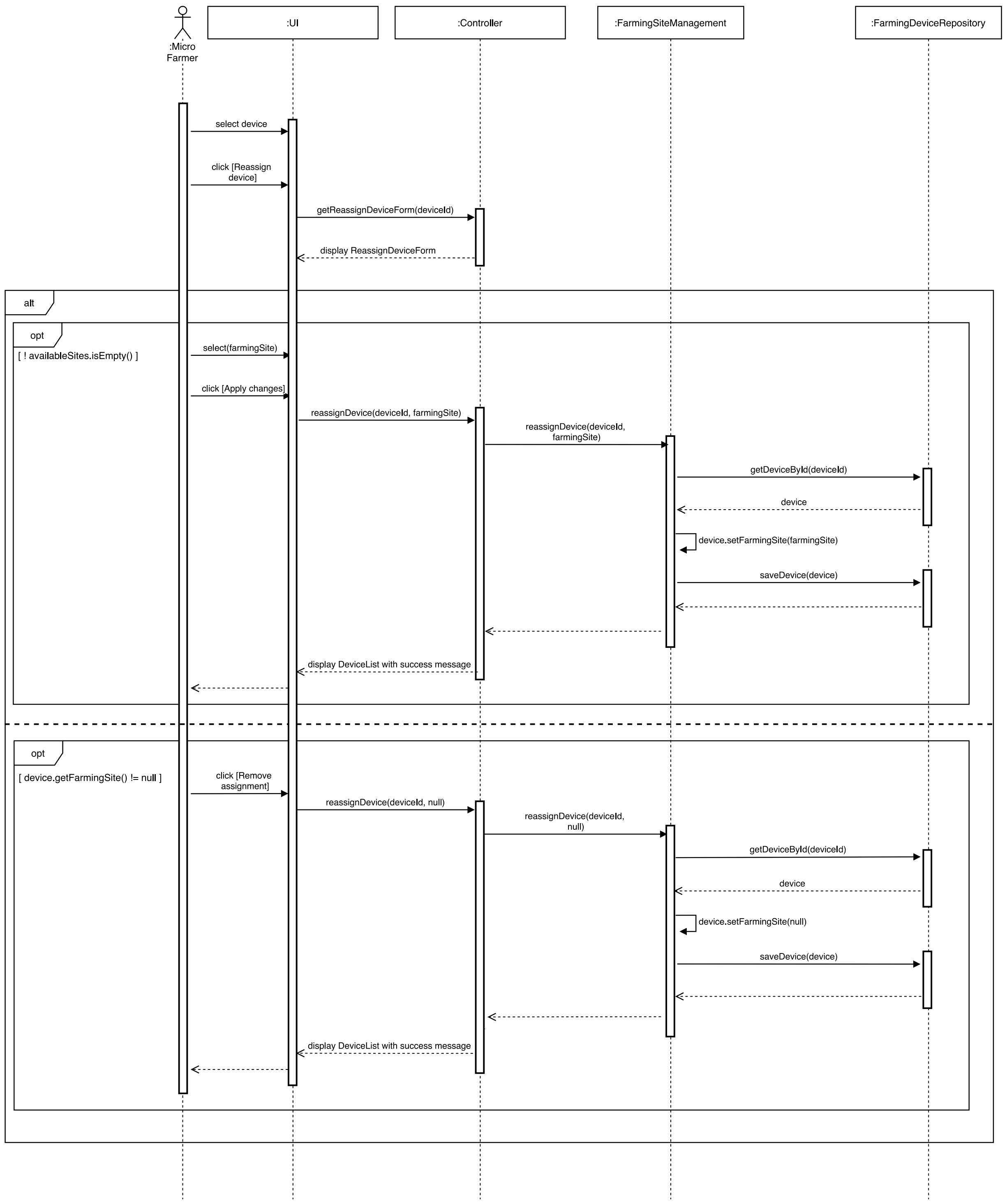

:Micro Farmer
:UI
:Controller
:FarmingSiteManagement
:FarmingDeviceRepository
select device
click [Reassign device]
getReassignDeviceForm(deviceId)
display ReassignDeviceForm
alt
opt
[! availableSites.isEmpty()]
select(farmingSite)
click [Apply changes]
reassignDevice(deviceId, farmingSite)
reassignDevice(deviceId, farmingSite)
getDeviceById(deviceId)
device
device.setFarmingSite(farmingSite)
saveDevice(device)
display DeviceList with success message
opt
[device.getFarmingSite() != null]
click [Remove assignment]
reassignDevice(deviceId, null)
reassignDevice(deviceId, null)
getDeviceById(deviceId)
device
device.setFarmingSite(null)
saveDevice(device)
display DeviceList with success message

Figure 12 Editing the assignment of a farming device

A Device supplier receives an error from a device through his user interface. He then sends a notification to the micro farmer to whom the device belongs to. The micro farmer receives the notification through his system user interface.

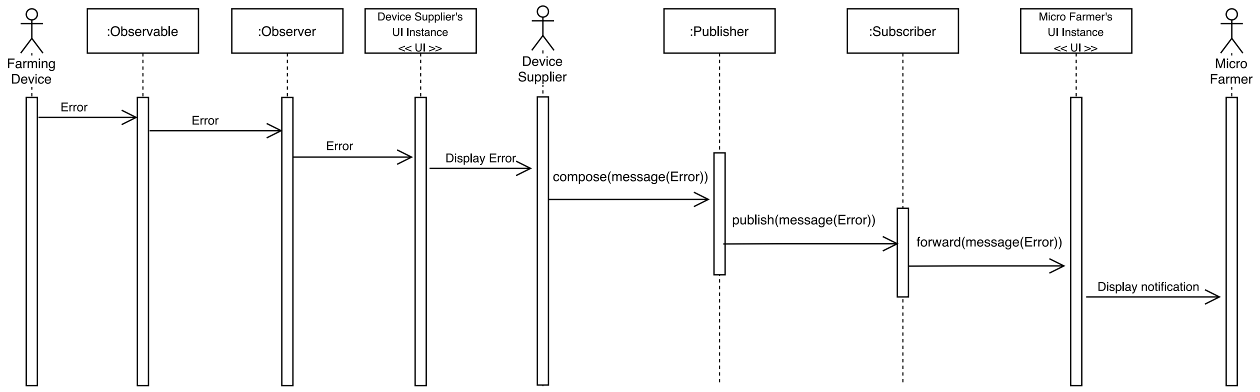


Figure 13 Asynchronous message (notification) propagation after an error occurred on the farming device

The system should support concurrent execution of multiple Grow Programs. This may be realized by utilizing Java Threads, either via implementing the Runnable interface or extending the Thread class for initializing worker threads. The sequence diagram below shows how "active" Grow Programs, i.e. the ones chosen by the user, are executed by the system.

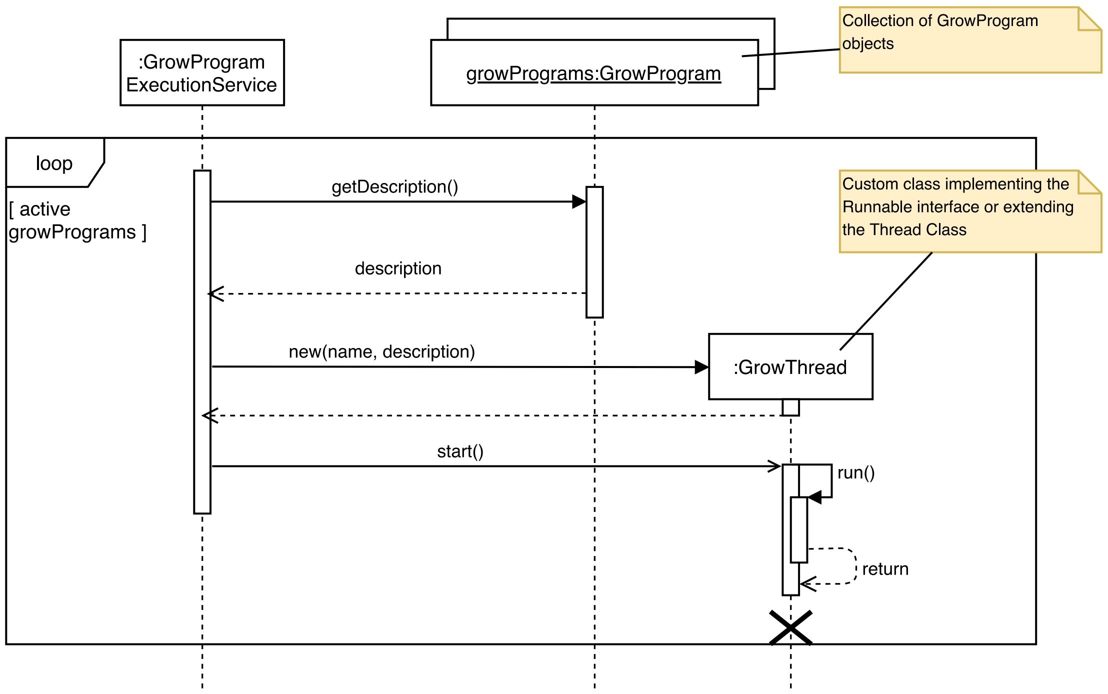

:GrowProgram ExecutionService
growPrograms:GrowProgram
Collection of GrowProgram objects
loop
[active growPrograms]
getDescription()
description
Custom class implementing the Runnable interface or extending the Thread Class
new(name, description)
:GrowThread
start()
run()
return

Figure 14 Simple view of grow programs' execution on java threads

A Micro Farmer can set actions to his farming devices. The process from setting an action to the end (completion) of an action by the device is depicted in the diagram below.

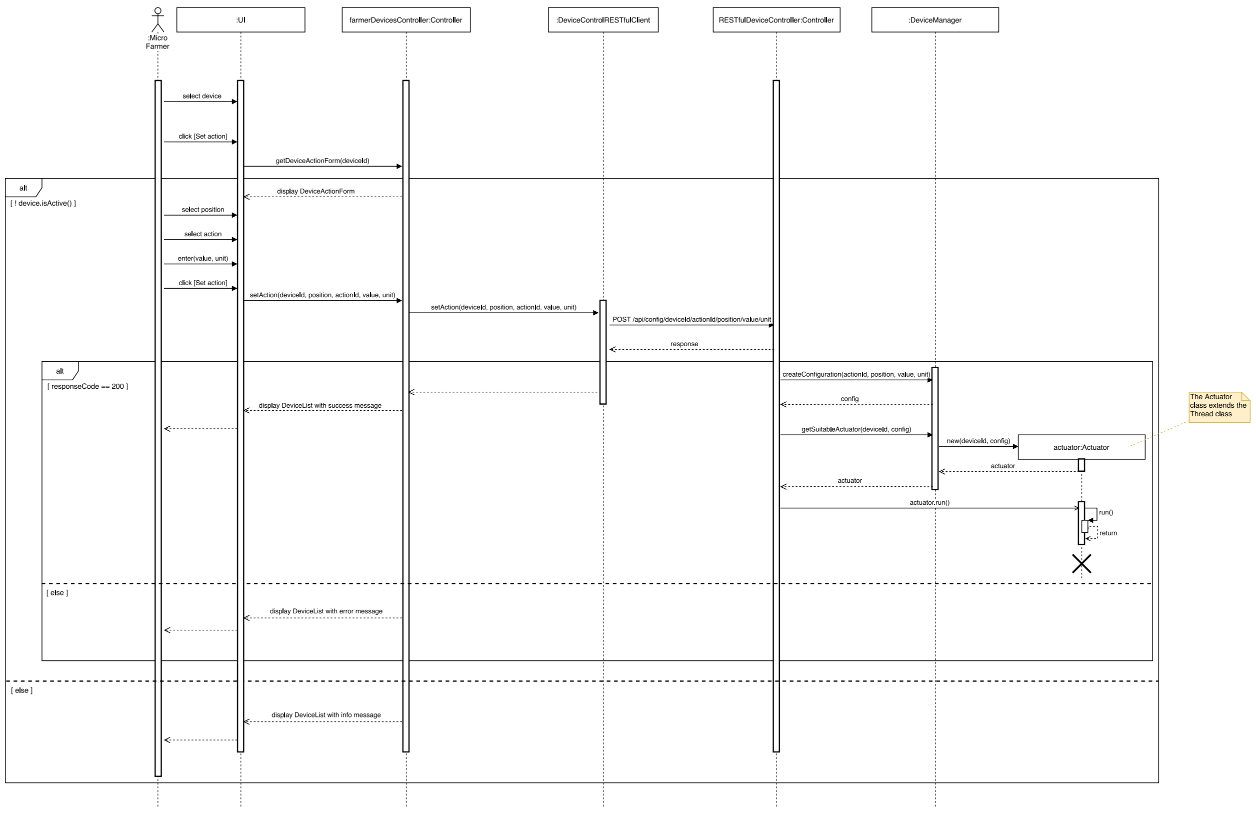

:Micro Farmer
:UI
farmerDevicesController:Controller
:DeviceControlRESTfulClient
RESTfulDeviceController:Controller
:DeviceManager
select device
click [Set action]
getDeviceActionForm(deviceId)
alt
[! device.isActive()]
display DeviceActionForm
select position
select action
enter(value, unit)
click [Set action]
setAction(deviceId, position, actionId, value, unit)
setAction(deviceId, position, actionId, value, unit)
POST /api/config/deviceId/actionId/position/value/unit
response
alt
[responseCode == 200]
createConfiguration(actionId, position, value, unit)
config
display DeviceList with success message
getSuitableActuator(deviceId, config)
new(deviceId, config)
actuator:Actuator
The Actuator class extends the Thread class
actuator
actuator
actuator.run()
run()
return
[else]
display DeviceList with error message
[else]
display DeviceList with info message

Figure 15 Setting an action to a farming device

A Device Supplier can access the maintenance data of farming devices and perform firmware updates

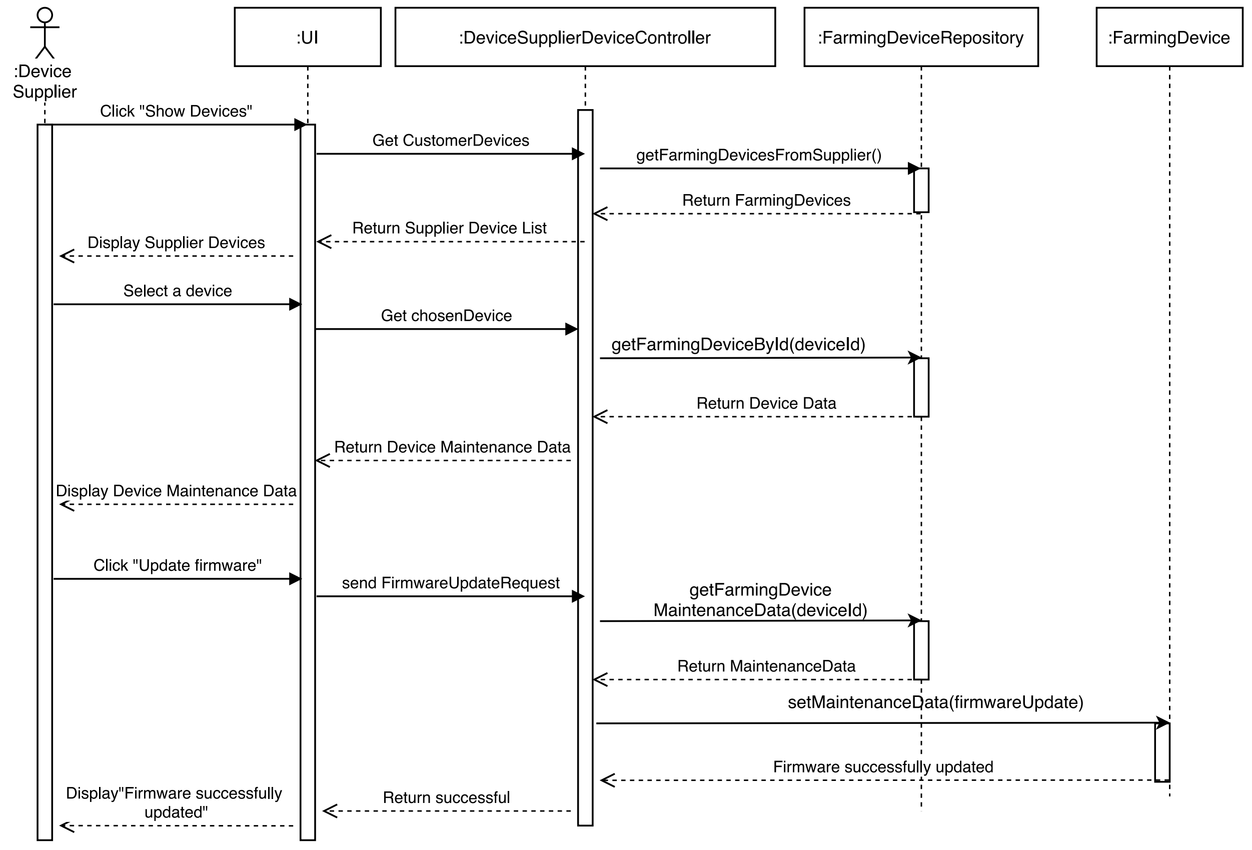

:Device Supplier
:UI
:DeviceSupplierDeviceController
:FarmingDeviceRepository
:FarmingDevice
Click "Show Devices"
Get CustomerDevices
getFarmingDevicesFromSupplier()
Return FarmingDevices
Return Supplier Device List
Display Supplier Devices
Select a device
Get chosenDevice
getFarmingDeviceById(deviceId)
Return Device Data
Return Device Maintenance Data
Display Device Maintenance Data
Click "Update firmware"
send FirmwareUpdateRequest
getFarmingDevice MaintenanceData(deviceId)
Return MaintenanceData
setMaintenanceData(firmwareUpdate)
Firmware successfully updated
Return successful
Display"Firmware successfully updated"

Figure 16 Firmware update use case

A micro farmer can visualize his expenses and incomes by clicking on “Visualize profit”. Then he has two choices, either he selects “View expenses” or “View income” where statistics and visualizations will be displayed.

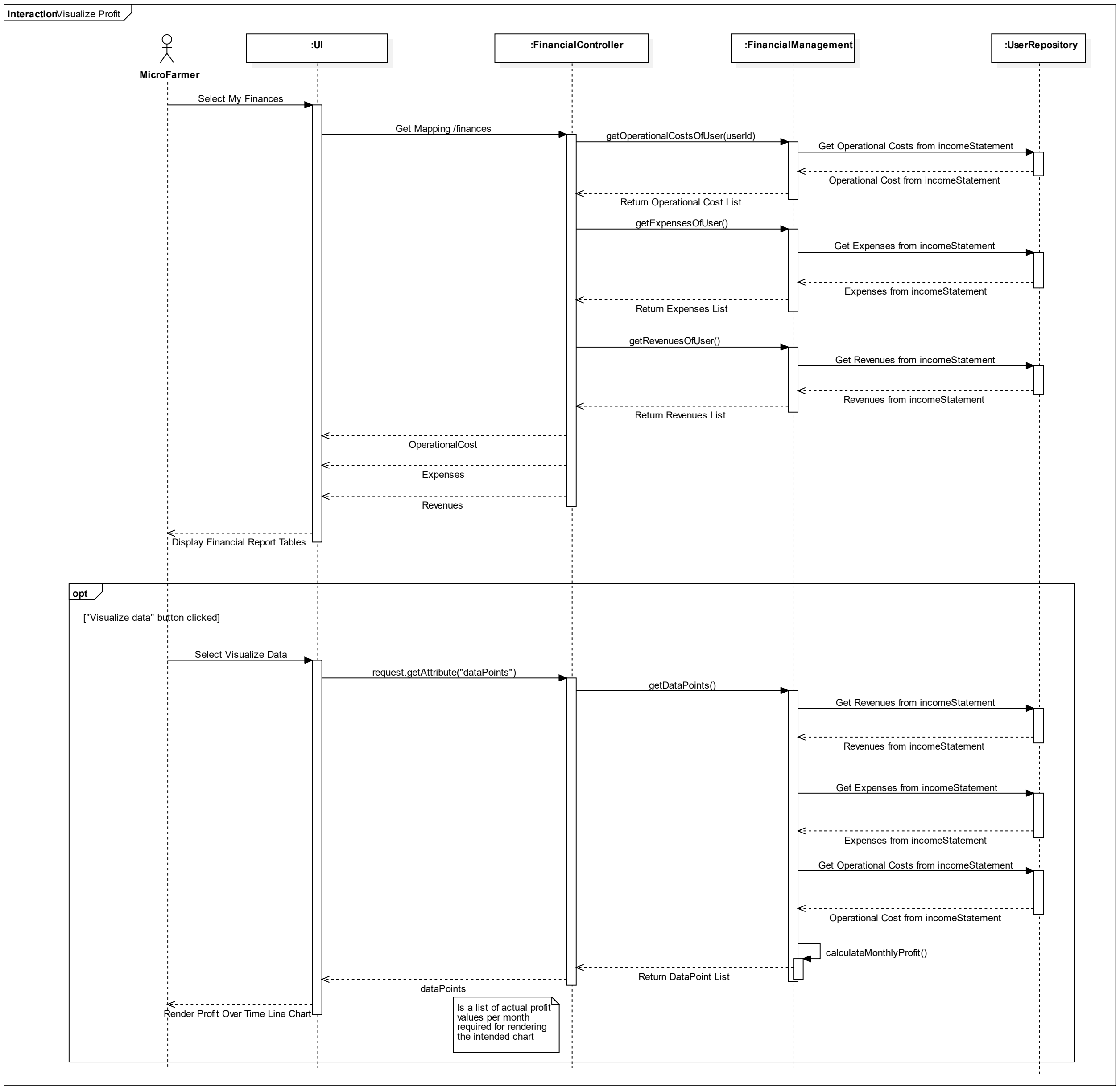

interaction Visualize Profit
MicroFarmer
:UI
:FinancialController
:FinancialManagement
:UserRepository
Select My Finances
Get Mapping /finances
getOperationalCostsOfUser(userId)
Get Operational Costs from incomeStatement
Operational Cost from incomeStatement
Return Operational Cost List
getExpensesOfUser()
Get Expenses from incomeStatement
Expenses from incomeStatement
Return Expenses List
getRevenuesOfUser()
Get Revenues from incomeStatement
Revenues from incomeStatement
Return Revenues List
OperationalCost
Expenses
Revenues
Display Financial Report Tables
opt
["Visualize data" button clicked]
Select Visualize Data
request.getAttribute("dataPoints")
getDataPoints()
Get Revenues from incomeStatement
Revenues from incomeStatement
Get Expenses from incomeStatement
Expenses from incomeStatement
Get Operational Costs from incomeStatement
Operational Cost from incomeStatement
calculateMonthlyProfit()
Return DataPoint List
dataPoints
Render Profit Over Time Line Chart
Is a list of actual profit values per month required for rendering the intended chart

Figure 17 Profit visualization use case

3.4 DEVELOPMENT VIEW

In Unified Modeling Language (UML), the component and package diagrams could be utilized to model and represent the development view. Hence, Figure 18 depicts the development view of the intended system modeled as a component diagram by representing the different components, ports, interfaces of the system and the existing relationship between them.

As it was specified, the application is going to be developed in form of a web-application that enables the users to interact with the system through a web browser on any device connected to the internet.

Based on the key architectural layers, In the presentation layer, the users interact with the web application which communicates with the webserver via REST API. Then the users are authenticated through the authentication component via credentials which are created through the account creation process and their roles are distinguished by the access control component. These components operate in the application layer.

The data is stored and retrieved in/from specific databases created for different purposes in the infrastructure layer. It is important to mention that the different databases such as the user, grow program, data log, farming device, farming site and plant databases could be physically implemented on a single database but due to the different types and the context of the information they are logically separated and considered as different components.

After login, based on the specified role of the users, different components dealing with the business logic which are classified into the domain layer have been identified and depicted. For instance, the Device Support component is dealing with the supporting of the farming devices (by device supplier). As depicted, the maintenance of the devices is specified as provided interface by the Device Support component which is required by the Farming Device Management component in order to perform properly. Moreover, management and maintenance of the farming devices is done based on the data log generated by the devices and stored and retrieved to/from the Data Log Database. The data log is consisting of two types of data which are maintenance data of farming devices and the data about actions performed by the devices. It is notable to mention that the device supplier has access only to the maintenance data through the Maintain Device interface and the data about the performed actions could be only accessed by the micro farmer. This data is managed through the Device Data Management component and passed to the Farming Device Management component.

Furthermore, as an instance, another component is Order Management that enables the users to purchase and sell different commodities such as grow-programs, agricultural yield, seeds and plant saplings based on their roles with the assistance of the external services including the payment and delivery services. The information of the orders is stored and retrieved in/from the Order Database.

There is also another component defined as Notification Management which is operating at the application layer and is responsible for generation and management of different types of system notifications such as failure notification for the micro farmer.

In the next stage, the physical view expounds on which hardware elements do these components reside.

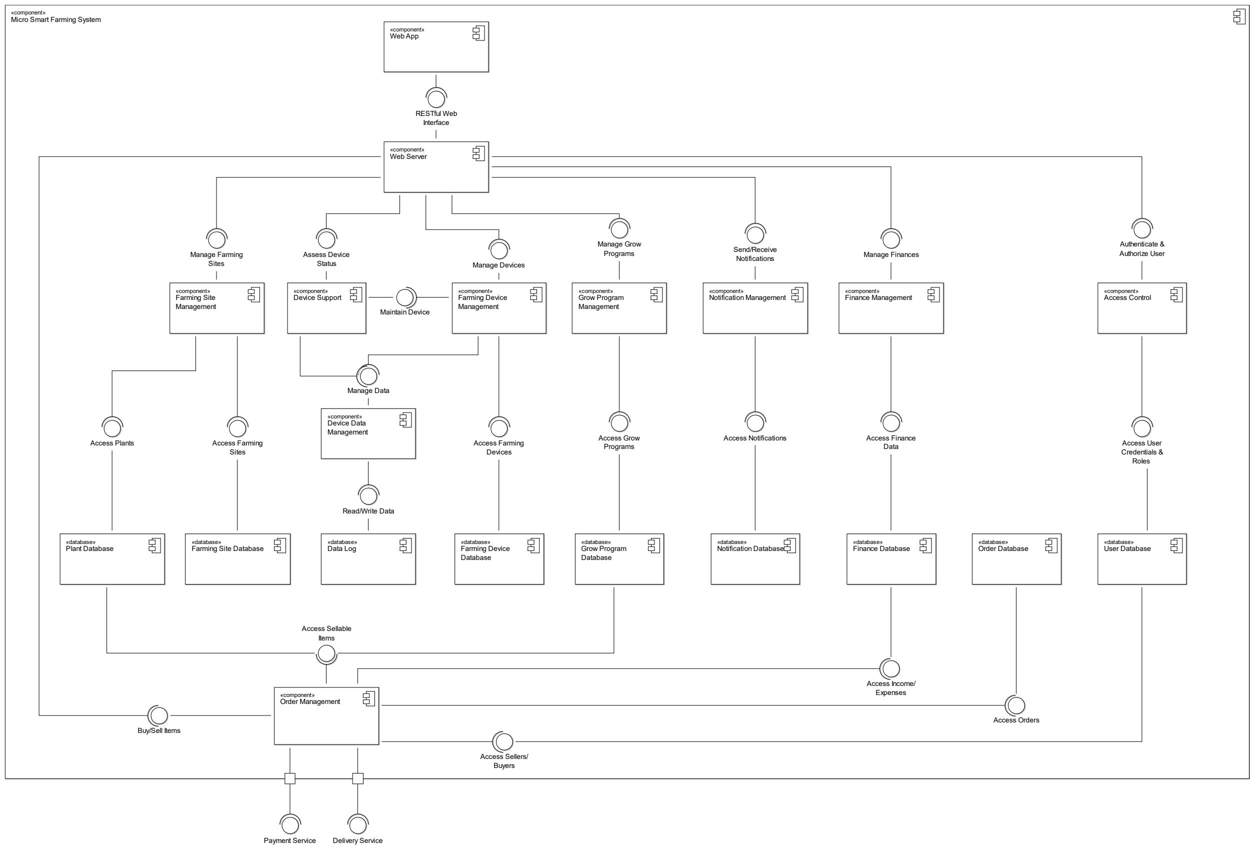

«component» Micro Smart Farming System
«component» Web App
RESTful Web Interface
«component» Web Server
Manage Farming Sites
Assess Device Status
Manage Devices
Manage Grow Programs
Send/Receive Notifications
Manage Finances
Authenticate & Authorize User
«component» Farming Site Management
«component» Device Support
Maintain Device
«component» Farming Device Management
«component» Grow Program Management
«component» Notification Management
«component» Finance Management
«component» Access Control
Manage Data
«component» Device Data Management
Access Plants
Access Farming Sites
Access Farming Devices
Access Grow Programs
Access Notifications
Access Finance Data
Access User Credentials & Roles
Read/Write Data
«database» Plant Database
«database» Farming Site Database
«database» Data Log
«database» Farming Device Database
«database» Grow Program Database
«database» Notification Database
«database» Finance Database
«database» Order Database
«database» User Database
Access Sellable Items
Access Income/ Expenses
«component» Order Management
Buy/Sell Items
Access Orders
Access Sellers/ Buyers
Payment Service
Delivery Service

Figure 18 Development View

3.5 PHYSICAL VIEW

The physical view focuses on hardware topology on which the system executes. A deployment diagram maps a system's software components to the hardware that will execute them and depicts the physical connections between these components, which reflects the system's distributed aspect.

According to the Deployment diagram implied by Figure 19, a User can access the MSFS Web Application via a Computer or a smartphone. The Web Browser uses http protocol to communication with the Application server. The Application Server performs the actual logistics of the system and controls the interconnection and exchange between the client and the system itself. The Application Server and farming devices communicate with each other over the REST API. The farming devices gather data from the surroundings with their sensors and influence the environment with their actuators. With mockup data, the farming devices and their behaviors are simulated. We have used the EMF persistence framework for model persistence. By default, EMF uses XMI (XML Metadata Interchange) to store the model contents.

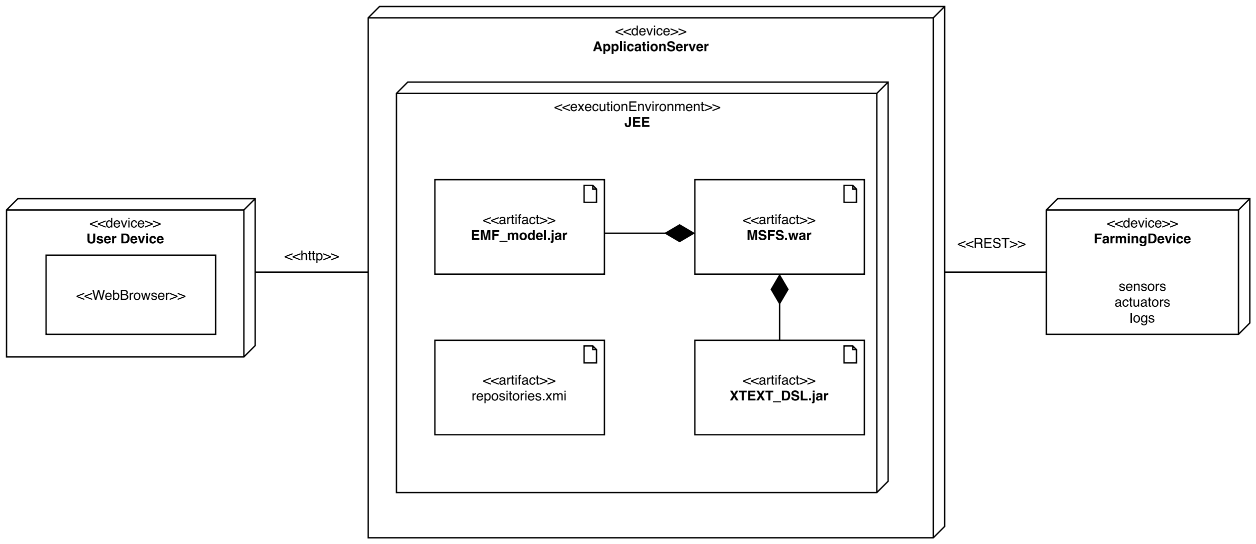


Figure 19 Physical View

4 DSL

4.1 GRAMMAR

```
grammar org.xtext.ase0401.gpdsl.MsfsDsl with org.eclipse.xtext.common.Terminals

generate msfsDsl "http://www.xtext.org/ase0401/gpdsl/MsfsDsl"

GrowProgram:
	'Grow' 'program' name=ID
	description=Description;

Description:
	'Plant' name=ID
	seedTreatment=SeedTreatment?
	seedlingTreatment=SeedlingTreatment?
	youngTreatment=YoungTreatment?
	adultTreatment=AdultTreatment?
	fruitTreatment=FruitTreatment?;

SeedTreatment:
	name='SEED'
	seeding=Seeding?
	replanting=Replanting?
	irrigating=Irrigating
	fertilizing=Fertilizing
	soilHumidtiy=SoilHumidity
	soilAcidity=SoilAcidity;

SeedlingTreatment:
	name='SEEDLING'
	irrigating=Irrigating
	fertilizing=Fertilizing
	soilHumidtiy=SoilHumidity
	soilAcidity=SoilAcidity;

YoungTreatment:
	name='YOUNG'
	irrigating=Irrigating
	fertilizing=Fertilizing
	soilHumidtiy=SoilHumidity
	soilAcidity=SoilAcidity;

AdultTreatment:
	name='ADULT'
	irrigating=Irrigating
	fertilizing=Fertilizing
	soilHumidtiy=SoilHumidity
	soilAcidity=SoilAcidity;

FruitTreatment:
	name='FRUIT'
	harvesting=Harvesting;

Replanting:
	'Replanting'
	'plant' 'out' 'after' plantOuAfter=TimePeriod unit=('day(s)' | 'week(s)');

Harvesting:
	'Harvesting'
	'mode:' harvestMode=HarvestMode;

Irrigating:
	'Irrigating'
	'water' 'every' period=TimePeriod unit=('day(s)' | 'week(s)')
	'amount:' amount=AmountWater ('L' | 'l');

Fertilizing:
	'Fertilizing'
	'fertilizer:' fertilizer=STRING
	'amount:' amount=AmountFertilizer unit=('TS' | 'g')
	'apply' 'every' period=TimePeriod timeUnit=('day(s)' | 'week(s)') ',' repetitions=INT 'times';

Seeding:
	'Seeding'
	'seed' 'at' seedAt=Depth unit=('cm' | 'mm');

SoilHumidity:
	'Soil' 'humidity'
	'ideally' humidity=Saturation '%';

SoilAcidity:
	'Soil' 'acidity'
	'ideally' 'pH' acidity=PH;

enum HarvestMode:
	onClick | manual | auto;
```

DESCRIPTION

A grow program is defined by its name.

A grow program contains an identifier for the plant which it is created for.

Every grow program has exactly 1 definition for each of the plant's life cycle stages.

The stages are marked as optional because of the potential case that the user/creator does not have information (or simply does not want to enter it) for a certain stage of the given plant.

Multiple actions are supported for describing the treatments of the plant in its different stages. Some actions are specific to a stage (e.g., seeding and replanting for the *SEED* stage), while others are common (irrigating and fertilizing). A special action is harvesting, for which different (enum) modes are available and supported by our system (auto, onClick and manual).

Two conditions are supported for describing the ideal environmental setting for the plant in a specific life-cycle stage.

The rules describing the different numerical values are certainly redundant in this form, but the high verbosity stuck after multiple iterations and changes in the DSL design.

```
AmountFertilizer:
            (DOUBLE | INT);

AmountWater:
            (DOUBLE | INT);

TimePeriod:
            (INT);

Depth:
            (DOUBLE | INT);

PH:
            (DOUBLE | INT);

Saturation:
            (DOUBLE | INT);

terminal DOUBLE:

            (INT '.' INT);
```

4.2 EXAMPLE INSTANCE

The example instance bellow demonstrates the full usage of our designed grammar.

```
Grow program MyCarrotProgram
Plant Carrot
SEED
        Seeding
                seed at 2 cm
        Replanting
                plant out after 2 week(s)
        Irrigating
                water every 1 day(s)
                amount: 1 L
        Fertilizing
                fertilizer: 'Urea'
                amount: 2 TS
                apply every 3 day(s), 3 times
        Soil humidity
                ideally 70%
        Soil acidity
                ideally pH 4
SEEDLING
        Irrigating
                water every 2 day(s)
                amount: 1 L
        Fertilizing
                fertilizer: 'Urea'
                amount: 2 TS
                apply every 5 day(s), 2 times
        Soil humidity
                ideally 60%
        Soil acidity
                ideally pH 4
```

```
YOUNG
	Irrigating
		water every 2 day(s)
		amount: 1 L
	Fertilizing
		fertilizer: 'Urea'
		amount: 2 TS
		apply every 7 day(s), 2 times
	Soil humidity
		ideally 50%
	Soil acidity
		ideally pH 4
ADULT
	Irrigating
		water every 3 day(s)
		amount: 0.5 L
	Fertilizing
		fertilizer: 'Urea'
		amount: 1 TS
		apply every 7 day(s), 2 times
	Soil humidity
		ideally 60%
	Soil acidity
		ideally pH 4
FRUIT
	Harvesting
		mode: manual
```

5 M2M TRANSFORMATION

This section describes our chosen method for the M2M transformation, as well as concrete implementation and realization of the chosen method.

Full documentation using the ADD template:

Table 3 Documenting M2M transformation realization design decision using the ADD template

M2M transformation realization	**Use or non-use of Eclipse ATL in M2M transformation**
Problem Statement	Use or non-use of Eclipse ATL for implementing a unidirectional M2M transformation from DSL Instances to the runtime instance of the EMF model
Decision drivers	Usability, availability, user-friendliness of the implemented method
Alternatives	1. Xtext code generation feature 2. Model transformation toolkit Eclipse ATL
Recommendation	Decide for non-use of Eclipse ATL
Decision Outcomes	
Status	Decided
Chosen alternative	1. Chosen alternative is (1) Xtext code generation feature
Justification	The chosen alternative enables easy and flexible integration into our app and allows us to demonstrate its usage and outcome directly via our GUI, without installing any additional tools.
Consequences	The benefits are as described in the Justification. In particular, this choice is in strong alignment with our NFRs (usability, availability and user-friendliness). We have not identified any significant downsides of this choice, especially not affecting our NFRs.
Assumptions	The main assumption was that the learning curve and/or effort for the integration process of the alternative method (ATL) would not be feasible.

As can be seen in the table, we decided to implement the transformation using the Xtext code generation feature. More concretely, we utilized its StandaloneSetup module to create an injector, and with that injector, initialize an XtextResourceSet instance to work with.

We implemented the possibility to enter a string (text) into a text area in our GUI, and the submitted input is converted (if valid, and adhering to our grammar) to a .msfsdsl file, which is then read into an EMF Resource, and then this Resource's contents are traversed and used to instantiate an appropriate GrowProgram instance of our EMF Ecore model.

The described procedure is carried out by a single handler class:

```
@Component
public class DslHandler {
          private Injector injector;
          private XtextResourceSet resourceSet;
          private Resource resource;

          @Autowired
          private GrowProgramManagement mgmt;

          public void setUpHandler() {
                    injector = new MsfsDslStandaloneSetup().createInjectorAndDoEMFRegistration();
                    resourceSet = injector.getInstance(XtextResourceSet.class);
          }

          public void generateDslFromString(String input) {
                    FileOutputStream fileOutputStream = null;

                    try {
                              fileOutputStream = new FileOutputStream("input.msfsdsl");
                              fileOutputStream.write(input.getBytes());
                              fileOutputStream.flush();
                              fileOutputStream.close();
                    } catch (IOException e) {
                              System.out.println("Exception thrown while trying to write file:");
                              e.printStackTrace();
                    }

          }

          public void setResourceFromFile(URI uri) {
                    resource = resourceSet.getResource(uri, true);
          }

          public boolean isModelValid() {
                    IResourceValidator validator = ((XtextResource) resource).getResourceServiceProvider().getResourceValidator();
                    List<Issue> issues = validator.validate(resource, CheckMode.ALL, CancelIndicator.NullImpl);
                    if (issues.isEmpty()) {
                              return true;
                    }
                    for (Iterator<Issue> iterator = issues.iterator(); iterator.hasNext();) {
                              Issue issue = (Issue) iterator.next();
                              System.out.println("Issue: " + issue.getMessage());
                    }
                    return false;
          }

          public void addGrowProgramFromDSL() {
                    EList<EObject> list = resource.getContents();
                    GrowProgram gp = null;
                    for (Iterator<EObject> iterator = list.iterator(); iterator.hasNext();) {
                              EObject eObject = (EObject) iterator.next();
                              if (eObject instanceof GrowProgram) {
                                        gp = (GrowProgram) eObject;
                                        break;
                              }
                    }

                    SeedTreatment seedTreatment = gp.getDescription().getSeedTreatment();
                    msfs_0401.SeedTreatment seedT;
                    if (seedTreatment == null) {
                              seedT = null;
                    } else {
                              seedT = mgmt.defineSeedTreatment(
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getSeeding().getSeedAt()),
                                                  gp.getDescription().getSeedTreatment().getSeeding().getUnit(),

          Double.valueOf(gp.getDescription().getSeedTreatment().getReplanting().getPlantOuAfter()),
                                                  gp.getDescription().getSeedTreatment().getReplanting().getUnit(),
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getSoilHumidtiy().getHumidity()),
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getSoilAcidity().getAcidity()),
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getIrrigating().getAmount()),
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getIrrigating().getPeriod()),
                                                  gp.getDescription().getSeedTreatment().getIrrigating().getUnit(),
                                                  gp.getDescription().getSeedTreatment().getFertilizing().getFertilizer(),
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getFertilizing().getAmount()),
                                                  gp.getDescription().getSeedTreatment().getFertilizing().getUnit(),
                                                  Double.valueOf(gp.getDescription().getSeedTreatment().getFertilizing().getPeriod()),
                                                  gp.getDescription().getSeedTreatment().getFertilizing().getTimeUnit(),
                                                  gp.getDescription().getSeedTreatment().getFertilizing().getRepetitions());
                    }

                    SeedlingTreatment seedlingTreatment = gp.getDescription().getSeedlingTreatment();
                    msfs_0401.SeedlingTreatment seedlingT;
                    if (seedlingTreatment == null) {
                              seedlingT = null;
                    } else {
                              seedlingT = mgmt.defineSeedlingTreatment(

          Double.valueOf(gp.getDescription().getSeedlingTreatment().getSoilHumidtiy().getHumidity()),

          Double.valueOf(gp.getDescription().getSeedlingTreatment().getSoilAcidity().getAcidity()),
                                                  Double.valueOf(gp.getDescription().getSeedlingTreatment().getIrrigating().getAmount()),
                                            Double.valueOf(gp.getDescription().getSeedlingTreatment().getIrrigating().getPeriod()),
                                            gp.getDescription().getSeedlingTreatment().getIrrigating().getUnit(),
                                            gp.getDescription().getSeedlingTreatment().getFertilizing().getFertilizer(),

        Double.valueOf(gp.getDescription().getSeedlingTreatment().getFertilizing().getAmount()),
                                            gp.getDescription().getSeedlingTreatment().getFertilizing().getUnit(),

        Double.valueOf(gp.getDescription().getSeedlingTreatment().getFertilizing().getPeriod()),
                                            gp.getDescription().getSeedlingTreatment().getFertilizing().getTimeUnit(),
                                            gp.getDescription().getSeedlingTreatment().getFertilizing().getRepetitions());
                }

                YoungTreatment youngTreatment = gp.getDescription().getYoungTreatment();

                msfs_0401.YoungTreatment youngT;
                if (youngTreatment == null) {
                        youngT = null;
                } else {
                        youngT = mgmt.defineYoungTreatment(

        Double.valueOf(gp.getDescription().getYoungTreatment().getSoilHumidtiy().getHumidity()),
                                            Double.valueOf(gp.getDescription().getYoungTreatment().getSoilAcidity().getAcidity()),
                                            Double.valueOf(gp.getDescription().getYoungTreatment().getIrrigating().getAmount()),
                                            Double.valueOf(gp.getDescription().getYoungTreatment().getIrrigating().getPeriod()),
                                            gp.getDescription().getYoungTreatment().getIrrigating().getUnit(),
                                            gp.getDescription().getYoungTreatment().getFertilizing().getFertilizer(),
                                            Double.valueOf(gp.getDescription().getYoungTreatment().getFertilizing().getAmount()),
                                            gp.getDescription().getYoungTreatment().getFertilizing().getUnit(),
                                            Double.valueOf(gp.getDescription().getYoungTreatment().getFertilizing().getPeriod()),
                                            gp.getDescription().getYoungTreatment().getFertilizing().getTimeUnit(),
                                            gp.getDescription().getYoungTreatment().getFertilizing().getRepetitions());
                }

                AdultTreatment adultTreatment = gp.getDescription().getAdultTreatment();

                msfs_0401.AdultTreatment adultT;
                if (adultTreatment == null) {
                        adultT = null;
                } else {
                        adultT = mgmt.defineAdultTreatment(

        Double.valueOf(gp.getDescription().getAdultTreatment().getSoilHumidtiy().getHumidity()),
                                            Double.valueOf(gp.getDescription().getAdultTreatment().getSoilAcidity().getAcidity()),
                                            Double.valueOf(gp.getDescription().getAdultTreatment().getIrrigating().getAmount()),
                                            Double.valueOf(gp.getDescription().getAdultTreatment().getIrrigating().getPeriod()),
                                            gp.getDescription().getAdultTreatment().getIrrigating().getUnit(),
                                            gp.getDescription().getAdultTreatment().getFertilizing().getFertilizer(),
                                            Double.valueOf(gp.getDescription().getAdultTreatment().getFertilizing().getAmount()),
                                            gp.getDescription().getAdultTreatment().getFertilizing().getUnit(),
                                            Double.valueOf(gp.getDescription().getAdultTreatment().getFertilizing().getPeriod()),
                                            gp.getDescription().getAdultTreatment().getFertilizing().getTimeUnit(),
                                            gp.getDescription().getAdultTreatment().getFertilizing().getRepetitions());
                }

                FruitTreatment fruitTreatment = gp.getDescription().getFruitTreatment();
                msfs_0401.FruitTreatment fruitT;
                if (fruitTreatment == null) {
                        fruitT = null;
                } else {
                        fruitT = mgmt.defineFruitTreatment(
                                            gp.getDescription().getFruitTreatment().getHarvesting().getHarvestMode().getName());
                }

                mgmt.addGrowProgram(gp.getName(), gp.getDescription().getName(), 0.0, false, seedT, seedlingT, youngT, adultT,
                                fruitT);

        }

}
```

The DSL instance from the previous section can be used to demonstrate the M2M transformation.

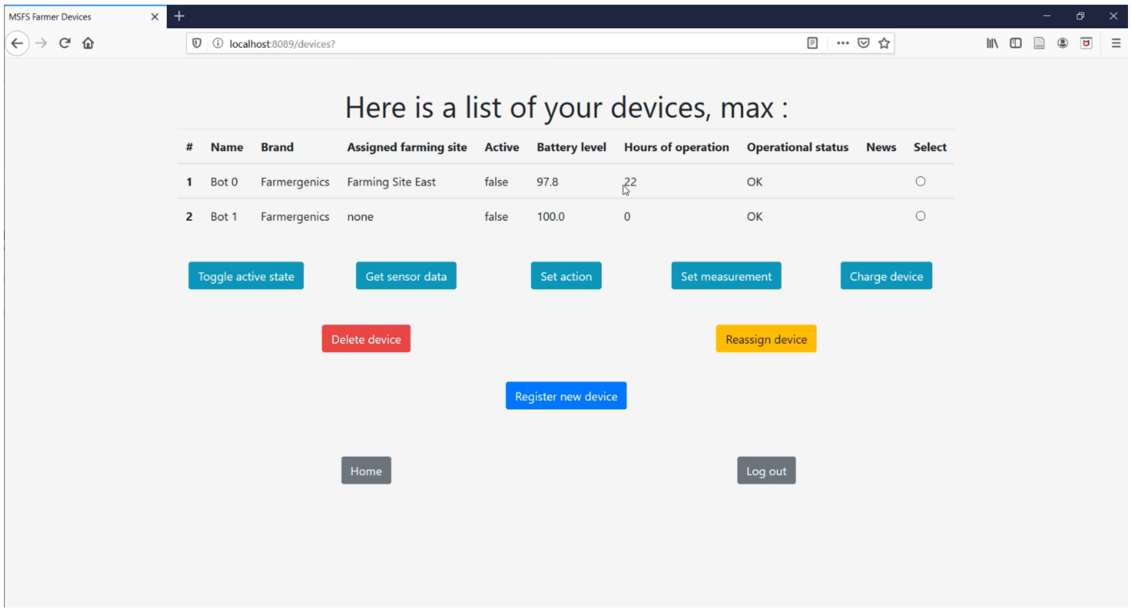
6 HOW TO

6.1 BUILD AND RUN

- The build- and run processes are simplified thanks to the Spring Boot framework
- The absolute requirements/prerequisites to run our system is downloading the following 2 projects from our GitLab repository:
    - `device`
    - `msfsdemo`
- Building and running the app(s) was only tested within the Eclipse IDE, but it should work outside the IDE as well, the only difference would be running the maven and spring-boot commands in the console.
- Building and running within Eclipse IDE:
    - The projects should be imported either via smart git import, or simply as maven projects.
    - First, the device project needs to be run:
        - Right-click on the project's root folder > Run As… > Maven build… > in the *Goals* field, enter "spring-boot:run"
        - Wait for the device to application to build and start
    - Then, run the msfsdemo project, repeating the steps from above:
        - Right-click on the project's root folder > Run As… > Maven build… > in the *Goals* field, enter "spring-boot:run"
        - Wait for the device to application to build and start
- Building and running from the CMD:
    - Navigate into the root folder where the pom.xml file is located
    - Open a CMD window in this directory (or Git Bash)
    - Enter the following command and press enter: "mvn spring-boot:run"
    - Perform these steps for the device project first, and for then the msfsdemo – the order matters!
- Our main web page will then be available at: http://localhost:8089
- Initial data is already provided in the msfsdemo's repository:
    - Test users:
        - Micro Farmer "max:max123"
        - Device Supplier "joe:joe123"
    - You can also register any type of user yourself.
- Have fun!

6.2 USE CASE DEMONSTRATIONS

One of the important use cases implemented during the development phase of the project is Manage Farming Devices in which the user can register, remove and reassign the devices to a farming site. Moreover, specific actions relating to the farming devices could be performed including charging a device, setting measurement, setting action, getting sensory data and toggling active state. These tasks are represented in the underneath image which is a screenshot of the panel which the user interacts with to perform the specified tasks of the Manage Farming Devices use case.

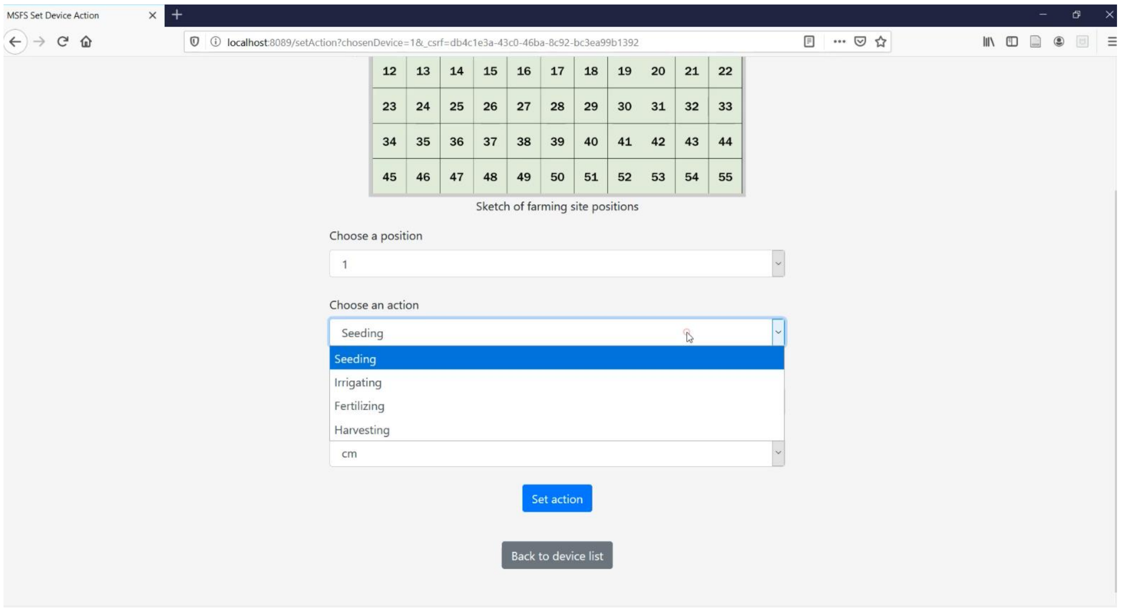


Here is a list of your devices, max :

#	Name	Brand	Assigned farming site	Active	Battery level	Hours of operation	Operational status	News	Select
1	Bot 0	Farmergenics	Farming Site East	false	97.8	22	OK		○
2	Bot 1	Farmergenics	none	false	100.0	0	OK		○

It is also important to mention that, to meet the requirement of the project which states that the system must support the combination of at least 4 different types of farming devices,

or at least 4 different capabilities, by selecting the Set Action option, the user is able to select one of the 4 device capabilities including the seeding, irrigating, fertilizing and harvesting as shown in the underneath image:



12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32	33
34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55

Sketch of farming site positions

Choose a position: 1

Choose an action: Seeding

cm