Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jonas Meyer

Design and development of a modular control software for a use in a greenhouse nutrient delivery and monitoring system for future moon exploration missions

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and Aeronautical Engineering

Jonas Meyer

Design and development of a modular control software for a use in a greenhouse nutrient delivery and monitoring system for future moon exploration missions

Bachelor thesis submitted as part of the Bachelor examination

in the Mechatronik degree program
at the Department of Automotive and Aircraft Engineering
of the Faculty of Engineering and Computer Science
at the University of Applied Sciences Hamburg

In collaboration with:
DLR e.V.
RY-AVS
Robert-Hooke-Str. 7
28359 Bremen

Primary examiner: Prof. Dr. Pawel Buzeck
Secondary examiner: Norbert Toth

Industrial supervisor: Norbert Toth

Date of submission: 20.05.2025

Kurzreferat

Jonas Meyer

Thema der Bachelorthesis

Design and development of a modular control software for a use in a greenhouse nutrient delivery and monitoring system for future moon exploration missions

Stichworte

Mond, Gewächshaus, Hardware-Abstraktion, Automatisierungstechnik, Controlled-environment agriculture

Kurzreferat

In Anbetracht des erneuten Interesses an bemannten Mondmissionen gewinnt die Erhaltung von menschlichem Leben in unwirtlichen Umgebungen an Bedeutung. Insbesondere die langfristige Versorgung solcher isolierten Populationen mit frischem Gemüse ist eine Herausforderung. Vor diesem Hintergrund entwickelt das Deutsche Zentrum für Luft- und Raumfahrt (DLR) ein Gewächshaus für zukünftige Mondexplorationsmissionen. Das Ziel dieser Arbeit ist die Entwicklung der Steuerungssoftware für das EDEN Luna Nutrient Delivery System (NDS). Dies beinhaltet neben der genannten Software auch die Entwicklung einer Software-Schnittstelle um Anpassbarkeit leicht zu gewährleisten.

Ende des Textes

Abstract

Jonas Meyer

Title of thesis

Design and development of a modular control software for a use in a greenhouse nutrient delivery and monitoring system for future moon exploration missions

Keywords

moon, green house, hardware abstraction, automation, Controlled-environment agriculture

Abstract

Considering the renewed interest in manned moon missions, the sustaining of human living in hostile environments gains in relevance. Especially the long-term supply of fresh vegetables of such isolated populations is challenging. Given this specific challenge the German Aerospace Center (DLR) is developing a greenhouse for future moon exploration missions. The goal of this thesis is the development of the control software for the EDEN Luna nutrient delivery system (NDS). This includes, besides the software mentioned, the development of an easily adaptable interface software and the selection of remote I/O devices.

End of text

Contents

List of abbreviations

***AMS* – Atmosphere Management System**

***AVS* – Avionics Systems Department**

***CAN* – Controller Area Network**

***CDH* – Command and Data Handeling**

***CEA* – Controlled Environment Agriculture**

***C.R.O.P.* – Combined Regenerative Organic food Production**

***DHCS* – Datahandeling and Control System**

***DLR* – Deutsches Zentrum für Luft- und Raumfahrt e.V.**

***EDEN* – Evolution and Design of Enviormentally-Closed Nutrition-Sources**

***FEG* – Future Exploration Greenhouse**

***FSM* – Final state machine**

***gtest* – googletest**

***ISRU* – in-situ resource utilization**

***ISS* – International Space Station**

***MEPA* – Mobile Emergency Plant-growing Application**

***MFT* – Mobile Test Facility**

***NASA* –**

***NDS* – Nutrtient Delivery Sytem**

***OBC* – On-Board Computer**

***OUTPOST* – Open modUlar sofTware PlatfOrm for SpacecrafT**

***pando* – PAcket Network DOcumentation model**

***PUS* – Telemetry and Telecommand Packet Utilization ECSS Standard**

***TC* – Telecommand**

***TCS* – Thermal Control System**

***TM* – Telemetry**

***UML* – Unified Modelling Language**

List of Figures

[1] personal communication

List of Listings

1 Introduction

For centuries, conventional agriculture has been the dominant method of food production. However, in the face of growing global challenges – including a rapidly increasing population, diminishing arable land, and stricter regulations for environmental and human health protection – traditional farming is reaching its limits. These constraints have sparked a growing interest in alternative agricultural systems that can ensure reliable, high-quality food production under controlled conditions.

One such approach is Controlled Environment Agriculture (CEA) , which decouples crop cultivation from external environmental factors. By directly managing key growth parameters such as watering, fertilization, climate, and lighting, CEA offers the potential for consistent yields, improved crop quality, and scalable production adapted to demand. In doing so, it addresses many of the limitations inherent in conventional open-field farming.

CEA systems achieve these advantages by creating a tightly controlled growing environment that minimizes external variability. Central to this approach is the precise regulation of critical factors that influence plant development.
The most common ones being:

- Watering and fertilization
- Climate
- Lighting

All this is done with the goal in mind to produce a fresher, higher yielding and more desirable crop with a predictable and schedulable time to market. In addition to this CEA also provides the producer with flexibility to scale output for the current demand.

In addition to economically motivated reasons, CEA also shows great potential beyond large-scale industrial agriculture. Since CEA systems deliberately isolate crop cultivation from external environmental conditions, they can be deployed in regions that are otherwise unsuitable for traditional farming.

Possible applications with a focus on self-sufficiency include supplying remote mining sites with fresh vegetables or enabling sustainable food production in disaster-stricken regions during the recovery phase. In general, self-contained CEA systems are particularly valuable in any context where the reliable supply of fresh food is challenged by substantial logistical constraints.

The Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) is actively pursuing research into such CEA applications, with a particular focus on extreme and resource-limited environments. Insights gained from the development of solutions for disaster relief – such as the Mobile Emergency Plant-growing Application (MEPA) project(Figure 1), which aims to ensure food supply in areas where conventional agriculture is no longer feasible - also inform research for space missions.

Current efforts concentrate on in-situ food production for astronauts during long-term spaceflight and extraterrestrial missions. These activities are brought together in the EDEN initiative (Section 2). As part of this initiative, the EDEN luna (Section 3) test facility is currently under construction.

Figure 1: MEPA Outdoor-Experiment[1]

1.1 Objectives

The primary objective of this thesis is the design and development of the control software for the EDEN luna Nutrtient Delivery Sytem (NDS) . This software is intended to autonomously manage essential plant care functions, with a particular focus on irrigation and fertilization. By ensuring the reliable and consistent provision of these parameters, the system contributes directly to the long-term sustainability and efficiency of the cultivation process. Detailed technical information regarding the subsystem architecture is provided in Section 4.

A further requirement of the control software is its modular architecture. This modularity is crucial in enabling straightforward adaption in the event of hardware modifications - particularly within the interpretation and connection layer, which acts as an interface between sensor hardware and system logic.

In addition to these software-related objectives, this thesis also aims to develop a preliminary concept for the interpretation and connection layer itself, which has not yet been finalized. This concept is not intended to represent a definitive solution, but rather to serve as a demonstrative implementation that can inform and support future development work.

To provide context for the in-house development decision, the thesis includes a brief overview of commercially available software solutions that could, in principle, be applied to the NDS . This comparison concludes with a summary of the reasons that ultimately led to the decision to pursue a custom, internally developed system.

2 EDEN

To better understand the relevance of the EDEN initiative, the following section provides an overview of the potential of in-situ production for future space exploration. In addition, the motivations, benefits, and objectives of the EDEN initiative are outlined.

For the foreseeable future, supplying astronauts with food and other necessities will remain essential. All goods required in space currently need to be transported from Earth, which entails significant logistical and financial effort. Consequently, space agencies are increasingly investigating the potential of in-situ resource utilization (ISRU) to reduce the dependency on Earth-based resupply. These efforts range from conceptual studies on fuel production on the Moon to more tangible experiments conducted onboard the International Space Station (ISS) . Notable examples include the in-space fabrication of tools[8] and the cultivation of plants under microgravity conditions[9]. Although these projects have been limited in scale, they provide valuable scientific data and lay the foundation for future research.

The relevance of these investigations will continue to grow as space missions become longer, more autonomous, and target destinations further away from Earth. In this context, in-situ production of food and other consumables offers a promising opportunity to reduce supply chain dependencies and mission costs.

This is where the EDEN initiative comes in. Its goal is to develop solutions for the autonomous cultivation of fresh produce in space, thereby reducing the need for frequent resupply missions. Beyond its space-related focus, the initiative also contributes to the broader field of self-contained CEA systems through ongoing research activities. The knowledge gained from the EDEN experiments benefits other related projects - such as the terrestrial MEPA system - which aim to adapt space-derived technologies for use in extreme environments on Earth[10].

In addition to logistical advantages, facilities developed under the EDEN initiative may also have a positive psychological impact on astronauts during long-duration missions. Participants in NASA's Veggie investigations reported increased morale when they were able to supplement their diets with freshly grown food and engage in plant care[11]. Similarly, crew members involved in the EDEN ISS mission described the presence of living plants and the availability of fresh produce as a significant psychological benefit[2].

The long-term goal of the EDEN initiative is to develop a fully validated, flight-ready facility for fresh food production on future lunar missions by 2030[12]. To support this objective, two major test facilities have already been built and operated for extended periods. These facilities are described in the following sections.

2.1 EDEN ISS

EDEN ISS MFT was a research facility located at the German Neumayer III Antarctic Station. Comprising two interconnected shipping containers, the facility operated from 2018 to 2023 and served as a testbed for closed-loop CEA systems under harsh environmental conditions[13]. During its operational period, EDEN ISS provided the Neumayer station crew with year-round access to fresh produce.

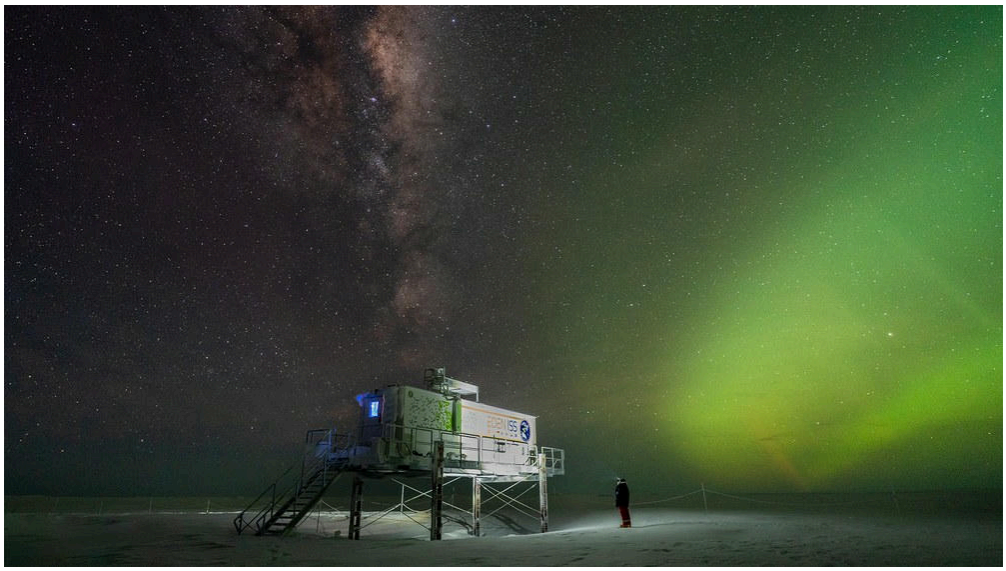


Figure 2: The EDEN ISS MFT container-based research facility at Neumayer III Station in Antarctica[2].

The facility employed a completely soil-less aeroponic system, supplying nutrients and water directly to the plant roots via misting. It also incorporated subsystems for nutrient delivery, lighting, air treatment, and monitoring of food quality and safety[14].

At the end of the five-year mission, the MFT was dismantled and returned to Bremen, Germany.

2.2 EDEN Lab

Currently, the EDEN initiative operates the EDEN Lab at the Planetary Infrastructure Laboratory on the DLR campus in Bremen(Figure 3). Established in 2014, this facility focuses on gaining expertise in plant cultivation within closed-loop CEA systems[10]. It incorporates lessons learned from the EDEN ISS mission and is continuously updated and improved.

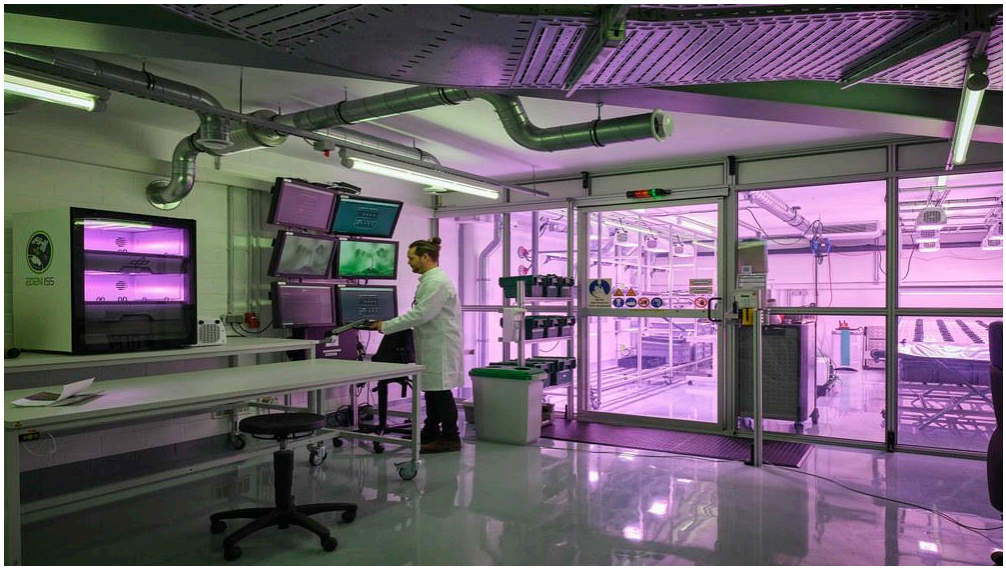


Figure 3: Interior view of the EDEN Lab facility at the DLR campus in Bremen with laboratory manager[3].

In addition to comprehensive monitoring systems, controllable lighting, irrigation, and temperature regulation, EDEN Lab also allows for the manipulation of atmospheric composition within the sealed growth chamber - e.g., by injecting CO_2. Like EDEN ISS, the lab uses aeroponics and omits any use of soil.

3 EDEN LUNA

EDEN LUNA(Figure 4) is the evolution of the EDEN ISS project. After concluding operations in Antarctica and arrival back in Bremen the EDEN ISS MFT is currently being refurbished.

Figure 4: EDEN LUNA with integration tent on the left.

Following this complete redesign of the interior and all subsystems the container is supposed to be integrated into the LUNA analog facility located in Cologne later this year.
The overall structure of the facility is comparable to the EDEN ISS facility. It is split into two sections. The first housing the majority of equipment and space for crop examination and the second comprised of the FEG . This FEG is where plants will be cultivated (see Figure 5).

Figure 5: FEG of EDEN LUNA, under construction

3.1 Project goals

Aim of this project is to gain further experience on how to integrate such a greenhouse into a luna habitat. As part of the LUNA analog facility EDEN LUNA is also supposed to be used for training future astronauts.
I addition to this, procedures for growing food in a luna setting are supposed to be developed. Another point of interest is the controlling of the facility from the EDEN control in Bremen.

This project in another step towards the end goal of the initiative to provide a mature hardware design by 2030.

3.2 System architecture

EDEN LUNA is composed of numerous independent subsystems (see Figure 6), each fulfilling a dedicated set of functions. These subsystems are being developed by specialized teams at the DLR .

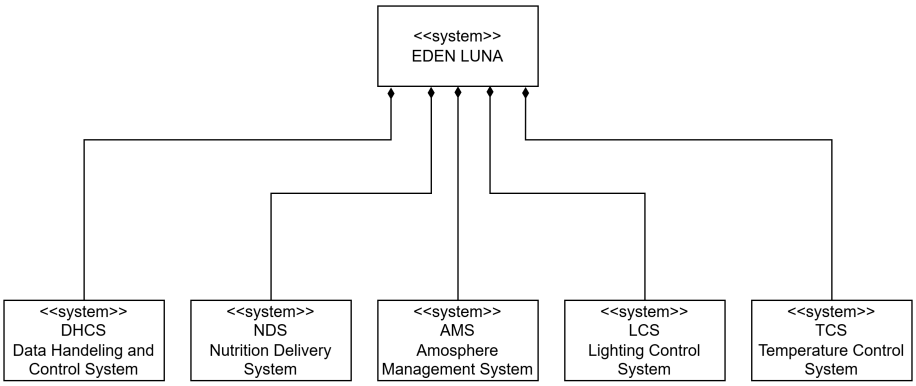


Figure 6: Reduced view of system architecture of EDEN LUNA

Most of the hardware and control units are developed and manufactured in-house to ensure system coherence, minimize dependencies, and allow for rapid iteration and optimization.

At the heart of the system lies an industrial-grade On-Board Computer (OBC) , which serves as the central interface to all subsystems. It communicates via a variety of standard protocols, including Ethernet, Controller Area Network (CAN) , and USB with these subsystems. This central unit also executes the main control software responsible for managing several critical subsystems, including the Atmosphere Management System (AMS) , Thermal Control System (TCS) , and NDS .

Due to its specific relevance to this thesis, the following section will provide a detailed discussion of the NDS . The other subsystems mentioned will not be described in full; however, key aspects will be introduced where necessary for context or system understanding.

4 NDS

The primary function of the NDS is to provide the crops cultivated in the FEG with water and nutrients. Reliability is of particular importance due to the system's aeroponic design: no growing substrate is used, and the plant roots must be regularly sprayed with a nutrient solution. This typically occurs every few minutes. It is critical that the roots do not dry out, as this would impair their ability to absorb nutrients and water.

If the NDS fails, the fine root hairs responsible for nutrient and water uptake will die off. While they can regenerate, their temporary absence significantly limits the plant's ability to sustain itself, causing stress that may lead to irreversible damage or death.

In addition to this core functionality, the NDS for EDEN LUNA must also fulfill the following tasks:

- Initial preparation of the nutrient solution
- Continuous conditioning and monitoring of the solution
- Adjustment of nutrient composition according to crop-specific needs
- Delivery of the nutrient solution to the plant roots
- Recovery and recycling of unused nutrient solution

Precise control of nutrient formulation is not only relevant for space applications such as EDEN LUNA. It also offers potential for terrestrial CEA systems. Several commercial providers offer fertigation systems aimed at these applications.

4.1 Industrial solutions

This section presents several commercially available fertigation systems that could meet the functional requirements of the NDS . For comparison, a baseline is defined based on the current hardware concept (Figure 11).

The system must support:

- Four separate nutrient stock solutions
- One base (alkaline component)
- One acid

Commercial systems are typically divided into two operating principles:

- Batch or mixing tank systems: These prepare nutrient solution in larger quantities and store it for later use.
- Inline or injection systems: These inject concentrated nutrients directly into the water stream during irrigation events[15].

Injection systems usually save space, as large tanks are not needed. However, in EDEN LUNA, where aeroponics is used and unused nutrient solution must be recirculated, the advantages of injection systems may be diminished. Additional hardware may be required to enable this recirculation. Nevertheless, both system types are discussed below.

4.1.1 Argus

Argus is a well-established company in the CEA sector, offering a broad range of solutions for greenhouse automation, including fertigation, irrigation, wireless sensors, and control software.

For fertigation, Argus provides the *Multifeed RM Nutrition Injection System*, a rack-mounted injection system supporting up to 16 injection heads, along with acid and base dosing modules. The system also includes integrated monitoring features[4].

Figure 7: Argus *Multifeed RM Nutrition Injection System*[4]

Systems from Argus were previously used in the EDEN ISS project.

4.1.2 Priva

Priva is another company offering automation solutions for both CEA and building management.

Within its product portfolio, Priva provides both mixing tank and injection-based fertigation systems. Relevant systems for the NDS include the *NutriFlex*(Figure 8) (batch) and *NutriJet* (injection) series. Both options can be equipped with at least ten dosing channels for delivering fertilizers or pH regulators such as acids and bases[16], [17].

Figure 8: Priva NutriFlex[5]

Priva systems are marketed for indoor plant cultivation. A NutriFlex-like system is currently used in the EDEN Lab. Monitoring and control software is included.

4.1.3 Spagnol

Spagnol is an Italian supplier of fertigation systems and other CEA technologies.

The *EvoMix* system(Figure 9) from Spagnol supports up to eight fertilizer channels and two acid/base channels. It was developed for use in closed-loop irrigation systems[6] in mind. An injection unit with identical features is offered under the name *EvoJet*.

Figure 9: Spagnol EvoMix[6]

Like other vendors, Spagnol offers software for system monitoring, control, and integration.

4.2 Reasons for in house development

Although commercial solutions exist and were used in EDEN ISS and EDEN Lab, the NDS for EDEN LUNA is being developed in-house. This decision is based on operational experience and internal evaluations at the DLR , including staff interviews and technical reports[2].

Adaptability

Modifying commercial systems was difficult and required direct vendor support. On-the-fly improvements based on operational findings were not feasible.

Integration difficulties

Commercial systems often lack compatibility with external control systems such as the OBC . At EDEN Lab, an external computer had to be installed for fertigation control. Integration of additional sensors was also limited by unsupported interfaces.

[2]personal communication

Remote monitoring limitations
During EDEN ISS operations in Antarctica, high data bandwidth was needed for vendor software. This proved problematic for remote control and software updates. For future lunar use, this limitation is a critical concern and must be addressed in the new design.

Support dependency
Vendor support was inconsistent and depended heavily on the motivation of individual contacts. Given the specialized nature of the project and low commercial interest, this is understandable, but it hampers long-term development.

5 Software requirements

Since the main aspect of this thesis is the design and implementation of the software running the NDS , this section describes the specific requirements for the software.

Furthermore, two important software libraries are introduced that are required to be used.

In the following, the software to be developed and all adjacent original code will be referenced as *the software*, unless stated otherwise.

5.1 Software required to use

The Avionics Systems Department (AVS) of the DLR develops software libraries and application-level software for spacecraft. In addition, AVS also develops subsystems for Command and Data Handeling (CDH) , power supply, and communication[18].

Since this thesis is written in collaboration with AVS , software produced by this department is to be used.

5.1.1 OUTPOST

Open modUlar sofTware PlatfOrm for SpacecrafT (OUTPOST) is the most important product of the AVS software development group. This flight software library serves as the basis for all software solutions provided by the group. The library covers a broad range of features, from hardware and operating system abstraction to driver interfaces and communication protocols.

The library was developed with the goal of enabling reliable, efficient, and cost-effective development of flight software for a variety of spacecraft[19].
OUTPOST is divided into two components: *OUTPOST-core* and *OUTPOST-satellite*. *OUTPOST-core* contains most of the features. *OUTPOST-satellite* contains closed-source code that may be subject to export control regulations[20].

Since the software developed in this thesis is intended to be integrated into the regular software produced by AVS , the use of OUTPOST is mandatory. As such, the software will heavily rely on this library for many of its implemented features.

5.1.2 pando

PAcket Network DOcumentation model (pando) is another library developed by the AVS software development group. It is part of the group's efforts to integrate model-based software engineering into their development workflow[19].
pando provides functionalities for defining a Telemetry (TM) / Telecommand (TC) model and generating compatible code for OUTPOST . The generated code ensures that the developed software correctly responds to incoming TCs and produces valid outgoing TM .

This functionality will be used in this thesis to enable telecommanding and remote monitoring via telemetry.

5.2 General requirements

Due to the fact that the NDS design was not finalized at the beginning of this thesis, the software design must offer a certain degree of flexibility.
Foundational aspects such as the make and model of the OBC and the overall sensor and actuator layout were already defined. However, the final specification of the interpretation and connection layer was still pending.

This layer is responsible for interfacing sensors and actuators with the OBC and, if required, for integrating additional networking components.
A simplified overview of the planned data flow, with the components still under development marked, is shown in Figure 10.

From this, it follows that the software architecture must allow for changes to this layer with minimal modifications to the application logic.
This decoupling ensures that future hardware updates or reconfigurations do not require deep structural changes to the core logic, thus reducing development effort and integration risk.

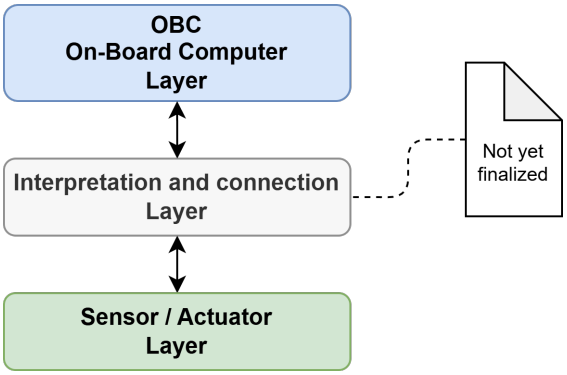


Figure 10: Simplified view of Sensor and Actuator data flow

In addition to these structural requirements, the software must support control of the two core functions of the NDS :

1. the regular irrigation of the crop
2. the preparation and monitoring of the nutrient solution

Furthermore, although hard real-time behavior is not required, the system is expected to operate deterministically within predictable time bounds.
This includes timely responses to incoming commands and periodic sensor-driven control routines.

Beyond these main features, the software must also meet the following additional requirements:

- The software must be executable on the mission's Ubuntu-based OBC .
- It must offer a similar feature set to the current EDEN Lab implementation.
- It must use OUTPOST to ensure compatibility.
- It must support telecommanding and telemetry-based monitoring.

5.3 Hardware constraints

In addition to the requirements listed in the previous section, the hardware design of the NDS imposes specific constraints - particularly regarding sensors and actuators.

Due to the evolving development status of the NDS , the software will be implemented with reference to Figure 11.
Only the sensors and actuators listed in this diagram will be considered for the current implementation.

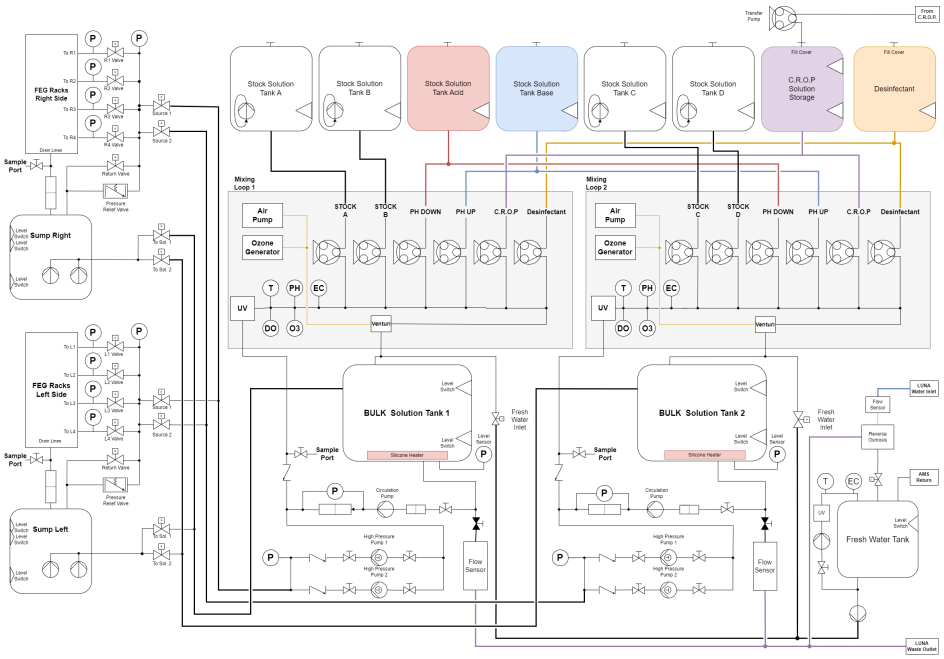


Figure 11: Schematic diagram of the NDS hardware design [7][3]

A detailed list of all identified sensors and actuators, along with additional information, can be found in Section A.2 in the appendix.

[3]personal communication

6 Design

The purpose of this section is to define a software design that satisfies the requirements outlined in Section 5.
The chapter is divided into three main sections:

First, the TMTC model is described, which defines how the system will be monitored and controlled remotely.
This part is largely independent of the remaining software structure.

Second, the main software architecture is presented, forming the technical backbone of the implementation.

Finally, the control algorithms are discussed, which are tightly coupled to specific hardware signals and subsystem behavior.

6.1 TMTC model

To enable remote control and monitoring of the subsystem, the software uses the Telemetry and Telecommand Packet Utilization ECSS Standard (PUS) .
All packets are received by the Datahandeling and Control System (DHCS) and distributed to the respective subsystems.
Packets relevant to the NDS are referred to in the following as *the packets*.

The pando utility is used to generate code for handling received telecommands and for assembling telemetry packets. This ensures the operability of the software as part of EDEN luna.

```
<enumeration name="Enum Stock Solution Id" uid="EnumStockSolutionId"
        width="8">
        <entry name="stock_a" value="0"></entry>
        <entry name="stock_b" value="1"></entry>
        <entry name="ph_acid" value="2"></entry>
        <entry name="ph_base" value="3"></entry>
        <entry name="crop" value="4"></entry>
        <entry name="desinfec" value="5"></entry>
</enumeration>
```

Figure 12: Section of `nds_luna.xml` showing the definition of `EnumStockSolution` data type

pando expects all packets to be defined in a structure description `.xml` file.
In addition to packet definitions, this file also specifies custom data types (Figure 12) and parameters that are used to describe and interpret these packets.

Together with a corresponding mapping `.xml` file, pando generates source files for telemetry packet generation and telecommand reception.

This approach ensures that telemetry data is transmitted in the correct format and that incoming commands trigger the appropriate system responses.

6.1.1 Telemetry

Telemetry refers to any data transmitted back from the spacecraft.
In this case, it is used to reflect the current state of the subsystem.

It includes:

1. Sensor data
2. Device and valve states
3. Other operational data

All sensors and actuators listed in Section A.2 in the appendix are part of the telemetry set.

6.1.2 Telecommands

Telecommands refer to any command packets sent to the spacecraft.
For the NDS , they are used to directly control all defined actuators.

This implementation is designed to provide users with a level of control and flexibility similar to that available in the current EDEN Lab environment. Command pairs are typically specified - for example, to open and close a valve or to turn a pump on and off.

Since the hardware design (Figure 11) makes frequent use of repeated structural elements - such as multiple dosing pumps in the mixing loop - many telecommands expect additional arguments. An example of such a command is shown in Figure 13.

```
<telecommand name="NDS turn on Stock Mixing Pump" uid="NdsTurnOnStockMixingPump">
        <description />
        <serviceType>8</serviceType>
        <serviceSubtype>1</serviceSubtype>
        <parameters>
                <parameterRef uid="s8_function_id" />
                <parameterRef uid="NdsStockMixingPumpId" />
        </parameters>
        <parameterValues>
                <parameterValue uid="s8_function_id">
                        <fixed value="1" />
                </parameterValue>
        </parameterValues>
</telecommand>
<telecommand name="NDS turn off Stock Mixing Pump" uid="NdsTurnOffStockMixingPump">
        <description />
        <serviceType>8</serviceType>
        <serviceSubtype>1</serviceSubtype>
        <parameters>
                <parameterRef uid="s8_function_id" />
                <parameterRef uid="NdsStockMixingPumpId" />
        </parameters>
        <parameterValues>
                <parameterValue uid="s8_function_id">
                        <fixed value="2" />
                </parameterValue>
        </parameterValues>
</telecommand>
```

Figure 13: Definition of NdsTrunOn/OffStockMixingPump packets

6.2 Software design

This section describes the overall design of the subsystem software. Special attention is given to measures taken to satisfy the requirement of easy adaptability.
The software is implemented in **C++17**. In addition, **SCons** is used as the build system to allow integration of existing code developed for the EDEN project and to align with the department's common development practices.

The structure of the software is largely based on the existing EDEN Lab implementation.
Figure 14 shows the structure of all components that were written or modified as part of this thesis. To produce a complete, running application, additional code components are required. Since these were externally provided, they are not shown in the figure.

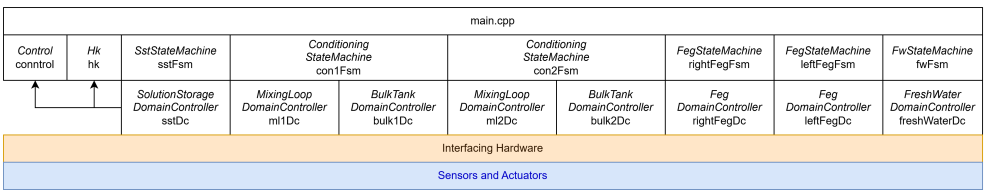


Figure 14: Software structure with underlying hardware architecture

The following subsections explain each software component and discuss the design decisions made.

6.2.1 Domain controller class

As already mentioned in Section 6.1.2, the hardware design of the NDS subsystem uses multiple instances of individual components and even duplicates entire assemblies.
This leads naturally to a further logical subdivision of the system into distinct sections (Figure 15).

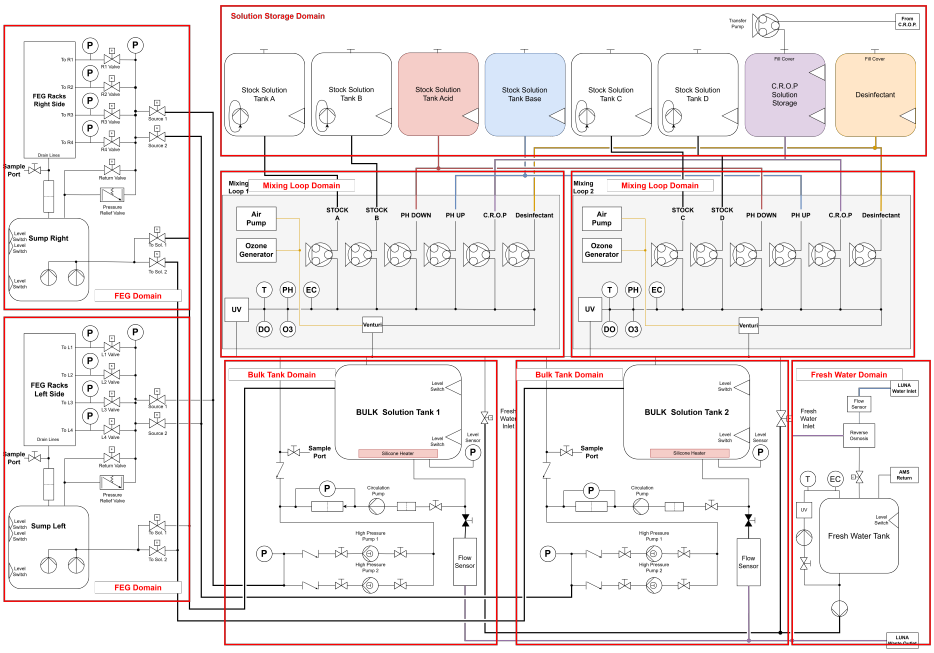


Figure 15: Definitions of Domains, modified Figure 11

These assemblies are referred to as *domains* in the following. In the figure above, the domains are marked with red boxes.
Each box corresponds to an instance of the respective domain class in the final software.

Five domain types are defined:

- Solutions Storage Domain
- Mixing Loop Domain
- Bulk Tank Domain
- FEG Domain
- Fresh Water Domain

To support easy hardware replacement and modification, an abstraction layer is introduced at this level.
For each domain, a *domain controller* is implemented. These are pure virtual classes that define

an interface to be implemented by concrete subclasses.
An example is shown in Figure 16 for the solution storage domain controller. The base class defines all required control methods and relevant data types.

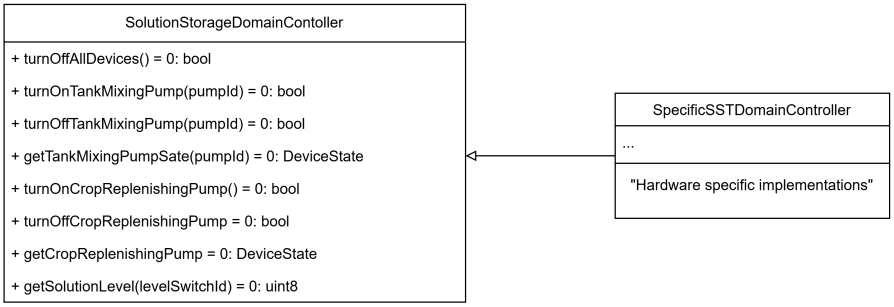


Figure 16: UML class diagram of SolutionStorageDomainController with inheritance

The child classes implement the hardware-specific functionality in addition to the inherited control methods.
This inheritance-based structure ensures a clean separation of concerns: hardware-specific behavior is encapsulated in the subclasses and is not exposed to higher-level control logic.

Because these specific domain controllers are a limited resources some kind of access control has to be implemented to avoid race conditions.

A runtime polymorphism approach was chosen due to its simplicity. Although a design based on static polymorphism was briefly considered - and would have offered potentially better execution performance[21] and alignment with DLR internal guidelines for reducing dynamic operations in flight software - it was ultimately not pursued.

The decision in favor of runtime polymorphism was based on the following considerations:

- The software is intended for terrestrial applications, so strict flight software requirements do not fully apply.
- Existing code developed for EDEN luna already uses runtime polymorphism.
- It simplifies support for new hardware configurations through straightforward inheritance.

The UML class diagrams of the other domain controllers can be found in Section A.4.

6.2.2 Housekeeping class

The housekeeping class `HK` is responsible for creating telemetry packets.
It collects current sensor values and serializes them for transmission.
These sensor values are accessed through domain controller references, which are passed initially to the constructor (see Listing 1).

```
Hk(uint8_t threadPriority,
     size_t stackSize,
     SolutionStorageDomainController& solutionStorageDc,
     MixingLoopDomainController& mixingLoop1Dc,
     MixingLoopDomainController& mixingLoop2Dc,
     BulkTankDomainController& bulkSolution1Dc,
     BulkTankDomainController& bulkSolution2Dc,
     FegDomainController& leftFegDc,
     FegDomainController& rightFegDc,
     FreshWaterDomainController& FreshWaterDc);
```

Listing 1: `Hk` constructor

Since housekeeping data must be provided at regular intervals, this class inherits from `outpost::rtos_utils::PeriodicThread`.
This thread type requires implementation of the pure virtual method `step()`, which is called at fixed intervals.
A period of 1 second was chosen, balancing the critical nature of the data and the slow response time of some sensors, such as those measuring water temperature.
The `step()` method gathers all telemetry data and sends the serialized output to the download channel.

Additionally, `Hk` implements several protected helper methods(see Listing 2) to modularize the logic within `step()`.
This structure simplifies unit testing of `step()` since the helper methods can be tested separately.

```
void
Hk::setNdsStockMixingPumpState(
    packets::nds::nds::NdsHousekeepingApplicationData& data) const
{
    data.setNdsStockMixingPumpAState(mSolutionStorageDc.getTankMixingPumpState(
            packets::nds::EnumStockMixingPumpId::mixTankA));
    data.setNdsStockMixingPumpBState(mSolutionStorageDc.getTankMixingPumpState(
            packets::nds::EnumStockMixingPumpId::mixTankB));
    data.setNdsStockMixingPumpCState(mSolutionStorageDc.getTankMixingPumpState(
            packets::nds::EnumStockMixingPumpId::mixTankC));
    data.setNdsStockMixingPumpDState(mSolutionStorageDc.getTankMixingPumpState(
            packets::nds::EnumStockMixingPumpId::mixTankD));
}
```

Listing 2: `Hk` helper function for Stock Mixing Pump States

The `Hk` class includes `telemetry.h`, which is generated by pando .
This header provides the telemetry data structure and type-safe setter functions (see Listing 2) to ensure correct serialization.

6.2.3 Control class

The control class is responsible for reacting to received telecommands.
It inherits from the pando-generated `NDSFunctionManagement` class, which defines handler methods for all supported telecommands.
pando also provides a template `.txt` file as a base for implementing the required C++ header.

Since telecommands directly trigger actuator operations, the `Control` class receives references to all domain controller instances via its constructor (see Listing 3).

```
Control(outpost::time::Clock& clock,
        SolutionStorageDomainController& SolutionStorage,
        MixingLoopDomainController& MixingLoop1,
        MixingLoopDomainController& MixingLoop2,
        FreshWaterDomainController& FreshWater,
        BulkTankDomainController& BulkTank1,
        BulkTankDomainController& BulkTank2,
        FegDomainController& FegLeft,
        FegDomainController& FegRight,
        Hk& hk);
```

Listing 3: `Control` constructor

Although large parts of the header can be derived from the template, the reaction logic must be implemented manually in the `.cpp` file.
Listing 4 shows an example. The method reads the command parameters and returns an appropriate verification status.

```
outpost::pus::FunctionVerification
Control::commandNdsTurnOnStockMixingPump(
                            outpost::pus::TelecommandIdentification,
                            NdsTurnOnStockMixingPumpPacketReader parameter)
  {
    if(mSolutionStorage.turnOnTankMixingPump(parameter.getNdsStockMixingPumpId())
            == EnumDeviceState::on)
    {
        return outpost::pus::FunctionVerification::success();
    }
    else
    {
        return outpost::pus::FunctionVerification::failure();
    }
  }
```

Listing 4: Definition of `Control::commandNdsTurnOnStockMixingPump`

For each telecommand defined in the TMTC model, one handler is generated and a corresponding response is implemented.

In addition, the `Control` class inherits from `outpost::pus::Application`.
This ensures correct handling of telemetry and telecommand packets in accordance with the PUS architecture.
This structure mirrors the design used in the current EDEN Lab implementation.

6.2.4 State machine classes

The state machine classes are responsible for the actual control flow of the subsystem.
In total four different types of state machine are implemented. See Section 6.3 for detailed informations on those types.

Regardless the type of state machine they are all based on the `outpost::rtos_utils::LoopingThread` class. The class defines a continuous looping thread similar to the aforementioned `PeriodicThread` but without a fixed period of dormancy between calling the `step()` method. Never the less, to free up system resources a `outpost::rtos::Thread::sleep(outpost::time::Milliseconds(500))` is called at the end of each state machine `step()` method. This is in the current implementation functional very similar to the behavior of `PeriodicThread` but could easily modified to facilitate a settable dormancy for the `Thread`.

Additionally, each state machine class provides methods to start or stop its thread by means of a `outpost::rtos::BinarySemaphore` (see below).

```
inline void
startFsm()
{
    mSemaphore.release();
}

inline void
stopFsm()
{
    mSemaphore.acquire();
}
```

Listing 5: Use of Semaphores for thread halting and releasing

A handy feature given the fact that the whole system may have to hibernate for prolonged time.

All classes heavily rely on the `outpost::time::Timeout` class for multiple types of time-triggered events.

Said this, all implemented state machine constructors need following arguments to be passed:

- `uint8_t threadPriority` and `size_t stackSize` for `Thread` constructor
- `outpost::time::Clock& clock` for use in `Timout` constructor

6.3 FSM Design

As mentioned in Section 6.2.4 a detailed description of each state machine type will be given in this section.

6.3.1 SstStateMachine

This state machine is responsible for the solution storage.
It is the simplest of all Final state machines (FSMs) . It only has to provide means to mix the stock solutions and replenishing the Combined Regenerative Organic food Production (C.R.O.P.) solution storage.
Since this FSM only needs access to sensors and actuators of the solution storage domain a `SolutionStorageDomainController` reference is passed in addition to the obligatory ones.

Triggered by a timer the mixing pumps of the stock tanks will be turned for a specified duration. Both timer durations are settable.

Refilling the C.R.O.P. storage is triggered by the installed level switch.

6.3.2 ConditioningStateMachine

Following the simplest FSM is the most complex. The `ConditioningStateMachine` is designed to handle everything regarding the production and monitoring of the nutrient solution. This FSM differs from the others by getting two different domain controller passed. To perform properly it requires both a `BulkTankDomainController` and `MixingLoopDomainController` reference.

To be able to receive new water, this FSM is provided with a way of requesting the `transferingAndFilling` state from the `FreshWaterStateMachine`.

6.3.3 FegStateMachine

The `FegStateMachine` class controls one of the two FEG sides.
Role of the FSM is to spray the plants regularly for a defined duration and make sure to empty the system sump.
It gets passed a `FegDomainController` reference of its corresponding side, in addition to references to both `BulkTankDomainController`.

Transitioning to and from the spraying state are triggered by timers. Pumping of the sump on the other hand only occurs when either the `top` or `max` level switch is triggered.

This class can request the `pressurizing` state from a `ConditioningStateMachine`.

6.3.4 FreshWaterStateMachine

The last type of FSM is responsible for fresh water management.
It is tasked with monitoring the already stored water and refilling the buffer tank. Simultaneous, it pretreats the water during filling.

This FSM only relies on its corresponding `FreshWaterDomainController`, which it gets passed as reference.

7 Software testing

To ensure the correct functionality of the developed software, a combination of unit testing and coverage analysis was employed.
This section briefly introduces the tools used and summarizes the testing efforts conducted during this thesis.

7.1 Unit testing

Unit testing is an established method for verifying the correctness of individual code components and is commonly used during software development.

As part of this thesis, googletest (gtest) was used to perform unit tests. gtest is an open-source testing framework for **C++**, developed by Google.
It is actively used within AVS to ensure code quality.

In addition to standard testing functionality, the framework also supports class mocking through gMock. This allows complex dependencies—such as hardware interfaces—to be replaced with simplified mock objects, enabling isolated and hardware-independent testing.

This approach was essential in this thesis, as the domain controllers were designed as pure virtual classes and no physical hardware was available during development. Mock classes were therefore used to simulate subsystem behavior.
An example of such a mock implementation can be found in Section A.5 in the appendix.

The primary goal of unit testing was to verify the correctness of control logic and state transitions under defined input conditions.
This includes verifying expected actuator behavior, proper reaction to domain-specific triggers.

7.2 Coverage testing

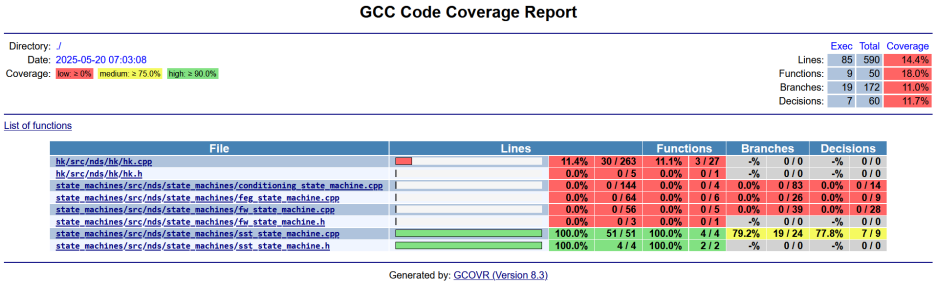
Coverage testing is another valuable technique for assessing software quality during development. In this context, it was used as a supplement to unit testing.

Coverage analysis provides insight into how much of the code base is executed during testing.
A distinction is made between line coverage, which tracks whether specific lines of code are executed, and decision coverage, which considers whether conditional branches are evaluated in both directions.

Within this thesis, coverage testing was primarily applied to evaluate the control logic implemented in the FSMs .
The tool used for this purpose was **gcovr**.

7.3 Testing summary

Testing was originally intended to focus on the custom control algorithms, with the goal of evaluating their functional correctness and robustness. However, due to time constraints, testing could only be conducted to a limited extent.



GCC Code Coverage Report

Directory: /
Date: 2025-05-20 07:03:08
Coverage: low: ≥ 0% medium: ≥ 75.0% high: ≥ 90.0%

	Exec	Total	Coverage
Lines:	85	590	14.4%
Functions:	9	50	18.0%
Branches:	19	172	11.0%
Decisions:	7	60	11.7%

List of functions

File	Lines		Functions		Branches		Decisions	
hk/src/nds/hk/hk.cpp	11.4%	30 / 263	11.1%	3 / 27	-%	0 / 0	-%	0 / 0
hk/src/nds/hk/hk.h	0.0%	0 / 5	0.0%	0 / 1	-%	0 / 0	-%	0 / 0
state_machines/src/nds/state_machines/conditioning_state_machine.cpp	0.0%	0 / 144	0.0%	0 / 4	0.0%	0 / 83	0.0%	0 / 14
state_machines/src/nds/state_machines/feg_state_machine.cpp	0.0%	0 / 64	0.0%	0 / 6	0.0%	0 / 26	0.0%	0 / 9
state_machines/src/nds/state_machines/fw_state_machine.cpp	0.0%	0 / 56	0.0%	0 / 5	0.0%	0 / 39	0.0%	0 / 28
state_machines/src/nds/state_machines/fw_state_machine.h	0.0%	0 / 3	0.0%	0 / 1	-%	0 / 0	-%	0 / 0
state_machines/src/nds/state_machines/sst_state_machine.cpp	100.0%	51 / 51	100.0%	4 / 4	79.2%	19 / 24	77.8%	7 / 9
state_machines/src/nds/state_machines/sst_state_machine.h	100.0%	4 / 4	100.0%	2 / 2	-%	0 / 0	-%	0 / 0

Generated by: GCOVR (Version 8.3)

Figure 17: Coverage result

As shown in Figure 17, only a small portion of the total code base was executed during testing. In particular, no boundary tests or negative tests were performed, such as invalid command sequences or simultaneous state requests.
As a result, potential edge-case behaviors may remain undetected.

Given the limited scope of testing, no conclusive statements can be made regarding the overall robustness or readiness of the software.
The current test setup is not sufficient for validating the system under realistic or fault-prone conditions.

Because of this limited testing no answer can be given in regards of code usability. More testing is needed.

8 Conclusion

This section summarizes the work presented in this thesis, evaluates the current development status, and outlines the improvements required for future implementation and use.

8.1 Evaluation

The goal of this thesis was to design and implement a modular and adaptable control software system for the EDEN LUNA NDS .
While significant progress was made in terms of architectural structure and conceptual design, the implementation cannot be considered deployable in its current form.
The following evaluation is based on the objectives outlined in Section 1.1.

Control software

The implemented classes cover the essential functionalities required for nutrient delivery and system actuation. However, due to the very limited extent of functional testing, no reliable conclusions can be drawn regarding the correctness or robustness of the system. The developed code therefore represents a sound concept, but not a flight- or field-ready implementation.

Modularity

The software architecture was successfully designed with modularity in mind.
The use of pure virtual domain controller classes and the ability to mock them during unit testing demonstrates a clear separation of concerns and high replaceability of hardware interfaces.
This flexibility will be essential for adapting the software to evolving hardware configurations.

Hardware concept

A basic hardware interface concept was established, including the selection of *brainboxes* remote I/O modules and the creation of a preliminary signal map (Section A.3). Domain controller classes specifically adapted to the hardware were not created, which would have been a valuable resource for future system integration.

Industrial solutions

Commercial fertigation systems were researched and compared.
The thesis provides an overview of operating principles and product specifications, and clearly justifies the decision to develop a customized NDS solution. While this section remains high-level, it effectively supports the in-house development decision.

Use of existing frameworks (OUTPOST / pando)

The decision to base the software on OUTPOST and pando allowed for close integration with existing software.
Standardized TMTC generation and scheduling features were successfully used.

8.2 Improvements

To develop the current concept into a functional, mission-capable control software, the following key improvements are required:

1. Comprehensive unit testing and increasing code coverage
2. Improve control algorithms for nutrient solution conditioning
3. Amend TMTC model. Multiple settable variables are currently not reachable with telecommands
4. Error handling and diagnostics

8.3 Final remark

Despite its limitations, this thesis provides a structured and expandable foundation for a modular control software architecture.
With targeted improvements, the presented concept can evolve into a solution for use in EDEN LUNA and related terrestrial or space-based applications.

In summary, this thesis contributes a well-structured and clearly reasoned software concept for the control of a nutrient delivery subsystem in a highly constrained and modular environment.
While it does not yet meet the requirements of a deployable control system, the architecture, documentation, and implementation choices provide a foundation for further development.

References

[1] O. Romberg, “MEPA.” Accessed: Apr. 29, 2025. [Online]. Available: https://www.dlr.de/de/irs/forschung-und-transfer/projekte/mepa

[2] F. Dambowsky, D. Schubert, J. M. Bunchek, and M. Witte, “EDEN ISS greenhouse returns to Bremen and has a new destination.” Accessed: May 16, 2025. [Online]. Available: https://www.dlr.de/en/latest/news/2023/03/eden-iss-greenhouse-returns-to-bremen-and-has-a-new-destination

[3] O. Romberg, “Labor für Planetare Infrastrukturen.” Accessed: May 19, 2025. [Online]. Available: https://www.dlr.de/de/irs/forschung-und-transfer/forschungsinfrastruktur/planetare-infrastrukturen-labor

[4] “Multi-Feed-RM-Brochure.” Accessed: May 17, 2025. [Online]. Available: https://arguscontrols.com/uploads/documents/Multi-Feed-RM-Brochure.pdf

[5] Priva, “Priva Nutri-Line | Perfect fertigation for healthy crops.” Accessed: May 17, 2025. [Online]. Available: https://www.priva.com/horticulture/solutions/greenhouse-water-systems/priva-nutri-line

[6] “EvoMix.” Accessed: May 18, 2025. [Online]. Available: https://www.spagnol.com/en-us/products/mixing-tank-fertigation-units/evomix

[7] N. Aksteiner, “EDEN_LUNA_NDS_Block_Reduced.”

[8] M. Gaskill, “3D Printing: Saving Weight and Space at Launch - NASA.” Accessed: Apr. 29, 2025. [Online]. Available: https://www.nasa.gov/missions/station/iss-research/3d-printing-saving-weight-and-space-at-launch/

[9] “Growing Plants in Space - NASA.” Accessed: Apr. 29, 2025. [Online]. Available: https://www.nasa.gov/exploration-research-and-technology/growing-plants-in-space/

[10] D. Schubert, “EDEN Initiative.” Accessed: Apr. 28, 2025. [Online]. Available: https://www.dlr.de/de/irs/ueber-uns/abteilungen/systemanalyse-raumsegment/planetare-infrastukturen/eden-initiative

[11] Space Station Research Integration Office, “Station Science 101: Plant Research - NASA.” Accessed: Apr. 29, 2025. [Online]. Available: https://www.nasa.gov/missions/station/ways-the-international-space-station-helps-us-study-plant-growth-in-space/

[12] D. Schubert, P. Zabel, J. Hauslage, and G. Bornemann, “Bio-regenerative Life Support Systems at DLR.” Accessed: Apr. 29, 2025. [Online]. Available: https://www.dlr.de/en/irs/downloads/images20/eden_initiative/DLR_BLSS_Roadmap_English_final_v1.0.pdf/@@download/file

[13] F. Dambowsky and D. Schubert, “EDEN ISS – DLR greenhouse in Antarctica.” Accessed: May 16, 2025. [Online]. Available: https://www.dlr.de/en/research-and-transfer/projects-and-missions/eden-iss

[14] F. Dambowsky and D. Schubert, “The EDEN ISS greenhouse.” Accessed: May 16, 2025. [Online]. Available: https://www.dlr.de/en/research-and-transfer/projects-and-missions/eden-iss/the-eden-iss-greenhouse

[15] “AUTOMATED FERTIGATION - Choosing the right system.” Accessed: May 04, 2025. [Online]. Available: https://arguscontrols.com/automated-fertigation-choosing-the-right-system

[16] “Priva Nutriflex: Smart, flexible and precise fertilization.” Accessed: May 18, 2025. [Online]. Available: https://www.priva.com/horticulture/solutions/greenhouse-water-systems/priva-nutri-line/nutriflex

[17] “Priva Nutrijet: Efficient dosing via fertilizer injection.” Accessed: May 18, 2025. [Online]. Available: https://www.priva.com/horticulture/solutions/greenhouse-water-systems/priva-nutri-line/nutrijet

[18] F. Dannemann, “Abteilung Avioniksysteme.” Accessed: May 05, 2025. [Online]. Available: https://www.dlr.de/de/irs/ueber-uns/abteilungen/avioniksysteme

[19] F. Dannemann, “Software Entwicklung.” Accessed: May 05, 2025. [Online]. Available: https://www.dlr.de/de/irs/ueber-uns/abteilungen/avioniksysteme/software-entwicklung

[20] “DLR-RY/outpost-core: OUTPOST - Open modUlar sofTware PlatfOrm for SpacecrafT.” Accessed: May 05, 2025. [Online]. Available: https://github.com/DLR-RY/outpost-core

[21] “Difference Between Compile Time And Run Time Polymorphism In C++.” Accessed: May 10, 2025. [Online]. Available: https://www.geeksforgeeks.org/compile-time-vs-run-time-polymorphism-difference-in-cpp/

A Appendix

A.1 Glossary

Aeroponics: Method of cultivating plants without the use of soil. The plant roots are sprayed regularly with a nutrient solution instead.

Fertigation: Fertigation is the process of applying mineral fertilizers to crops along with the irrigation water.

A.2 Sensor Actuator List

Domain	Sensor/Actuator	Name	ID
Solution Storage	S	Stock Solution Tank A Level Switch	SST-A-LEVEL
Solution Storage	S	Stock Solution Tank B Level Switch	SST-B-LEVEL
Solution Storage	S	Stock Solution Tank Acid Level Switch	SST-ACID-LEVEL
Solution Storage	S	Stock Solution Tank Base Level Switch	SST-BASE-LEVEL
Solution Storage	S	Stock Solution Tank C Level Switch	SST-C-LEVEL
Solution Storage	S	Stock Solution Tank D Level Switch	SST-D-LEVEL
Solution Storage	S	C.R.O.P Solution Tank Level LOW Switch	SST-CROP-LOW-LEVEL
Solution Storage	S	C.R.O.P Solution Tank Level HIGH Switch	SST-CROP-HIGH-LEVEL
Solution Storage	S	Disinfectant Tank Level Switch	SST-DESIN-LEVEL
Solution Storage	A	Stock Solution Tank A Mixing Pump	SST-A-MIX
Solution Storage	A	Stock Solution Tank B Mixing Pump	SST-B-MIX
Solution Storage	A	Stock Solution Tank C Mixing Pump	SST-C-MIX
Solution Storage	A	Stock Solution Tank D Mixing Pump	SST-D-MIX
Mixing Loop 1	S	Mixing Loop 1 Temperatur Probe	ML-1-TEMP
Mixing Loop 1	S	Mixing Loop 1 pH-Probe	ML-1-PH
Mixing Loop 1	S	Mixing Loop 1 Electric Conductivity Probe	ML-1-EC
Mixing Loop 1	S	Mixing Loop 1 Desolved Oxygen Probe	ML-1-DO
Mixing Loop 1	S	Mixing Loop 1 Ozone Probe	ML-1-O3
Mixing Loop 1	A	Mixing Loop 1 Air Pump	ML-1-AP
Mixing Loop 1	A	Mixing Loop 1 Ozone Generator	ML-1-OG
Mixing Loop 1	A	Mixing Loop 1 UV Lamp	ML-1-UV
Mixing Loop 1	A	Mixing Loop 1 Stock A Pump	ML-1-PUMP-STA
Mixing Loop 1	A	Mixing Loop 1 Stock B Pump	ML-1-PUMP-STB
Mixing Loop 1	A	Mixing Loop 1 PH Up Pump	ML-1-PUMP-PH_UP
Mixing Loop 1	A	Mixing Loop 1 PH Down Pump	ML-1-PUMP-PH_DOWN
Mixing Loop 1	A	Mixing Loop 1 CROP Pump	ML-1-PUMP-CROP
Mixing Loop 1	A	Mixing Loop 1 Desinfectant Pump	ML-1-PUMP-DESIN
Mixing Loop 2	S	Mixing Loop 2 Temperatur Probe	ML-2-TEMP
Mixing Loop 2	S	Mixing Loop 2 pH-Probe	ML-2-PH
Mixing Loop 2	S	Mixing Loop 2 Electric Conductivity Probe	ML-2-EC
Mixing Loop 2	S	Mixing Loop 2 Desolved Oxygen Probe	ML-2-DO
Mixing Loop 2	S	Mixing Loop 2 Ozone Probe	ML-2-O3
Mixing Loop 2	A	Mixing Loop 2 Air Pump	ML-2-AP
Mixing Loop 2	A	Mixing Loop 2 Ozone Generator	ML-2-OG
Mixing Loop 2	A	Mixing Loop 2 UV Lamp	ML-2-UV
Mixing Loop 2	A	Mixing Loop 2 Stock A Pump	ML-2-PUMP-STA
Mixing Loop 2	A	Mixing Loop 2 Stock B Pump	ML-2-PUMP-STB
Mixing Loop 2	A	Mixing Loop 2 PH Up Pump	ML-2-PUMP-PH_UP
Mixing Loop 2	A	Mixing Loop 2 PH Down Pump	ML-2-PUMP-PH_DOWN
Mixing Loop 2	A	Mixing Loop 2 CROP Pump	ML-2-PUMP-CROP
Mixing Loop 2	A	Mixing Loop 2 Desinfectant Pump	ML-2-PUMP-DESIN
BULK Solution Tank 1	S	Bulk Solution Tank 1 Level Switch High	BST-1-LS-HIGH
BULK Solution Tank 1	S	Bulk Solution Tank 1 Level Switch Low	BST-1-LS-LOW

Figure 18: Sensor/Actuator List

Domain	Sensor/Actuator	Name	ID
BULK Solution Tank 1	S	Bulk Solution Tank 1 Level Sensor	BST-1-LEVEL
BULK Solution Tank 1	S	Bulk Solution Tank 1 Feeder Pressure Sensor	BST-1-P-FEED
BULK Solution Tank 1	S	Bulk Solution Tank 1 High Pressure Sensor	BST-1-P-HIGH_PRES
BULK Solution Tank 1	S	Bulk Solution Tank 1 Waste Water Flow Sensor	BST-1-FS-WASTE
BULK Solution Tank 1	A	Bulk Solution Tank 1 Feeder Pump	BST-1-PUMP-FEED
BULK Solution Tank 1	A	Bulk Solution Tank 1 High Pressure Pump Nominal	BST-1-PUMP-HIGH_PRES-NOM
BULK Solution Tank 1	A	Bulk Solution Tank 1 High Pressure Pump Redundant	BST-1-PUMP-HIGH_PRES-RED
BULK Solution Tank 1	A	Bulk Solution Tank 1 Heater	BST-1-HEATER
BULK Solution Tank 1	A	Bulk Solution Tank 1 Fresh Water Inlet	BST-1-VALVE-FRESH
BULK Solution Tank 2	S	Bulk Solution Tank 2 Level Switch High	BST-2-LS-HIGH
BULK Solution Tank 2	S	Bulk Solution Tank 2 Level Switch Low	BST-2-LS-LOW
BULK Solution Tank 2	S	Bulk Solution Tank 2 Level Sensor	BST-2-LEVEL
BULK Solution Tank 2	S	Bulk Solution Tank 2 Feeder Pressure Sensor	BST-2-P-FEED
BULK Solution Tank 2	S	Bulk Solution Tank 2 High Pressure Sensor	BST-2-P-HIGH_PRES
BULK Solution Tank 2	S	Bulk Solution Tank 2 Waste Water Flow Sensor	BST-2-FS-WASTE
BULK Solution Tank 2	A	Bulk Solution Tank 2 Feeder Pump	BST-2-PUMP-FEED
BULK Solution Tank 2	A	Bulk Solution Tank 2 High Pressure Pump Nominal	BST-2-PUMP-HIGH_PRES-NOM
BULK Solution Tank 2	A	Bulk Solution Tank 2 High Pressure Pump Redundant	BST-2-PUMP-HIGH_PRES-RED
BULK Solution Tank 2	A	Bulk Solution Tank 2 HEATER	BST-2-HEATER
BULK Solution Tank 2	A	Bulk Solution Tank 2 Fresh Water Inlet	BST-2-VALVE-FRESH
FEG Right	S	FEG Racks Right R1 Pressure Sensor	FEG-R-P-R1
FEG Right	S	FEG Racks Right R2 Pressure Sensor	FEG-R-P-R2
FEG Right	S	FEG Racks Right R3 Pressure Sensor	FEG-R-P-R3
FEG Right	S	FEG Racks Right R4 Pressure Sensor	FEG-R-P-R4
FEG Right	S	FEG Racks Right Manifold Pressure Sensor	FEG-R-P-MANF
FEG Right	S	Sump Right Level Switch HIGH NOM	FEG-R-LS-HN
FEG Right	S	Sump Right Level Switch HIGH RED	FEG-R-LS-HR

Figure 19: Figure 18 (continued)

Domain	Sensor/Actuator	Name	ID
FEG Right	S	Sump Right Level Switch LOW	FEG-R-LS-L
FEG Right	A	FEG Racks Right R1 Valve	FEG-R-VALVE-R1
FEG Right	A	FEG Racks Right R2 Valve	FEG-R-VALVE-R2
FEG Right	A	FEG Racks Right R3 Valve	FEG-R-VALVE-R3
FEG Right	A	FEG Racks Right R4 Valve	FEG-R-VALVE-R4
FEG Right	A	FEG Racks Right Return Valve	FEG-R-VALVE-Return
FEG Right	A	FEG Racks Right Source 1 Valve	FEG-R-VALVE-Source_1
FEG Right	A	FEG Racks Right Source 2 Valve	FEG-R-VALVE-Source_2
FEG Right	A	Sump Right to Tank 1 Valve	FEG-R-VALVE-To_Tank_1
FEG Right	A	Sump Right to Tank 2 Valve	FEG-R-VALVE-To_Tank_2
FEG Right	A	Sump Right Return Pump Nominal	FEG-R-PUMP-RETURN-NOM
FEG Right	A	Sump Right Return Pump Redundant	FEG-R-PUMP-RETURN-RED
FEG Left	S	FEG Racks Left R1 Pressure Sensor	FEG-L-P-R1
FEG Left	S	FEG Racks Left R2 Pressure Sensor	FEG-L-P-R2
FEG Left	S	FEG Racks Left R3 Pressure Sensor	FEG-L-P-R3
FEG Left	S	FEG Racks Left R4 Pressure Sensor	FEG-L-P-R4
FEG Left	S	FEG Racks Left Manifold Pressure Sensor	FEG-L-P-MANF
FEG Left	S	Sump Left Level Switch HIGH NOM	FEG-L-LS-HN
FEG Left	S	Sump Left Level Switch HIGH RED	FEG-R-LS-HRL
FEG Left	S	Sump Left Level Switch LOW	FEG-L-LS-L
FEG Left	A	FEG Racks Left R1 Valve	FEG-L-VALVE-R1
FEG Left	A	FEG Racks Left R2 Valve	FEG-L-VALVE-R2
FEG Left	A	FEG Racks Left R3 Valve	FEG-L-VALVE-R3
FEG Left	A	FEG Racks Left R4 Valve	FEG-L-VALVE-R4
FEG Left	A	FEG Racks Left Return Valve	FEG-L-VALVE-Return
FEG Left	A	FEG Racks Left Source 1 Valve	FEG-L-VALVE-Source_1
FEG Left	A	FEG Racks Left Source 2 Valve	FEG-L-VALVE-Source_2
FEG Left	A	Sump Left to Tank 1 Valve	FEG-L-VALVE-To_Tank_1
FEG Left	A	Sump Left to Tank 2 Valve	FEG-L-VALVE-To_Tank_2
FEG Left	A	Sump Left Return Pump Nominal	FEG-L-PUMP-RETURN-NOM
FEG Left	A	Sump Left Return Pump Redundant	FEG-L-PUMP-RETURN-RED
SES Fresh Water	S	Fresh Water Inlet Flow Sensor	FW-INLET-FLOW
SES Fresh Water	S	Fresh Water Temperature Sensor	FW-TEMP
SES Fresh Water	S	Fresh Water EC Sensor	FW-EC
SES Fresh Water	S	Fresh Water Level Sensor	FW-LEVEL-SENSOR
SES Fresh Water	A	Fresh Water Circulation Pump	FW-CIRC
SES Fresh Water	A	Fresh Water Distribution Pump	FW-DIST
SES Fresh Water	A	Fresh Water Fill Valve	FW-FILL
SES Fresh Water	A	Fresh Water RO Control	FW-RO

Figure 20: Figure 19 (continued)

A.3 Signal brainbox map

ID	TM data type	Interface	Bridge
SST-A-LEVEL	uint8	IN - Digital	ED-549-SST
SST-B-LEVEL	uint8	IN - Digital	ED-549-SST
SST-ACID-LEVEL	uint8	IN - Digital	ED-549-SST
SST-BASE-LEVEL	uint8	IN - Digital	ED-549-SST
SST-C-LEVEL	uint8	IN - Digital	ED-549-SST
SST-D-LEVEL	uint8	IN - Digital	ED-549-SST
SST-CROP-LOW-LEVEL	uint8	IN - Digital	ED-549-SST
SST-CROP-HIGH-LEVEL	uint8	IN - Digital	ED-549-SST
SST-DESIN-LEVEL	uint8	IN - Digital	ED-549-SST
SST-A-MIX	EnumDeviceState	OUT - Relay	ED-538-SST
SST-B-MIX	EnumDeviceState	OUT - Relay	ED-538-SST
SST-C-MIX	EnumDeviceState	OUT - Relay	ED-538-SST
SST-D-MIX	EnumDeviceState	OUT - Relay	ED-538-SST
ML-1-TEMP	uint16	IN - Analog	ED-549-ML-1
ML-1-PH	uint16	IN - Analog	ED-549-ML-1
ML-1-EC	uint16	IN - Analog	ED-549-ML-1
ML-1-DO	uint16	IN - Analog	ED-549-ML-1
ML-1-O3	uint16	IN - Analog	ED-549-ML-1
ML-1-AP	EnumDeviceState	OUT - Relay	ED-538-ML-1
ML-1-OG	EnumDeviceState	OUT - Relay	ED-538-ML-1
ML-1-UV	EnumDeviceState	OUT - Relay	ED-538-ML-1
ML-1-PUMP-STA	EnumDeviceState	OUT - Digital	ED-527-ML-1
ML-1-PUMP-STB	EnumDeviceState	OUT - Digital	ED-527-ML-1
ML-1-PUMP-PH_UP	EnumDeviceState	OUT - Digital	ED-527-ML-1
ML-1-PUMP-PH_DOWN	EnumDeviceState	OUT - Digital	ED-527-ML-1
ML-1-PUMP-CROP	EnumDeviceState	OUT - Digital	ED-527-ML-1
ML-1-PUMP-DESIN	EnumDeviceState	OUT - Digital	ED-527-ML-1
ML-2-TEMP	uint16	IN - Analog	ED-549-ML-2
ML-2-PH	uint16	IN - Analog	ED-549-ML-2
ML-2-EC	uint16	IN - Analog	ED-549-ML-2
ML-2-DO	uint16	IN - Analog	ED-549-ML-2
ML-2-O3	uint16	IN - Analog	ED-549-ML-2
ML-2-AP	EnumDeviceState	OUT - Relay	ED-538-ML-2
ML-2-OG	EnumDeviceState	OUT - Relay	ED-538-ML-2
ML-2-UV	EnumDeviceState	OUT - Relay	ED-538-ML-2
ML-2-PUMP-STA	EnumDeviceState	OUT - Digital	ED-527-ML-2
ML-2-PUMP-STB	EnumDeviceState	OUT - Digital	ED-527-ML-2
ML-2-PUMP-PH_UP	EnumDeviceState	OUT - Digital	ED-527-ML-2
ML-2-PUMP-PH_DOWN	EnumDeviceState	OUT - Digital	ED-527-ML-2
ML-2-PUMP-CROP	EnumDeviceState	OUT - Digital	ED-527-ML-2
ML-2-PUMP-DESIN	EnumDeviceState	OUT - Digital	ED-527-ML-2
BST-1-LS-HIGH	uint8	IN - Digital	ED-588-BST-1
BST-1-LS-LOW	uint8	IN - Digital	ED-588-BST-1

Figure 21: brainbox Signal List

ID	TM data type	Interface	Bridge
BST-1-LEVEL	uint16	IN - Analog	ED-549-BST-1
BST-1-P-FEED	uint16	IN - Analog	ED-549-BST-1
BST-1-P-HIGH_PRES	uint16	IN - Analog	ED-549-BST-1
BST-1-FS-WASTE	uint16	IN - Analog	ED-549-BST-1
BST-1-PUMP-FEED	EnumDeviceState	OUT - Relay	ED-538-BST-1
BST-1-PUMP-HIGH_PRES-NOM	EnumDeviceState	OUT - Relay	ED-538-BST-1
BST-1-PUMP-HIGH_PRES-RED	EnumDeviceState	OUT - Relay	ED-538-BST-1
BST-1-HEATER	EnumDeviceState	OUT - Relay	ED-538-BST-1
BST-1-VALVE-FRESH	EnumValveState	OUT - Digital	ED-588-BST-1
BST-2-LS-HIGH	uint8	IN - Digital	ED-588-BST-2
BST-2-LS-LOW	uint8	IN - Digital	ED-588-BST-2
BST-2-LEVEL	uint16	IN - Analog	ED-549-BST-2
BST-2-P-FEED	uint16	IN - Analog	ED-549-BST-2
BST-2-P-HIGH_PRES	uint16	IN - Analog	ED-549-BST-2
BST-2-FS-WASTE	uint16	IN - Analog	ED-549-BST-2
BST-2-PUMP-FEED	EnumDeviceState	OUT - Relay	ED-538-BST-2
BST-2-PUMP-HIGH_PRES-NOM	EnumDeviceState	OUT - Relay	ED-538-BST-2
BST-2-PUMP-HIGH_PRES-RED	EnumDeviceState	OUT - Relay	ED-538-BST-2
BST-2-HEATER	EnumDeviceState	OUT - Relay	ED-538-BST-2
BST-2-VALVE-FRESH	EnumValveState	OUT - Digital	ED-588-BST-2
FEG-R-P-R1	uint16	IN - Analog	ED-549-FEG-R
FEG-R-P-R2	uint16	IN - Analog	ED-549-FEG-R
FEG-R-P-R3	uint16	IN - Analog	ED-549-FEG-R
FEG-R-P-R4	uint16	IN - Analog	ED-549-FEG-R
FEG-R-P-MANF	uint16	IN - Analog	ED-549-FEG-R
FEG-R-LS-HN	uint8	IN - Digital	ED-538-FEG-R
FEG-R-LS-HR	uint8	IN - Digital	ED-538-FEG-R

Figure 22: Figure 21 (continued)

ID	TM data type	Interface	Bridge
FEG-R-LS-L	uint8	IN - Digital	ED-538-FEG-R
FEG-R-VALVE-R1	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-R2	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-R3	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-R4	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-Return	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-Source_1	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-Source_2	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-To_Tank_1	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-VALVE-To_Tank_2	EnumValveState	OUT - Digital	ED-527-FEG-R
FEG-R-PUMP-RETURN-NOM	EnumDeviceState	OUT - Relay	ED-538-FEG-R
FEG-R-PUMP-RETURN-RED	EnumDeviceState	OUT - Relay	ED-538-FEG-R
FEG-L-P-L1	uint16	IN - Analog	ED-549-FEG-L
FEG-L-P-L2	uint16	IN - Analog	ED-549-FEG-L
FEG-L-P-L3	uint16	IN - Analog	ED-549-FEG-L
FEG-L-P-L4	uint16	IN - Analog	ED-549-FEG-L
FEG-L-P-MANF	uint16	IN - Analog	ED-549-FEG-L
FEG-L-LS-HN	uint8	IN - Digital	ED-538-FEG-L
FEG-R-LS-HRL	uint8	IN - Digital	ED-538-FEG-L
FEG-L-LS-L	uint8	IN - Digital	ED-538-FEG-L
FEG-L-VALVE-L1	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-L2	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-L3	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-L4	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-Return	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-Source_1	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-Source_2	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-To_Tank_1	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-VALVE-To_Tank_2	EnumValveState	OUT - Digital	ED-527-FEG-L
FEG-L-PUMP-RETURN-NOM	EnumDeviceState	OUT - Relay	ED-538-FEG-L
FEG-L-PUMP-RETURN-RED	EnumDeviceState	OUT - Relay	ED-538-FEG-L
FW-INLET-FLOW	unit16	IN - Analog	ED-549-FW
FW-TEMP	uint16	IN - Analog	ED-549-FW
FW-EC	uint16	IN - Analog	ED-549-FW
FW-LEVEL-SENSOR	uint8	IN - Digital	ED-538-FW
FW-CIRC	EnumDeviceState	OUT - Relay	ED-538-FW
FW-DIST	EnumDeviceState	OUT - Relay	ED-538-FW
FW-FILL	EnumValveState	OUT - Digital	ED-538-FW
FW-RO	EnumDeviceState	OUT - Digital	ED-538-FW

Figure 23: Figure 22 (continued)

A.4 Domain Controller class diagrams

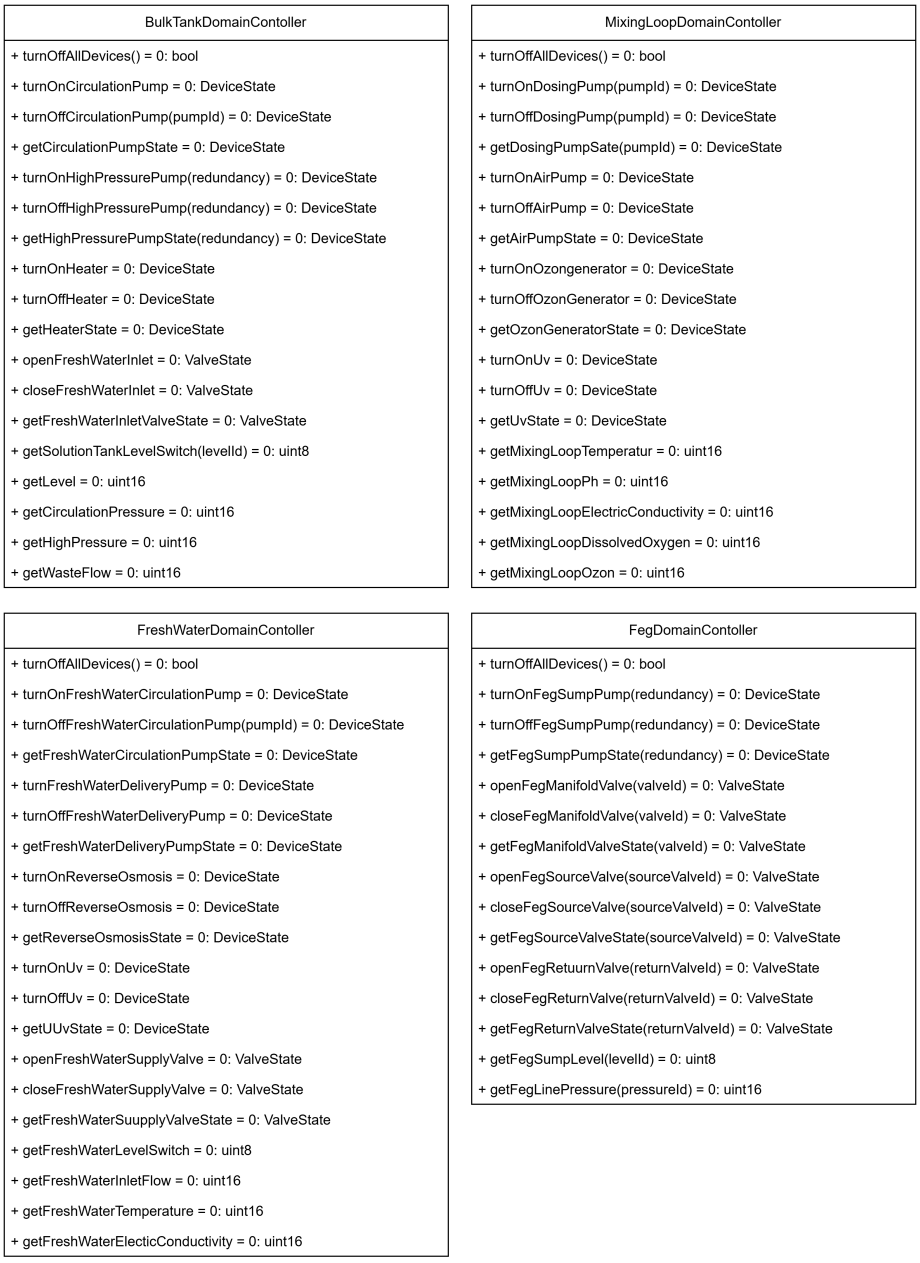


Figure 24: Remaining domain controller class diagrams

A.5 Mock class example

```
class MockSolutionsStorageDc : public eden::nds::SolutionStorageDomainController
{
private:
    /* data */
public:
    // MockSolutionsStorageDc(/* args */);
    // ~MockSolutionsStorageDc();

    MOCK_METHOD(bool, turnOffAllDevices, (), (override));

    MOCK_METHOD(packets::nds::EnumDeviceState,
                turnOnTankMixingPump,
                (packets::nds::EnumStockMixingPumpId pumpId),
                (override));
    MOCK_METHOD(packets::nds::EnumDeviceState,
                turnOffTankMixingPump,
                (packets::nds::EnumStockMixingPumpId pumpId),
                (override));
    MOCK_METHOD(packets::nds::EnumDeviceState,
                getTankMixingPumpState,
                (packets::nds::EnumStockMixingPumpId pumpId),
                (override));

    MOCK_METHOD(packets::nds::EnumDeviceState, turnOnCropReplenishingPump, (),
                  (override));
    MOCK_METHOD(packets::nds::EnumDeviceState, turnOffCropReplenishingPump, (),
                  (override));
    MOCK_METHOD(packets::nds::EnumDeviceState, getCropReplenishingPumpStatus, (),
                  (override));

    MOCK_METHOD(uint8_t, getSolutionLevel, (LevelSwitchID levelSwitchId), (override));
};
```

Listing 6: Mock of SolutionStorageDomainControll

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der ist erfolgt durch:

_____	_____	_____
Ort	Datum	Unterschrift im Original