DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

RaHM-Lab
Positionsregelung Drohne

Projekt Software Engineering 2

für die Prüfung zum
Bachelor of Science

im Studiengang Informatik
an der DHBW Karlsruhe

von
Michael Maag
Theresa Uhlmann
17.05.2022

Gutachter der DHBW Karlsruhe Daniel Lindner

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

USB	Universal Serial Bus

Tabellenverzeichnis

Code-Verzeichnis

1 Einleitung

Diese Ausarbeitung wird als Prüfungsleistung für das Modul *Software Engineering 2* angefertigt. Die Inhalte basieren maßgeblich auf der zugrundeliegenden Vorlesung. Die Prüfungsleistung wird auf entstandenem Code einer Studienarbeit ausgeführt.

BESCHREIBUNG! *noch auf die Gliederung eingehen*

2 Problemstellung

Diese Ausarbeitung baut auf dem Code einer Studienarbeit auf.

Die Problemstellung der zugrundeliegenden Studienarbeit lautet wie folgt: „

BESCHREIBUNG! *Text hier einfügen!*

“[]

BESCHREIBUNG! *Darf das hier überhaupt so? Immerhin wird SWE vor der Studienarbeit abgegeben... Also brauchen wir eine Dependency Inversion für das Zitat :D*

3 Architektur

3.1 Dependency Rule

Wie in Abbildung 1 (Seite 3) zu sehen, zeigen alle Abhängigkeiten in die gleiche oder eine darunterliegende Schicht. Aufrufe werden mittels *virtuellen* Klassen und einer entsprechenden *Dependency Inversion* in weiter außen liegende Klassen ermöglicht.

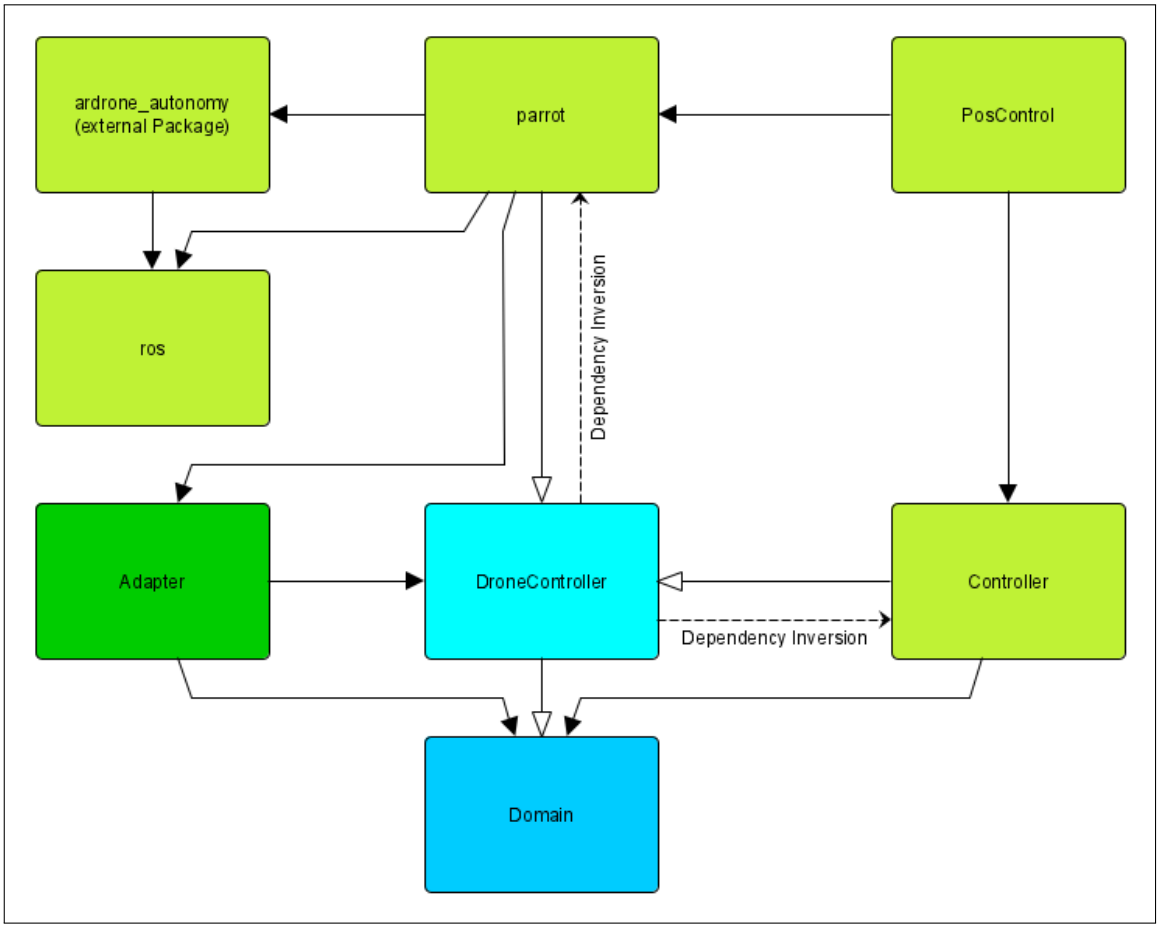

Abbildung 1: Architektur-Pakete des Projektes

Abbildung 1 zeigt das Konzept der Architektur. Die Bedeutungen der Pfeile orientieren sich an UML-Klassendiagrammen. Die Farbgebung unterscheidet die verschiedenen Schichten der *Clean Archirecture*: *Domain-Layer* (blau), *Application-Layer* (hellblau), *Adapter-Layer* (grün), *PlugIn-Layer* (hellgrün).

Ein detallierteres Verständnis der Interaktion der Klassen soll durch Abbildung 2 (Seite 4) vermittelt werden.

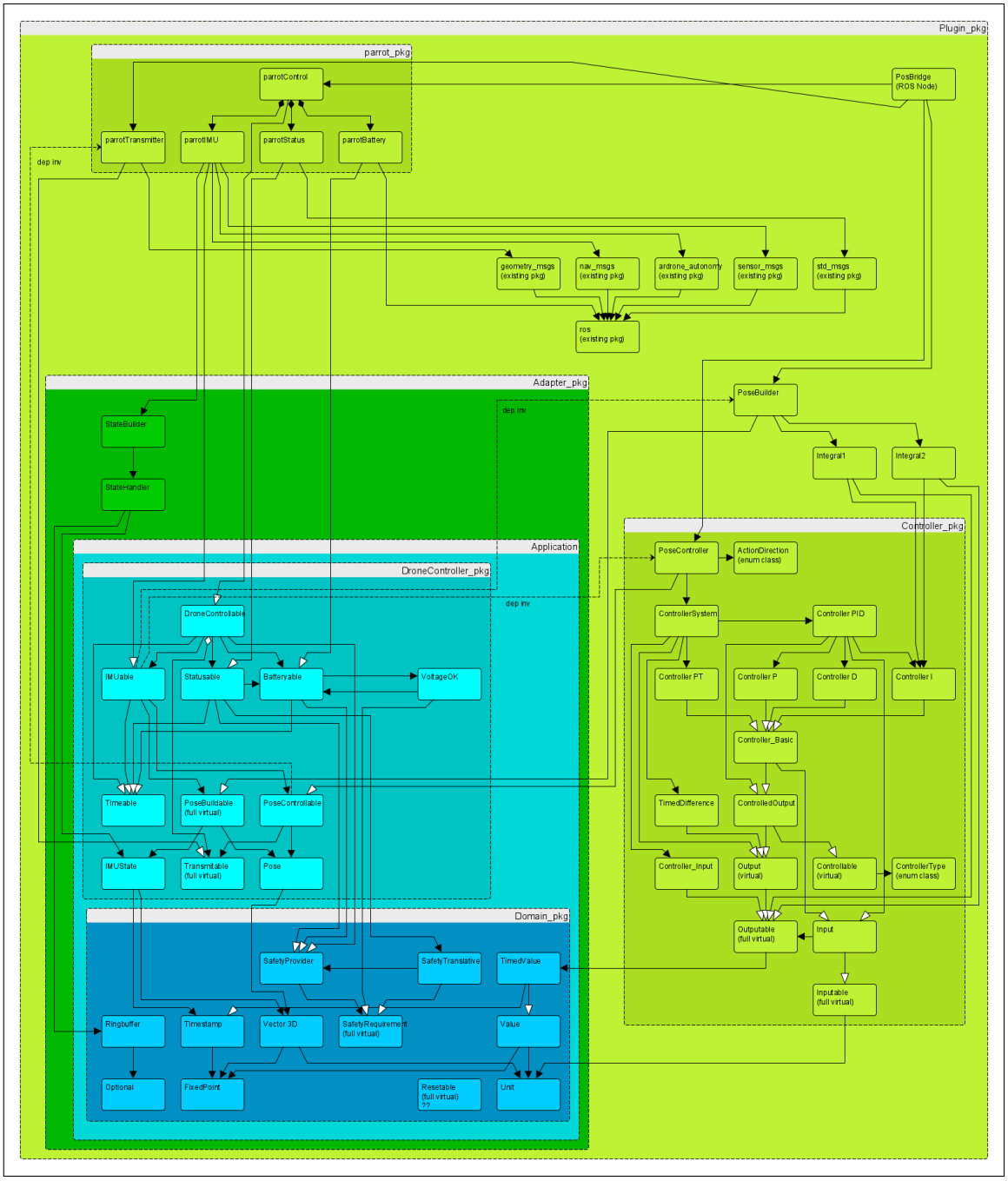

Abbildung 2: Architektur des Projektes

3.2 Domain Layer

Um ein Anlegen des *Abstraction Layers* aus Angeberei zu vermeiden[1], wurden die beiden Schichten *Abstraction Layer* und *Domain Layer* vereinigt. Tatsächlich könnten folgende Klassen in einem *Abstraction Layer* untergebracht werden:

- FixedPoint

[1]Siehe Seite 23 in den Vorlesungsfolien zu *Clean Architecture*

- Unit
- Value
- Vector3D
- Timestamp
- TimedValue
- Optional

Da die Klasse `SafetyRequirement`, Klasse `SafetyProvider` und Klasse `SafetyTranslative` den Use Case ‚`überwacht einen sicheren Start beziehungsweise Flug`‘ abbilden, sind diese Klassen entgegen der Aufteilung innerhalb des betrachteten Projekts dem *Application Layer* zuzuordnen.

3.3 Application Layer

Der *Application Layer* bildet eine *Bridge* ab. Hierdurch kann die Hardware im zukünfigten Projektverlauf ausgetauscht werden, ohne den Kern der Anwendung zu beeinflussen. Darüber hinaus bietet das Paket *`DroneController`* die grundlegenden Funktionalitäten für die Positionsregelung einer Drohne an. Umfangreichere Funktionen, wie zum Beispiel die Kommunikation mit der Hardware, werden in dafür vorgesehenden Paket des *PlugIn Layers* implementiert.

Die Klasse `IMUState` und die Klasse `Pose` bilden Daten ab. Somit sind diese entgegen der Darstellung in Abbildung 2 (Seite 4) dem *Domain Layer* zugehörig.

3.4 Adapter Layer

Die Klasse `StateBuilder` ermöglicht eine Transformation der Informationen der *IMU-Message* in die dafür vorgesehene Klasse `IMUState`. Die entstehende Instanz wird anschließend zur weiteren Verarbeitung in die Implementierung der Klasse Klasse `PoseBuildable` übergeben.

3.5 PlugIn Layer

Das Framework *ROS* wurde gezielt in den *PlugIn Layer* aufgenommen, um etwaigen Änderunges dieses Frameworks mit mäßigem Aufwand begegnen zu können. *Anmerkung:* „The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot

applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source."[2]

Des Weiteren wird die vollständige Implementierung der einzusetztendes Drohne, sowie des Controllers im *PlugIn Layer* geführt. Somit ist ein einfacher Austausch möglich.

4 Entwurfsmuster

BESCHREIBUNG! *Blabla für Einleitung des Kapitels*

Bei der Beschreibung der einzelnen Entwurfsmuster orientiert sich diese Arbeit an der Beschreibungsstruktur der GoF (Gang of Four).

4.0.1 Fassade

Die Klasse Klasse `PoseController` kann als Fassade angesehen werden, da hier die Interaktion mit den darin verwalteten Klassen gekapselt wird.

BESCHREIBUNG! *fällt evtl raus??*

Klassifikation

Struktur und beteiligte Akteure

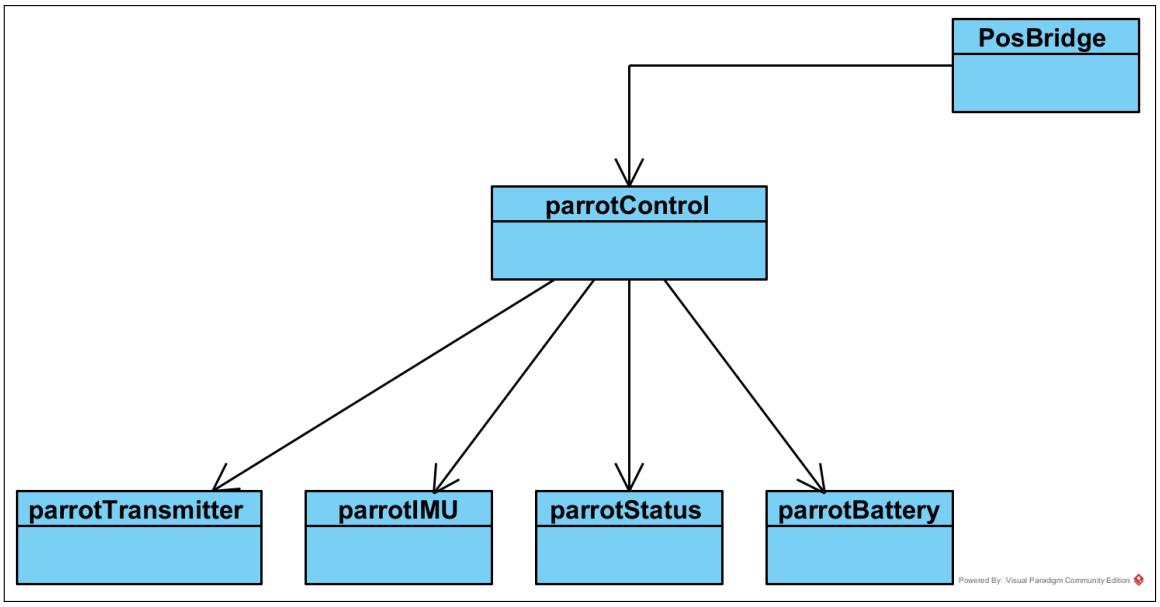

Abbildung 3: Fassade im Projekt

Abbildung 3 zeigt BESCHREIBUNG!

Motivation

Auch hier Design-Entscheidung. Interaktion mit einer Klasse ist einfacher, als sich um alles selbst kümmern zu müssen.

4.0.2 Brücke

DroneController_pkg zu parrot_pkg

„Eigentlich basiert jede Dependency Inversion auf einer Brücke“

Klassifikation

Struktur und beteiligte Akteure

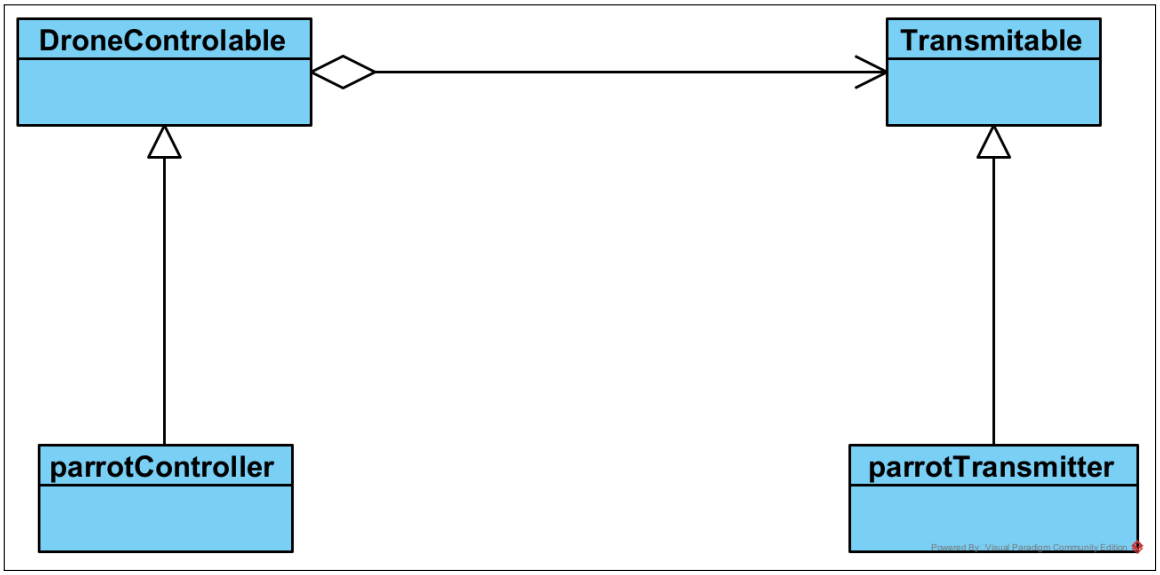

Abbildung 4: Brücke im Projekt

Abbildung 4 zeigt BESCHREIBUNG!

Motivation

Das ist eine Design-Entscheidung um im weiteren Lebenszyklus der *Application* verschiedene Dronen als *PlugIn* einbinden zu können.

4.0.3 Beobachter

BESCHREIBUNG! *hier einfach mal ROS blabla. evtl von der ROS Homepage übersetzen und so*

Klassifikation

Struktur und beteiligte Akteure

Motivation

BESCHREIBUNG! *Ist halt Grundlage von ROS*

Die *ROS-Publisher*und *Subscriber*kommunizieren über *ROS-Topics* miteinander.

(vgl. [3])

The and the turtleteleopkey node are communicating with each other over a ROS Topic. turtleteleopkey is publishing the key strokes on a topic, while turtlesim subscribes to the same topic to receive the key strokes.

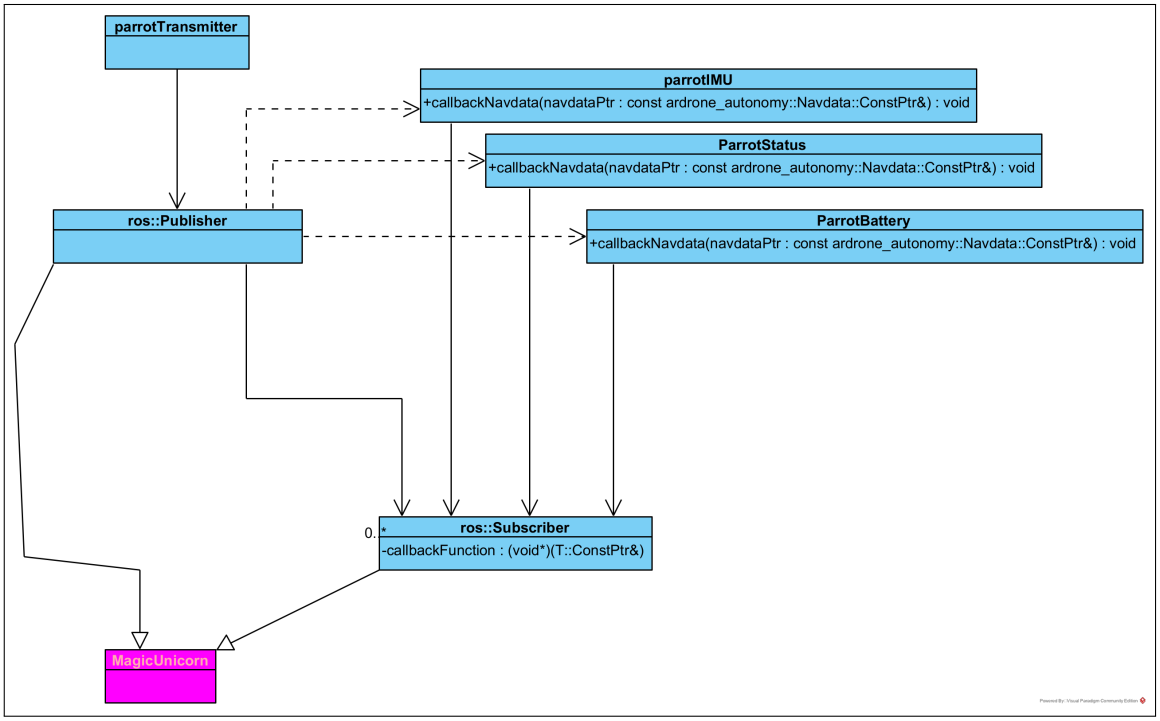

Abbildung 5: Observer im Projekt

Abbildung 5 zeigt BESCHREIBUNG!

BESCHREIBUNG! *evtl überarbeiten ;)* Alles, was wir nicht verstanden haben (oder nicht genauer erörtern wollten) wird durch den Platzhalter *MagicUnicorn* veranschaulicht. Ernsthaft, bei Nachfragen selbst mal den ROS-Jungel durchkämmen.

4.0.4 Eventbus

BESCHREIBUNG!

ACHTUNG, dieses Kapitel ist ein Spass-Kapitel!!

5 Domain Driven Design

Unter *Domain Driven Design* (*DDD*) versteht sich eine Vorgehensweise für Software-Modellierung.

Um Software zu modellieren, welche nach *DDD* entworfen werden soll, wird ein Domänenmodell von der Rolle *Domänen-Expert*in* erstellt. Personen, die diese Rolle bekleiden benötigen keine Programmierkenntnisse. Die Implementierung einer Software bildet die grundlegenden Zusammenhänge des Domänenmodells ab.

Die nachfolgenden Unterkapiteln wird die Anwednung dieser Vorgehensweise auf dem zugrundeliegenden Code analysiert.

BESCHREIBUNG! *Sem5 VL7*

BESCHREIBUNG! *word-cloud basteln?*

BESCHREIBUNG! *DOmain Services??*

5.1 Domänenmodell

Um Personen der Rolle *Domänen-Expert*in* vor unnötigen Programmier-Details (Komplexität) zu schützen, wird ein *Domänenmodell* entwickelt. Hier bildet sich lediglich die *inhärente Komplexität* ab.

Im vorliegenden Projekt können grundsätzlich zwei verschieden abgegrenzte Domänenmodelle zugrunde gelegt werden:

- Ein Modell, welches sich auf die regelungstechnischen Zusammenhänge des Projekts bezieht.
- Ein Modell, welches die Interaktion mit der Drohne beschreibt.

Das Modell in Abbildung 6 (Seite 11) begrenzt sich auf die Betrachtung der Regelung einer Drohne.

5.2 Ubiquitous Language

Die Ubiquitous Language ist eine einheitliche Sprache, auf die sich Entwickler und Domänenexperten einigen und gemeinsam verwenden. In diesem Projekt ist aus Zeitgründen keine solche einheitliche Sprache definiert worden, da die Rollen *Domänen-Expert*in* und *Entwickler*in* von der gleichen Person eingenommen wurden. Verständ-

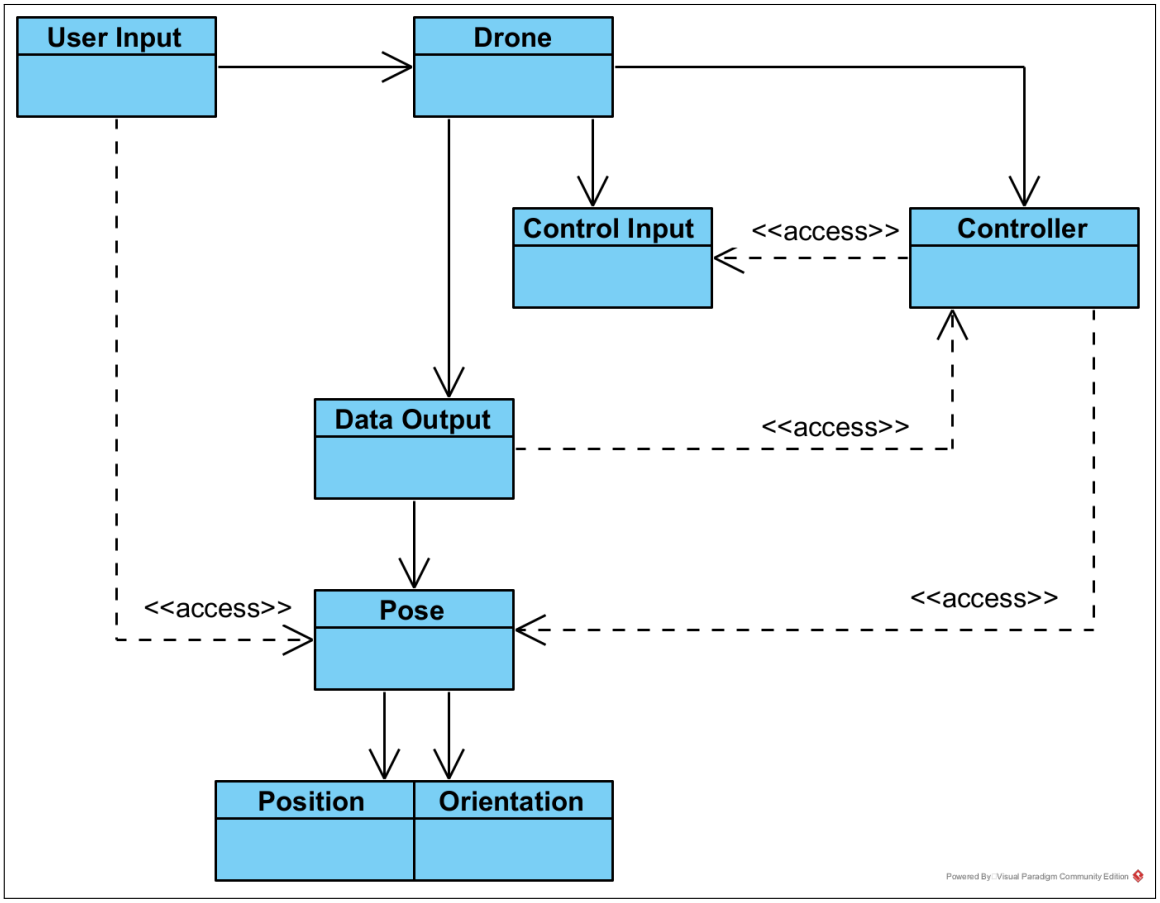

Abbildung 6: Domänenmodell des Projekts

Die gestrichelten Pfeile deuten eine Nutzung der entsprechenden Klasse an, wobei hier kein Attribut innerhalb der ausgehenden Klasse vorgesehen.

nisprobleme sind hierbei nicht zu erwarten.[2]

Für außenstehende hier eine kleine Übersicht: BESCHREIBUNG! *einmal in schöner schreiben. danke*

Domänenmodell	Klassenbezeichnung
Drohne	parrotControl
Data Output	parrotIMU
Control Input	parrotTransmitter
Controller	PoseController
Position	Vector3D
Orientation	Vector3D
User Input	MainClass

5.3 Sichtbarkeitsstufen

BESCHREIBUNG! *Sem5 VL7-2 Min41*

[2] Aber nie ganz auszuschließen.

Muss das hier mit rein? eher optional, weil das nur Lindners Prinzip ;)

BESCHREIBUNG! *listing von SafetyProvider.h und SafetyProvider.cpp, hierin beschrieben, welche Variablen bzw Methoden wie sichtbar sind und wie man das in cpp erkennen kann. (Stufen 0 - 2)* BESCHREIBUNG! *Sprachfeature von cpp: alles, was in einer Klasse Public ist, ist auch untmittelbar in Stufe 3-5*

5.4 Domain Service

BESCHREIBUNG! *Was genau ist das eigentlich und wie relevant ist es, das hier zu beschreiben?*

5.5 Adapter Interface Domain Service

Hamma ned, weil keine Zustände gespeichert werden (im Domainen Modell - in der Pure Fabrication schon ;))

5.6 Repositories

BESCHREIBUNG! *Sem5 VL9-1 Min12*

BESCHREIBUNG! *Keine Interaktion mit persistentem Speicher => Keine Repos im Code.*

5.7 Aggregates

BESCHREIBUNG! *Sem5 VL8-2 Min33*

BESCHREIBUNG! *Sem5 VL9-1 Min6*

BESCHREIBUNG! VL DDD S74: Jede Entity gehört zu einem Aggregat – selbst wenn das Aggregat nur aus dieser Entity besteht

Wenn keine Entities, dann auch keine Aggregate.

5.8 Entities

BESCHREIBUNG! *Sem5 VL7-3 Min46*

BESCHREIBUNG! *Sem5 VL8-1 Min84*

Identitäten sind im Code nicht als solches Abgebildet (es existieren keine Schlüssel).

5.8.1 Surrogatschlüssel

BESCHREIBUNG! *Sem5 VL8-2 Min7*

5.9 Value Objects

Ein beliebiges Objekt ohne eigene Identität werden auch als *Value Object* (*VO*) bezeichnet. Die Kategorisierung als solches ergibt sich durch die *immutable*-Eigenschaft. Heraus leitet sich eine Gleichheit der Objekte bei gleichen Attribut-Werten ab.

BESCHREIBUNG!

Sind immer gleich bei gleichen Variablen-Werten.

FixedPoint Unit Value TimeStamp Vector3D TimedValue Alle ROS_messages

BESCHREIBUNG! *Bei Beispielen: Header-Datei zeigen, weil dort nicht set-Methode vorkommt.*

6 Programming Principles

6.1 SOLID

BESCHREIBUNG! *Sem5 VL4*

6.1.1 Single Responsibility Principle

BESCHREIBUNG! *Was genau soll hier beschrieben werden? nochmal Vorlesung anhören!*

BESCHREIBUNG! *Hier evtl eine Tabelle mit allen Klassen (nach Layer und Alphabet geordnet?) und dem von uns definierten Zweck?*

Beispiel: Klassen aus DroneController_pkg mit jeweiligen Aufgaben beschreiben. => Michael

6.1.2 Open/Closed Principle

BESCHREIBUNG! *Vorlesung Sem5 VL xx*

Offen für Erweitgerung. Geschlosen gegenüber Veränderungen.

BESCHREIBUNG! *TODO: VORLESUNG ANHÖREN!*

Beispiel: Value und TimedValue ??

Beispiel: Der ganze Input/Output Kram für die Controller

Beispiel Closed: ControllerSystem

Gegenbeispiel: Integral1 und Integral2, Erklärung: zu Aufwändig und idR nicht benötigt in diesem Anwendungsfall.

6.1.3 Liskov Substitution Principle

BESCHREIBUNG! *Vorlesung Pricipals Folie 23.*

Kompletter Code ist theoretisch Kovariant.

Hier als Beispiel das Controller-Array in ControllerSystem.

6.1.4 Interface Segregation Principle

implizit Single responsibility.

Klient soll nur das gezeigt bekommen, was er tatsächlich auch benötigt.

Beispiel: DroneController_pkg

6.1.5 Dependency Inversion Principle

BESCHREIBUNG! *Transmitable in PoseController*

Die Brücke.

IMUable => PoseBuilder PoseControlable => parrotTransmitter

PoseControlable ist zwar maßgeblich virtual, aber implenentiert eine Methode (transmitAction). Damit sind wir uns unklar, ob das noch als dependecy inversion zählt. Auch so bei DroneControlable => parrotStatus

6.2 GRASP

6.2.1 Kopplung

PoseController ist ein Monolith.

ROS ist strukturell sehr lose gekoppelt (verschiedene Prozesse, die mit Topics (Observer) miteinander komunizieren).

Brücke ist eher lose gekoppelt.

6.2.2 Kohäsion

mit niedriger Kohäsion wird das single responsibility principle. Wird angewandt. Beispiel: Controller-Aufbau in Domain.

6.3 DRY

Don't repeat yourself

Aus aufwändiger Domain-Architektur ergibt sich, dass das umgesetzt wurde => viele kleine Klassen.

Sichtbar beonders in der Umsetzung von Full Virtual Klassen (Interfaces in Java) und darauf aufbauend Klassen, die diese Methoden implementieren. Beide werden in ‚höherwertigen"Klassen eingesetzt. zum Beispiel Outputable und Output.

We DRY nicht anwendet, ist halt bescheudert - weil faule Menschen wollen doch nicht extra Mehrarbeit. BESCHREIBUNG! *evtl andere fragen, wie hier argumentiert wurde*

BESCHREIBUNG! *Umsetzung mit Duplication Checker Tool*

https://cppdepend.com/blog/?p=556

Duplication Commit `edd7726743b9eb67e159112919224d54c54be670` für die Klasse `FixedPoint<T>`

6.4 KISS

BESCHREIBUNG! *Sem5 VL6*

evtl Vector3D mit den 3 rotate_ Methoden...? Statt eine massive Methode, die alle 3 erlaubt...

6.5 YAGNI

BESCHREIBUNG! *Sem5 VL6-2 Min36*

YAGNI (*You ain't gonna need it*) ist ein Prizip, nach dem Code nur erzeugt wird, wenn dieser unmittelbar benötigt wird. Auch absehbar benötigte Funktionalität ist zurückzustellen.

In diesem Projekt wurde dieses Prinzip aus der Präferenz des Entwicklers heraus nicht angewandt: Grundlegende Funktionalitäten werden hinzugefügt, *WEIL* sie benötigt werden, nicht *WENN* sie benötigt werden.

BESCHREIBUNG! Gedanklich hinundher springen?? Widerspricht guter Planung (Danke Theresa :))

Abweichend hiervon wurde dieses Prinzip im Zusammenhang mit der Anwendung von TDD umgesetzt (siehe Kapitel 7.2 *TDD* (Seite 18)).

6.6 Conway's Lay

Die Entwicklung der betrachteten Software basiert auf einer Person, daher kann davon ausgegangen werden, dass kein Kommunikationsproblem vorliegt. Kommunikation mit Personen, die nicht Teil des Entwicklungsteams sind (Auftraggeber beziehungsweise Betreuer der Studienarbeit), wird hierbei nicht betrachtet.

7 Software Tests

Es gibt folgende Test-Arten:

- Unit Test
- Integration Test
- System Test
- Acceptance Test
- BESCHREIBUNG! *fehlt hier was?*

Allerdings wird in dieser Ausarbeitung nur auf Unit Tests eingegangen.

7.1 Unit Tests

Wahl fällt auf *banditcpp*, da es sich hierbei um ein *headers only* Framework handelt. Daraus folgt, dass die Test-Umgebung nicht in den Produktiv-Code eingebunden werden muss. Es kann quasi als PlugIn „aufgesetzt“ werden.

BESCHREIBUNG! *Bandit bietet leider keine Code Coverage...*

7.1.1 Umgesetzte Unit Tests

verweis auf eine tst.cpp Datei (oder mehrere?)

Mocks

Test callTransmitter Test_Application/test.cpp

7.1.2 ATRIP-Regeln

wurde angewandt. Verweis auf einen beliebigen Test.

7.1.3 Code Coverage

BESCHREIBUNG! *Reka fragen*

using `coverlet` with `Visual Studio`

Getting Started

- installiere Microsoft.NET.Sdk

- installiere Microsoft.NET.Test.Sdk (NuGet.org)

using `Google Test` with `Visual Studio`

7.2 TDD

Test Driven Development (*TDD*) ist ein Prozess in der Software-Entwicklung, wonach jede Klasse beziehungsweise Funktion im produktiven Code bereits im Vorfeld durch einen Test abgedeckt wird.

Hierbeit unterscheidet sich das *TDD* von *Test First* dadurch, dass *TDD* lediglich jeweils einen Test vor dem produktiven Code liegt, wobei *Test First* eine beliebige Anzahl an Tests bereitstellen kann, bevor produktiver Code entsteht. BESCHREIBUNG! *Quelle die letzte Vorlesung - Sem6 VL 3 oder so?*

Im zugrundeliegenden `git`-Repository wird die Entwicklung einer Klasse nach dem *Test First*-Prinzip im Branch `createFixedPoint` mit dem Commit `a4f9bd383c32a7d6d1b0be2319764a6` gestartet.

8 Refactoring

8.1 Code Smells

8.1.1 Duplicated Code

Wie bereits in Kapitel 6.3 *DRY* (Seite 15) beschrieben, finden sich Code-Duplikate im Code. Der hier zu erwähnende Code befindet sich in folgender Klasse: Klasse `FixedPoint<T>` (Commit `edd7726743b9eb67e159112919224d54c54be670`)

+ Funktionen (arythmetische Operatoren)

8.1.2 Long Method

Klasse `PoseBuilder`, Methode `updatePose` Commit `73f3400f4866b27200a37be5d3a80fa8394b2c48`

Wie innerhalb der Methode ersichtlich ist wird ein manipulierender Code durch einen Block eingefasst. Dieser Block kann in eine eigene Methode verschoben werden und somit in der betrachteten Methode zu einer Funtionalität umgeändert werden.

8.1.3 Large Class

Commit `73f3400f4866b27200a37be5d3a80fa8394b2c48` Klasse `FixedPoint<T>`

Sehr viele Methoden / Funktionen, um die Funktionalität gewährleisten zu können. Ist eine Klasse, die Arithmetik implementiert. BESCHREIBUNG! *bitte in schönem Deutsch*

8.1.4 Switch Statement

BESCHREIBUNG! *Theresa denkt sich einen wundervollen Namen für PosBridge aus und dann den Commit mit diesem Namen hier angeben.*

Methode `callbackKeys`

Grundsätzlich sollen *Switch Statements* vermieden werden. In diesem Fall wird davon ausgegangen, dass keine weitere Funktionalität zur Drohne hinzugefügt wird.

8.1.5 Code Comments

An deiser Stelle sollen lediglich Links aufgezählt werden.

- https://github.com/MobMonRob/ROSLabDrohne/blob/d39df8957ec8659b8a6683d8d8ba853d
- https://github.com/MobMonRob/ROSLabDrohne/blob/d39df8957ec8659b8a6683d8d8ba853d

- https://github.com/MobMonRob/ROSLabDrohne/blob/d39df8957ec8659b8a6683d8d8ba853d

Informative Comments und damit sinnvoll:

- https://github.com/MobMonRob/ROSLabDrohne/blob/d39df8957ec8659b8a6683d8d8ba853d
- https://github.com/MobMonRob/ROSLabDrohne/blob/d39df8957ec8659b8a6683d8d8ba853d

8.2 Refactoring

8.2.1 Method Extraction

Angewandt auf die in Kapitel 8.1.2 *Long Method* (Seite 19) beschriebene Methode. Commit `9dd6821a9c63bf1682806b8ab319d6004247d3ca`

8.2.2 Interface Extraction

Da sich die Hardware (Drohnen-PlugIn) im Projektverlauf geändert hat, wurde diese Chance genutzt, eine Bridge einzuführen. Dies hatte den Hintergrund, einen Umstieg auf andere Hardware zukünfitig zu erleichtern. Hierzu sind diverse, teilweise virtuelle[3], Klassen entstanden.

Commit `2b1bccb0f9b2671ace7fb91b6ed86e2be2284596`

BESCHREIBUNG! *Outputable und Inputable suchen... Da wurde das nochmal angewandt.*

8.2.3 Rename Class

In der Vorlesung wird das Umbenennen einer Methode angesprochen. Ziel ist hier, die Verständlichkeit des Codes zu erhöhen. In dem dieser Programmentwurf zugrundeliegenden Code wurden Klassen gefunden, bei welchen die Bezeichnung nicht selbstsprechend erscheint. Um dieser Problematik entgegenzuwirken, soll eine dieser betreffenden Klassen umbenannt werden. Hierbei handelt es sich um die Klasse Klasse `PosBridge`. Commit

BESCHREIBUNG! *muss noch gemacht werden :D*

[3] *virtuelle* Klassen in der Sprache c++ entsprechen *abstrakten* Klassen in Java.

9 Fazit und Ausblick

War alles kacke, viel zu spät angefangen, das aufzuschreiben. Ging aber irgendwie auch nicht früher, weil der Code länger gebraucht hat.

FUUUUUUUUUUUUUUUU

Literaturverzeichnis

[1] Intel's First Microprocessor: Its invention, introduction, and lasting influence,
online, https://www.intel.de/content/www/de/de/history/museum-story-of-intel-4004.html
veröffentlicht -unbekannt-, verändert 08.04.2020, abgefragt 06.09.2021

[2] ROS - Robot Operating System,
online, http://ros.org
veröffentlicht -unbekannt-, abgefragt 10.04.2022

[3] Understanding ROS Topics,
online, http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics
veröffentlicht -unbekannt-, verändert 18.07.2019, abgefragt 27.03.2022

[4] Saks D., Better even at the lowest levels,
online, https://www.embedded.com/better-even-at-the-lowest-levels/
veröffentlicht 01.11.2008, verändert 05.12.2020, abgefragt 28.07.2021

[5] Application Note Object-Oriented Programming in C,
online, https://www.state-machine.com/doc/AN_OOP_in_C.pdf
veröffentlicht 06.11.2020, abgefragt 28.07.2021

[6] Kirk N., How do strings allocate memory in c++?,
online, https://stackoverflow.com/questions/18312658/how-do-strings-allocate-memory-in-c
veröffentlicht 19.08.2013, abgefragt 17.08.2021

[7] Bansal A., Containers in C++ STL (Standard Template Library),
online, https://www.geeksforgeeks.org/containers-cpp-stl/
veröffentlicht 05.03.2018, verändert 12.07.2020, abgefragt 17.08.2021

[8] Automatic Storage Duration,
online, https://www.oreilly.com/library/view/c-primer-plus/9780132781145/ch09lev2sec2.html
veröffentlicht -unbekannt-, abgefragt 17.08.2021

[9] Noar J., Orda A., Petruschka Y., Dynamic storage allocation with known durations,
online, https://www.sciencedirect.com/science/article/pii/S0166218X99001754
veröffentlicht 30.03.2000, abgefragt 17.08.2021

Anmerkung: Wird hier ein Veröffentlichungsdatum als "-unbekannt-" markiert, so konnte diese Angabe weder auf der entsprechenden Webseite, noch in deren Quelltext ausfindig gemacht werden.

Anhang