

**Seminararbeit**

**Von C# nach Python: Software-Konzeptionierung einer  
robotergestützten Lagerverwaltung**

Analyse bestehender Software und Konzeptionierung einer integrierten Python-Anwendung mit  
kameragestützten Validierungsprozessen in der Industrie 4.0-Plattform Modellfabrik  $\mu$ Plant

Lennart Schink

Matrikelnummer:

33237484

Gutachter:

Univ.-Prof. Dr.-Ing. Andreas Kroll

Betreuer:

Dip.-Ing. Axel Dürrbaum

Tag der Abgabe:

10. November 2023

MRT-Nr.:

N.N



# Inhaltsverzeichnis

---

<b>Abkürzungsverzeichnis</b>	<b>XI</b>
<b>Abkürzungsverzeichnis</b>	<b>XII</b>
<b>1. Motivation und Zielsetzung</b>	<b>1</b>
<b>Symbolverzeichnis</b>	<b>1</b>
<b>Index</b>	<b>1</b>
<b>2. Kurzbeschreibung der Lagerzelle</b>	<b>3</b>
<b>3. Softwarearchitekturanalyse bestehender Programme</b>	<b>5</b>
3.1. Lagerverwaltung 3.0 . . . . .	6
3.1.1. Klassenstruktur des Datenmodells . . . . .	7
3.1.2. Klassenstruktur zur Steuerung des ABB Industrieoboter IRB 140 . . . . .	10
3.1.3. Klassenstruktur Modbus TCP/IP . . . . .	13
3.1.4. GUI . . . . .	15
3.2. Controller . . . . .	18
3.2.1. Klassenstruktur des Controllers . . . . .	18
3.2.2. GUI . . . . .	18
3.3. RFID Server . . . . .	22
3.3.1. Klassenstruktur des RFID Servers . . . . .	22
3.3.2. GUI . . . . .	22
<b>4. Funktions- und Anforderungsanalyse</b>	<b>25</b>
4.1. Lagerverwaltung 3.0 . . . . .	25
4.1.1. Bedienfunktionen . . . . .	25
4.1.2. Informationsdarstellung . . . . .	26
4.2. Controller . . . . .	26
4.3. RFID Server . . . . .	27
<b>5. Konzeptionierung einer integrierten Python Anwendung</b>	<b>29</b>
5.1. Datenmodellierung . . . . .	29
5.1.1. Datenklassen . . . . .	32
5.1.2. Konstante Daten . . . . .	34
5.1.3. Programmeinstellungen . . . . .	34
5.2. Konzepte für Controller- und Serviceklassen . . . . .	34
5.3. GUI Konzeptionierung . . . . .	35
5.3.1. PySide6 und QuickQml 2.0 . . . . .	35
5.4. GUI - Modellierung . . . . .	41
5.5. Teilautomatisierte Code Dokumentation mit Sphinx . . . . .	44
5.5.1. Sphinx Erweiterungen . . . . .	44

5.6. Sonstige Festlegungen . . . . .	46
5.6.1. Virtual Environment . . . . .	46
5.6.2. Verwendete Programme . . . . .	46
<b>6. Analyse zur Fehlerbehandlung</b>	<b>49</b>
6.0.1. Fehleridentifikation . . . . .	49
6.0.2. Erkennung fehlerhafter Benutzereingaben . . . . .	50
<b>7. Ideensammlung zu kameragestützten Validierungsprozessen in der Lagerverwaltung</b>	<b>51</b>
7.0.1. Konzepte . . . . .	51
7.0.2. Abgeleitete Anforderungen an die Kamera . . . . .	51
7.0.3. Kameraauswahl . . . . .	51
<b>8. Zusammenfassung und Ausblick</b>	<b>53</b>
<b>A. Dies ist der erste Anhang</b>	<b>XIII</b>
<b>Literaturverzeichnis</b>	<b>XV</b>

# Tabellenverzeichnis

---

3.1. Werte der Klasse ModbusBaseAddress . . . . .	13
5.1. Benötigte Serviceklassen . . . . .	35
5.2. Werte der Klasse ModbusBaseAddress . . . . .	35
6.1. Benutzerin角度ben, mögliche Fehler und ihre Erkennung . . . . .	50



# Abbildungsverzeichnis

---

3.1. Ansicht des Startbildschirms . . . . .	7
3.2. Klassendiagramm Datenmodells . . . . .	8
3.3. Klassendiagramm ABBRobotics Controller . . . . .	11
3.4. Klassendiagramm Modbus . . . . .	14
3.5. Klassendiagramm des Controllers . . . . .	19
3.6. Ansicht des Controller- Startbildschirm . . . . .	20
3.7. Klassendiagramm des Programms RFID Server . . . . .	23
3.8. Startbildschirm des Programms RFID-Server . . . . .	24
5.1. Beispiel: Referenzielle Integrität . . . . .	31
5.2. Objektdiagramm . . . . .	32
5.3. Klassendiagramm Datenmodell . . . . .	33
5.4. Model-View Konzept mit zusätzlichem Controller und Service . . . . .	40
5.5. Mockup des Startbildschirms . . . . .	42
5.6. Mockup des Menüs für die manuelle Lagersteuerung . . . . .	43
5.7. Mockup der Commission List . . . . .	43





# Listings

---

3.1. Formatierung eines Kommandos an den IRB140 Roboterarm . . . . .	12
3.2. Erzeugung eines beschreibenden Strings für den Eventlog . . . . .	12
5.1. Beispiel einer einfachen App mittels PySide6. Zunächst wird eine Instanz der Application-Klasse und der QmlEngine erzeugt. Danach werden Objekte eines Datenmodells und eines Controllers erzeugt und als rootContext der QmlEngine registriert. Anschließend wird die QML-Datei des Hauptfensters geladen, was die App startet. . . . .	38
5.2. Beispiel einer QML-DateiDiese QML-Datei erzeugt ListView QML-Datentyp, der zum Anzeigen der Daten in einem ListModel verwendet wird. Innerhalb eines Rechtecks mit farbigem Rand werden in einem RowLayout Type die Datensätze des Datenmodells gerendert und über die Funktionslogik von Signalen der QML-Datentyps und der Controllerklasse farblich markiert. . . . .	38



# Abkürzungsverzeichnis

---

Abkürzung	Bedeutung
GUI	<b>G</b> raphical <b>U</b> ser <b>I</b> nterface
UML	<b>U</b> nified <b>M</b> odeling <b>L</b> anguage
MES	<b>M</b> anufacturing <b>E</b> xecuting <b>S</b> ystem
C#	An object oriented, component oriented programming language
C++	A high level, general purpose programming language
QML	<b>Q</b> t-Project's <b>I</b> nterface <b>M</b> arkup <b>L</b> anguage
LZ	<b>L</b> agerzelle der $\mu$ Plant
FZ	<b>F</b> ertigungszelle der $\mu$ Plant
AuE	<b>A</b> bfüll- und <b>E</b> ntleer - Station der $\mu$ Plant
RFID	<b>R</b> adio <b>F</b> requency <b>I</b> Dentification
TCP/IP	<b>T</b> ransmission <b>C</b> ontrol <b>P</b> rotocoll / <b>I</b> nternet <b>P</b> rotocoll
MVC	<b>M</b> odel - <b>V</b> iew - <b>C</b> ontroller, Ein Design-Konzept für Software
URI	<b>U</b> niform <b>R</b> essource <b>I</b> dentifier, Im Qt Framework kann dies eine beliebige Ressource sein. Z.B. eine URL, ein Bild oder ein Programmteil.
QML Type	Ein QML Basiselement. Enthält alle für die Visualisierung nötigen Attribute. Eine Liste aller QML Types findet sich hier <a href="#">[1]</a>
RAPID	Eine speziell für die Robotersteuerung entwickelte Programmiersprache
RST	<b>R</b> estructured <b>T</b> ext Eine einfache markup language die in Sphinx benutzt wird



# 1 Motivation und Zielsetzung

---

Das Institut für Mess- und Regelungstechnik an der Universität Kassel hat in den letzten Jahren eine Modellfabrik  $\mu$ Plant gebaut. Aus über 70 Einzelarbeiten ist ein modernes Industrie-4.0 Konzept geschaffen worden. Teil der  $\mu$ Plant ist ein vollautomatisiertes Lager. Das Lager besteht aus einem abgetrennten Raum, dessen Zugang über eine Tür mit einem Türschalter überwacht ist. In diesen Bereich können autonome mobile Roboter (Turtlebots) einfahren. In dem abgetrennten Bereich steht ein Industrieroboter Typ ABB IRB 140 und ein Lagerregal mit ausgewiesenen 18 Lagerplätzen. Außerdem befindet sich neben einer Andockstation für den Turtlebot ein Kommissioniertisch.

Ein pneumatischer Greifer kann Paletten, die je mit bis zu zwei Bechern bestückt werden können, zwischen dem mobilen Roboter und dem Lagerregal transportieren. Von einem PC-Arbeitsplatz aus können mittels Software die Lagerprozesse überwacht werden. Im Fehlerfall kann eingeschritten werden oder es können manuell Prozesse ausgelöst werden. Die Software ist derzeit in 3 Programme aufgeteilt: Einerseits gibt es die Lagerverwaltung 3.0 - die Hauptsoftware. Sie bildet die automatisierten Prozesse ab und verfügt über ein GUI welches u.A. den Bestand visualisiert. Daneben gibt es den Warehouse Controller, der dazu verwendet wird, manuelle Prozesse auszulösen. Zudem gibt es ein Programm „RFID-Server“ mit dem über RFID Leser der Fa. Feig Tags der Transportbehälter ausgelesen werden können.

Mit dem Wechsel des Betriebssystems von Windows 7 auf Windows 10 ist die Kompatibilität der C# Implementierung nicht mehr gegeben. Außerdem laufen Teilfunktionen des Programms nicht fehlerfrei oder tolerieren kaum Fehlbedienungen. Die Dreiteilung der Software ist im Allgemeinen auch nicht mehr erwünscht.

Diese Seminararbeit beschäftigt sich mit der Analyse der bestehenden Software: Es wird ermittelt, aus welchen Programmteilen und Funktionen die Software besteht. Aus den Erkenntnissen wird ein Konzept entwickelt, welches die Funktionen der drei Software Teile zusammenführt. Dies soll die Grundlage für eine Migration der Software nach Python schaffen.

Erkenntnisse aus der studentischen Arbeit von Sebastian Hübler aus dem Jahr 2019 [2] sollen überprüft und vertieft werden um Anforderungen an Kameras und arUco Marker zu ermitteln, die später eine automatisierte Inventur ermöglichen sollen.



## 2 Kurzbeschreibung der Lagerzelle

---

Die Lagerzelle ist ein abgesperrter Bereich innerhalb der  $\mu$ Plant. Sie hat eine überwachte Zugangstür für Personen und eine Andockstation für einen mobilen Roboter, die für Personen nicht passierbar ist. Die Andockstation ist mit einem RFID-Lesegerät der Fa. FEIG GmbH und zwei induktiven Näherungssensoren ausgestattet, sowie einer Ladestation.

Direkt dahinter steht ein ABB Industrieroboter IRB 140 auf seinem Fundament.

Aus Sicht des Roboters um 90° dazu befindet sich ein Kommissionier-Tisch mit Platz für bis zu zwei Paletten. Die Paletten-Plätze sind mit „K1“ und „K2“ gekennzeichnet. Für die Becher sind die Plätze „a“ und „b“ auf der vom Roboterarm abgewandten Seite des Tisches angebracht.

Wiederum 90° versetzt zum Kommissioniertisch, also gegenüber der Andockstation befindet sich ein Lagerregal. Es hat drei Böden mit jeweils sechs Lagerplätzen für Paletten. Die Lagerplätze sind mit „L1“ (oben links) bis „L18“ (unten rechts) gekennzeichnet. An der dem Benutzer zugewandten Seite sind die Becherplätze „a“ (vorne) und „b“ (hinten) angebracht.

Der Roboterarm hat einen pneumatischen Greifer, mit dem er eine Palette oder einen Becher aufnehmen kann.





# 3 Softwarearchitekturanalyse bestehender Programme

---

Analysemethoden der Informatik für Software sind in der Regel für die verschiedenen Design-Phasen entwickelt worden. Eine von mir durchgeführte Internet-Recherche ergab, dass sich Analyse-Tools und Methoden für bestehende Software vor allem darauf fokussieren, die Performance, Speichermanagement und Benutzererfahrung zu bewerten. Die Architektur einer Software spielt dabei eine untergeordnete Rolle. Für einen Nachfolger der Lagerverwaltung 3.0 soll jedoch zunächst ihre Softwarearchitektur untersucht werden. Die Performance und der Speicherverbrauch spielen für die  $\mu$ Plant eine untergeordnete Rolle. Eine Bewertung der Programmkomponenten sollt also anhand folgender Kriterien bewertet werden:

- Ihrem Nutzen für den Anwender
- Zuverlässigkeit im Betrieb
- Umgang mit erwartbaren Fehlern

Der Programmcode sollte zudem

- Leicht lesbar und verständlich,
- Zuverlässig und robust, und
- gut erweiterbar sein.

In einem C# Projekt sind GUI und Buisnesslogik in getrennten Dateien implementiert. XAML-Dateien gehören zu Microsofts .NET Plattform. Ihr Dateiformat ähnelt denen von XML-Dateien. Sie legen fest wie das GUI gerendert wird und werden im integrierten Editor der Visual Studio IDE bearbeitet. .XAML.cs Dateien implementieren die Controller-Logik des XAML-Inhalts. Ihre Programmiersprache ist C#.

Am Beispiel des Startbildschirms des Programms „Lagerverwaltung 3.0“ wird der Programmaufbau geschildert. Aufgrund der Komplexität und des fehlenden Mehrwerts wird weiteren Verlauf dieser Arbeit darauf verzichtet. Aus dem gleichem Grund wird auf die detaillierte Beschreibung der Funktionsweise von grafischen ELelementen ihrer Events sowie ihrer Eventhandler verzichtet.

In den erstellten Klassendiagrammen ist gut ersichtlich, welche Klassen Events nutzen. Sie erben von der Klasse `INotifyPropertyChanged`. Die Klasse ist ein Interface der .NET Plattform und wird immer dann eingesetzt, wenn eine GUI-Komponente über eine Änderung informiert werden soll. Sie wird im Allgemeinen dafür benutzt um das GUI mit dem hinterlegten Datenmodell zu verknüpfen.

Wenn diese Klasse implementiert wird, muss die Methode `OnPropertyChanged` überschrieben werden, die immer dann aufgerufen wird, wenn sich ein Wert ändert.

---

## 3.1. Lagerverwaltung 3.0

Wie in dem einleitenden Abschnitt angekündigt wird zunächst der grundlegende Aufbau des Programms beschrieben.

Die Datei `App.xaml` ist der Einstiegspunkt des Programms. In ihr wird ein Objekt der Application-Klasse mit allen benötigten Ressourcen erzeugt. Als Start URL ist `MainWindow.xaml` angegeben.

In der Datei ist beschrieben, wie der Startbildschirm gerendert wird.

Zunächst wird ein Banner gerendert, bestehend aus dem Titel des Programms, dem Logo des Instituts und der  $\mu$ Plant (Siehe Abb.3.1 Bereich „A“). Es werden außerdem alle benötigten Datenobjekte erzeugt. Sie lassen sich wie folgt einteilen:

- Objekte und Variablen, die dem Lager zugeordnet sind:
  - Ein Objekt `inventory` der Klasse `Inventory` für das Inventar mit `null` initialisiert.
  - Ein Objekt `storageMatrix` von der Klasse `PalletMatrix` erzeugt.
  - Ein Objekt `commissionMatrix` von der Klasse `PalletMatrix` erzeugt.
  - Ein Objekt `mobileRobot` von der Klasse `MobileRobot` erzeugt.
  - Außerdem eine Variable `lastCupRead` vom Datentyp `ushort` (16-Bit-Ganzzahl, vorzeichenlos) mit 0 initialisiert.
- Objekte und Variablen initialisiert, die dem ABB Controller zugeordnet sind:
  - Ein Objekt `commands` von der Klasse `controllerCommandList`.
  - Ein Objekt `controllerProperties` von der Klasse `RobotControllerProperties`.
  - Ein Objekt `controllerBase` von der Klasse `RobotControllerBas`, mit dem Initialisierungswert `null`.
  - Ein Objekt `controllerSim` von der Klasse `RobotSimulator`.

Im Constructor der Klasse `MainWindow` werden zudem der ModBus und der Roboter Controller initialisiert und ihre GUI-Elemente gerendert (Abb.3.1 Bereiche „B“ und „C“).

Die Produktliste wird aus der Datei `Produkte.db` geladen und gerendert (Abb.3.1 Bereich „D“).

Daten des Lagers und des mobilen Roboters sowie die Kommissionsdaten werden aus der Datei `CommissionData.db` geladen und anschließend das Inventar gerendert (Abb.3.1 Bereich „E“ und „G“).

Bereich „F“ ist der Eventlog der Anwendung, dort werden alle Ereignisse der Software als Text angezeigt. Das können Fehler sein aber auch Fortschritte im Programmablauf.

Im mittleren Bereich ist die Anordnung von Roboter, Andockstation und Kommissioniertisch symbolisiert. Der „Start“-Knopf startet den Automatikbetrieb. Nach dem Start der Automatik kann der selbe Knopf zum Stoppen des Automatikbetriebs verwendet werden.

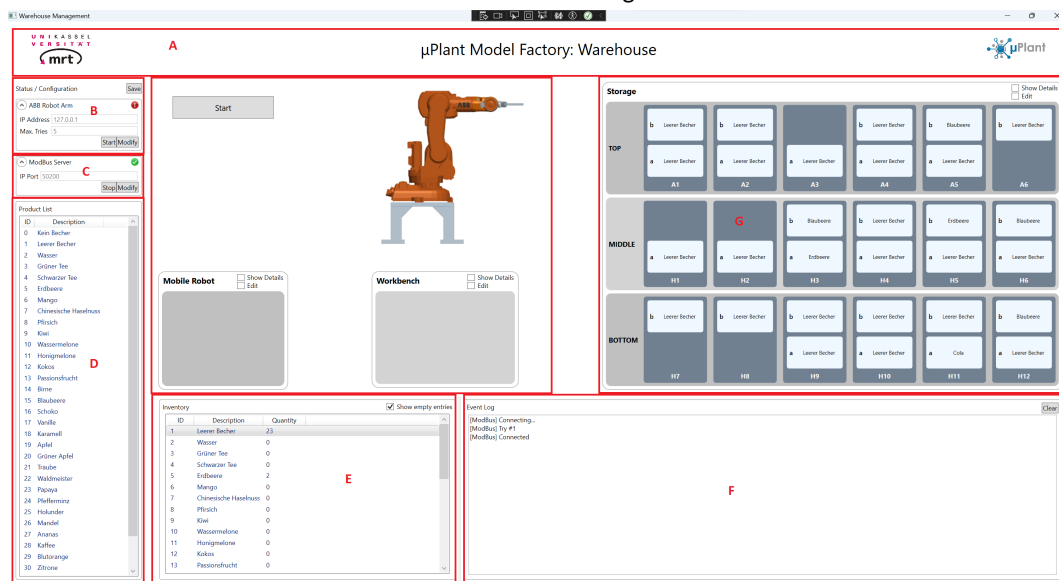
Wenn keine Verbindung zum Modbus hergestellt werden kann, wird dem Benutzer angeboten die Vorgänge zu simulieren. Im Klassendiagramm Abb.3.2 wird jedoch schnell deutlich, dass dieser Simulationsbetrieb nicht für einen Testbetrieb geeignet ist, da dazu eine ganz andere Klasse verwendet wird.

Im Bereich „Mobile Robot“ wird nach erfolgreichem Andocken das erkannte Produkt angezeigt. Dem Bediener wird hier angeboten die Daten manuell zu manipulieren oder Details ein- bzw. auszublenden.

Im Bereich „Workbench“ werden bis zu zwei Paletten mit ihrem Inhalt gerendert. Auch hier wird dem Benutzer angeboten, die Daten per Hand zu manipulieren.

Im Folgenden werden die wichtigsten Klassenstrukturen beschrieben. Auf die Beschreibungen von Hilfsklassen und solche, die für die .NET Plattform oder die Programmiersprache C# spezifisch sind, wird verzichtet.

**Abbildung 3.1.:** Startbildschirm der Anwendung, zur besseren Beschreibung sind die Bedienbereiche rot umrandet und mit Buchstaben gekennzeichnet



### 3.1.1. Klassenstruktur des Datenmodells

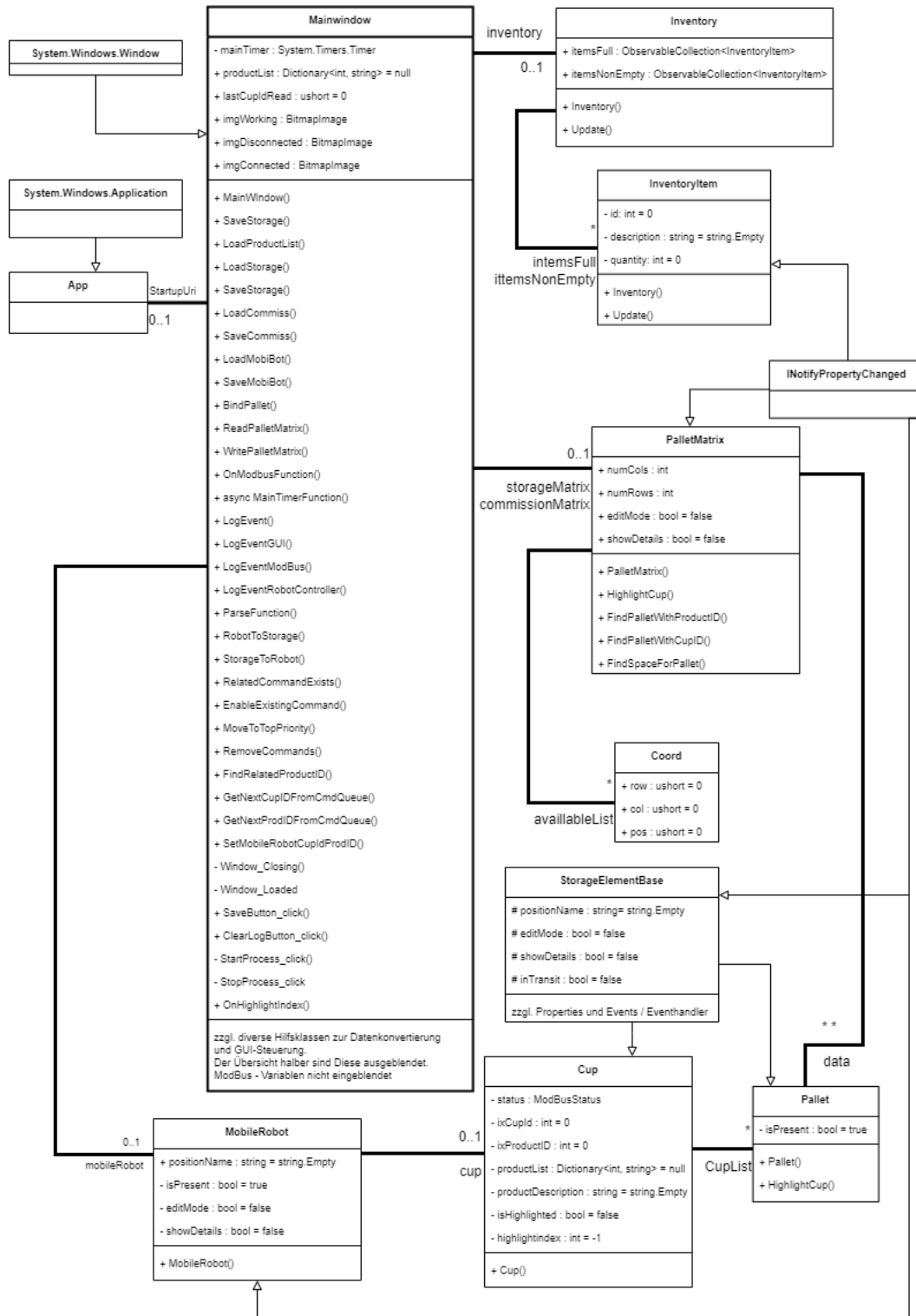
Die bestehende Software dient dazu Lagerpakete vom mobilen Roboter auf die Werkbank oder ins Lager zu bewegen. Oder alle möglichen Kombinationen davon. Ein Lagerpaket ist einer der nachfolgenden Varianten:

- Ein Becher
- Eine leere Palette
- eine Palette mit einem oder zwei Bechern

Der Programmierer hat sich diese Struktur angeeignet und in der Datenmodellierung umgesetzt. In Abb.3.2 ist gut ersichtlich, dass die Klasse `StorageElementBase` an die Klassen `Cup` und `Palett` vererbt (schmale Linie mit leerem Pfeil, in Anlehnung an UML). Eigenschaften, die sowohl Palette als auch den Becher betreffen, sind in dieser Klasse implementiert.

Weiterhin findet sich das Lager als eigene Klasse `Inventory` und der mobile Roboter als `MobileRobot` wieder. Die `Inventory`-Klasse ist jedoch nicht das Lager im Sinne von Abb.3.2 Bereich „G“, sondern auf die Liste in Bereich „E“.

**Abbildung 3.2.:** Klassendiagramm der MainWindow-Klasse mit vererbenden- und Datenmodell-Klassen



---

Zentrales Element ist die `MainWindow` Klasse. Sie implementiert eigentlich die Interface-Klasse `System.Windows.Window`. Der Programmierer hat sie allerdings auch als Daten-Hub verwendet.

Wie zu Beginn des Abschnitts geschildert werden beim Rendern des Fensters alle benötigten Datenobjekte erzeugt oder aus Dateien geladen.

- In der Klasse `Inventory` wird der Inhalt der Datei `Produkte.db` in zwei Listen geladen, sodass eine Liste mit lagernden Produkten und eine vollständige Produktliste gespeichert werden. Eine Instanz `inventory` wird zur Laufzeit erzeugt. Wenn die Dateien `Produkte.db` und `StorageData.db` zu dem Zeitpunkt nicht verfügbar ist, stürzt das Programm ab.
- In der Klasse `PalletMatrix` wird die Datei `StorageData.db` bzw. `CommissionData.db` geladen um einen zweidimensionalen Array `data` zu erzeugen. Jedes Array-Element ist ein Objekt der Klasse `Pallet` und enthält eine Liste `CupList` von Objekten der Klasse `Cup`. Diese Struktur wird dazu verwendet, um das reale Lager zu visualisieren. Zur Laufzeit werden zwei Objekte der Klasse `PalletMatrix` erzeugt:
  - `storageMatrix` bildet das Datenmodell um die Visualisierung in [Abb.3.2](#) Bereich "G," zu realisieren.
  - `commissionMatrix` bildet das Datenmodell für die Visualisierung des mobilen Roboters und der Workbench.

---

### 3.1.2. Klassenstruktur zur Steuerung des ABB Industrieroboter IRB 140

Um dem Industrieroboter ABB IRB 140 Kommandos zu senden, muss das Programm mit der Steuerung IRC5 kommunizieren. Die Steuerung ist Baujahr 1991 und hat eine Firmware „RobotWare 5.15.1004.01“ installiert. Die Kommunikation erfolgt über ein verschlüsseltes TCP/IP Protokoll.

Im Programmcode finden sich dazu wie in Abb.3.3 gezeigt, die Klassen der Bibliothek ABBRobotics. Instanzen dieser Bibliotheksbestandteile werden in der Klasse RobotController erzeugt, die von der Klasse RobotControllerBase erbt.

Interessant ist, dass in der MainWindow Klasse kein Objekt von RobotController erzeugt wird, sondern eins von der Klasse RobotControllerBase „controllerBase“, welches mit null initialisiert wird. Diese Klasse hat Properties für die Näherungssensoren der Andockstation und virtuelle Funktionen um den Status abzurufen, den mobilen Roboter als abfahrbereit zu melden und Kommandos auszuführen.

In der Methode StartProcess\_click - also mit Klick auf den Start Button - wird das Objekt verändert mit der Zuweisung

$$\text{controllerBase} = \text{robotControl.controlHandler};$$

Die Zuweisung löst die Anmeldung des PC's mit der Steuerung aus und gibt als Rückgabewert ein Objekt der Klasse RobotController zurück.

#### Klassen und Methoden der Kommandos

Die Klasse ControllerCommandList erbt von einer Liste der Klasse ControllerCommand. Beide Klassen gehören zum namespace der RobotControllerBase.

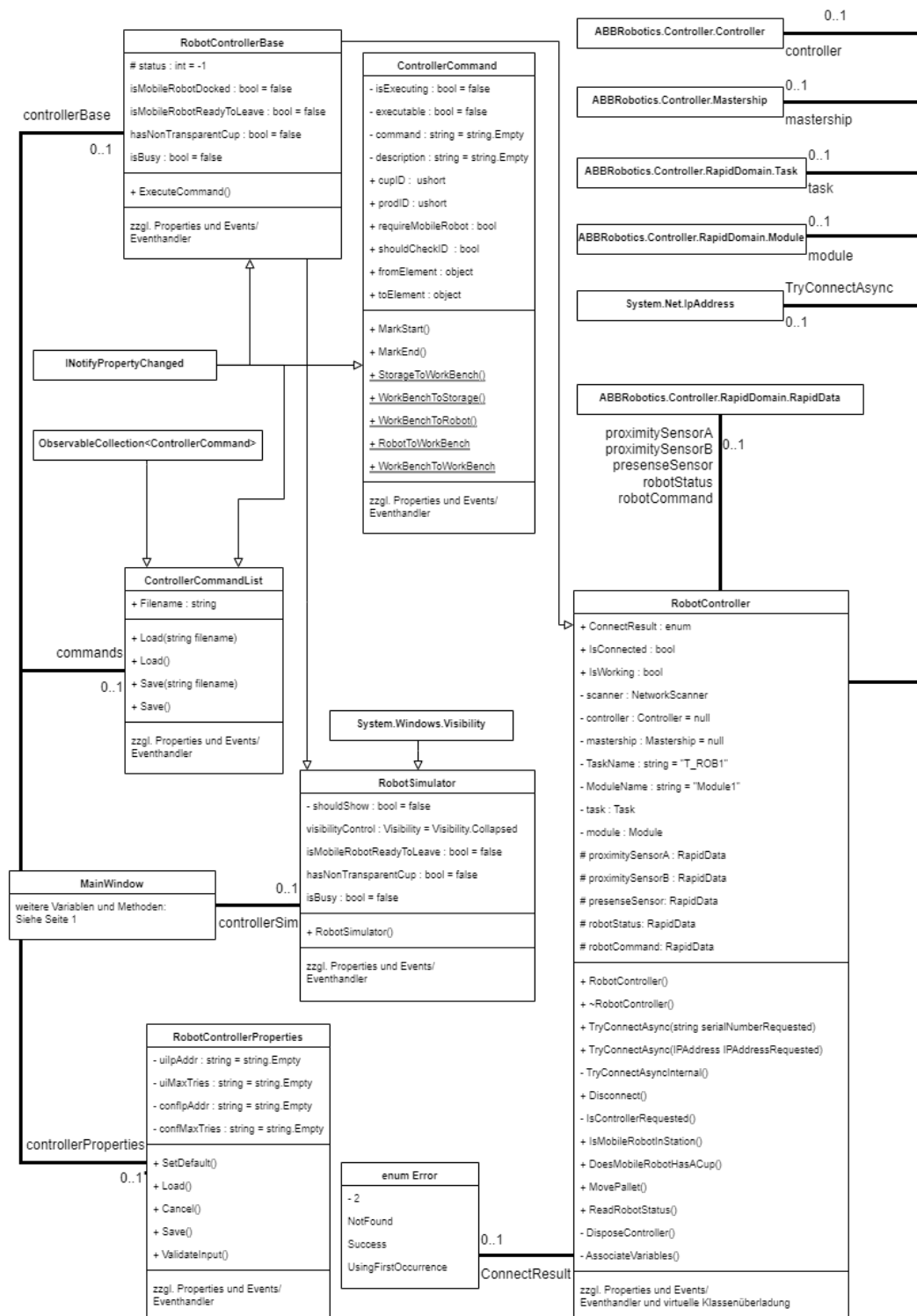
Beim Initialisieren des Programms wird eine leere ControllerCommandList erzeugt. Die Klasse ControllerCommand hat Properties, die bspw. den Status, ein Kommando-Datenstring und eine Beschreibung des Kommandos enthalten. Sie implementiert demnach das Datenmodell der Kommandos.

Die Methoden und Variablen in der Klasse implizieren, dass die Commands aus einer Datei geladen werden. Sie lassen sich aber sowohl in der IDE Visual Studio als auch in der IDE Rider keine Verweise zu den Methoden finden.

Die Klasse ControllerCommand hat 5 statische Methoden. Jede von ihnen gibt ein Objekt der Klasse ControllerCommand zurück.

- StorageToWorkBench übermittelt der Steuerung den Befehl eine Palette vom Lager zum Kommissioniertisch zu transportieren.
- WorkBenchToStorage übermittelt der Steuerung den Befehl eine Palette vom Kommissioniertisch zum Lager zu transportieren.
- WorkBenchToRobot übermittelt der Steuerung den Befehl eine Palette vom Kommissioniertisch auf den mobilen Roboter zu platzieren.
- RobotToWorkBench übermittelt der Steuerung den Befehl eine Palette vom mobilen Roboter auf den Kommissioniertisch zu transportieren.
- WorkBenchToWorkBench übermittelt der Steuerung den Befehl, dass eine Palette

**Abbildung 3.3.:** Klassendiagramm der MainWindow-Klasse mit Klassen aus dem ABBRobotics Controllerframework.



---

den Platz auf dem Kommissioniertisch wechseln soll.

Als Parameter in der Methodensignatur werden die Koordinaten der Start- und Endpunkte übergeben sowie Pallet Objekte am Start- und Endpunkt.

In dem ControllerCommand Objekt werden zwei Strings erzeugt, die am Beispiel StorageToWorkBench erklärt werden:

**Listing 3.1:** Formatierung eines Kommandos an den IRB140 Roboterarm

```
command = string.Format(
    "Palette_{0}_{1}_1_{2}",
    station, position, w_col + 1
)
```

station ist ein Integer mit dem Wert von 3, wenn die oberste Regalreihe ausgewählt wurde, oder 2 sonst.

position ist ein Integer, der sich aus der Spalte des Lagerregals errechnet.

w\_col+1 ist die Angabe des Ortes am Kommissioniertischs.

**Listing 3.2:** Erzeugung eines beschreibenden Strings für den Eventlog

```
Description = string.Format(
    "[{0}.{1}]_{2}({3})_{4}({5})",
    cupID,
    prodID,
    str_title_Storage,
    PositionName,
    str_title_Commissioning,
    PositionName
)
```

Mit 3.2 wird ein String erzeugt, der eine Beschreibung des Vorgangs enthält.

Die ControllerCommands werden in Methoden der MainWindow Klasse erzeugt. Die Parameter der Funktionssignatur stammen aus dem Objekt commissionMatrix welche in Kapitel 3.1.1 bereits beschrieben wurde.

Die MainWindow Klasse hat eine Timer-gesteuerte Funktion MainTimerFunction, die die Commands fallweise abarbeitet. Wie genau die Daten vom Modbus Server in die commissionMatrix geschrieben werden konnte ich mit meinen eher bescheidenen C# Kenntnissen nicht ermitteln.



---

### 3.1.3. Klassenstruktur Modbus TCP/IP

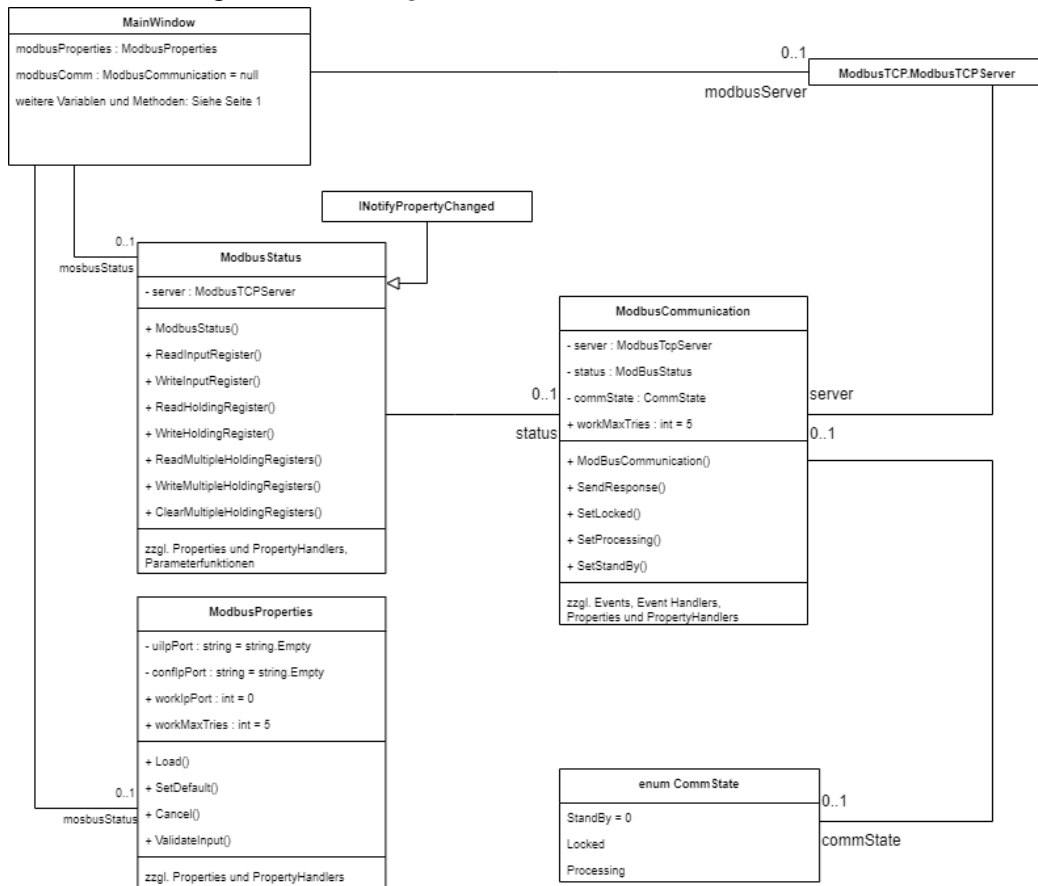
Modbus ist ein open-source Kommunikationsprotokoll welches 1979 von Modicon (heute Schneider Electric) veröffentlicht wurde. Es wird dazu verwendet Client-Server Verbindungen einzurichten und ist laut Modbus Organization de facto Standard in industriellen Herstellungsprozessen. Modbus unterstützt verschiedene Kommunikationsstrukturen. Modbus TCP/IP ist lediglich eine Variante, die 1999 entwickelt und veröffentlicht wurde. TCP/IP ist das übliche Transportprotokoll des Internets und besteht aus einer Reihe von Layer Protokollen, die einen zuverlässigen Datentransport zwischen Maschinen bereitstellen. Die Verwendung von Ethernet TCP/IP in der Fabrik ermöglicht eine echte Integration mit dem Unternehmensintranet und MES-Systemen, die die Fabrik unterstützen. Die Protokollspezifikation und Implementierungsanleitung sind auf der Seite der Modbus Organization frei verfügbar[3].

Lars Kistner [4] hat in seiner Bachelorarbeit aus dem Jahr 2016 die Kommunikationsstruktur der  $\mu$ Plant festgelegt. Wie er beschreibt, existiert ein zentraler Modbus Server, der die „Kommandos“ in die entsprechenden Modbus Adressen und/oder Funktionsregister schreibt. Diese Adressen sind in der Datei `ModbusBaseAddress.cs` niedergeschrieben und um die Adressen des mobilen Roboters ergänzt.

**Tabelle 3.1.:** Werte der Klasse `ModbusBaseAddress`

Adresse	Wert	Beschreibung
Status	1	
KeepAlive	2	
Working	3	
FunctionReady	5	
ResponseReady	6	
Done	7	
AgentLockID	8	
AgentLockRequest	9	
FunctionID	10	
FunctionParameters	11	
FunctionParametersLength	32	
ResponseID	43	
ResponseParameters	44	
ResponseParametersLength	32	
StorageCup	1024	
StorageProduct	1280	/// 1024 + 256
CommissionCup	2048	
CommissionProduct	2080	/// 2048 + 32
MobileRobotCup	2560	
MobileRobotProduct	2568	/// 2560+8
MobileRobotDocked	2576	
MobileRobotReadyToLeave	2580	

**Abbildung 3.4.:** Klassendiagramm der MainWindow-Klasse mit Mosbus-Klassen



Schaut man sich das Klassendiagramm der Modbus - Implementierung 3.4 an, stellt man fest, dass es die Klasse **MainWindow** Objekte von drei verschiedenen Klassen instanziiert, die sich dem Modbus Protokoll zuordnen lassen.

Sie besitzt ein Objekt der Klasse **ModbusStatus** welches ein Objekt der Klasse **ModbusCommunication** besitzt.

`modbusTCP.modbusTCP`Server wird ebenfalls als Objekt in **MainWindow** erzeugt. Gleichzeitig wird sie aber auch in **ModbusCommunication** instanziiert.

Die dritte Klasse ist **ModbusProperties**. Sie ist eine Datenklasse und enthält die IP-Adresse und den Port des Modbus Servers.

**CommissionMatrix** Die Klasse **RobotController** übersetzt diese in Strings und schickt sie an die IRC5 Steuerung. Wenn diese den Befehl ausgeführt hat, meldet sie „Fertig“ zurück.

Im Anhang B.5 seiner Arbeit sind die Modbus-Adressen, die dem Lager zugeordnet sind, aufgelistet. Der Adressenbereich ist lückenlos zwischen 1024 und 1105. Jeder Ablageort eines Bechers (z.B. L1a für Lagerort L1, vorne) hat zwei Adressen: Eine für die Cup ID und eine für die Product ID. Das schließt den Kommissioniertisch und den mobilen Roboter mit ein.

---

### 3.1.4. GUI

Das GUI besteht aus einem Fenster, welches in 8 Bereiche unterteilt ist (siehe Abb.3.1). Der Bereich „A“ hat keine Bedienfunktion.

#### ABB Controller Einstellungen

Der Bereich „B“ ermöglicht dem Benutzer die Verbindungsdaten zur Steuerung des ABB Roboter IRB140 zu konfigurieren und die Verbindung herzustellen. Die Verbindungsdaten sind voreingestellt und die Felder sind standardmäßig für die Eingabe deaktiviert. Die Aktivierung des Felds erfolgt über die Taste „Modify“. Mit der Taste „Save“ können Änderungen gespeichert werden.

Mit der „Start“ Taste wird die Verbindung hergestellt. Die Taste verändert daraufhin ihren Text zu „Connecting...“ und bei erfolgreich hergestellter Verbindung zu „Stop“. Ein Symbolbild informiert dabei über den Verbindungsstatus.

#### Modbus Einstellungen

Im Bereich „C“ kann der IP-Port des Modbus-Servers über die Taste „Modify“ angepasst werden. Direkt nach Programmstart, noch vor der Betätigung der Start-Taste, ist das Symbolbild für den Verbindungsstatus grün und suggeriert eine erfolgreiche Verbindungsherstellung. Nach meinem Verständnis handelt es sich hierbei um eine Fehl-Initialisierung.

Die Verbindungstaste zum Starten der des Automatikbetrieb steht zum Programmstart auf „Stop“.

#### Produktliste

Im Bereich „D“ befindet sich eine scrollbare Liste aller möglichen Produkte mit zwei Spalten. Es wird neben dem Produktnamen die Produkt-ID angezeigt. Vermutlich dient die Liste als Nachschlagewerk oder Übersicht, denn es sind ansonsten keine Bedienfunktionen verfügbar.

#### Inventaranzeige

Im Bereich „E“ ist die Gleiche Liste um eine Spalte erweitert, in der die gelagerte Menge aufgeführt ist. Die Liste bildet somit eine bessere Übersicht über das gelagerte Inventar und ist auch als „Inventory“ gekennzeichnet. Mit der Checkbox „Show empty entries“ können Produkte ohne eingelagerte Menge ausgeblendet werden.

#### Eventloganzeige

Bereich „F“ ist ein Eventlog. Hier werden von dem Programm aus verschiedenste Mitteilungen dem Benutzer angezeigt. Mit der Taste „clear“ werden alle existierenden Einträge gelöscht.

---

## Lagervisualisierung

Bereich „G“ ist die Visualisierung des Lagers. Die drei Reihen „TOP“, „MIDDLE“ und „BOTTOM“ sollen die drei Regalböden des Lagerregals nachbilden. Sie sind als hellgraues Rechteck gerendert.

Die Slots A1...A6, H1...H6 sowie H7...H12 sind die Plätze für je eine Palette, die wiederum Platz für je zwei Becher hat. Der Ursprung dieser Bezeichnung konnte während der Vorbereitungen auf diese Arbeit nicht geklärt werden. Wie ich in 2 beschrieb habe, weicht diese Art der Bezeichnung der Lagerplätze von der realen Lagerzelle ab.

Das dunkelgraue Rechteck mit der Beschriftung des Lagerorts symbolisiert die Palette. Die beiden weißen Rechtecke darauf symbolisieren je einen Becher. Sie sind mit „a“ für vorne und „b“ für hinten gekennzeichnet und zeigen den Produktnamen an.

Ist ein Slot für einen Becher leer, wird das entsprechende Rechteck nicht gerendert. Ein leerer Becher ist ein Produkt im Sinne des Programms.

Ist keine Palette vorhanden, verbleibt der gesamte Bereich der Palette leer.

In der oberen, rechten Ecke kann der Benutzer mit der Checkbox „Show Details“ zusätzlich die Becher-ID und die Produkt-ID ein- und ausblenden.

Mit der Checkbox „Edit“ kann der Benutzer alle Daten (Palette vorhanden/ nicht vorhanden, Becher-ID, Produkt-ID) als Eingabefeld sehen und ändern. Der Produktname wird dabei über die Produkt-ID referenziert. Die Änderungen werden gespeichert, indem die Checkbox einfach wieder abgewählt wird.

## Prozessvisualisierung

Der Bereich der Prozessvisualisierung ist in der Mitte des Bildschirms.

Im Automatikbetrieb ist dieser grob in 4 Bereiche eingeteilt.

**Oben Rechts** ist ein Symbolbild des Industrieroboters IRB 140 dargestellt. Das Bild bietet keinerlei Bedienfunktion.

**Oben links** befindet sich die Start-Taste und im Simulationsbetrieb weitere GUI-Elemente. Wenn die Verbindungen zum Modbus und zur IRC5-Steuerung hergestellt sind, kann mit ihr der Automatikbetrieb gestartet werden. Ist der Modbus nicht verbunden, erscheint nach dem Klick ein Dialogfenster. Der Benutzer wird gefragt, ob aufgrund der fehlenden Verbindung der Controller simuliert werden soll. Klickt man nun auf „Ja“, wird über dem mobilen Roboter ein weiteres Rechteck gerendert. Es enthält zwei Checkboxes „Mobile Robot is present“ und „Robot has non-transparent cup“. Erstere rendert nach dem Klick einen mobilen Roboter (schwarzes Oval) und gibt dem Benutzer die Möglichkeit, wie gewohnt über die beiden Checkboxes im oberen rechten Bereich, Daten eines Bechers und Produkts einzugeben. Klickt man nun wieder auf die Taste, die nun mit „Stop“ beschriftet ist, erscheint für etwa eine Sekunde der Schriftzug „Wait...“, bevor das Programm wieder in den Ausgangszustand wechselt.

**Unten links** befindet sich die Visualisierung des mobilen Roboters.

**Unten rechts** befindet sich die Visualisierung des Kommissioniertisches.

Beide Lagerbereiche, für den mobilen Roboter und den Kommissioniertisch, geben dem

---

Benutzer die Möglichkeit die Becher- bzw. Produkt-ID ein- und auszublenden oder zu überschreiben.

Weiterhin bietet das Programm keine Möglichkeit den Roboter zu steuern.

---

## 3.2. Controller

Der Controller ist eine kleine Anwendung, die die manuelle Bedienung der Lagerzelle ermöglicht. Die Implementierung der RFID - Funktionen können anhand des Klassendiagramms nicht gezeigt werden. In Vorbereitung auf diese Arbeit habe ich nicht ausprobiert, ob das Programm diese Funktionen so erfüllt wie sie in dem GUI impliziert wird. Für die weitere Arbeit würde dies auch keinen Mehrwert bieten, wie sich in Kapitel 5 noch zeigen wird.

### 3.2.1. Klassenstruktur des Controllers

Der Controller ist genau wie die Lagerverwaltung 3.0 in C# zusammen mit der .NET-Plattform implementiert. Da der Controller keinen eigenen Modbus-Server hostet sondern die Klasse `ModbusClient` nutzt, kann er ohne die Anwendung „Lagerverwaltung 3.0“ nicht benutzt werden.

In Abb.3.5 sieht man, dass das Programm aus zwei wesentlichen Klassen neben dem `MainWindow` besteht. `ModbusClient` implementiert den Modbus Clienten zur Kommunikation mit den Modbus Servern, die in 3.1.3 beschrieben sind. Die Klasse implementiert auch das in [4] beschriebene Agentensystem. `ModbusInfo` dagegen hält die Daten, die zur Erzeugung der Kommandos benötigt werden.

Mit dem Konstruktor der Klasse `MainWindow` werden die Klassen `ModbusClient` und `ModbusInfo` instanziiert.

### 3.2.2. GUI

Wie in 3.6 ersichtlich, ist das Fenster einfach strukturiert und übersichtlich. Es ist in vier Bereiche unterteilt, die durch eine grau umrandete Box und eine Überschrift abgegrenzt sind.

#### Modbus Einstellungen

Dieser Bereich ist ganz oben und durch die Überschrift „Modbus Configuration“ gekennzeichnet. In zwei Textfeldern können Modbus-IP und -Port eingegeben werden.

Rechts daneben befinden sich untereinander zum Einen ein Label, dass den Verbindungsstatus anzeigt und zum Anderen ein Knopf mit dem die Verbindung hergestellt und beendet werden kann.

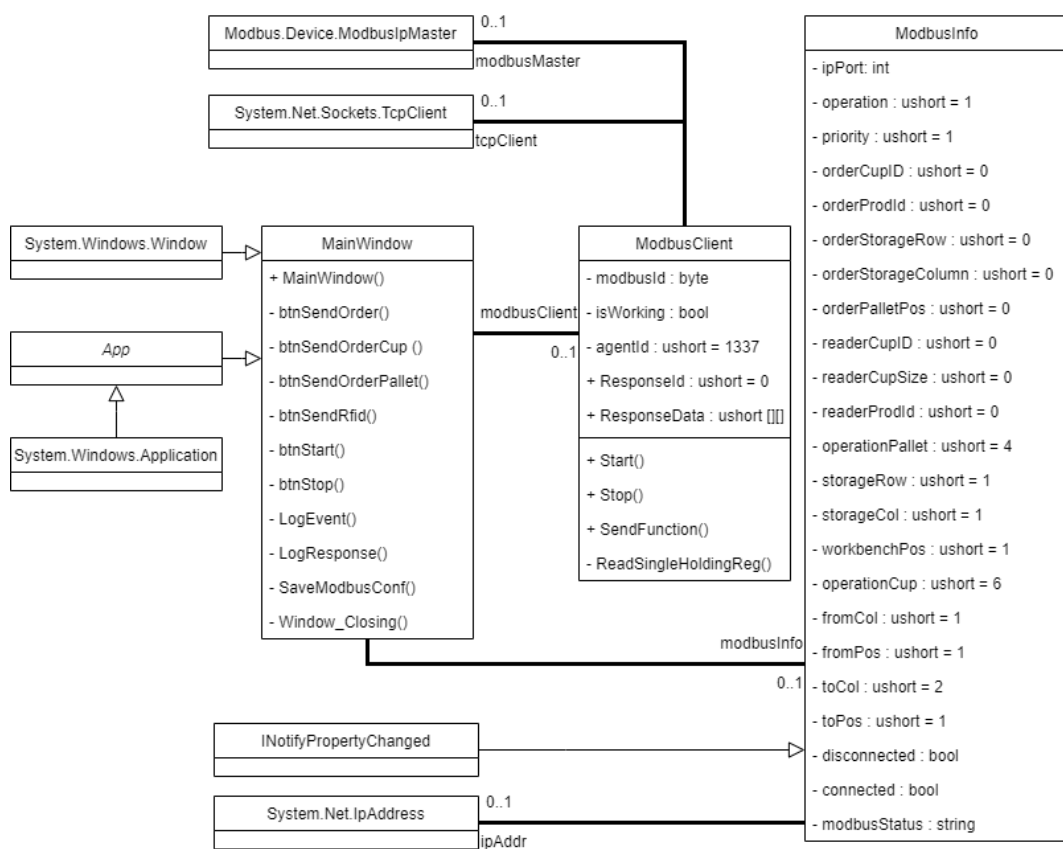
Nach dem Programmstart ist die Beschriftung der Taste „Start“.

Mit Klick auf die Taste wird die Verbindung hergestellt und das Label darüber wechselt zu „Connecting...“. Ist die Verbindung erfolgreich hergestellt, wechselt das Label auf „Connected“ und die Taste wird mit „Stop“ beschriftet.

#### Kommando erzeugen

Direkt unter dem Modbus Einstellungsbereich befindet sich der Bereich „Order“. Der Bereich ist durch ein Reitermenü ausgefüllt, dessen Reiter die Überschriften „Basic“,

**Abbildung 3.5.:** Klassendiagramm des Controllers mit allen vererbenden Klassen, jedoch ohne Klassen die lediglich Datentypen konvertieren.



**Abbildung 3.6.:** Ansicht des Controller Startbildschirms.

Warehouse Controller

Modbus Configuration

IP Address

127.0.0.1

IP Port

502

Disconnected

Start

Order

Basic

Pallet

Cup

Operation

Robot → Storage

Request Type

Prepare

Cup ID

0

Product ID

0

Storage Position (optional)

Any

Any

Cup Position (optional)

Any

Note: Cup ID or Product ID must be set.

Send

RFID Server

Cup ID

0

Cup Size (optional)

Any

Product ID (optional)

0

Send

Event Log

Error connecting to modbus server.



---

„Pallet“ und „Cup“ tragen.

Im Reiter „Basic“ befinden sich sieben Auswahlfelder und eine Taste.

Mit dem Feld „Operation“ kann über ein Drop-Down Menü ausgewählt werden, ob ein Becher oder Produkt vom Roboter ins Lager transportiert werden soll oder umgekehrt.

Mit dem Feld „Request Type“ kann über ein Drop-Down Menü ausgewählt werden, ob der Transport nur Vorbereitet („Prepare“) oder auch ausgeführt („Execute“) werden soll.

Darunter befinden sich Eingabefelder über die die relevante ID (Becher oder Produkt) eingegeben werden kann.

Optional kann darunter über ein Drop-Down Menü die Lagerposition des Lagers oder Bechers ausgewählt werden.

Mit der Taste „Send“ wird das Kommando an die Steuerung gesendet.

Im Reiter „Pallet“ kann analog der Transport einer Palette angefordert werden.

Im Reiter „Cup“ kann analog der Transport eines Bechers angefordert werden.

### **RFID Server Tool**

Mit dem RFID Server Tool, dargestellt unter dem Reitermenü, können Speicherblöcke eines RFID-Tags mit der Becher-ID und/oder Produkt-ID beschrieben werden. Voraussetzung dafür ist, dass der mobile Roboter mit Becher in der Andockstation des steht.

Die IDs werden in einem Eingabefeld als Zahl eingegeben. Optional kann die Bechergröße ausgewählt werden.

Mit der Taste „Send“ werden die Daten auf das RFID-Tag geschrieben.

### **Eventlog**

Dies ist ein Textbereich in dem die angeforderten Kommandos aufgelistet werden und eine Mitteilung wenn sie erfolgreich ausgeführt wurden.

---

### 3.3. RFID Server

Dieses kleine Programm implementiert einen Modbus TCP/IP Client wie in 3.1.3 beschrieben. Dieser wird genutzt um die gelesenen RFID-Tags für alle Stationen der  $\mu$ Plant verfügbar zu machen.

In jeder Station der  $\mu$ Plant gibt es einen RFID Leser der Fa. Feig.

Stationen in der  $\mu$ Plant können einen Tag am Becher beschreiben wenn sie das Produkt im Becher manipulieren oder die Tags auslesen. Anschließend können Stationen die einen Becher erhalten, den angekündigten Becher und das Produkt beim Eintreffen in der Station verifizieren. Die Verfahren dazu sind in [4] gut beschrieben.

Die Daten der Tags sollen beim Schreiben und Lesen auf die Modbus Adressen des Modbus Servers geschrieben werden.

Die Auftragsverwaltung der  $\mu$ Plant kann die Daten der Tags dann vom Modbus Server abrufen. Die RFID-Lesegeräte werden über TCP/IP mittels des SDK der Fa. Feig angesprochen.

#### 3.3.1. Klassenstruktur des RFID Servers

Die Programmstruktur ist anhand des Klassendiagramms in 3.7 erklärt. Eine Klasse `MainWindow` hält eine Instanz der Klasse `RFIDProducerCollection`.

Diese implementiert die oben beschriebene Liste welche Objekte der Klasse `RFIDProducer` speichert.

Ein `RFIDProducer` Objekt enthält ein Objekt der Klasse `ModbusClient` für die Kommunikation über TCP/IP und ein Objekt der Klasse `TagReader` für die RFID Kommunikation.

#### 3.3.2. GUI

Das GUI des RFID-Servers ist einfach gehalten und ist in Abb.3.8 dargestellt. Das Hauptelement ist eine Liste. Jedes Listenelement enthält Daten zu einem RFID Modbus Server und einem Tag mit seinen Daten. Mit der Taste „Add New“ können neue, leere, Listenelemente erzeugt werden.

Standardmäßig sind die Eingabefelder deaktiviert, können jedoch mit der Taste „Modify“ zum Bearbeiten freigeschaltet werden.

Mit der Taste „Start“ wird die Verbindung hergestellt. Ein Feedback, ob die Verbindung erfolgreich hergestellt werden konnte ist zunächst nicht ersichtlich.

Jedes Listenelement enthält eine Checkbox mit der es ausgewählt werden kann.

Mit den Tasten „Select All“ und „Select None“ können alle Listenelemente an- oder abgewählt werden.

Im unteren, rechten Bereich können mit den Tasten „Start Selected“ und „Stop Selected“ mehrere Server gleichzeitig gestartet und gestoppt werden.

Mit der Taste „Remove Selected“ können alle ausgewählten Listenelemente gelöscht werden.

**Abbildung 3.7.:** Klassendiagramm des Programms RFID Server mit allen vererbenden Klassen jedoch ohne solche die lediglich Typen konvertieren.

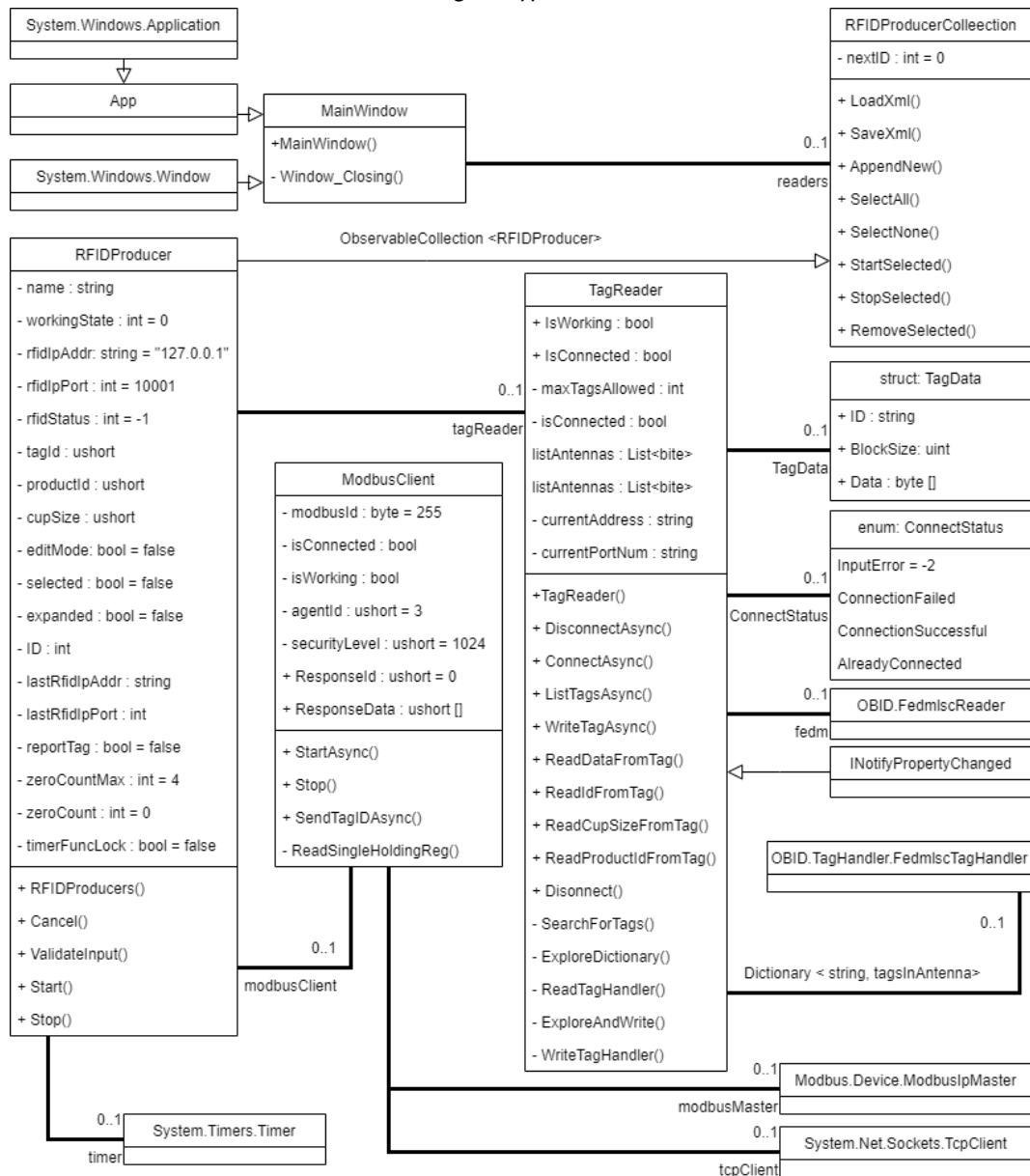





Abbildung 3.8.: Startbildschirm des Programms RFID-Server

RFID Server

Add New

^

☐ LZ

Tag Info 0 . 0 . 0   

**Name**

**Tag Reader**

IP Address

IP Port

**End Point**

IP Address

IP Port

Modbus Address

Start

Modify

▼

☐ FZ links (entleer)

Tag Info 0 . 0 . 0   

▼

☐ FZ rechts (befüll)

Tag Info 0 . 0 . 0   

▼

☐ AuE1

Tag Info 0 . 0 . 0   

▼

☐ AuE2

Tag Info 0 . 0 . 0   

Select All

Select None

Remove Selected

Start Selected

Stop Selected

24

# 4 Funktions- und Anforderungsanalyse

---

In diesem Kapitel werden auf Grundlage der vorangegangenen Kapitel die notwendigen Funktionen und ihre Anforderungen ermittelt. Der daraus entstehende Funktions- und Anforderungsumfang dient als Grundlage für das Kapitel 5.

## 4.1. Lagerverwaltung 3.0

Dieses Programm hat ein Anwendungsfenster welches dem Benutzer einen umfassenden Überblick über die LZ gibt.

Das Fenster hat eine Mindest- und Maximalgröße, ist dazwischen aber beliebig skalierbar.

Dialoge werden genutzt, wenn Einträge gelöscht werden, Verbindungen nicht aufgebaut werden können und der Simulationsbetrieb gestartet werden kann.

Felder sind standardmäßig deaktiviert und müssen über Checkboxes aktiviert werden.

Dateneingaben erfolgen über Dropdown Menüs wenn eine Auswahl festgelegter Werte möglich ist. Ansonsten werden Eingabefelder benutzt.

### 4.1.1. Bedienfunktionen

Der Benutzer kann über das GUI folgende Eingriffe vornehmen:

- Die Verbindung zu einem Modbus Server konfigurieren, starten und beenden.
- Die Verbindung zur ABB IRC5 Steuerung konfigurieren, speichern, starten und beenden.
- In der Produktliste kann gescrollt werden.
- In der Inventarliste kann gescrollt werden und es können leere Produkte ein- und ausgeblendet werden.
- Die vergangenen Einträge im Eventlogger können gelöscht werden. Im Eventlogger kann gescrollt werden.
- In der Lagervisualisierung können an den einzelnen Lagerorten
  - Paletten hinzugefügt oder entfernt werden
  - Becher hinzugefügt oder entfernt werden
  - Produkt ID und Cup ID der Becher können angezeigt und konfiguriert werden
  - Durch Klick auf einen Becher werden alle Becher gleichen Produkts in der Lagervisualisierung und auch im Inventar blau hervorgehoben.

- 
- Palette, Becher und Produkt auf dem Kommissioniertisch können konfiguriert werden
  - Becher und Produkt auf dem mobilen Roboter können konfiguriert werden.
  - Der Automatikbetrieb kann gestartet werden
  - Wenn die Verbindung zum Modbus Server nicht hergestellt wurde kann ein Simulationsbetrieb gestartet werden
  - Im Simulationsbetrieb kann zudem
    - Die Anwesenheit eines mobilen Roboters Simuliert werden
    - ausgewählt werden, dass der Becher auf dem mobilen Roboter nicht transparent ist (Funktion unklar)

#### **4.1.2. Informationsdarstellung**

Der Benutzer wird über folgende Inhalte informiert:

- Die Verbindungseinstellungen des Modbus Servers und den Verbindungsstatus
- Die Verbindungseinstellungen des ABB-Controllers und ihren Verbindungsstatus
- Alle möglichen Produkt ID's und die zugehörigen Produktnamen
- Produkte und ihre Lagermengen im Inventarbereich.
- In der Lagervisualisierung:
  - Symbolisierung einer Palette durch dunkelgraues Rechteck, wenn eine Palette vorhanden ist
  - Symbolisierung der Becher durch weißes Rechteck, wenn ein Becher vorhanden ist
  - Produktnamen und auf Wunsch auch Produkt ID und Becher ID
- Im Eventlogger werden folgende Informationen bereit gestellt:
  - Fehler und Events der Verbindung über Modbus TCP/IP
  - Fehler und Events der Verbindung zur Steuerung IRC5 des Industrieroboters
  - Programmfortschritte und Events im Automatikbetrieb oder der Simulation

#### **4.2. Controller**

Der Benutzer kann folgende Bedienvorgänge durchführen:

- Die Verbindungseinstellungen des Modbus Servers konfigurieren und starten
- Transportbefehle können erzeugt werden:
  - Transportbefehle eines einzelnen Bechers zwischen Lager, Kommissioniertisch und Roboter
  - Transportbefehle für einer Palette zwischen Lager und Kommissioniertisch

---

Die beiden Befehle können einerseits direkt ausgeführt werden, andererseits können sie auch als Eingabe für die Lagerverwaltungssoftware benutzt werden.

- Ein Becher in der Andockstation kann mittels einem RFID- Lesegerät beschrieben werden.

Der Benutzer wird in diesem Programm nur über seine Eingaben informiert. Die Eingaben erfolgen wo möglich über ein Dropdown Menü

### 4.3. RFID Server

Der Benutzer kann folgende Bedienvorgänge durchführen:

- Einen Neuen Listeneintrag erzeugen
- Einen oder alle Listeneintrag markieren
- Markierte Listeneinträge abwählen
- ausgewählte Listeneinträge Löschen, Verbindung starten oder stoppen
- Je Listeneintrag kann der Benutzer folgende Einstellungen vornehmen:
  - IP und Port des Tag Readers
  - IP, Port und Modbus Adresse des Endpoints
- Ein einzelner Listeneintrag kann gestartet werden
- Mit der Taste „Modify“ kann ein Listeneintrag zum Bearbeiten freigeschaltet werden
- Mit der Taste „Save“ können Änderungen gespeichert werden.

Der Benutzer sieht in diesem Programme eine Liste aller eingetragenen RFID Reader, Endpoints und Modbus Adressen. Jeder Listeneintrag kann auf eine Zeile minimiert werden die lediglich den Namen des Endpunkts und die gelesenen Tag Daten anzeigt. Im ausgeklappten Zustand sind die Konfigurationen sichtbar, aber grau hinterlegt um anzuzeigen, dass die Bearbeitung gesperrt ist.





# 5 Konzeptionierung einer integrierten Python Anwendung

---

Die Basisanforderungen einer integrierten LZ-Verwaltung lassen sich aus den Kapiteln 3 und 4 ableiten.

Hinzu kommen die später in Kapitel 5 beschriebenen Funktionen.

Neben den Kriterien Modularität, Erweiterbarkeit und Wartbarkeit ist darüber hinaus wichtig, dass auch Personen, die bei der ursprünglichen Entwicklung der Software nicht involviert waren, die Software warten und weiterentwickeln können.

Eine saubere Struktur und eine gute Dokumentation der Software ist daher unerlässlich. Eine objektorientierte Programmierung bietet sich an, da andere Teile der  $\mu$ Plant dieses Programmierparadigma nutzen.

Das Programm „Lagerverwaltung 3.0“ nutzte eine GUI-Klasse als Datenhub und verfügt ansonsten über kein zentrales Datenmodell. Dadurch entstand ein unübersichtliches Klassenkonstrukt, das es schwer macht den Datenfluss zu ermitteln.

Zum Beispiel wird anhand der Klassendiagramme deutlich, dass Daten der Modbus Adressen in die `commissionMatrix` geschrieben werden müssen, weil sie ausschließlich dort zu RAPID Kommandos für den Industrieroboter übersetzt werden. Wie genau das passiert, lässt sich jedoch anhand des C# Codes nicht nachvollziehen.

## 5.1. Datenmodellierung

Um den Anforderungen gerecht zu werden, empfiehlt es sich auf übliche Design Patterns der Softwareentwicklung zu setzen. Das MVC Konzept sieht vor, dass Daten (Model) und GUI (View) keinerlei Zugriff aufeinander erlauben. Die Schnittstelle zwischen den Daten und dem Benutzer ist ein Controller, der die Programmlogik implementiert. Für die Daten gilt, dass sie in einer Objektstruktur modelliert werden und nicht von außerhalb dieser Objekte verändert werden.

Soll das Datenmodell geändert werden, muss dafür eine öffentliche Methode in dem Modell selbst existieren, die alle möglichen Fehler behandelt und referenzielle Integrität herstellt, sodass das Datenmodell nach jeder berechtigten Änderung konsistent bleibt und unberechtigte Änderungen nicht zulässt.

Die drei Klassen in Abb. 5.1 besitzen private Attribute, was durch das „-“ angedeutet ist.

Dafür besitzen sie öffentliche Methoden ( durch das „+“ gekennzeichnet), die einerseits die Attribute zurückgeben (get-Methoden) oder einen zu verändernden Wert übernehmen (set - Methoden).

---

Werden Listen verwendet, können Objekte mit den `with-` und `without-` Methoden hinzugefügt oder entfernt werden.

Ein Objekt der Klasse `Pallet` kann sich also nicht direkt selbst in das entsprechende Feld eines Objekts der Klasse `Cup` eintragen, sondern muss dazu die entsprechende Methode `SetPallet()` mit sich selbst als Argument aufrufen.

Diese Methode muss mindestens die folgenden Kriterien erfüllen um referenzielle Integrität herzustellen:

1. Die übergebenen Parameter müssen gültig sein.
2. Das zu schreibende Attribut darf nicht schon ein anderes Objekt enthalten.
3. Die Änderungen müssen allen betroffenen Klassen mitgeteilt werden. Eventuelle Fehler müssen behandelt werden.

Wird z.B. einem Objekt der Klasse `Cup` je ein Objekt der Klasse `Pallet` und `Product` übergeben, dann muss am Ende der Änderung das `Cup` Objekt in der Liste `cups` der Klasse `Product` stehen.

Das Objekt der Klasse `Pallet` muss ebenfalls als Feldwert des Attributs `cup` das Objekt der `Cup` Klasse haben.

In Python gibt es nicht die Möglichkeit Attribute und Methoden von Klassen zu schützen bzw. zu verstecken wie z.B. in `C#`. Es ist also jederzeit möglich, dass ein Softwareentwickler die referenzielle Integrität außer Kraft setzt.

Für Methoden wird die übliche Konvention verwendet, die festlegt, dass

- Methoden, die mit einem „`_`“ beginnen, als `private` gekennzeichnet sind.
- Methoden, die mit einem „`__`“ beginnen, als `protected` gekennzeichnet sind.
- Methoden, die weder mit einem „`_`“ noch mit einem „`__`“ beginnen, als `public` gekennzeichnet sind.

Ferner wird für dieses Projekt festgelegt, dass alle Attribute in Klassen des Datenmodells nur über entsprechende `get-` und `set-` Methoden gelesen und geschrieben werden dürfen. Die Ordnerstruktur des Projekts muss so angelegt sein, dass Datenklassen eindeutig erkenntlich sind.

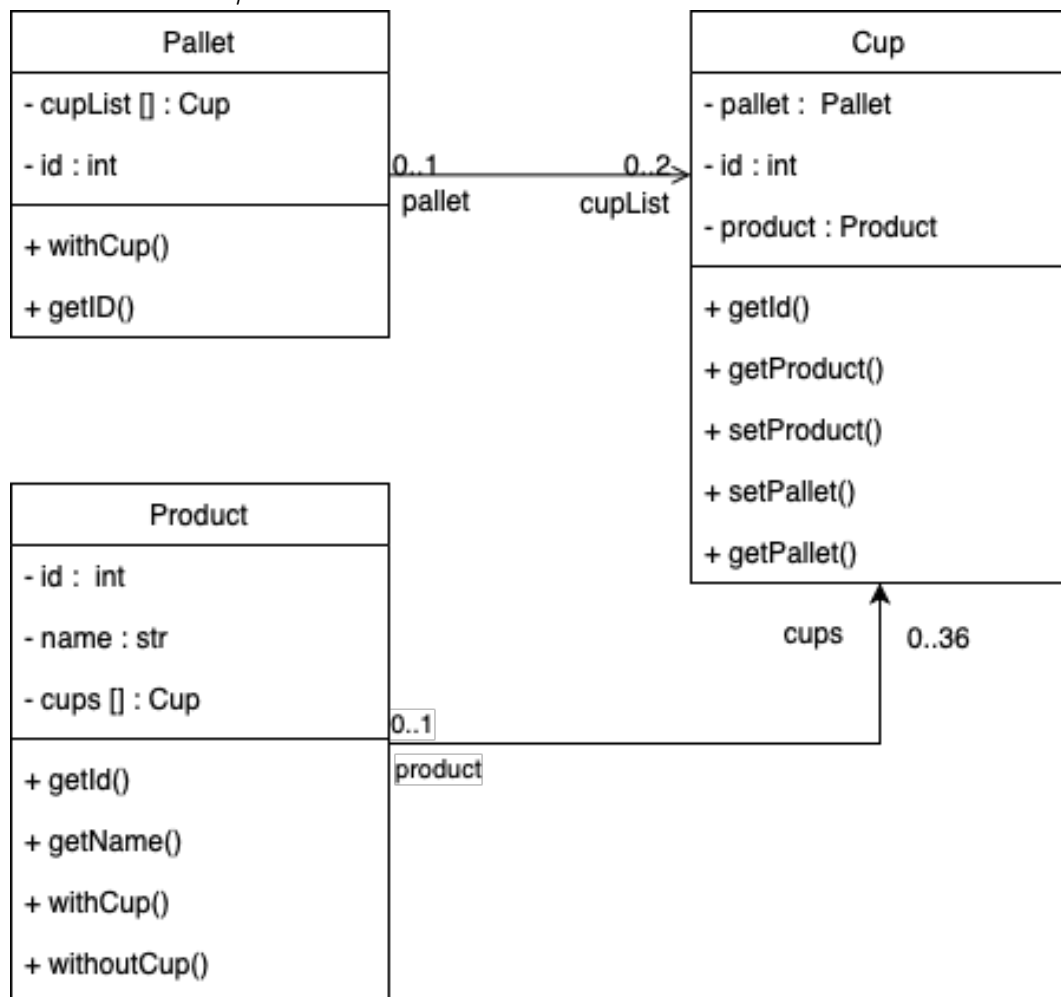
In die Klassen der Datenmodelle sollen nur Klassen gespeichert werden, die dynamisch sind. D.h. sie werden zur Laufzeit geändert und/oder erzeugt bzw. gelöscht. Für die Modellierung der Daten wird das `MVVC` Konzept angewandt und um Serviceklassen erweitert.

Bei der Anlieferung eines Bechers wird dieser in eine Palette gestellt und anschließend eingelagert. Ein Objektdiagramm dieser Situation ist in [Abb.5.2](#) dargestellt.

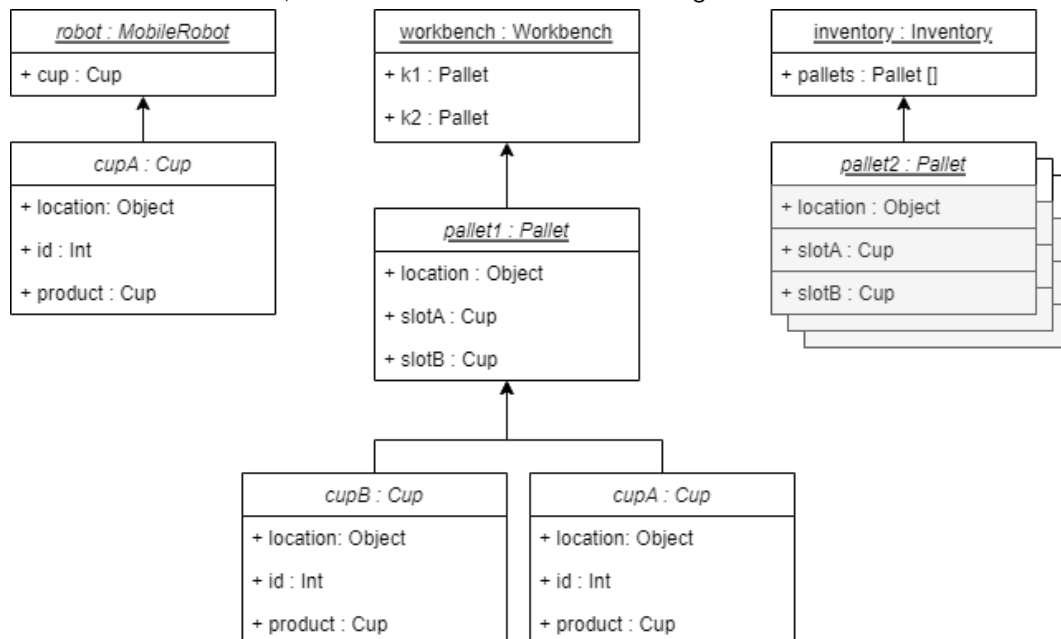
Eine Palette kann keinen, einen oder zwei Becher beinhalten. Die Becher in einer Liste in dem Paletten-Objekt zu speichern hat den Nachteil, dass man die Becher in der Palette als Listeneinträge verarbeiten muss und der Slot als weitere Listenspalte geführt wird.

Die Becher als zwei separate Attribute des Paletten-Objekts zu führen hat den Nachteil, dass zwei Orte abgefragt werden müssen. Andererseits ist die Zuordnung des Bechers

**Abbildung 5.1.:** Referenzielle Integrität einer Palette / Becher / Produkt - Kombination wie sie in der  $\mu$ Plant auftreten könnte.



**Abbildung 5.2.:** Die Abbildung zeigt ein Objektdiagramm aus Ablageorten für Paletten und Becher. Es ist ebenfalls ersichtlich, dass keine Verbindungslinien zwischen dem mobilen Roboter, dem Kommissioniertisch und dem Lager existieren.



zum Slot dadurch inhärent. Daher wird dieser Ansatz gewählt.

Abb. 5.2 zeigt auch, dass jedes Objekt der Klasse Palette und Cup ein Feld `location` braucht. Hier wird das Objekt gespeichert in dem sich die Palette oder Becher gerade befindet.

Dies ist erforderlich um referenzielle Integrität zu gewährleisten und erleichtert die Implementierung der Controller, wie eingangs in Kapitel 5 beschrieben.

Aus dem Diagramm wird auch deutlich, dass zwischen den Objekten der Klassen `MobileRobot`, `WorkBench` und `Inventory` keinerlei Assoziationen gibt.

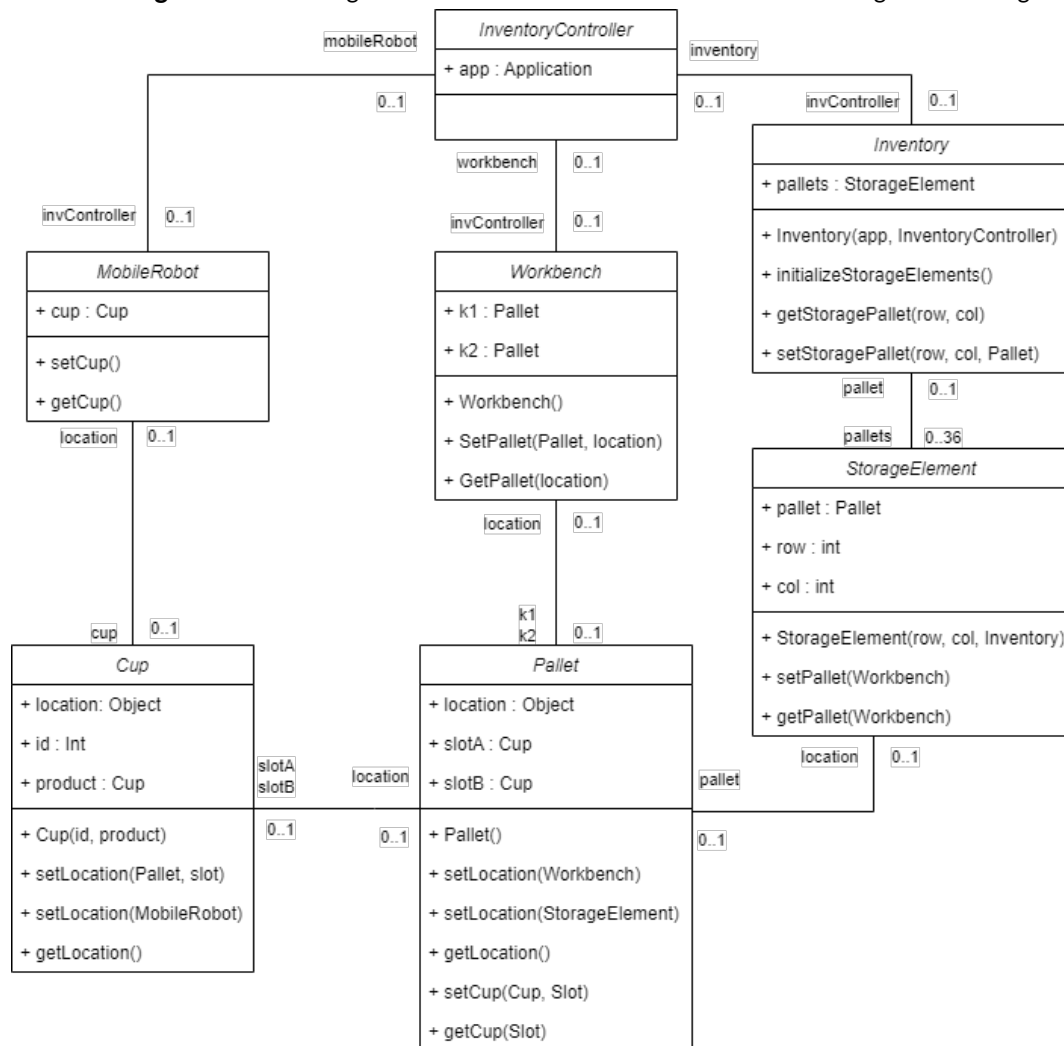
Es wird also ein weiteres Objekt, z.B. ein `InventoryController`, gebraucht um aktiv Objekte von einem Lagerort zu einem Anderen zu schreiben. Immer wenn ein Service Zugriff auf das Datenmodell braucht, wird der entsprechende Controller diese Zugriffe verarbeiten. Als Ergebnis dieser Überlegungen leitet sich ein (Daten-)Klassendiagramm ab (Siehe Abb.5.3).

### 5.1.1. Datenklassen

In den Datenklassen sollen alle realen Objekte abgebildet werden, die durch das Lager bewegt sind oder daran aktiv/passiv beteiligt sind. Das betrifft:

1. Becher.
2. Paletten, weil sie Becher beinhalten können und mit oder ohne Becher bewegt werden.
3. Greifer, weil er die Becher und Paletten bewegt. Ein Ausfall der Lagerzelle kann bewirken, dass ein Becher / Palette auf dem Greifer verbleibt.

**Abbildung 5.3.:** Klassendiagramm für die Datenstruktur der Software der Lagerverwaltung.



- 
4. Kommissioniertisch, weil er Ziel von Transportbewegungen ist.
  5. mobile Roboter, weil er Quelle und Ziel von Transportbewegungen ist.

### 5.1.2. Konstante Daten

Daten, die über die im Rahmen der Entwicklung festgelegt werden und sich üblicherweise nicht Ändern, sollen in einer eigenen Klasse gespeichert werden und `constants` heißen. Dort soll gespeichert werden:

- Dateipfade zu allen Dateien, die das Programm verwendet oder erzeugt. Dies können z.B. Konfigurationsdateien, Logdateien oder Dateien zum Abspeichern von Daten sein.
- Strategische Werte, die in der Entwicklung festgelegt und mehrfach verwendet werden.

### 5.1.3. Programmeinstellungen

Programm-Einstellungen sollen in einer eigenen Klasse `preferences` gespeichert werden. Die Einstellungen sollen in einer Datei gespeichert werden, sobald der Benutzer eine Änderung speichert.

## 5.2. Konzepte für Controller- und Serviceklassen

Die Kommunikation über Modbus wird durch OPC UA ersetzt. Für die Kommunikation über diese Schnittstelle wird ein `OpcUaService` implementiert.

Ein Service übernimmt die Kommunikation mit den RFID-Lesegeräten in der  $\mu$ Plant und stellt die ausgelesenen Daten über den `OpcUaService` den anderen Stationen zur Verfügung.

Das in [4] eingeführte Agentensystem wird ebenfalls als eigener Service implementiert. Da die LZ nicht mit anderen Stationen kommunizieren muss, reicht die Implementierung des Clients. Dabei wird die Kommunikation über OPC UA realisiert.

Controller werden gebraucht um Schnittstellen zwischen Service und Daten oder zwischen Service und GUI zu implementieren. Eine besondere Rolle nimmt der `InventoryController` ein.

Er muss zum Start des Programms das Inventar aus Dateien laden und damit das Datenmodell initialisieren. Aus Gründen die in Kapitel 6 erörtert werden, sollte der Controller nach jeder vollständigen Änderung des Datenmodells die Änderung einerseits in der Datei sichern.

Eine Liste aller benötigter Service Klassen ist in Tab. 5.1 aufgelistet. Die Controller finden sich in Tab. 5.2.

---

**Tabelle 5.1.:** Benötigte Serviceklassen

Klassenname	Beschreibung
Modbus Service	Implementiert die Modbus Kommunikaion, sendet Modbus informationen an entsprechende Controller
OPC UA Service	Hält OPC UA Server und Client, organisiert Kommunikation zu anderen OPC UA entities
RFID Service	Erledigt alles was mit RFID zu tun hat

**Tabelle 5.2.:** Werte der Klasse ModbusBaseAddress

Klassenname	Beschreibung
InventoryController	Führt jede Operation bezüglich des Datenmodells aus
EventlogController	Erzeugt Meldungen für den Eventlog

### 5.3. GUI Konzeptionierung

Neben der Datenmodellierung und der Programmlogik ist die GUI ein wichtiger Bestandteil der Software.

#### 5.3.1. PySide6 und QuickQml 2.0

Laut der Qt Wiki Website [5] wurde das Qt Framework geboren, als ihre Schöpfer Haavard Nord und Eric Chambe-Eng im Sommer 1990 in Norwegen an einem GUI für eine Ultraschalldatenbank arbeiteten.

Die Software sollte damals in C++ implementiert auf Mac, Unix und Windows laufen. Fünf Jahre später veröffentlichten Sie das erste Qt Framework unter dem Firmennamen Troll Tech. Seitdem gewann das Framework immer mehr Popularität. Im Jahr 2006 übernahm Nokia die Firma Troll Tech und verkaufte das Qt Project in den Jahren 2011 und 2012 erst teilweise, dann vollständig an den Digia Konzern. Seit 2014 ist Qt als Tochterunternehmen des Digia Konzerns unter dem Namen „The Qt Company“ ein eigenständiges Unternehmen.

Das Qt Framework ist in C++ implementiert. Die neue Software für die  $\mu$ Plant soll jedoch in Python implementiert werden. Für diese Zwecke hat Qt u.A. das Framework PySide6 veröffentlicht, welches einen Wrapper für Python Projekte bietet.

GUI's können in PySide6 zwei Paradigmen folgend erstellt werden.

Eine Möglichkeit ist es, das GUI über Widgets[6] zu erstellen. Dabei werden GUI Elemente als eigene Klasse direkt im Python implementiert. Dies ist eine schnelle und unkomplizierte Art einfache GUIs zu erstellen.

Die zweite Möglichkeit ist QtQuick [7] zu nutzen. Neben der Instanz einer `QGuiApplication` wird eine `QQmlApplicationEngine` instanziiert. GUI Elemente werden in einer separaten QML-Datei erstellt und mit dieser geladen und gerendert. Dies bietet den Vorteil, dass für das GUI eine deklarative Sprache verwendet wird, die sich von der Programmlogik trennt. Die Bibliothek QtQuick 2.0 beinhaltet viele grundlegende QML-Datentypen, die beliebig zu GUIs kombiniert werden können.

Für die Umsetzung eines MVVC-Design Patterns empfiehlt sich die Verwendung von

---

QML. Durch die Verwendung des Frameworks wird die konsequente Trennung zwischen Interface und Datenmodell erzwungen. Veranschaulicht wird dies in Abb. 5.4.

Um die anwendungsspezifischen Datenmodelle (Model) in einem GUI (View) zu verwenden, muss das Datenmodell als Ressource der `QQmlApplicationEngine` zur Verfügung gestellt werden. Zu diesem Zweck werden Klassen implementiert, die von Qt-Klassen erben, die wiederum von `QtAbstractModel` abgeleitet sind. Die Klasse muss der Datenstruktur des Datenmodells entsprechen. In diesem Projekt sind `QAbstractListModel` für Listen und `QAbstractTableModel` für tabellenartige Datenstrukturen relevant.

In diesen Klassen müssen bestimmte Methoden überschrieben werden. Manche andere Methoden sind optional. Welche Methoden das sind ist in der Referenzdokumentation des Qt Frameworks nachzulesen.

Die Implementierungen der Methoden ist individuell und hängt von der Datenstruktur ab. Die Datentypen der Argumente und Rückgabewerte sind jedoch vorgegeben.

Wenn in einem GUI z.B. eine Liste sortiert oder gefiltert werden soll, dann muss zusätzlich ein sog. `QSortFilterProxyModel` implementiert werden.

Bevor die Instanz der `QQmlApplicationEngine` die erste QML-Datei lädt - also vor Programmstart - muss entweder die Instanz des `QSortFilterProxyModel` oder die Instanz des `QAbstractModel` als Ressource der `QQmlApplicationEngine` registriert werden. Diese Instanz ist dann das Viewmodel. Die Implementierten Methoden dienen dazu die Daten des Datenmodells beim Rendern zu verarbeiten.

Über das GUI können die Daten des Viewmodels geändert, gelöscht oder hinzugefügt werden - ohne zwangsweise das Datenmodell selbst zu ändern.

Wenn ein Viewmodel mit der `QQmlApplicationEngine` verknüpft ist, kann es über seine URI in jeder QML-Datei angesprochen werden.

In einer QML-Datei wird einem entsprechenden QML-Datentyp, z.B. `ListView` für eine einfache Liste, über das Property `model` die URI des Datenmodells zugewiesen. Dadurch kennt das `ListView` Objekt die Indices des Datenmodells und erhält beim Rendern der Liste nur die benötigten Daten.

Jede weitere Aktion, die durch den Benutzer auf ein Listenelement ausgelöst wird, bezieht sich auf ein Delegate des Datenmodells. Jede Änderung an dem Delegate wird zunächst gerendert und überprüft bevor das Datenmodell geändert wird. Diese Basisfunktion kommt mit dem Qt Framework an sich und muss nicht implementiert werden. Wie jedoch das Datenmodell mit den geänderten Daten umgeht oder ob die Änderung des Datenmodells automatisch ein Update der Daten auf dem Server oder der Datei auslöst, muss vom Entwickler implementiert werden!

Die Kommunikation des GUIs mit der Anwendung erfolgt über Signale und Slots. Signale sind Teil des Signal/Slot Prinzips des Qt Framework [8] und stellen die Funktion eines Events dar. Wird ein Signal an beliebiger Stelle im GUI emittiert, kann es an jeder anderen Stelle als Event genutzt werden. Dadurch muss man in der Regel keine zusätzlichen Callbacks oder Lambda-Ausdrücke benutzen.

Außerhalb des QML-Contexts, z.B. in einer Python Klasse, muss das Signal der Klasse bekannt sein, indem das Signal an die Klasse gebunden wird.

Eine Funktion, die mit der Annotation „`@Slot()`“ versehen ist, wird als Slot behandelt und kann mit dem Signal verknüpft werden, sodass diese bei Auftreten des Signals ausgeführt



---

werden.

Ist eine Methode einer Klasse ein Slot und ist diese Klasse als Ressource der `QQmlApplicationEngine` registriert, dann kann im QML-Code der Slot der Klasse auch direkt aufgerufen werden.

An Signale und Slots können Daten übergeben werden. Bei der Deklaration eines Signals oder Slots müssen als Argumente für jedes übergebene Datum der Datentyp angegeben werden. z.B.:

`@Slot(str, int)`

Anzahl und Datentyp der Argumente müssen bei deklarierten Signalen und Slots jederzeit übereinstimmen. Dass Datentypen in Python dynamisch sind kann unter Umständen zum Problem werden. Der Datentyp `None` ist im QML-Kontext nicht bekannt und führt zu einer Fehlermeldung. Es kann also erforderlich sein die zu übergebenen Daten auf die Datentypen zu casten oder in sinnvolle Werte umzuwandeln.

In meiner Vorbereitung auf diese Arbeit hat sich eine intuitive Vorgehensweise entwickelt, die ich für die Implementierung der Software empfehlen möchte:

Beim Initialisieren des Programms werden alle Datenmodelle (`QAbstractModel` und abgeleitete Klassen), Controller und Serviceklassen instanziiert und als Root-context der `QQmlApplicationEngine` gesetzt. Siehe dazu das Beispiel [5.3.1](#). Damit stehen sie dem Programmierer in jeder QML-Datei der Anwendung zur Verfügung.

In einer QML-Datei können die Kontexte der `QQmlEngine` nun unter ihrer URI referenziert werden. Dies ist in [Beispiel 5.3.1](#) gezeigt:

In dem QML-Datentyp „`ListView`“ wird dem Property `model` die URI des `ListModels` aus [5.3.1](#) zugewiesen. Im weiteren Verlauf des Codes findet sich ein `MouseArea` QML-Datentyp.

Wird in dem Bereich ein Klick mit linker Maustaste durchgeführt, wird das Signal `onClicked` emittiert. Innerhalb des Codes im Funktionskörper der `MouseArea` ist definiert, dass nach einem Mausklick die Funktion `selectRow(message)` aufgerufen wird. `message` ist dabei der übergebene Parameter.

Das Beispiel braucht dabei keinerlei Importe anderer Klassen, was daran liegt, dass sowohl das Datenmodell als auch das Objekt der Controller Klasse als Ressource der `QQmlEngine` registriert wurden.

Im weiteren Verlauf des Codes wird ein QML-Datentyp `Connections` dazu benutzt, das Signal des Controllers `onRowClicked` mit einer Funktion zu verbinden. Die Funktion wird innerhalb des `Connections`-Datentyps in JavaScript implementiert.

Gemäß der Namenskonvention ist das Signal im Controller selbst als `RowClicked` deklariert! Innerhalb der QML-Datei werden die Signale dann mit dem Präfix „on“ erfasst.

Der Signalname wird als Name einer JavaScript-Funktion verwendet, die mit `function` gekennzeichnet ist. Der Code innerhalb des Funktionskörpers beschreibt dann die Funktionslogik im JavaScript-Syntax. Im Fall des Beispiels wird ein bool'sches Property umgeschaltet, wenn die `id` des Datenmodell-Delegates mit der `message` des Signals übereinstimmt.

---

**Listing 5.1:** Beispiel einer einfachen App mittels PySide6. Zunächst wird eine Instanz der Application-Klasse und der QmlEngine erzeugt. Danach werden Objekte eines Datenmodells und eines Controllers erzeugt und als rootContext der QmlEngine registriert. Anschließend wird die QML-Datei des Hauptfensters geladen, was die App startet.

```
'''Create Basic Application Class and QMLEngine'''
app = QtGuiApplication(sys.argv)
engine = QQmlApplicationEngine()

# Create simple ProductListModel from path to database file
listModel = ProductListModel(getProducts(PRODUCTLIST))
engine.rootContext().setContextProperty("listModel", listModel)

# create InventoryController instance
listController = ProductListController(listModel)
engine.rootContext().setContextProperty("listCon", listController)

# define load main.qml file to start application
qml_file = "../src/qml/main.qml"
engine.load(qml_file)

if not engine.rootObjects():
    sys.exit(-1)

sys.exit(app.exec())
```

**Listing 5.2:** Beispiel einer QML-Datei Diese QML-Datei erzeugt ListView QML-Datentyp, der zum Anzeigen der Daten in einem ListModel verwendet wird. Innerhalb eines Rechtecks mit farbigem Rand werden in einem RowLayout Type die Datensätze des Datenmodells gerendert und über die Funktionslogik von Signalen der QML-Datentyps und der Controllerklasse farblich markiert.

```
ListView {
    id: inventoryList
    model: listModel
    anchors.fill: parent
    anchors.margins: 10
    clip: true
    spacing: 5
    Layout.fillWidth: true
    delegate: Rectangle {
        id: rect1
        width: ListView.view.width
        height: 50
        property bool selected: false
        color: selected ? "#4FC3F7": "white"

        RowLayout {
            id: row
            anchors.fill: parent
            Text {
                id: id
                text: model.id
                font.pixelSize: 20
                verticalAlignment: Text.AlignVCenter
                Layout.fillHeight: true
                Layout.fillWidth: true
                Layout.preferredWidth: 50
            }
        }
    }
}
```

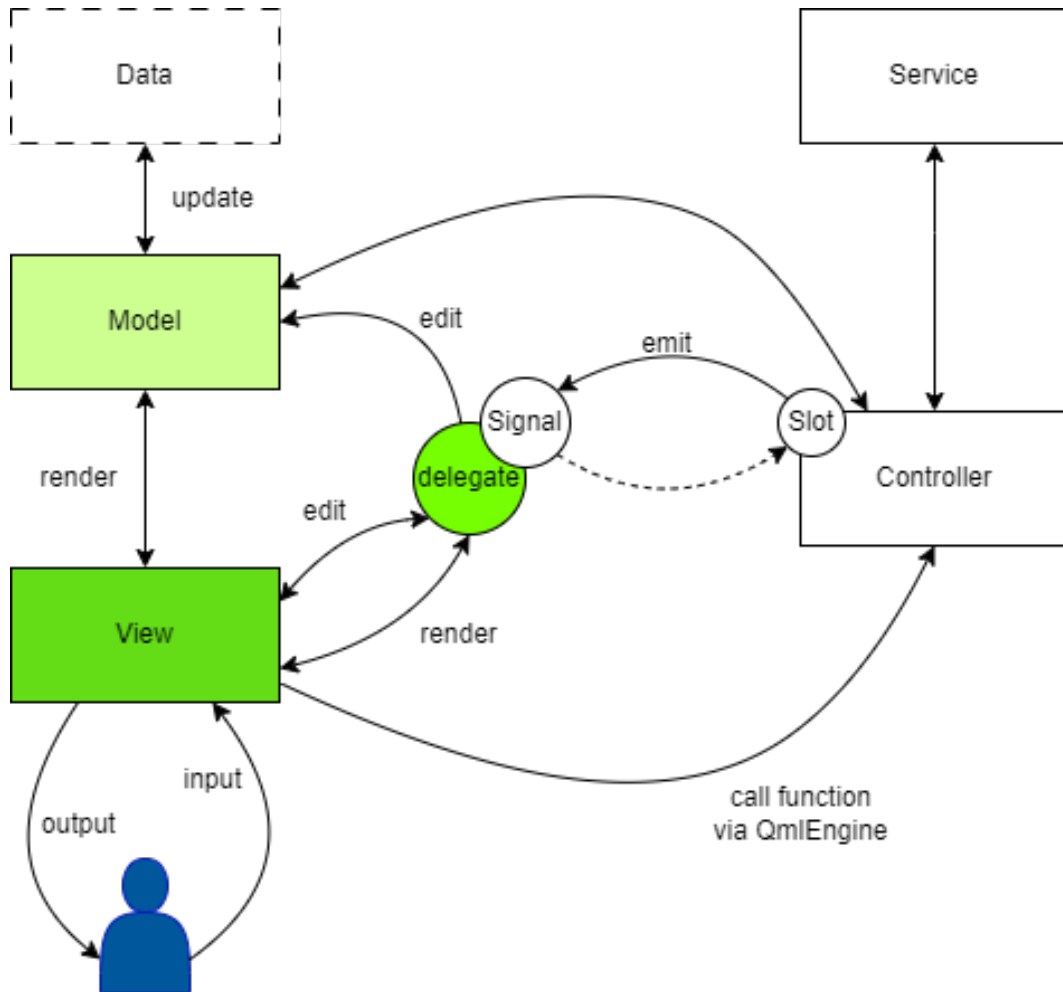
---

```

    }
    Text {
        id: name
        text: model.name
        font.pixelSize: 20
        verticalAlignment: Text.AlignVCenter
        Layout.fillHeight: true
        Layout.fillWidth: true
        Layout.preferredWidth: 400
    }
    Text {
        id: quantity
        text: model.quantity
        font.pixelSize: 20
        verticalAlignment: Text.AlignVCenter
        Layout.fillHeight: true
        Layout.fillWidth: true
        Layout.preferredWidth: 100
    }
}
MouseArea {
    anchors.fill: parent
    onClicked: {
        if(!rect1.selected) {
            listCon.selectRow(model.id)
            rect1.selected= true
        }
    }
}
Connections {
    target: listCon
    function onRowClicked(message) {
        if (model.id !== message) {
            rect1.selected = false
        }
        if(parseInt(model.id) === parseInt(message)) {
            rect1.selected = true
        }
    }
}
}
}
}
}
}

```

**Abbildung 5.4.:** Die Abbildung zeigt das in PySide6 verwendete Model-View Konzept, welches um einer Controllerklasse und einer Service Klasse erweitert wurde.



Aus den Beispielen 5.3.1 und 5.3.1 lässt sich eine Systematik erkennen: Die Daten sind hinter einem ViewModel geschützt und werden der View zur Verfügung gestellt.

Über die QML-Dateien erfolgt die GUI Modellierung und Events werden mit dem Signal/Slot Prinzip behandelt. Beliebige Ressourcen können als Teil der QmlEngine registriert werden, um Sie an anderer Stelle verfügbar zu machen.

Das künftige Programm soll jedoch auch Programmteile aufweisen, die nicht unbedingt mit den Daten oder dem GUI verknüpft sind.

Z.B. die Kommunikation über ModBus und zum ABB Industrieroboter oder das Übersetzen der Modbus Werte in RAPID Befehle. Diese Programmteile werden als Service Klassen bezeichnet. Wenn sich der Programmcode auf ein GUI auswirkt, wird ein Controller gebraucht, der dieses Verhalten steuert.

Die Funktionen eines Services in den Controller zu integrieren würde die Wartbarkeit und Erweiterbarkeit verschlechtern.

Nach der Systematik wie in Abb. 5.4 abgebildet, können beliebig viele Datenmodelle, GUI's, Controller und Services parallel existieren ohne sich gegenseitig zu beeinflussen.

---

Als Nachteil kann angeführt werden, dass sich mit zunehmendem Kontextregister der QmlEngine die Performance des Programms verschlechtern wird. Bei dem eher gering erwarteten Funktionsumfang der zu erstellenden Software wird dies jedoch untergeordnet behandelt und könnte durch das Aufteilen der Funktionen in Threads reduziert werden.

## 5.4. GUI - Modellierung

Die Hauptfunktion der Software ist das automatisierte Abarbeiten der Kommissionsaufträge, die bisher über Modbus TCP/IP von der Auftragsverwaltung der  $\mu$ Plant übergeben werden. Während der Automatikbetrieb läuft, wird ein Benutzer allenfalls einen Soll-Ist-Vergleich zwischen dem Stand der Software und der realen Lagerzelle durchführen. Die Idee ist daher, dass das neue GUI im Vergleich zum Alten aufgeräumter und übersichtlicher wird. Informationen die unmittelbar mit dem Prozess verbunden sind, sollen leicht ersichtlich und verfügbar sein. Alle anderen Informationen sollen nur bei Bedarf angezeigt werden.

In Abb. 5.5 zeigt meinen Entwurf zu dieser Idee:

Links ist die Andockstation des mobilen Roboters simuliert mit einem RFID Gerät darüber. Die eingelesenen Daten des RFID -Lesers könnten rechts daneben angezeigt werden, sobald der Benutzer mit der Maus darüber hovers. Wenn aktuell ein Tag gelesen wird, könnte das Symbol die Farbe wechseln.

Der Kommissioniertisch ist mit seinen beiden Plätzen „K1“ und „K2“ daneben symbolisiert.

Die Visualisierung des Lagers nimmt die gesamte rechte Bildschirmhälfte ein.

Die Produktliste, wie sie in 3.1 Bereich „D“ dargestellt ist, ist im GUI nicht mehr vorhanden. Muss der Bediener ein Produkt an irgendeiner Stelle des Programms überschreiben, so wird ihm nicht mehr die Produkt-ID angezeigt, sondern direkt der Produktname. Zusätzlich kann die Produktliste als Neues Fenster über die Toolbar eingeblendet werden.

Sämtliche Einstellungen wie IP und Port der Kommunikationsschnittstellen werden nur selten gebraucht. Über die Toolbar sind die Einstellungen über den QML-Datentyp `QDialog` einsehbar.

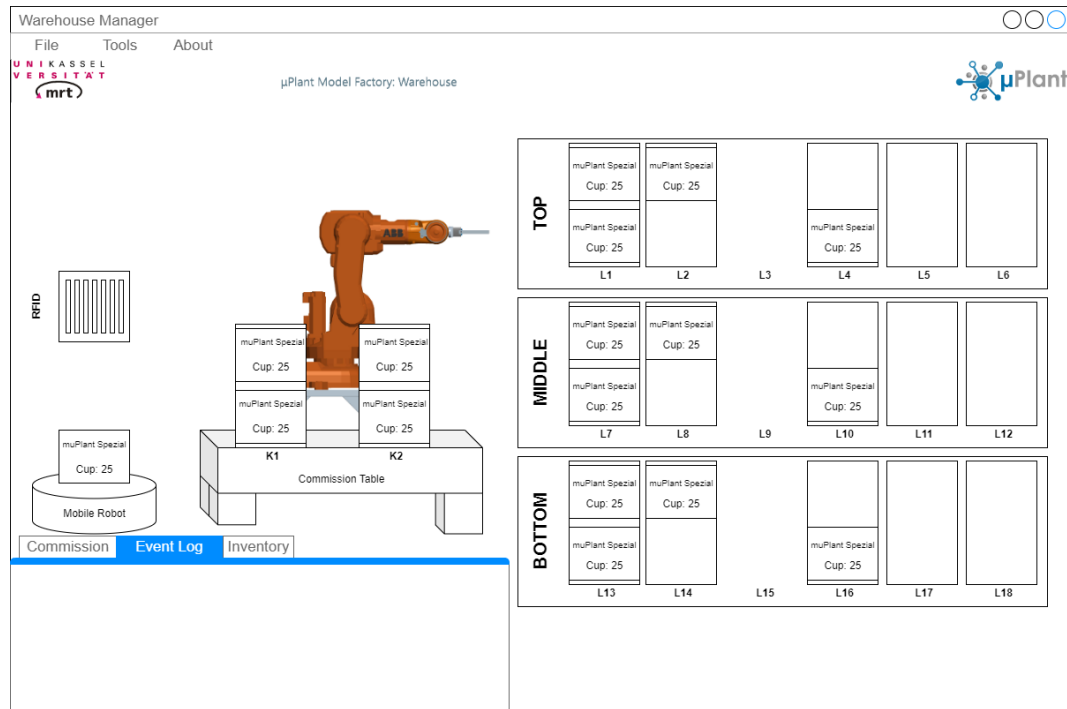
Eingegebene Daten müssen bestätigt werden und können auch wieder verworfen werden.

Die Bereiche „E“ (Inventarliste) und „F“ (Eventlog) aus Abb. 3.1 müssen nicht gleichzeitig sichtbar sein. Sie werden in einem Register integriert und mit dem QML-Datentyp `StackView` oder einem `Loader` angezeigt. Das Erscheinungsbild der beiden GUI Elemente kann aus der alten Software übernommen werden. Als Drittes Register soll zusätzlich zum alten Startbildschirm eine Commission List (siehe Abb. 5.7) hinzugefügt werden. Sie soll eine Übersicht über die von der Auftragsverwaltung eingetroffenen Aufträge und ihren Status anzeigen.

Um manuelle Überschreibungen der Produkte und Becher vorzunehmen, wird statt des Checkbox-Menüs ein Zahnrad-Symbol erscheinen, wenn man über dem entsprechenden Bereich den Mauszeiger führt (Hovering). Mit Klick auf das Zahnrad-Symbol soll ein Objekt des QML-Datentyps `QDialog` eingeblendet werden in dem die alten Feldwerte angezeigt und editiert werden können, aber auch das Editieren verworfen werden kann.

Die Funktion des ehemaligen Controllers wird über die Toolbar aufgerufen. Der Begriff

**Abbildung 5.5.:** Die Abbildung zeigt wie der Startbildschirm aussehen könnte um die Funktion des Automatikbetriebs abzubilden:





„Controller“ an sich ist im Zusammenhang mit der neu konzipierten Software irreführend und wird daher verworfen. Der stattdessen wird der Titel „Manual Processing“ gewählt. Hiermit sollen manuell Transportaufträge erzeugt werden wie z.B. „Räume den Becher aus Lagerort L3a nach L6b“. Dies soll für jede sinnvolle Transportoperation möglich sein. Mit Klick auf den entsprechenden Eintrag in der Toolbar öffnet sich ein Fenster oder ein Dialog, wie in Abb.5.6. Die so neu erzeugten Transportaufträge werden ebenfalls in der Commission List (siehe Abb. 5.7) angezeigt.

Das GUI des RFID Server Programms war sinnvoll und praktisch und kann so wie bisher umgesetzt werden. Der Aufruf des Programms soll über die Toolbar implementiert werden.

**Abbildung 5.6.:** Die Abbildung zeigt, dass entweder ein Becher oder eine Palette (mit oder ohne Becher) vom Startort zum Zielort transportiert werden kann. Die Eingabe der Zielorte wird hier über zwei Dropdown Menüs realisiert. Das Mockup impliziert, dass bei Tatigung einer Eingabe eine Validierung der Eingaben erfolgt.

Manual Processing

Select Target:

☐ Pallet

☐ Cup


\*Pallet transport will transport also eventual containing cups!

Start Destination:

Mobile Robot

▼

-

▼

Pallet at Start Destination:

YES

Product in a:

Apfelsaft

Product in b:

Kein Becher

End Destination:

L17

▼

b

▼

WARNING: Illegal Operation!

End Destination is not empty!

Cancel

Create Command

**Abbildung 5.7.:** Die Abbildung zeigt, wie die Commission List aussehen konnte. Sie konnte als Tab in einem Register eingebunden werden.:

Commission List Show processed orders

ID	Task	Source	Status
23	Cup 54 from L15 b to K1 a	Manual Process	In Progress...
24	Cup 67 from MR b to L15 a	Modbus	Pending
25	Cup 87 from K2 a to L1 a	Manual Process	Pending
26	Cup 5 from L5 a to MR	Modbus	Pending

---

## 5.5. Teilautomatisierte Code Dokumentation mit Sphinx

Code-Dokumentation ist ein wichtiger Bestandteil jedes Softwareprojekts und dient dazu den Code anderen Entwicklern zugänglich zu machen. Die Erstellung ist unter Umständen aufwändig, kann aber zumindest teilweise automatisiert werden. Eine Möglichkeit besteht darin, das Paket sphinx zu verwenden.

Sphinx ist ein Tool, das es Entwicklern ermöglicht, Dokumentationen in verschiedenen Formaten wie HTML, PDF und ePub aus Kommentaren im Code des Python Projekts zu erstellen. Dazu werden die in Python üblich Docstrings verwendet [9].

Das Paket kann über PyPi mit folgendem Kommando über die Eingabeaufforderung installiert werden:

```
pip install -U sphinx
```

Das Sphinx- Paket benutzt Themes um die Dokumentation grafisch darzustellen und zu strukturieren. In Vorbereitung auf diese Arbeit habe ich mir das rtd-theme (Read the Docs [10]) angeschaut und ausprobiert. Es ist ein beliebtes Theme für Sphinx-Dokumentationen, da eine klare und übersichtliche Darstellung der Dokumentation bietet und einfach zu verwenden ist. Um das rtd-theme in einem Sphinx-Projekt zu verwenden, muss es zunächst installiert werden. Dies kann über den Befehl

```
pip install sphinx_rtd_theme
```

erfolgen, dabei wird das Sphinx Paket auf die Version 6.4.1 angepasst.

Um Sphinx zu nutzen, muss es zunächst konfiguriert werden. Mit dem Kommando

```
sphinx-quickstart
```

wird man über die Konsole aufgefordert einige Optionen festzulegen mit denen die Dokumentation aufgesetzt wird. Anschließend befinden sich in dem ausgewählten Ordner:

- ein Makefile
- eine Datei `modules.rst`
- eine Datei `index.rst`
- eine Datei `conf.py`

Die Index-Datei ist später der Einstiegspunkt. Im RST-Format [11] kann die Seite gestaltet werden.

In der `modules.rst` dagegen werden die Module aufgelistet, die dokumentiert werden sollen. Nach der Installation des rtd-theme kann es in der Konfigurationsdatei von Sphinx aktiviert werden. Dazu muss die Zeile `html_theme = 'sphinx_rtd_theme'` in die Datei `conf.py` eingefügt werden. Sobald das Theme aktiviert ist, wird die Dokumentation nach dem build Vorgang im rtd-Theme angezeigt.

### 5.5.1. Sphinx Erweiterungen

Erweiterungen werden nach der Installation des Pakets in der Konfigurationsdatei `conf.py` aktiviert. Im Rahmen habe ich viele Erweiterungen getestet und empfehle die nachfolgenden Erweiterungen zu verwenden. Die nachfolgende Zeile aktiviert alle Erweiterungen aus meiner Empfehlung.



---

```
extensions = [
    'sphinx.ext.autodoc',
    'sphinx.ext.viewcode',
    'sphinx_copybutton']
```

## Autodoc

Autodoc wird dazu verwendet um die Docstrings im Python-Code als Dokumentationstext zu erfassen. Damit dies funktioniert muss ein Modul in der Datei `modules.rst` mit folgendem Syntax hinzugefügt werden:

```
.. automodule:: src.cameraApplication.cameraProcessing
    :members:
    :undoc-members:
    :show-inheritance:
```

`:members:` sorgt dafür, dass alle Klassen und Funktionen des Moduls in der Dokumentation erscheinen. Die verwendeten Docstrings werden zusammen mit dem Namen und ggf. Argumenten und Rückgabewerten angezeigt.

`:undoc-members:` sorgt dafür, dass auch Funktionen und Klassen ohne Docstrings in der Dokumentation erscheinen. Die Klassen und Funktionen werden dann ohne Beschreibung angezeigt.

`:show-inheritance:` sorgt dafür, dass die Vererbungshierarchie der Klassen angezeigt wird. Dies ist in Verbindung mit dem Qt Framework oft hilfreich. Da viele Qt-Klassen eine große Vererbungskette haben, kann dies manchmal auch unübersichtlich werden.

Die Dateipfade der Module werden wie im Beispiel vom Makefile aus zur Python-Datei angegeben, jedoch ohne Dateierweiterung. Docstrings in den Modulen werden direkt als HTML gerendert. Inline-Kommentare werden nicht angezeigt.

QML-Dateien können auf diese Weise nicht gerendert werden. Um sie dennoch in der Dokumentation anzuzeigen, habe ich sie in der Datei `modules.rst` mit folgendem Syntax hinzugefügt:

```
qml.main.qml
-----

The main QML-file creates the main application window.

.. literalinclude:: src/qml/main.qml
    :language: qml
```

Dadurch wird der gesamte Inhalt der QML Datei in der Dokumentation angezeigt.

## viewcode

Um den Quellcode des Moduls in dem HTML Dokument einblenden zu können wird die Erweiterung `viewcode` genutzt. Unterhalb der Modulüberschrift im HTML-Dokument werden zunächst alle Docstrings gerendert. Darunter erscheint nun ein grauer Bereich, in dem Python Code mit Farbformatierung angezeigt wird. Es werden auch andere Sprachen erkannt, jedoch gibt es leider keine Option QML Code farblich formatiert anzuzeigen.

---

QML-Dateien werden im Allgemeinen von Autodoc nicht erkannt. Die Einzige Option ist ein `literalinclude` Befehl wie oben schon beschrieben. Wie leicht zu erkennen werden Docstrings nicht erkannt und müssen daher manuell eingefügt werden. Kommentare im JavaScript-Syntax und Properties werden farblich hervorgehoben.

### **sphinx\_copybutton**

Diese Erweiterung fügt in Code-Blöcken der HTML-seite eine Kopier-Taste hinzu, mit denen z.B. Kommandozeilenbefehle in den Zwischenspeicher kopiert werden können. Code-Blocks werden wie folgt angegeben:

```
.. code-block:: bash
    pip install -r requirements.txt
```

## **5.6. Sonstige Festlegungen**

Die zu verwendenden Farben sollen dem *mu*Plant Logo oder dem Logo der Universität Kassel entnommen werden.

### **5.6.1. Virtual Environment**

Die Entwicklung der Software soll mit einer virtuellen Entwicklungsumgebung (venv) stattfinden. Alle benötigten Pakete werden dort installiert.

Die installierten Pakete werden nicht in das Git Repository gepusht. Stattdessen wird mit

```
pip freeze > requirements.txt
```

Eine Textdatei mit allen verwendeten Bibliotheken erstellt. Läd sich ein Benutzer das Repository herunter, können mit dem Befehl

```
pip install -r requirements.txt
```

alle benötigten Pakete installiert werden, sodass das Projekt lauffähig kompiliert werden kann. Der Nachteil ist, dass es im Aufgabenbereich des Entwicklers liegt, nicht verwendete Pakete zu deinstallieren, sodass sie später nicht unbeabsichtigter Weise installiert werden.

### **5.6.2. Verwendete Programme**

Nachfolgend werden alle verwendeten Programme und Software aufgelistet, die zur Erstellung der Software benutzt werden

- PyCharm Community Edition von JetBrains wird zur Editierung von Code verwendet. Das betrifft nicht nur Python Code, sondern auch alle anderen Sprachen und Syntaxe exklusiv QML. Sogar zur Erstellung von LaTeX Dokumenten wird PyCharm mit dem PlugIn TeXiFy verwendet.
- Qt Creator kann für alle Qt kompatiblen Programmiersprachen verwendet werden. Es beinhaltet außerdem den Qt Designer, mit dem QML-Dateien grafisch editiert werden können. Das Programm ist teilweise unbequem und umständlich, ist aber

---

für QML in Python die beste Lösung um intuitiv Einstellungen an QML-Datentyps auszuprobieren.

- Visual Studio 2022 wird dazu verwendet den C# Code der alten Software zu analysieren. Das PlugIn „Class Designer“ wird dazu verwendet um die Projektmappen in Klassendiagramme darzustellen.
- Die Homepage Diagrams.net bietet eine gute Lösung um schnell Diagramme in UML und anderen Standards zu erstellen und in gewünschten Dateiformaten zu exportieren.
- Pictogramme und Symbole werden mit Adobe Illustrator erstellt.



# 6

## Analyse zur Fehlerbehandlung

---

Fehler können systematischer oder sporadischer Natur sein. Die effektive Fehlersuche und Fehlerbehebung ist ein wichtiger Bestandteil der Softwareentwicklung, der den Entwickler ständig begleitet.

### 6.0.1. Fehleridentifikation

Syntaxfehler und Fehler in der Benutzung der Programmiersprache Python lassen sich in der Regel gut finden und beheben. Der größte Teil wird bereits bei der Erstellung des Quellcodes erkannt und farblich markiert. Andere Fehler führen bei jedem Ausführen des Codes zu einem Ausnahmefehler, den der Interpreter erkennt und (meistens) eine brauchbare Fehlermeldung in der Konsole ausgibt.

Weitere Fehler sind oft in der Logik des Programms zu finden. Diese zu identifizieren ist in der Regel schwieriger. Es ist jedoch möglich, nach dem Eingabe-Verarbeitung-Ausgabe Prinzip vorzugehen. Dabei bewertet man engmaschig die Eingabe und die Ausgabe und vergleicht Soll mit Ist.

Eingaben können in einem Wertebereich möglich sein. Hier muss der Entwickler alle erwartbaren Sonderfälle abdecken. Für systematische Fehler können in der Softwareentwicklung Tests definiert werden, die das Programmverhalten für ein Spektrum an Eingaben überprüfen. Die Ausgaben werden mit den erwarteten Ausgaben durch Assertions verglichen. In Python ist dies beispielsweise mit dem Paket `pytest` (siehe [12]) oder `unittest` (siehe [13]) möglich. Anhand einer kurzen Recherche beider Internetauftritte werde ich `pytest` verwenden. Einerseits liefert `pytest` bei fehlgeschlagenen Tests eine ausführlichere Analysemeldung, andererseits wird `pytest` von Qt empfohlen. Mit `pytest` lassen sich aber nur Python Module testen.

Für einen GUI Test muss das `QtTest` Framework verwendet werden. Leider lässt sich zu diesem Paket keine umfangreiche Dokumentation. Ich habe versucht dieses Framework zusammen mit dem Qt Creator zu benutzen. Entweder erhielt ich Fehlermeldungen, die ich nicht mit meinem Code in Verbindung bringen konnte oder das Ausführen des Programms ignorierte die geschriebenen Tests. Dies bewegt mich zu der Empfehlung von diesem Framework ohne weitere Updates und Dokumentation seitens Qt abzusehen.

Sporadische Fehler können nicht unbedingt vorhergesehen werden. Sie rühren meistens aus einer unerwarteten Kombination von Eingaben die selten auftritt oder durch einen Fehler in einem eingebundenen Dienst. Eine gut dokumentierte API weist auf Fehlermöglichkeiten hin und gibt Hinweise zur Fehlerbehandlung. Die erwarteten Quellen sind Eingabefehler und Kommunikationsfehler.

---

## 6.0.2. Erkennung fehlerhafter Benutzereingaben

Werden im Programm falsche Daten eingegeben eingelesen, soll dies, soweit wie möglich, entdeckt und abgefangen werden. Die Tabelle 6.1 listet alle erwarteten Benutzereingaben und mögliche Methoden die Benutzereingaben zu validieren.

**Tabelle 6.1.:** Benutzerinagenben, mögliche Fehler und ihre Erkennung

Art	Typ	Wertebereich	Fehlererkennung
IP Adresse	Formatierter String aus Integers	0 ... 255 bzw. 0 ... 9	Gültigkeitsprüfung bei Dateneingabe, Validierung bei Verbindungsaufbau
IP Port	Integer	0...65536	Wertebereich bei Dateneingabe, Validierung bei Verbindungsaufbau
Produkt ID	Integer	0...99	Eingabe anhand Dropdown Menü mit Anzeige des Produktnamens beschränkt Wertebereich auf zulässige Werte. Validierung nur im Softwarebetrieb durch Soll-Ist Abgleich.
Becher ID	Integer	0...99	Eingabe kann nicht auf gültigen Wertebereich beschränkt werden. Soll-Ist-Vergleich mittels RFID ist möglich.
Palette	Bool	True / False	Abgleich mit Anwesenheit von Bechern, arUco Marker Erkennung mittels Kamera

Wenn manuell Transportaufträge eingegeben werden, kann es zu Konflikten kommen. Z.B. Könnte im Abholort kein Becher oder keine Palette sein. Umgekehrt könnte am Abstellort eine Palette oder Becher stehen. Für diese Problematiken können :

- Inventardaten zur Überprüfung herangezogen werden
- Kameratestützte Validierungsprozesse in Kapitel 7 Entworfen werden.

# **7**

## **Ideensammlung zu kameragestützten Validierungsprozessen in der Lagerverwaltung**

---

**7.0.1. Konzepte**

**7.0.2. Abgeleitete Anforderungen an die Kamera**

**7.0.3. Kameraauswahl**





# 8

## Zusammenfassung und Ausblick

---

Hier wird die Arbeit zusammengefasst und ein Ausblick auf offene Fragestellungen gegeben.



# **A** Dies ist der erste Anhang

---

Hier Text einfügen.



# Literaturverzeichnis

---

- [1] „Liste aller QML Types, <https://doc.qt.io/qt-6/qtquick-qmlmodule.html>, Accessed: [15.06.2023].
- [2] S. Hübler, „BPS Tätigkeits- und Erfahrungsbericht: Objekterkennung und Verfolgung mittels Kamera in Produktionsanlagen am Beispiel der Modellfabrik µPlant, 2019.
- [3] „Modbus Organization, <http://www.modbus.org>, Accessed: [12.06.2023].
- [4] L. Kistner, „Konzeptionierung und Umsetzung einer serviceorientierten verteilten Automatisierungs-Architektur für die heterogene Modellfabrik µPlant, 2017.
- [5] „Qt History on Qt Wiki, [https://wiki.qt.io/Qt\\_History](https://wiki.qt.io/Qt_History), Accessed: [13.06.2023].
- [6] „PySide6 QtWidgets, <https://doc.qt.io/qtforpython-6/PySide6/QtWidgets/index.html>, Accessed: [14.06.2023].
- [7] „PySide6 QtQuick, <https://doc.qt.io/qtforpython-6/PySide6/QtQuick/index.html>, Accessed: [14.06.2023].
- [8] „Signal and Slot in PySide6 Framework, <https://doc.qt.io/qtforpython-6/PySide6/QtCore/Slot.html>, Accessed: [13.06.2023].
- [9] „PEP 257 – Docstring Conventions, <https://peps.python.org/pep-0257/>, Accessed: [19.06.2023].
- [10] „Read The Docs Homepage, <https://readthedocs.org/>, Accessed: [19.06.2023].
- [11] „Restructured Text Syntax, [https://thomas-cokelaer.info/tutorials/sphinx/rest\\_syntax.html](https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html), Accessed: [19.06.2023].
- [12] „Pytest Framework, <https://docs.pytest.org/en/7.3.x/>, Accessed: [19.06.2023].
- [13] „Python 3 unittest - Unit testing framework, <https://docs.python.org/3/library/unittest.html>, Accessed: [19.06.2023].