UNIKASSEL
VERSITÄT

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 μPlant

Lennart Schink

Matrikelnummer:	33237484
Gutachter:	Univ.-Prof. Dr.-Ing. Andreas Kroll
Betreuer:	Dip.-Ing. Axel Dürrbaum
Tag der Abgabe:	12. Juni 2023
MRT-Nr.:	N.N

mrt

isac

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Motivation und Zielsetzung

Das Institut für Mess- und Regelungstechnik an der Universität Kassel hat in den letzten Jahren eine Modellfabrik μPlant gebaut. Aus über 70 Einzelarbeiten ist ein modernes Industrie-4.0 Konzept geschaffen worden. Teil der μPlant 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 [Hügler] sollen überprüft und vertieft werden um Anforderungen an Kameras und arUco Marker zu ermitteln, die später eine automatisierte Inventur ermöglichen sollen.

2 Softwarearchitektur der bestehenden Software

Analysemethoden der Informatik für Software sind in der Regel für die verschiedenen Design-Phasen entwickelt worden. Eine von mir durchgeführte 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 die Neuentwicklung der bestehenden Software wird allerdings im Folgenden die Programmstruktur dargestellt und analysiert. Die Performance und der Speicherverbrauch spielen für die μPlant eine untergeordnete Rolle. Eine Bewertung der Programmkomponenten sollte 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 UI und Logik in getrennten Dateien implementiert. XAML Dateien sind eine angepasste Form von XML Dateien und legen somit fest, wie das GUI gerendert wird. XAML.CS Dateien enthalten die dahinterliegende Logik und Schnittstellen zu anderen Programmteilen. Am Beispiel des Startbildschirms des Programms „Lagerverwaltung 3.0“wird der Programmaufbau geschildert. Im Weiteren Verlauf dieser Arbeit wird, wegen der Komplexität und des fehlenden Mehrwerts, darauf verzichtet. Aus gleichem Grund wird auf die detaillierte Beschreibung der Funktionsweise von grafischen ELementen ihrer Events sowie ihrer Eventhandler verzichtet.
In den erstellen Klassendiagrammen ist gut ersichtlich, welche Klassen Events nutzen. Sie erben von der Klasse `INotifyPropertyChanged`, welche die benötigten Listeners dafür bereitstellt.

2.1. Lagerverwaltung 3.0

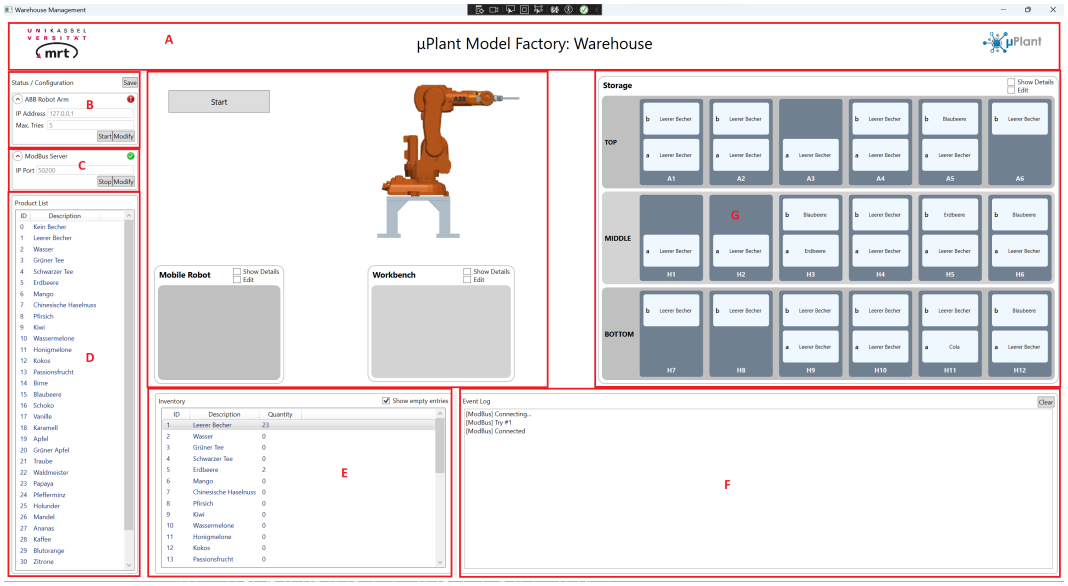
Wie in dem einleitenden Abschnitt angekündigt wird zunächst der grundlegende Aufbau des Programms beschrieben.
Die Datei `App.xaml` ist der Einstiegspunkt des Programms. In ihr wird ein Objekt der

Application-Klasse mit allen benötigten Ressourcen erzeugt. Als Start URL ist `MainWindow.xaml` angeben.
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 μPlant (Siehe Abb.2.1 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 Object `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 gerendert (Abb.2.1 Bereiche „B“und „C“). Die Produktliste wird aus der Datei `Produkte.db` geladen und gerendert (Abb.2.1 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.2.1 Bereich „E “und „G “).
Im Bereich „F“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. Wenn keine Verbindung zum Modbus hergestellt werden kann, wird dem Benutzer angeboten die Vorgänge zu simulieren. m Klassendiagramm Abb.2.2 wird jedoch schnell deutlich, dass dieser Simulationsbetrieb nicht dazu genutzt werden kann, um etwaige Fehler zu erkennen, 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 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.

Abbildung 2.1.: Startbildschirm der Anwendung, zur besseren Beschreibung sind die Bedienbereiche rot umrandet und mit Buchstanben gekennzeichnet



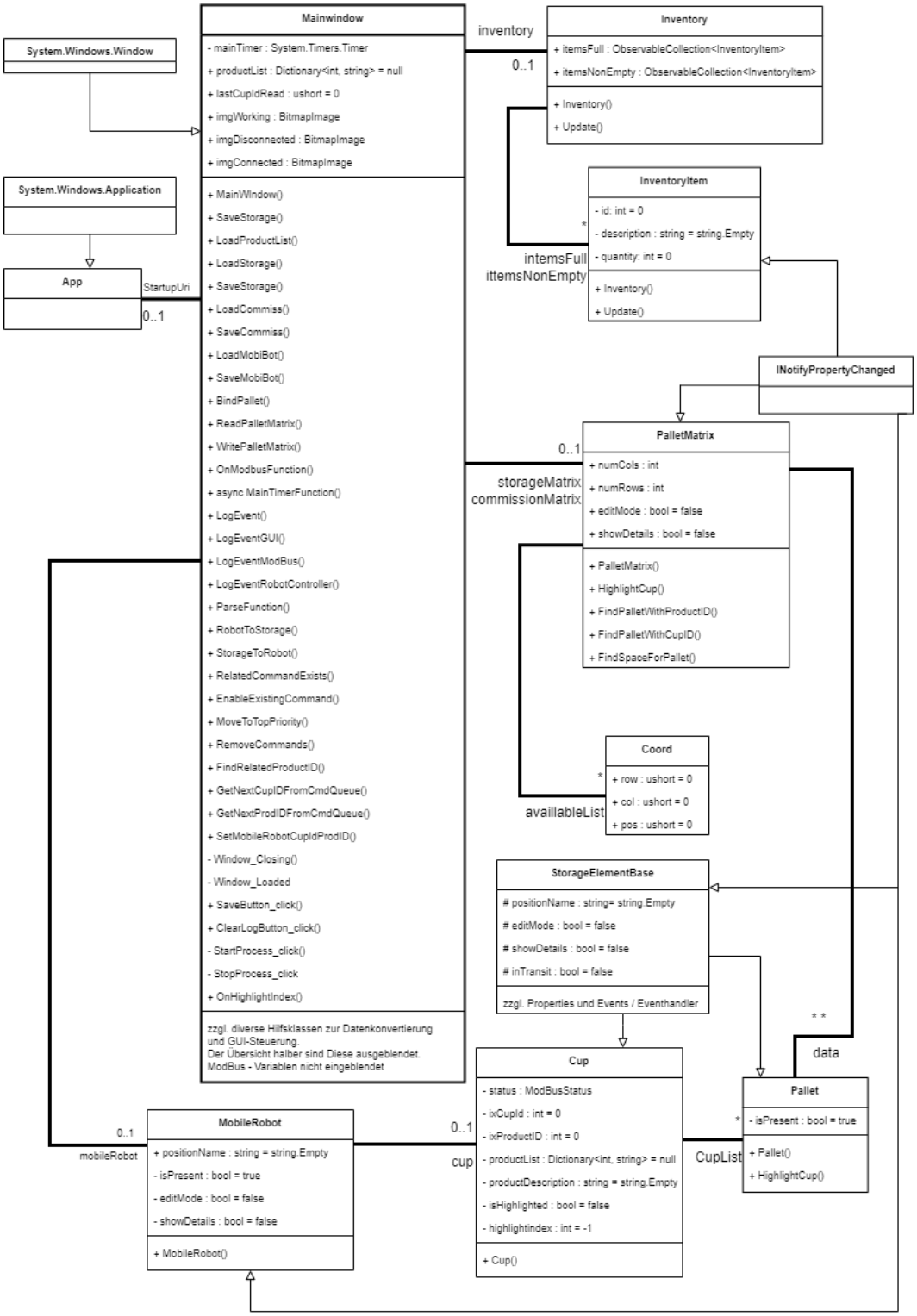
2.1.1. Klassenstruktur des Datenmodells

Die bestehende Software dient dazu Lagerpakete vom mobilen Roboter auf die Werkbank oder ins Lager zu bewegen. Oder alle möglichen Kombinationen davon. Das Konzept der μPlant beschränkt sich dabei auf eine Palette, die so ausgeführt ist, dass der Greifer des Industrieroboters sie sicher bewegen kann. Im Wesentlichen ist es ein Quader mit zwei seitlichen Längsnuten. Eine Palette enthält zwei senkrechte Bohrungen, die es ermöglichen je einen Becher aufzunehmen. Ein Becher ist ein transparentes, zylindrisches Gefäß aus Acrylglas mit einem Absatz, der etwa mittig in der Höhe angebracht ist, sodass der Greifer die Becher einzeln bewegen kann. Es wird also immer entweder

- Eine Palette
- Ein Becher
- eine Palette mit einem oder mehreren Bechern

bewegt. Der Programmierer hat sich diese Struktur angeeignet und in der Datenmodellierung umgesetzt. In Abb.2.2 ist gut ersichtlich, dass die Klasse `StorageElementBase` an die Klassen `Cup` und `Pallet` 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.2.2 Bereich „G“, sondern auf die Liste in Bereich „E“.

Abbildung 2.2.: 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 als Daten-Hub verwendet. Wie oben 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.2.2 Bereich “G „zu realisieren.
  - `commissionMatrix` bildet das DatenModell für die Visualisierung des mobilen Roboters und der Workbench.

2.1.2. Klassenstruktur zur Steuerung des ABB Industrieoboter IRB 140

Um dem Industrieroboter ABB IRB 140 Befehle zu erteilen, muss das Programm mit der Steuerung IRC5 kommunizieren. Im Programmcode finden sich dazu wie in Abb.2.3 gezeigt, die Klassen der Bibliothek `ABBRobotics`. Instanzen dieser Bibliothekbestandteile werden in der Klasse `RobotController` erzeugt, die von der Klasse RobotControllerBase erbt. Interessanter Weise wird in der `MainWindow` Klasse kein Objekt von RobotController erzeugt, sondern von der Klasse `RobotControllerBase`. Die Klasse `ControllerCommandList` erbt von einer Liste der Klasse `ControllerCommand`. Beim Initialisieren des Programms wird eine leere `ControllerCommandlist` erzeugt. Die Methoden und Variablen in der Klasse implizieren, dass die Commands aus einer Datei geladen werden. Diese Methoden werden jedoch nie genutzt. 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 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:

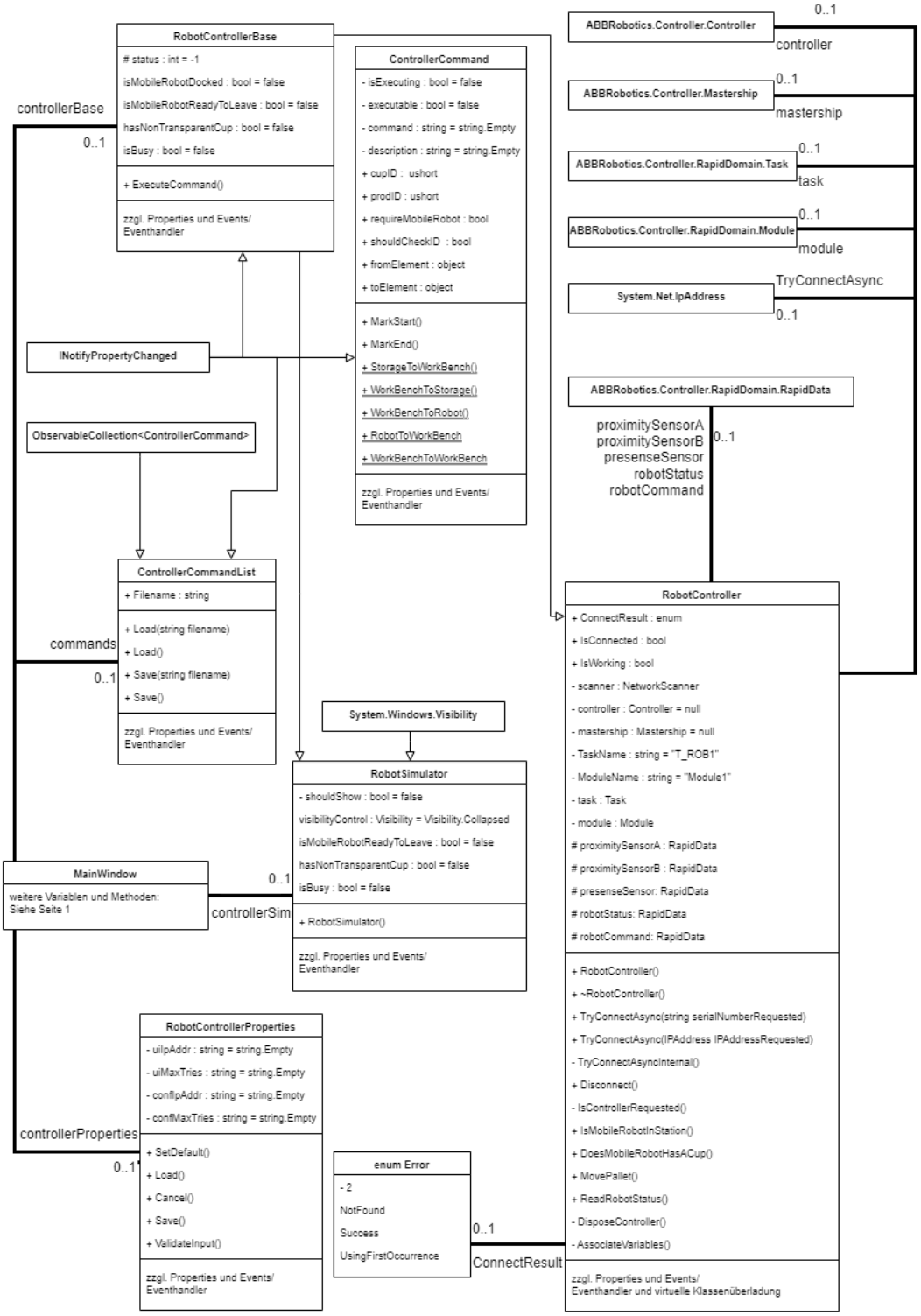
```
command = string.Format("Palette_{0}_{1}_1_{2}",
station, position, w_col +1)
```

`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.

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

Damit 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 2.1.1 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 ließ sich mit meinen eher bescheidenen C# Kenntnissen nicht evaluieren.

Abbildung 2.3.: Klassendiagramm der MainWindow-Klasse mit Klassen aus dem ABBRobotics Controllerframework.



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.

Lars Kistner [1] hat in seiner Bachelorarbeit aus dem Jahr 2016 die Kommunikationsstruktur der μPlant 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 und um die Adressen des mobilen Roboters ergänzt. Schaut man sich das Klassen-

Tabelle 2.1.: Werte der Klasse ModbusBaseAddress

Adresse	Wert	Beschreibung
Status	1	Modbus Status
KeepAlive	2	Dieser Wert wird vom Server auf den Wert 10 gesetzt, der Client zählt den Wert runte
Working	3	
FunctionReady	5	

diagramm der Modbus - Implementierung 2.4 an, sieht man schnell, dass die Klasse `MainWindow` nur zwei wesentliche Objekte hat: Zum Einen hat sie ein Objekt der Klasse `ModbusClient` als `modbusClient` gespeichert, zum Anderen ein Objekt `modbusInfo` der Klasse `ModbusInfo`. `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. Das muss anhand von Code gezeigt werden.

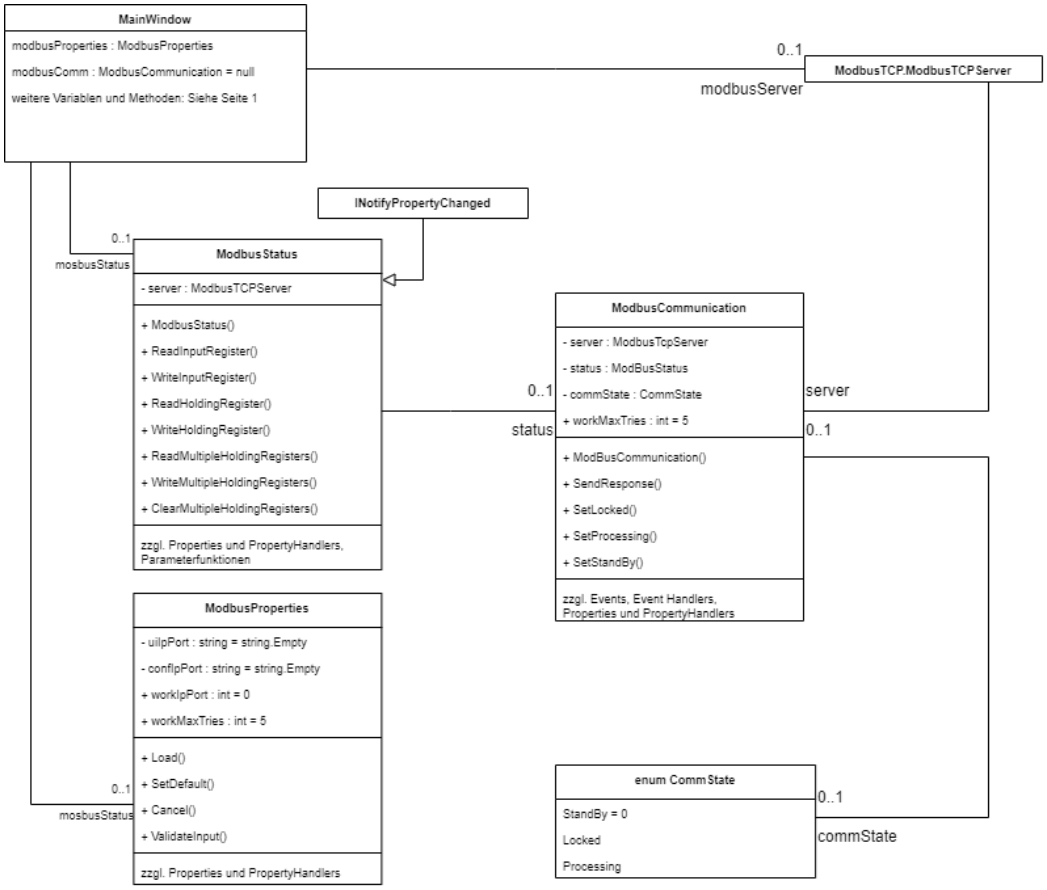
Im Anhang B.5 seiner Arbeit sind die Modbus-Adressen, die dem Lager zugeordnet sind, aufgelistet. Der Adressenbereich ist lückenlos zwischen 1024 und 1105. Jeder Ablageort eines Bechers (z.B. L1a für Lagerort L1, vorne) hat zwei Adressen: Eine für die Cup ID und eine für die Product ID. Das schließt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

2.1.3. GUI

Das GUI besteht aus einem Fenster, welches in 8 Bereiche unterteilt ist (siehe Abb.2.1). Der Bereich„A "hat keine Bedienfunktion.

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 Felder können über die Taste „Modify "wird die Eingabe freigeschaltet. Mit der Taste

Abbildung 2.4.: Klassendiagramm der MainWindow-Klasse mit Mosbus-Klassen



„Save “können die Änderungen gespeichert werden. Mit der „SStart"“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.

Im Bereich „C “kann der Ip Port des Modbus Servers über die Taste „Modify “angepasst werden. Nach meiner Beobachtung funktioniert der Bereich nicht wie er soll, denn die Verbindung wird direkt nach Programmstart direkt als hergestellt angezeigt. Die Verbindungstaste zum Starten steht zum Programmstart auf „Stop“.

Im Bereich „D “befindet sich eine scrollbare Liste aller möglicher Produkte mit zwei Spalten. Es wird neben dem Produktnamen die Produkt ID angezeigt. Wahrscheinlich wurde die Liste als Nachschlagewerk gebaut, denn es sind ansonsten keine Bedienfunktionen verfügbar.

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

Bereich „F “ist ein Eventlog. Hier werden von dem Programm aus verschiedenste MItteilungen an den Beutzer getätigt. Mit der Taste „clear “werden alle existierenden Einträge gelöscht.

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. Das reale Pendant ist mit L1. . . L18 gekennzeichnet.
Das dunkelgraue Rechteck mit der Lagerort Beschriftung 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 kein entsprechendes Rechteck 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 Cup ID und die Produkt ID ein- und ausblenden. Mit der Checkbox „Edit “kann der Benutzer alle Daten (Palette vorhanden/ nicht vorhanden, Cup 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.

Im Mittleren Bereich ist links der Bereich für den mobilen Roboter gerendert. Rechts ist der Bereich des Kommissioniertischs gerendert. Beide Bereiche geben dem Benutzer, wie das Lager auch, die Möglichkeit über die Checkboxen „Show Details “und „Edit“Die Cup- bzw. Produkt ID ein- und auszublenden oder die Daten manuell zu überschreiben. Darüber ist der Roboter IRB 140 als symbolbild gerendert. Das Bild bietet keinerlei Bedienfunktion. Links daneben befindet sich die „Start “Taste. Sie startet bei verbundenem Modbus den Automatikbetrieb. Ist der Modbus nicht verbunden, erscheint nach dem Klick ein Dialogfenster. Dieses fragt den Benutzer ob wegen 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 Checkboxen „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 Checkboxen 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.

Weiterhin bietet das Programm keine Möglichkeit den Roboter zu steuern.

2.2. Controller

Abbildung 2.5.: Ansicht des Controller Startbildschirms.

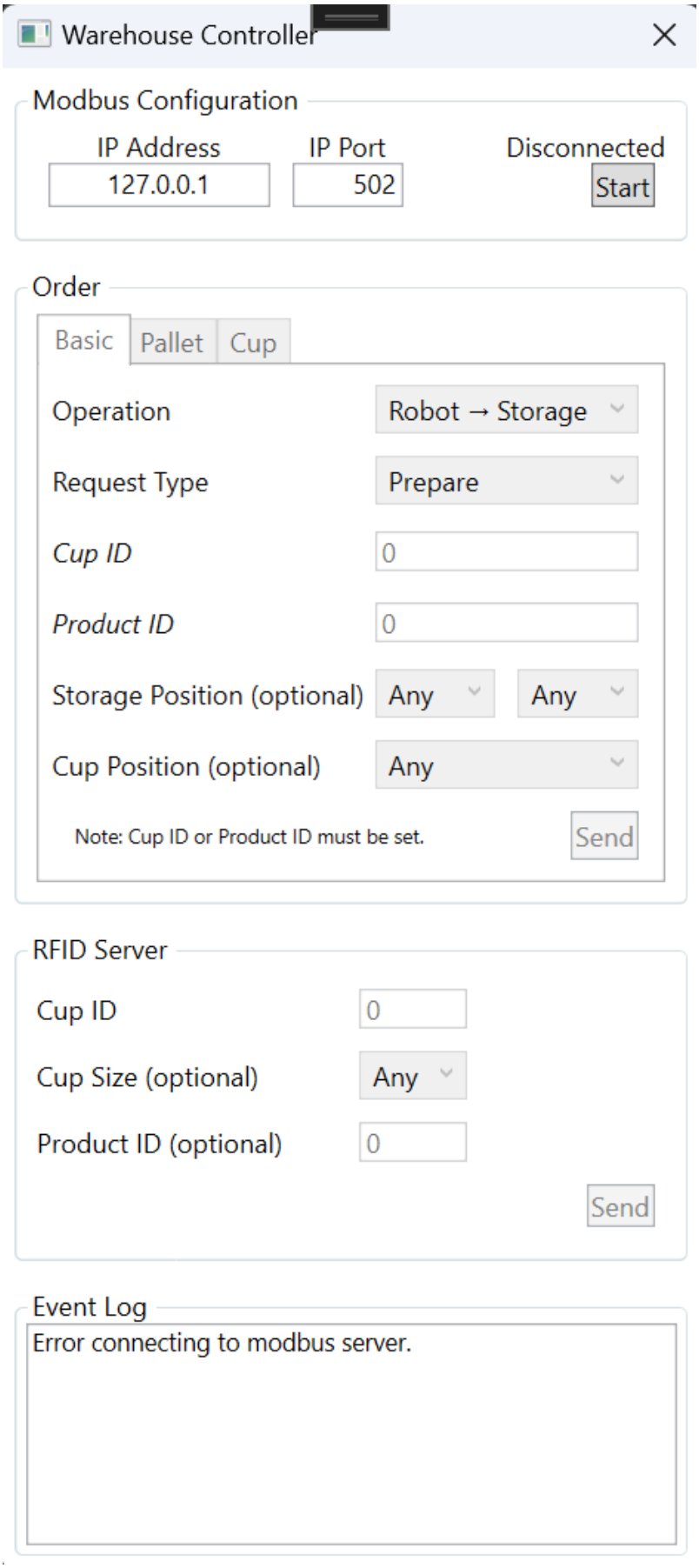


Abbildung 2.6.: Klassendiagramm des Warehouse-Controllers mit allen vererbenden Klassen jedoch ohne Klassen die lediglich Datentypen konvertieren.

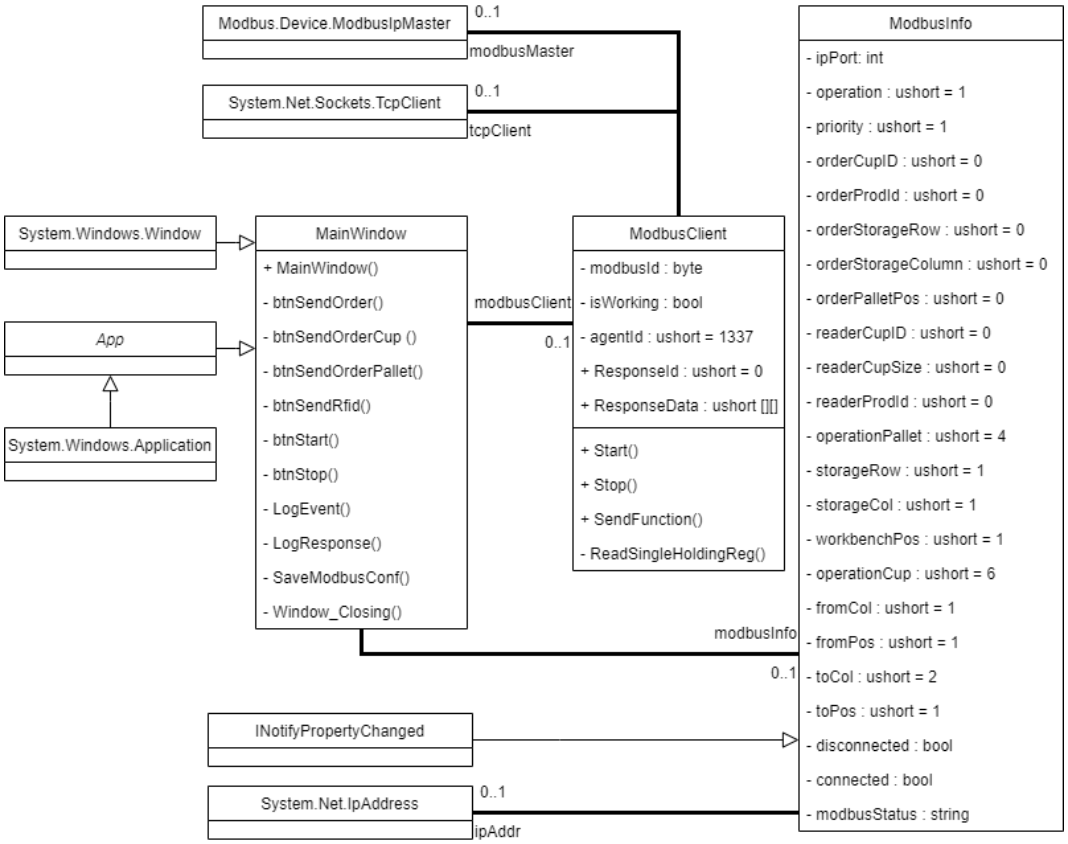
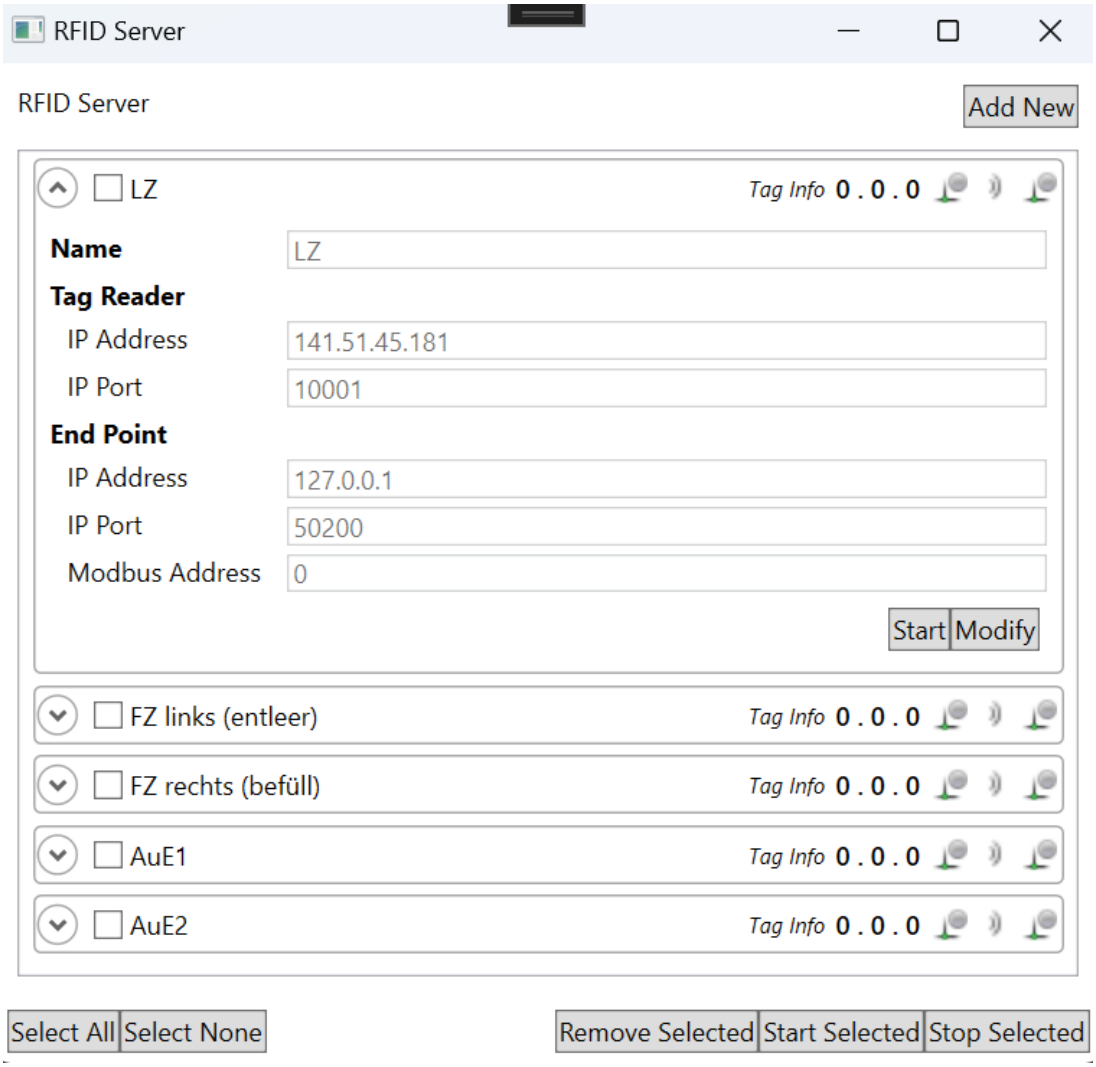
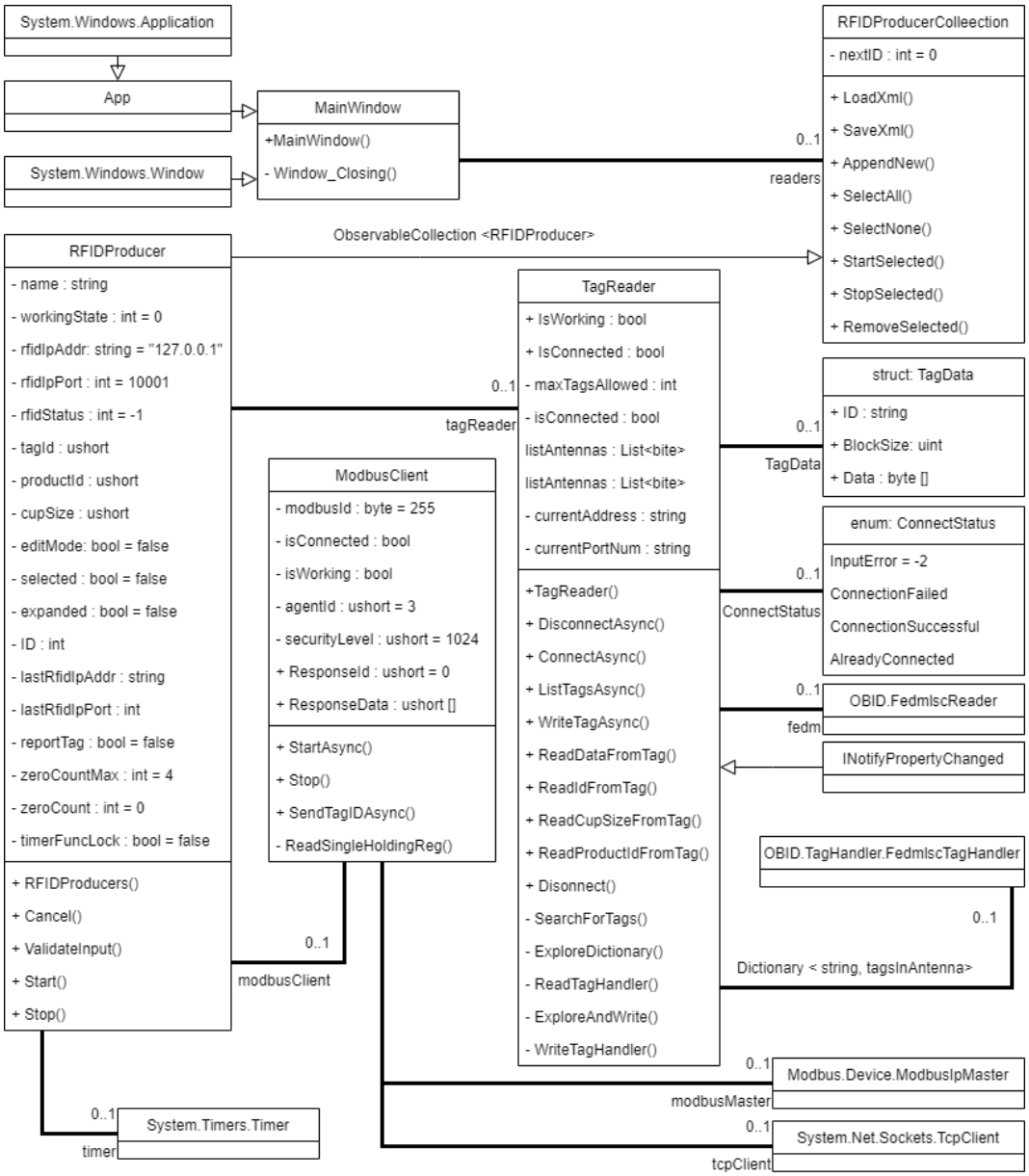


Abbildung 2.7.: Startbildschirm des Programms RFID-Server



2.3. RFID Server

Abbildung 2.8.: Klassendiagramm des Programms RFID Server mit allen vererbenden Klassen jedoch ohne solche die lediglich Typen konvertieren.



3 Konzeptionierung einer integrierten Python Anwendung

3.1.]

PySide6 und QuickQml 2.0

3.2. GUI - Konzeptionierung

3.3. Konzepte zur Datenmodellierung

3.4. Konzepte für Controller- und Serviceklassen

3.5. Teilautomatisierte Code Dokumentation

4 Analyse zur Fehlerbehandlung

5 Ideensammlung zu kameragestützten Validierungsprozessen in der Lagerverwaltung

5.0.1. Konzepte

5.0.2. Abgeleitete Anforderungen an die Kamera

5.0.3. Kameraauswahl

6 Zusammenfassung und Ausblick

Hier wird die Arbeit zusammengefasst und eun Ausblick auf offene Fragestellungen gebeben.

A Dies ist der erste Anhang

Hier Text einfügen.

Literaturverzeichnis

[1] L. Kistner.