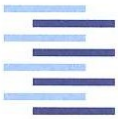

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Dominik Löffler

Automatisierung einer Modellanlage unter Verwendung einer grafischen Modellierungssprache mit objektorientierter Programmierung

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Dominik Löffler

Automatisierung einer Modellanlage unter Verwendung einer grafischen Modellierungssprache mit objektorientierter Programmierung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Elektrotechnik und Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Ulfert Meiners
Zweitgutachter: Prof. Dr. Marc Hensel

Abgegeben am 12. März 2018

Dominik Löffler

Thema der Bachelorarbeit

Automatisierung einer Modellanlage unter Verwendung einer grafischen Modellierungssprache mit objektorientierter Programmierung

Stichworte

Objektorientierte Programmierung, Raspberry Pi, CODESYS, UML, Automatisierung

Kurzzusammenfassung

Diese Arbeit beschreibt die Programmierung einer Modellanlage mit Hilfe von CODESYS, sowie der Erweiterung CODESYS UML. Als Laufzeitsysteme werden drei Raspberry Pi verwendet. Im Vordergrund steht der objektorientierte Programmentwurf, der zu einer besseren Wiederverwendbarkeit des Programmcodes und einem geringeren Programmieraufwand führt. In der Automatisierungstechnik findet die Objektorientierung noch nicht viel Anwendung, obwohl es gerade in diesem Bereich Sinn ergibt, reale mechanische Komponenten, wie z.B. Aktoren und Sensoren, in Klassen abzubilden.

Dominik Löffler

Title of the paper

Automation of a model system using a graphical modelling language with object-oriented programming

Keywords

Object-oriented programming, Raspberry Pi, CODESYS, UML, automation

Abstract

This thesis describes the programming of a model installation using CODESYS and the extension CODESYS UML. Three Raspberry Pi are used as runtime devices. The focus is on the object-oriented program design, which leads to a better reusability of the program code and less programming effort. In automation technology, object orientation has not been popular, although it makes sense to map real mechanical components, such as actuators and sensors, into classes.

Danksagung

Zuallererst möchte ich mich ganz herzlich bei Prof. Dr.-Ing. Ulfert Meiners und Prof. Dr. Marc Hensel für Ihre freundliche und kompetente Betreuung bedanken.

Zudem bedanke ich mich bei der Firma Prinovis GmbH & Co. KG für die Unterstützung im Laufe des Studiums.

Letztendlich bedanke ich mich bei meinen Freunden und besonders bei meiner Familie, die mich nicht nur während der Erstellung der Bachelorarbeit, sondern auch im Verlauf des Studiums immer unterstützt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

CODESYS	Controller Development System
CPU	Central Processing Unit
CTU	Count-Up
DUT	Benutzerdefinierter Datentyp
EEPROM	Electrically Erasable Programmable Read-Only Memory
EtherCAT	Ethernet for Control Automation Technology
GPIO	General Purpose Input/Output
GVL	Globale Variablenliste
I/O	Input/Output
NVL	Netzwerkvariablenliste
OOP	Objektorientiertes Programmieren
PAA	Prozessabbild der Ausgänge
PAE	Prozessabbild der Eingänge
PLC	Programmable Logic Controller
POU	Program Organization Unit
UDP	User Datagram Protocol
UML	Unified Modeling Language
RAM	Random Access Memory
SC	Statechart
SDRAM	Synchronous Dynamic Random Access Memory
SPS	Speicherprogrammierbare Steuerung

1 Einführung

1.1 Motivation

Die stetige Entwicklung industrieller Prozesse führt zu einem Anstieg der Anforderungen, die an die Automatisierungstechnik gestellt werden. Bei gleichbleibendem Zeit- und Kostendruck steigt der Anspruch an Variabilität und Qualität. Gleichzeitig entwickeln sich auch die dazugehörigen Steuerungen weiter und leistungsfähigere CPUs ermöglichen die Implementierung komplexerer Automatisierungslösungen, die zuvor auf spezielle externe Hardware ausgelagert werden mussten.

Die steigende Komplexität führt zu einem deutlichen Anstieg der Engineeringkosten im Bereich der Erstellung und Wartung von Software. Um diesen Anstieg entgegenzuwirken, stellt sich die Frage, wie *„die zunehmende Realisierung der Automatisierungstechnik in Software vor dem Hintergrund kürzerer Entwicklungszeiten und niedrigerer Kosten [...] für automatisierungstechnische Systeme besser (Qualität, Effizienz) entwickelt, implementiert, in Betrieb genommen und gewartet bzw. erweitert werden kann“*. [14]

Als eine mögliche Lösung haben sich objektorientierte Konzepte zur Steigerung der Effizienz und der Qualität bewährt. Im Bereich der Hochsprachenprogrammierung ist objektorientiertes Programmieren (OOP) in Sprachen wie Java oder C++ bereits etabliert, während in der Steuerungstechnik die speicherprogrammierbaren Steuerungen (SPS) meist in prozeduralen Sprachen der IEC 61131-3 Norm programmiert werden. Dabei bietet es sich gerade in diesem Anwendungsbereich an, Maschinen und Anlagen in ihre mechanischen Komponenten zu zerlegen und diese in Softwaremodulen abzubilden. Diese Modularisierung trägt maßgeblich zur Wiederverwendbarkeit von Software und somit zu einer Verringerung der Engineeringkosten bei. Die Unified Modeling Language (UML) könnte bei dieser Formulierung eine mögliche Lösung darstellen.

1.2 Ziel der Arbeit

Mit der Einführung der Entwicklungsumgebung „CODESYS V3“ von 3S-Smart Software Solutions stehen die Konzepte der objektorientierten Programmierung auch den Programmierern von speicherprogrammierbaren Steuerungen zur Verfügung. Zusätzlich wird die Entwicklungsumgebung seit der Einführung der Erweiterung „CODESYS UML“ um zwei Sprachen der Unified Modeling Language, dem Klassendiagramm und dem Zustandsdiagramm, ergänzt.

Ziel dieser Bachelorarbeit ist die Automatisierung einer an der Hochschule für angewandte Wissenschaften vorhandenen Modellanlage unter Verwendung von OOP mit CODESYS und der Erweiterung CODESYS UML. Es werden drei Raspberry Pis 3 als Laufzeitsysteme genutzt, die jeweils mit einem Anlagenteil über EtherCAT kommunizieren. Zusätzlich soll eine Bewertung der Erweiterung CODESYS UML hinsichtlich Bedienbarkeit und Nutzen vorgenommen werden.

1.3 Gliederung der Arbeit

Im **Kapitel 2** werden dem Leser die Grundlagen über speicherprogrammierbare Steuerungen, dem Raspberry Pi 3 und der Kommunikation via EtherCAT und UDP vermittelt. Des Weiteren wird hier auf die CODESYS UML Erweiterung eingegangen.

Nach den technischen Grundlagen wird im **Kapitel 3** die Modellanlage detailliert beschrieben.

Die Definition der Anforderungen, die an die Programmierung und Funktion der Anlage gestellt werden, sind im **Kapitel 4** näher erläutert.

Der Aufbau des Netzwerks und die notwendige Konfiguration von CODESYS und der Raspberry Pis sind Inhalt von **Kapitel 5**. Dabei wird die Konfiguration lediglich für diesen konkreten Anwendungsfall unter der Verwendung von CODESYS V3.5 erläutert.

Im **Kapitel 6** wird die objektorientierte Herangehensweise bei der Erstellung der Software und die Implementierung in CODESYS beschrieben.

Kapitel 7 dient der Dokumentation der Funktionstests, die an der Anlage durchgeführt werden. Die Ergebnisse werden mit Hilfe der Trace-Funktion von CODESYS aufgezeichnet.

Das abschließende **Kapitel 8** beinhaltet eine Zusammenfassung der Ergebnisse und eine Reflexion zur Umsetzung der Software mit der CODESYS UML Erweiterung.

2 Grundlagen

Für ein besseres Verständnis der Arbeit soll in diesem Kapitel auf die Grundlagen eingegangen werden. Dazu zählen Informationen über speicherprogrammierbare Steuerungen, dem Raspberry Pi 3 und die Kommunikation zwischen den verschiedenen Teilnehmern. Außerdem wird hier die UML Erweiterung von CODESYS vorgestellt.

2.1 Speicherprogrammierbare Steuerungen

Als Klassiker der verwendeten Laufzeitsysteme in der Steuerungstechnik gilt die speicherprogrammierbare Steuerung (engl. Programmable Logic Controller, PLC). Aus der Industrie sind diese Steuerungen heutzutage kaum wegzudenken und tragen maßgeblich zu einer fortschreitenden Automatisierung bei.

2.1.1 Definition einer Steuerung

Bei einer Steuerung wird der Stellwert $\vec{y}(k)$, der an den Prozess gegeben wird, durch den eingehenden Sollwert $\vec{w}(k)$ und den in der SPS implementierten Algorithmus beeinflusst. Hierbei spricht man von einem offenen Wirkungsweg. Wird der Prozess zusätzlich durch Sensorik überwacht, die Messwerte $\vec{x}(k)$ an das Steuergerät übermittelt und im Algorithmus berücksichtigt, entsteht ein geschlossener Wirkungsweg. In Abbildung 2.1 ist der Steuerkreis schematisch dargestellt.

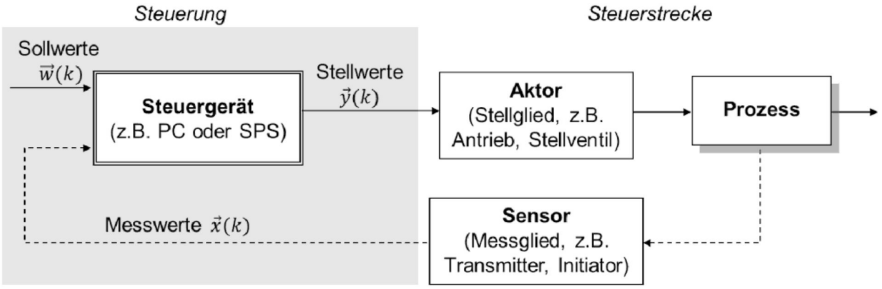

Abbildung 2.1: Schematischer Aufbau eines Steuerkreises [13]

Im Gegensatz zur Regelung findet bei der Steuerung kein kontinuierlicher Vergleich zwischen der Prozessgröße und dem Sollwert statt. Alle Werte werden zu zeitdiskreten Abtastpunkten k eingelesen bzw. ausgegeben. [12][13]

2.1.2 Bauarten

Das Hauptanwendungsgebiet für Steuerungen in der Automatisierungstechnik ist das Messen, Steuern und Regeln von Produktionsanlagen. Grundsätzlich werden speicherprogrammierbare Steuerungen in drei Bauarten unterschieden, die im Folgenden kurz erläutert werden.

Die Hardware-SPS ist die in der Industrie am häufigsten anzutreffende Steuerung. Durch eine kompakte und robuste Bauform kann diese direkt in Schaltschränken einer Produktionsanlage eingebaut werden. Ihre Kernkomponenten bestehen aus einer CPU, die für die Ausführung der Steuerungsprogramme zuständig ist, einem RAM, der die aktuellen Prozessparameter und das abzuarbeitende Programm enthält, und einem EEPROM, der alle Anwender- und Betriebssystemprogramme enthält. Eine Besonderheit ist die Möglichkeit zur Erweiterung der SPS durch Ein- und Ausgabebaugruppen, die beispielsweise das Verarbeiten von Analogwerten oder das Anbinden von Sensorik über Kommunikationsbaugruppen ermöglichen.

Eine Slot-SPS beinhaltet alle Komponenten einer Hardware-SPS und ist auf einer Einsteckkarte für einen PC untergebracht. Die Besonderheit hierbei ist ein multi-ported RAM, der ein Zugreifen der CPU des PCs auf die aktuellen Prozessvariablen des Steuerungsprogramms ermöglicht.

Die letzte Variante ist die Soft-SPS, die als reine Software auf einem PC läuft. Ein bedeutender Vorteil dieser Variante ist, dass ein PC zur Programmierung, Steuerung und Visualisierung genutzt werden kann und sich preiswertere Automatisierungslösungen ergeben. Dabei muss beachtet werden, dass der PC die Kriterien, die an eine in der Industrie verwendeten SPS gestellt werden, erfüllen muss. [13] [18]

2.1.3 Funktionsweise

Eine SPS verarbeitet das Steuerungsprogramm und die ein- und ausgehenden Informationen zyklisch. Die einzelnen Verarbeitungsschritte folgen dabei dem sogenannten „EVA“-Prinzip:

Einlesen
Im ersten Verarbeitungsschritt wird das Prozessabbild der Eingänge erstellt. Dabei wertet die CPU alle Eingangskanäle aus und legt die Prozessvariablen im RAM ab.

Verarbeiten
Im zweiten Verarbeitungsschritt wird das Programm sequentiell abgearbeitet. Dabei werden die Prozessvariablen im RAM gelesen und die Variablen des Prozessabbilds der Ausgänge geschrieben.

Ausgeben
Im letzten Schritt wird das Prozessabbild der Ausgänge an die Hardware geleitet und die angeschlossenen Aktoren werden angesteuert.

Ändert sich der Status einer Prozessvariable nachdem der erste Verarbeitungsschritt abgeschlossen ist, wird diese Änderung erst beim nächsten Zyklusdurchlauf berücksichtigt.
[1] [11] [13]

2.2 Raspberry Pi 3

Der Raspberry Pi 3 ist ein Einplatinen-Computer, der alle wesentlichen Komponenten eines vollwertigen PCs enthält. Die Idee hinter der Entwicklung war eine preisgünstige Lernplattform zu schaffen, mit der Schüler und Studenten das Programmieren und den Umgang mit Computern erlernen können. Der Raspberry Pi 3 Model B ist in Abbildung 2.2 zu sehen.

Abbildung 2.2: Raspberry Pi 3 Model B [10]

Ausgestattet ist der Raspberry Pi 3 mit einem Quad-Core-Prozessor mit 1,2 GHz von Broadcom und einem SDRAM-Arbeitsspeicher mit 1 GByte. Die Besonderheit im Vergleich zu den Vorgängermodellen besteht darin, dass WLAN und Bluetooth onboard integriert sind.
Für diese Arbeit sind lediglich die USB2.0-Ports und der RJ45-Ethernet-Anschluss von Bedeutung, da über diese Schnittstellen die Kommunikation zur Modellanlage und zwischen den Raspberry Pis realisiert werden. [10]

CODESYS bietet eine speziell für den Raspberry Pi angepasste Laufzeitumgebung. Diese muss installiert werden, um den Raspberry Pi als SPS mit dem CODESYS Development System programmieren zu können. Die Möglichkeit auf die GPIO-Pins zuzugreifen oder weitere Hardware mittels SPI-, I^2C- oder one-wire-Schnittstelle anzuschließen, ist ebenfalls mit der CODESYS Entwicklungsumgebung realisierbar. [2]

2.3 Kommunikation

Für die Kommunikation der verschiedenen Teilnehmer kommen zwei unterschiedliche Kommunikationsprotokolle zum Einsatz, die nachfolgend kurz erläutert werden.

2.3.1 EtherCAT

EtherCAT ist ein Ethernet-basiertes Feldbussystem, das die Forderungen nach Echtzeitfähigkeit und geringe Kosten erfüllt.

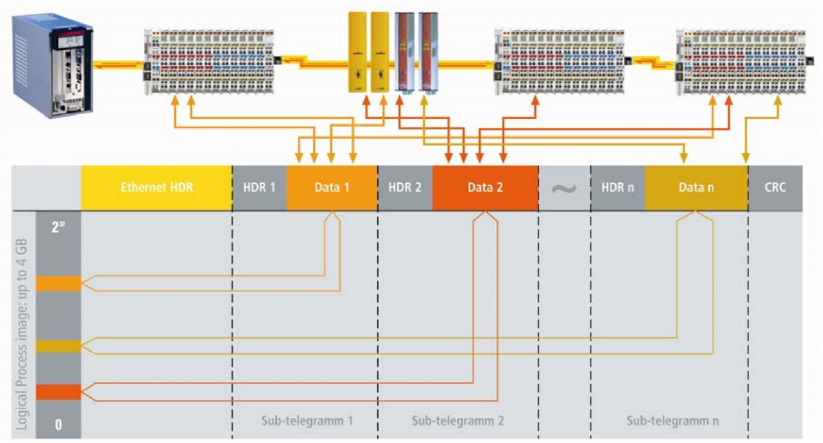

Abbildung 2.3: Funktionsweise von EtherCAT [6]

Herkömmliche Ethernet Pakete werden von jedem Busteilnehmer empfangen, ausgewertet und die Prozessdaten weitergegeben. Dies führt zu höheren Latenzen. Während ein EtherCAT-Telegramm die Teilnehmer durchläuft, entnimmt jeder Slave nur die Daten, die explizit an ihn gerichtet sind. Zusätzlich können dabei neue Daten in das Telegramm eingefügt werden. Die Verzögerung, die so beim Durchlaufen eines Teilnehmers entsteht, liegt bei wenigen Nanosekunden. Es wird demnach jedem Teilnehmer des Netzwerks ein Sub-Telegramm zugeordnet, in dem die zugehörigen Daten liegen (siehe Abbildung 2.3). Die Reihenfolge der Sub-Telegramme muss nicht zwangsläufig der Reihenfolge der Teilnehmer im Netzwerk entsprechen, sondern kann frei adressiert werden. Eine direkte Kommunikation zwischen Slaves sowie Multicast und Broadcast werden ebenfalls unterstützt.

Die Verwendung eines speziellen Ethertypes ermöglicht den Transport der Daten direkt im Ethernet-Frame. Diese Variante ist dann zu wählen, wenn höchste Performance gefragt ist und sich alle Teilnehmer im selben Netzwerk wie die Steuerung befinden. Ist das nicht der

Fall, wird das EtherCAT Datagramm durch IP/UDP-Datagramme erweitert (siehe Abbildung 2.4) und kann somit auch über das Netzwerk der Steuerung hinaus versendet werden. [6]

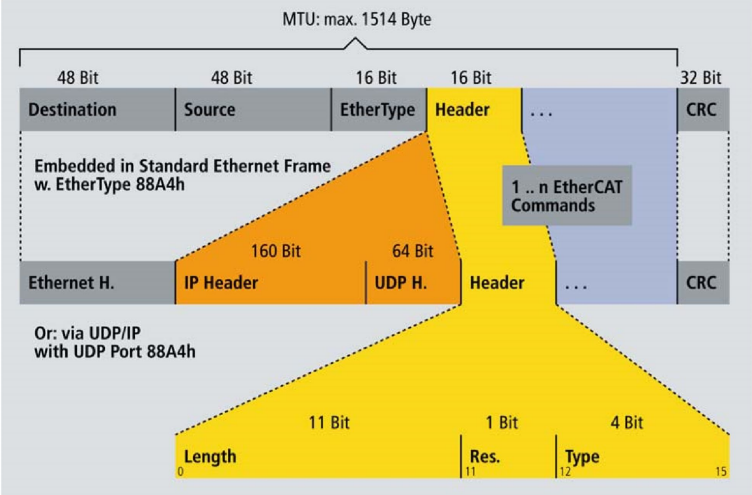

Abbildung 2.4: Aufbau eines EtherCAT Telegramms [6]

2.3.2 User Datagram Protocol

Das User Datagram Protocol (UDP) ist ein Transport-Protokoll für das Versenden und Empfangen von Daten in IP-basierten Netzwerken. Der Aufbau eines UDP-Telegramms ist in Abbildung 2.5 zu sehen. Es werden für die Kommunikation Ports genutzt, die eine Zustellung an die gewünschten Teilnehmer ermöglichen. Zusätzlich bietet UDP die Möglichkeit zur Erkennung einer fehlerhaften Übertragung durch das Mitsenden einer Prüfsumme.

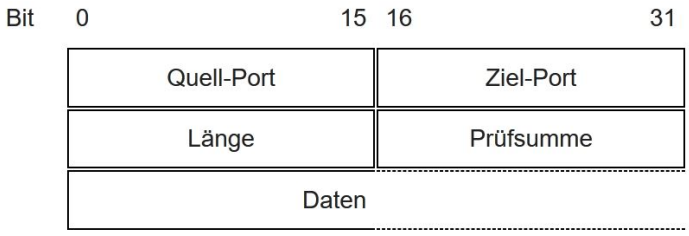

Abbildung 2.5: Aufbau des UDP-Headers [5]

UDP verzichtet beim Datenaustausch, abgesehen von der Prüfsumme, auf jegliche Kontrollfunktionen. Das bedeutet, dass nicht sichergestellt werden kann, ob die Daten beim Empfänger eingegangen sind, die Pakete in der richtigen Reihenfolge ankommen oder ob die

Daten sicher vor der Manipulation Dritter sind. Daher liegt es im Verantwortungsbereich der Anwendung, sicherzustellen, dass der Empfänger gegenüber verlorenen oder unsortierten Daten unempfindlich ist sowie über entsprechende Sicherheitsmaßnahmen verfügt. Der Vorteil einer UDP-Übertragung liegt darin, dass die Teilnehmer keine Verbindung zueinander aufbauen müssen und so die Übertragung der Daten schnell beginnen kann. Besonders in der Steuerungstechnik, in der gewöhnlich eher kleinere Datenmengen zeitkritisch ausgewertet werden müssen, bringt dies entscheidende Vorteile. [5]

2.4 Einsatz von UML-Diagrammen in der Steuerungstechnik

Wie bereits in der Einleitung dieser Arbeit beschrieben, steigt die Komplexität von Maschinen und Anlagen und somit wird auch die Steuerungs-Software immer komplexer. *„Neben textuellen Sprachen haben sich grafische Sprachen etabliert, mit denen der Bruch der prozeduralen Programmierung zwischen Design und Software [...], überbrückt wird.“* [15] Die Unified Modeling Language ist eine solche grafische Sprache, die bereits in vielen Bereichen des Softwareengineerings eingesetzt wird.

An dieser Stelle sei darauf hingewiesen, dass die Abschnitte 2.4.2 und 2.4.3 auf der CODESYS Online Hilfe basieren und deren Bestimmungen übernommen werden.

2.4.1 Übersicht über die Unified Modeling Language

Die Unified Modeling Language dient der Spezifikation, Visualisierung, Konstruktion und Dokumentation für objektorientierte Softwaresysteme. Sie hat den Anspruch eine standardisierte und allgemein verständliche Diskussionsbasis für die Erstellung von Systemen zu sein. Die UML wird von der Object Management Group entwickelt. [7]

Grundsätzlich lassen sich die Diagramme in zwei Hauptdiagrammtypen unterscheiden: Strukturdiagramme und Verhaltensdiagramme. Sie fassen jeweils weitere Diagrammtypen zusammen. Abbildung 2.6 stellt eine Übersicht dieser Diagrammtypen dar.

Abbildung 2.6: Übersicht der Diagrammtypen der UML

Häufig werden UML Diagramme zur Systemmodellierung bzw. -analyse verwendet, wobei die resultierenden Modelle als Diskussionsbasis, für die Dokumentation oder zur Spezifikation von Systemanforderungen dienen. [15] Eine andere Möglichkeit zur Nutzung der UML besteht in der Erstellung von ausführbaren Modellen. Hierbei werden die Diagramme soweit ausformuliert, dass diese direkt in eine Programmiersprache umgewandelt werden können. CODESYS verknüpft beide Anwendungsszenarien durch die Integration der UML in die Entwicklungsumgebung.

Die grün markierten Diagrammtypen, das Klassendiagramm und das Zustandsdiagramm, sind mit der CODESYS UML Erweiterung zusätzlich zu den IEC 61131-3 Editoren im Controller Development System verfügbar. Die Modellierungsmöglichkeiten mit den beiden neu hinzugefügten Editoren werden nachfolgend näher beschrieben.

2.4.2 CODESYS UML - Klassendiagramm

Mit Hilfe eines Klassendiagramms lässt sich die Struktur eines komplexen Systems visualisieren und leicht verständlich darstellen. Es können Klassen entworfen und die Beziehun-

gen zu anderen Klassen definiert werden. Zusammenhänge zwischen den Klassen, wie Vererbungs- oder Implementierungsbeziehungen, werden übersichtlich dargestellt.

Der Klassendiagrammeditor von CODESYS (Abbildung 2.7) bietet alle notwendigen Tools zur Abbildung von objektorientierten Projekten. Neben der Möglichkeit ein bestehendes Projekt analysieren zu können, kann das Klassendiagramm auch als Entwicklungstool für die Struktur neuer Projekte verwendet werden.

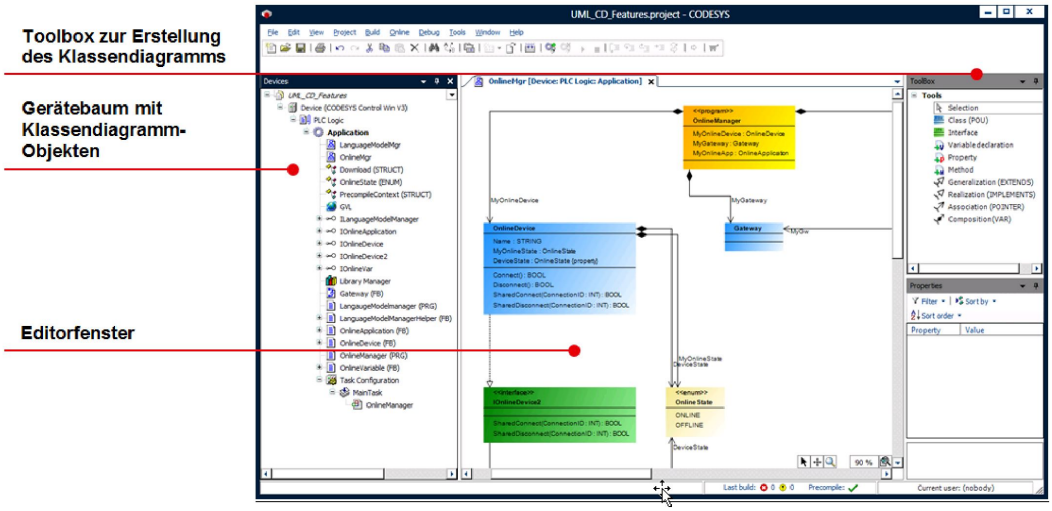

Abbildung 2.7: Programmoberfläche Klassendiagramm in CODESYS [4]

Die Toolbox zur Erstellung eines Klassendiagramms bringt einige neue Werkzeuge in die Entwicklungsumgebung von CODESYS. Diese erfordern eine kurze Erläuterung.

Klasse

Die Klasse, das zentrale Element eines Klassendiagramms, kapselt die Daten und Operationen einer Einheit. Sie stellt einen eigenen Datentyp dar, der instanziiert werden kann.

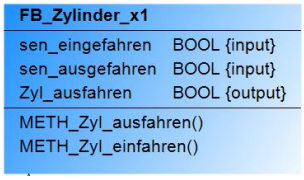

Abbildung 2.8: Klasse eines Pneumatikzylinders

Eine Klasse verfügt über eine eigene Variablendeklaration auf die alle Elemente der Klasse selbst (z.B. Eigenschaften, Methoden oder Actions) zugreifen können. Zusätz-

lich kann eine Klasse Eigenschaften (engl. Properties) besitzen, die bei der Erstellung eine Get- und Set-Methode erhält. Bei einem lesenden oder schreibenden Zugriff auf eine Eigenschaft werden diese Methoden implizit aufgerufen. Unterhalb der Trennlinie (siehe Abbildung 2.8) sind die Methoden und Actions, die zu einer Klasse gehören, aufgelistet.

Schnittstelle

Eine Schnittstelle (engl. Interface) umfasst eine Menge an Methoden-, Action- und Eigenschaftsdeklarationen. Sie ähnelt einem Funktionsbaustein, der keine Variablendeklaration besitzt und keinerlei Implementierung enthält. Gekennzeichnet ist die im Klassendiagrammeditor durch die grüne Farbe und der Kennzeichnung *«interface»* (siehe Abbildung 2.9).

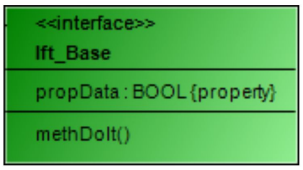

Abbildung 2.9: Beispiel einer Schnittstelle [3]

Ein Interface ist immer dann sinnvoll, wenn Oberklassen so allgemein gehalten sind, dass ihre Implementierungen stark voneinander abweichen können, aber die externe Schnittstelle der Instanzen gleich bleibt. Die Definition eines Interfaces entspricht der Schnittstellenfestlegung zwischen Objekten und der Klasse, die das Interface implementiert, ohne dabei eine Umsetzung der eigentlichen Funktion mitzuliefern.

Globale Variablenliste (GVL)

Eine globale Variablenliste ermöglicht den projektweiten Lese- und Schreibzugriff auf die enthaltenen Variablendefinitionen. Im Editor sind globale Variablenlisten rosa dargestellt und mit der Überschrift *«global»* gekennzeichnet.

Benutzerdefinierter Datentyp (DUT)

Ein benutzerdefinierter Datentyp ist im Editor gelb dargestellt. Handelt es sich dabei um eine Struktur, steht *«struct»* in der Überschrift, ist es eine Aufzählung (engl. Enumeration), steht dort *«enum»*.

Um die Beziehungen zwischen den Objekten des Klassendiagramms näher beschreiben zu können, bietet der Editor verschiedene Relationstypen. Für jede Relationsbeziehung, die in den Editor gezeichnet wird, wird automatisch der entsprechende Code erzeugt. Gleiches gilt für die automatische Zeichnung der Verbindungen, wenn der Code geändert wird. Es stehen vier verschiedene Beziehungstypen zur Verfügung, die in der nachstehenden Auflistung erläutert werden.

Komposition

Eine Komposition drückt ein Enthalten der Klasse aus, auf die der Pfeil zeigt (siehe Abbildung 2.10). Enthalten bedeutet, dass das eine Element eine Instanz des anderen Elements erzeugt.

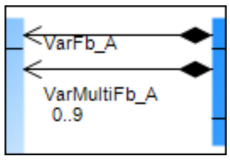

Abbildung 2.10: Komposition [3]

Es ist auch möglich, dass eine Klasse mehrere Instanzen einer anderen Klasse enthält. In diesem Fall enthält der Pfeil zusätzlich die Kardinalität[1]. Für eine Kardinalität größer Eins, wird in der Variablendeklaration ein Array erzeugt.

Assoziation

Die Assoziation drückt ein Kennen einer anderen Klasse aus. Das bedeutet, dass ein Zeiger erzeugt wird, der auf die andere Klasse verweist. In der Variablendeklaration entspricht der Zeiger der *POINTER TO*-Anweisung. Auch hier ist es möglich über die Kardinalität mehrere Zeiger zu erzeugen.

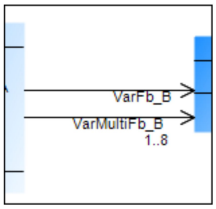

Abbildung 2.11: Assoziation [3]

Generalisierung

Eine Generalisierung beschreibt die Spezialisierung einer Klasse. Im Softwareengineering wird hierbei häufiger der Begriff Vererbung anstelle von Spezialisierung verwendet. Die Vererbung ist eines der wichtigsten Konzepte der Objektorientierung. Dabei werden alle Attribute einer Klasse (Methoden, Actions und Variablendeklarationen) an die erbende Klasse weiter gegeben. Die vererbende Klasse wird meist als Superklasse und die erbende Klasse als Subklasse bezeichnet. Im Editor ist die Verbindung an einer durchgezogenen Linie mit einem Pfeil zur Superklasse zu erkennen.

[1] Anzahl der Elemente

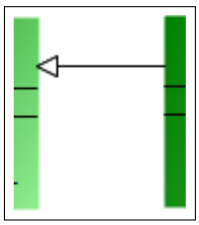

Abbildung 2.12: Generalisierung [3]

Es ist möglich die Methoden und Actions der Superklasse in der Subklasse zu überschreiben und damit neu zu definieren. Daher kann die Subklasse als Erweiterung oder Einschränkung der Superklasse gesehen werden.

Für die Vererbung in CODESYS gibt es Einschränkungen, die es zu beachten gilt:

- Ein Funktionsbaustein kann von einem anderen Funktionsbaustein erben. [3]
- Eine Schnittstelle kann von einer anderen Schnittstelle erben. [3]
- Ein Benutzerdefinierter Datentyp kann von einem anderen benutzerdefinierten Datentyp erben. [3]
- Programme und Funktionen können weder vererben noch erben. [3]

Im Programmcode ist eine Generalisierung an dem Schlüsselwort *EXTENDS* zu erkennen.

Realisierung

Wenn eine POU vom Typ *FUNCTION_BLOCK* von einer Schnittstelle erbt, handelt es sich um eine Realisierung. Die realisierende Klasse erbt dabei die Attribute und Operationen der Schnittstelle. Im Code wird hierfür das Schlüsselwort *IMPLEMENTS* verwendet.

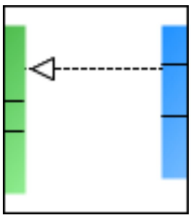

Abbildung 2.13: Realisierung [3]

2.4.3 CODESYS UML - Zustandsdiagramm

„Die möglichen Zustände, Zustandsübergänge, Ereignisse und Aktoren im »Leben« eines Systems, werden in einem Zustandsdiagramm modelliert. Zustandsdiagramme basieren auf dem Konzept der deterministischen, endlichen Automaten.“ [8] Sie sind im Bereich der Programmierung und Spezifikation von ereignisdiskreten Systemen weit verbreitet.

Der Markt bietet bereits kommerzielle und nicht kommerzielle Editoren für Zustandsdiagramme an, die den Programmcode für unterschiedliche Zielplattformen generieren können. CODESYS führt mit der UML-Erweiterung die Programmiersprache UML-Zustandsdiagramm (Statechart SC) ein, die eine direkte Implementierung der Zustandsdiagramme ermöglicht. [3][15]

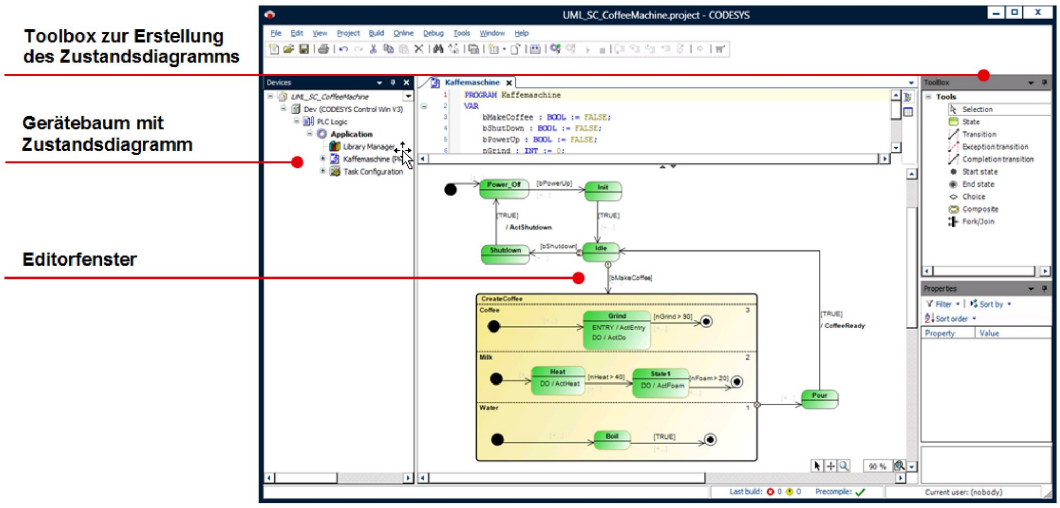

Abbildung 2.14: Programmoberfläche Zustandsdiagramm [4]

Mit dieser grafischen Programmiersprache können in CODESYS Programme, Funktionsbausteine, Funktionen, Methoden, Aktionen oder Eigenschaften erstellt werden. Ein Zustandsdiagramm ist ein Graph, der im Wesentlichen aus Zuständen und Transitionen zur Beschreibung der Systemdynamik besteht. Neben diesen beiden Werkzeugen stellt der Editor noch weitere Werkzeuge zur Verfügung. In der folgenden Auflistung werden alle Elemente des Zustandsdiagrammeditors von CODESYS aufgezeigt.

Startzustand

Ein Startzustand markiert den initialen Zustand und damit den Startpunkt für den Ablauf des Zustandsdiagramms. Startzustände befinden sich auf der obersten Ebene, der Diagrammebene, sowie in zusammengesetzten Zuständen und den Regionen orthogonaler Zustände. Es wird auf jeder dieser Ebenen genau einen Startzustand benötigt. Im Editor ist der Startzustand durch einen schwarzen Kreis gekennzeichnet.

Endzustand

Ein Endzustand markiert das Ende der Ausführung eines Bereichs, der diesen Endzustand enthält. Ein Endzustand muss nicht zwingend vorhanden sein. Abhängig davon in welcher Ebene sich der Endzustand befindet, weicht die Funktionalität voneinander ab.

Wird der Endzustand auf Diagrammebene erreicht, führt dies zu einer Beendigung des Statecharts. Auch durch ein erneutes Aufrufen wird das Zustandsdiagramm nicht fortgesetzt. Erst durch das Setzen der impliziten Variable „ReInit“ wird das Zustandsdiagramm beim nächsten Zyklus wieder von vorne bearbeitet. Ist eine durchgängige Bearbeitung gewünscht, empfiehlt sich die Rückführung zum initialen State durch Transitionen.

Beim zusammengesetzten Zustand markiert der Endzustand den Aussprungpunkt aus dem zusammengesetzten Zustand. Hierbei ist keine Reinitialisierung notwendig.

In einem orthogonalen Zustand führen die Endzustände erst dann zu einem Sprung aus dem orthogonalen Zustand, wenn alle Regionen ihren jeweiligen Endzustand erreicht haben. Bei einem erneuten Aufruf ist ebenfalls keine Reinitialisierung notwendig und alle Regionen starten den Ablauf vom Startzustand aus.

Ein Endzustand ist im Editor durch einen schwarzen Kreis gekennzeichnet, der von einem weiteren, leeren Kreis umgeben ist.

Zustand

Ein Zustand bildet das Hauptelement eines Zustandsdiagramms. Zur Laufzeit werden bei einem Zustand drei Zustände durchlaufen. Der erste Zustand ist das Betreten (Entry-Aktion), der zweite Zustand die Ausführung (Do-Aktion) und der letzte Zustand ist das Verlassen (Exit-Aktion). In jedem dieser drei Zustände lassen sich Methoden oder Aktionen aufrufen. Es ist immer nur einer dieser Zustände zur selben Zeit aktiv.

CODESYS bietet zwei Formen von einfachen Zuständen an: zyklusinterne und normale Zustände. Bei einem normalen Zustand ist der Task, in der er aufgerufen wird, ausschlaggebend für das Schaltverhalten. Wenn die Bedingung für das Schalten in den nächsten Zustand erfüllt ist, wird der Zustandswechsel erst im nächsten Taskzyklus durchgeführt. Das Schaltverhalten eines zyklusinternen Zustands ist unabhängig vom Task. Das bedeutet, dass sobald die Aktionen des Zustands abgeschlossen sind,

die Weiterschaltbedingung geprüft wird. Ist diese erfüllt, wird direkt in den Zielzustand gewechselt.

Zusammengesetzter Zustand

Zusammengesetzte Zustände (engl. Composite States) dienen zur Gruppierung der Zustände, die sie umfassen. Sie können beliebig tief geschachtelt werden. Bedingung hierfür ist, dass es in allen verschachtelten Zuständen nur eine Region gibt. Die Gruppierung von Zuständen kann der logischen, visuellen Einteilung dienen, oder zur Bearbeitung gemeinsamer Fehlerverhalten eingesetzt werden.

Werden Composite States um Regionen erweitert, werden die resultierenden, in Regionen gegliederte Statecharts als orthogonale Zustände bezeichnet. Die Orthogonalen Zustände modellieren parallele Abläufe. Da das Programm zyklisch durchlaufen wird, muss das modellierte parallele Verhalten sequentialisiert werden. Dies geschieht durch eine Vergabe von Prioritäten an die Regionen. In Abbildung 2.15 ist ein Beispiel für orthogonale Zustände zu sehen.

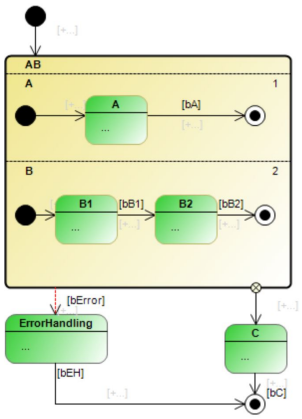

Abbildung 2.15: Zusammengesetzter Zustand [3]

Gabelung/Verbindung

Eine Gabelung/Verbindung kann verwendet werden, um mehrere orthogonale Zustände zeitgleich anzusprechen. Es kann nur eine Transitionsbedingung gestellt werden, da alle von einer Gabelung ausgehenden Transitionen bedingungslos sind.

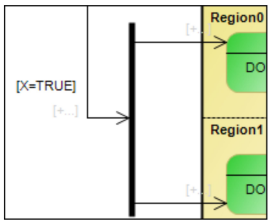

Abbildung 2.16: Gabelung/Verbindung [3]

Auswahl

Eine Auswahl ermöglicht es eine Überwachungsbedingung an eine Variable zu binden, bei der es zu mehreren Ergebnissen führen kann. Sie hat mindestens eine eingehende und eine ausgehende Transition. Der Einsatz einer Auswahl ist auf Diagrammebene, in zusammengesetzten Zuständen und innerhalb orthogonaler Zustände möglich. Die Auswahl wird mit Hilfe eines Beispiels in Abbildung 2.17 verdeutlicht.

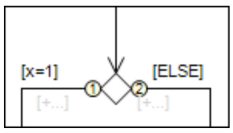

Abbildung 2.17: Auswahl [3]

Transition

Transitionen verfügen über eine Wächterbedingung (Guard) und können zusätzlich eine Aktion (Action) besitzen. Sie dienen der Steuerung des Übergangsverhaltens zwischen Zuständen. Ist die Wächterbedingung erfüllt, kann die Transition überschritten werden und der vorangegangene Zustand wird verlassen. Wenn eine optionale Aktion vorhanden ist, wird diese einmalig vor der Aktivierung des Folgezustands ausgeführt. Es ist möglich, dass ein Zustand mehrere abgehende Transitionen besitzt. In diesem Fall werden den Transitionen Prioritäten zugeordnet, die die Reihenfolge der Evaluierung der Wächterbedingung bestimmt.

Abschlusstransition

Eine Abschlusstransition besitzt im Gegensatz zur normalen Transition keine Wächterbedingung. Sobald der Quellzustand vollständig abgearbeitet ist, schaltet sie in den nächsten Zustand. Abschlusstransitionen sind beim Übergang aus dem Startzustand oder aus einer Gabelung/Verbindung vorgeschrieben. Eine Aktion kann auch bei dieser Transition optional angegeben werden.

Ausnahmetransition

Eine Ausnahmetransition ähnelt funktionell der normalen Transition. Auch sie besitzt eine Wächterbedingung und eine optionale Aktion. Üblicherweise wird die Ausnahmetransition verwendet, um im Fehlerfall oder bei einer Ausnahme (Exception) in einen Zustand zu schalten, der das Fehler- oder Ausnahmeverhalten definiert. Durch ein Schalten der Ausnahmetransition werden alle laufenden Zustände und Subzustände (in zusammengesetzten Zuständen oder orthogonalen Zuständen) ohne Zykluswechsel unterbrochen. Im Zustandsdiagrammeditor ist sie durch einen dünnen Pfeil, der rot gestrichelt ist, gekennzeichnet.

Bei der Erstellung eines Zustandsdiagramms wird automatisch eine Struktur erstellt, die die impliziten Variablen des Zustandsdiagramms enthält. Im Projekt gehört die Struktur zur Instanz des Zustandsdiagramms und trägt den Namen UML_SC_<Objektname>. Die impliziten Variablen geben Auskunft über den aktuellen Status und bieten Möglichkeiten das Zustandsdiagramm zur Laufzeit zu steuern oder zu beobachten.

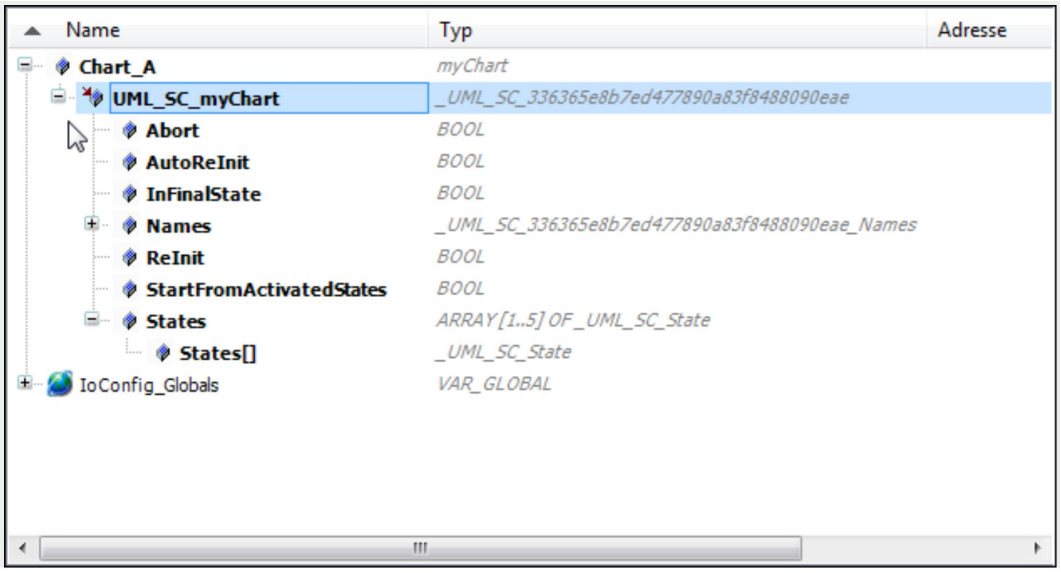

Abbildung 2.18: Implizite Variablen eines Zustandsdiagramms [4]

3 Beschreibung der Modellanlage

Dieses Kapitel dient der Vorstellung der Modellanlage (Abbildung 3.1), die unter Verwendung objektorientierter Programmierung automatisiert werden soll. Die Anlage ist aus dem Bestand des Labors für Automatisierungstechnik der Hochschule für angewandte Wissenschaften Hamburg und findet derzeit keine Anwendung.

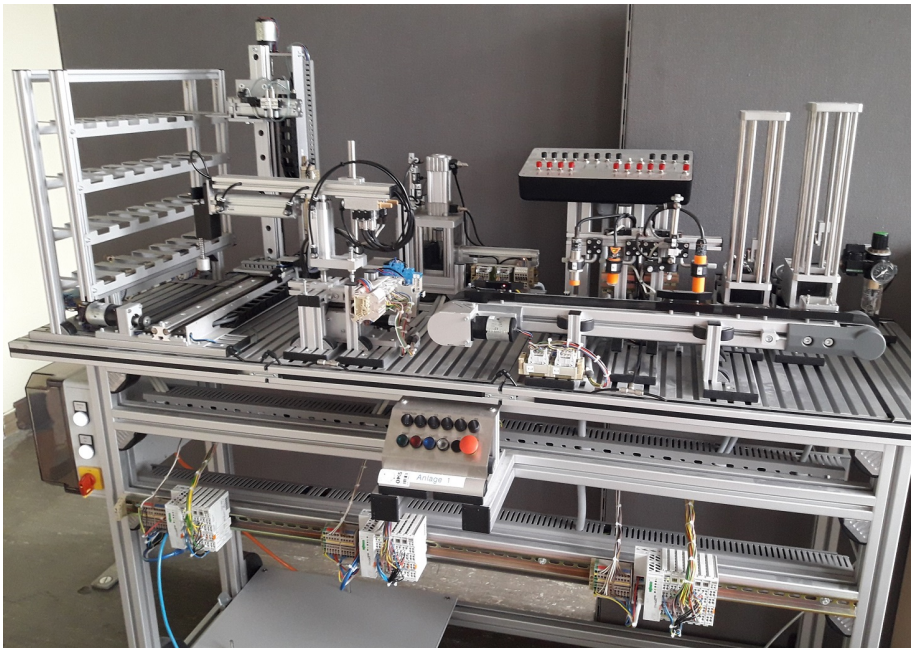

Abbildung 3.1: Modellanlage

Zunächst wird in den folgenden Abschnitten der dynamische Ablauf des Prozesses und die Gliederung der Modellanlage erläutert. Eine genaue Analyse der Anlage ist eine entscheidende Grundlage für die objektorientierte Softwarestrukturierung. Aus diesem Grund werden die Aggregate[1] im einzelnen beschrieben.

[1] Einheit von mehreren zusammenwirkenden Maschinen oder Geräten

3.1 Dynamischer Ablauf des Prozesses

Die fertigungstechnische Aufgabe der Anlage ist die Pressung der in Abbildung 3.2 dargestellten Werkstücke und die anschließende Einlagerung in das Hochregallager.

Abbildung 3.2: Werkstücke

Die Werkstücke werden in Bodenteile und Deckel unterschieden. Das Bodenteil besteht aus Metall und ist an den Befestigungsstiften zu erkennen. Die Deckel sind in weißer und schwarzer Variante vorhanden. Die Werkstücke können in zwei Magazine eingelegt werden, wobei ein Magazin für die Bodenteile und ein Magazin für die Deckel zur Verfügung steht. Der erste Schritt sieht einen Transport der Werkstücke zur Sensoreinheit vor. Hierfür werden sie aus dem Magazin ausgestoßen und auf dem Transportband positioniert. Sollte das Werkstück die Prüfung durch die Sensoreinheit nicht bestehen, transportiert das Band das Werkstück in die entgegengesetzte Richtung, bis es vom Band fällt. Nach einer erfolgreichen Prüfung des Werkstücks wird dieses durch den Schwenkarm zur Presse transportiert und dort abgelegt. Nun kann der Deckel nach der gleichen Vorgehensweise zur Presse transportiert werden. Befinden sich beide Werkstücke in der Ablage der Presse, kann der Pressvorgang gestartet werden. Im Anschluss an diesen Prozess transportiert der Schwenkarm das Werkstück zum Hochregallager, wo es auf der Lagereinheit abgelegt wird. Der letzte Schritt ist die Einlagerung des Werkstücks beginnend in der untersten Etage, bis das Lager vollständig gefüllt ist. Das Auslagern von Werkstücken ist nicht Bestandteil dieser Bachelorarbeit.

3.2 Gliederung der Modellanlage

Bei genauerer Betrachtung des Prozesses lassen sich fünf Module herausarbeiten: Das Magazin, das Transportband mit Sensoreinheit, der Schwenkarm, die Presse und das Hochregallager. Da die Signale aller Aktoren, Sensoren und Bedienelemente des Magazins und des

Transportbands mit Sensoreinheit, sowie des Schwenkarms und der Presse jeweils mit einem gemeinsamen EtherCAT-Koppler der Firma WAGO Kontakttechnik verbunden sind, werden diese Module zu einer Einheit zusammengefasst. Es resultieren demnach drei Module in die die Modellanlage von nun an gegliedert wird. Die Module werden in den Abschnitten von 3.3 bis 6.7.6 im einzelnen beschrieben.

3.3 Magazin und Förderband

Das Magazin und das Förderband dienen der Zuführung von Werkstücken zum Pressvorgang und der anschließenden Einlagerung. Der Anlagenteil ist in Abbildung 3.3 dargestellt.

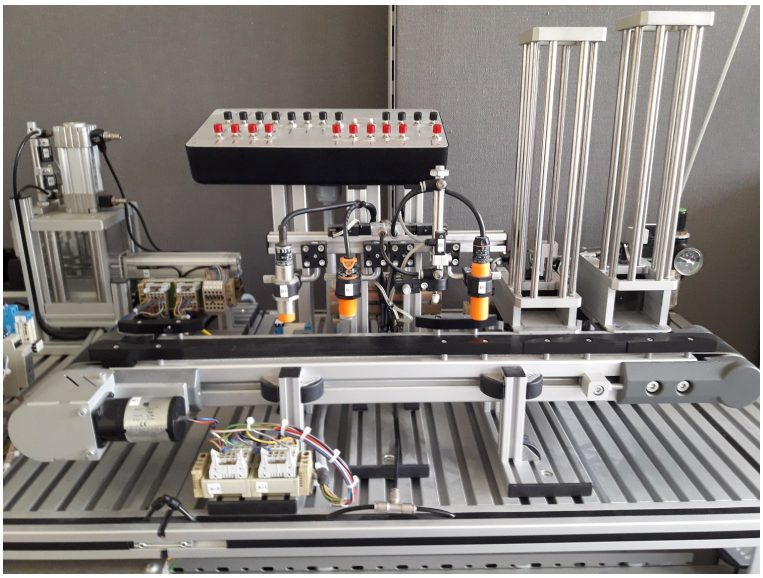

Abbildung 3.3: Magazin und Förderband mit Sensoreinheit

Das Modul lässt sich in drei Subkomponenten aufteilen, die für den Prozess relevant sind. Die erste Komponente ist die Lagereinheit, die die Werkstücke beinhaltet. Die zweite Komponente ist das Transportband mit der Sensoreinheit zur Überprüfung der Werkstücke. Die letzte Komponente ist die Bedieneinheit der Anlage, die alle für den Benutzer notwendigen Steuersignale beinhaltet.

3.3.1 Lagereinheit

Die Lagereinheit besteht aus zwei Magazinen, deren Aufbau nahezu identisch ist. Das rechte Magazin enthält die Bodenteile mit den entsprechenden Befestigungsstiften, daher ist dieses Magazin etwas größer. Die Werkstücke werden von oben in die Magazine eingelegt und stapeln sich dort. Für die Erkennung, ob ein Werkstück eingelegt wurde, befindet sich ein Positionsschalter im Materialspeicher, der durch das Werkstück aktiviert wird. Des Weiteren besitzt jedes Magazin einen doppeltwirkenden Pneumatikzylinder mit einer Endlagenüberwachung für den eingefahrenen und ausgefahrenen Zustand. Diese haben die Aufgabe, die Werkstücke aus dem Materiallager auszustoßen und auf dem Transportband zu positionieren. Die Pneumatikzylinder werden über zwei 5/2-Wegeventile angesteuert und können durch boolesche Variablen in beide Richtungen getrieben werden.

3.3.2 Transportband mit Sensoreinheit

Das Transportband verfügt über einen DC-Getriebemotor der durch zwei boolesche Variablen linksdrehend oder rechtsdrehend angesteuert werden kann. Befindet sich ein Werkstück auf dem Transportband, wird es zur Sensoreinheit transportiert. Die Sensoreinheit besteht aus insgesamt drei Sensoren und einem Prüfzylinder mit Reedschalter. Die Prüfung des Werkstücks geschieht in nachstehender Reihenfolge.

1. Der kapazitive Sensor B1 detektiert das Werkstück als erstes und dient lediglich der Feststellung, ob ein Werkstück die Sensoreinheit erreicht.
2. Danach wird das Werkstück unterhalb des Prüfzylinders positioniert und der Prüfzylinder wird ausgefahren. Der Reedschalter B2 ist so am Zylinder positioniert, dass festgestellt werden kann, ob es sich bei dem darunterliegenden Werkstück um ein Bodenteil oder Deckel handelt. Nur bei einem Bodenteil meldet der Reedschalter ein positives boolesches Signal zurück.
3. Mit Hilfe der Lichtschranke B3 kann die Farbe der Deckel bestimmt werden. Bei einem weißen Deckel liefert die Auswertung der Lichtschranke eine logische eins zurück. Um Fehler bei der Auswertung zu vermeiden, sollte das Werkstück während der Überprüfung der Lichtschranke still stehen.
4. Der letzte Sensor ist der induktive Sensor B4. Er dient der Feststellung, das Werkstück aus Metall oder Kunststoff besteht.

Sollte das Werkstück nicht den in Kapitel 4 definierten Anforderungen entsprechen, wird es in entgegengesetzter Transportrichtung vom Band befördert. Ansonsten wird es weiter zur Endlage des Transportbands befördert, wo es durch den Schwenkarm abgeholt werden kann. Ob das Werkstück die Endlage erreicht hat, wird durch einen Sensor überwacht.

3.3.3 Bedieneinheit

Die Bedieneinheit enthält alle für den Bediener relevanten Steuersignale der Modellanlage und ist in Abbildung 3.4 zu sehen.

Abbildung 3.4: Bedieneinheit der Modellanlage

Die wichtigsten Bedienelemente sind die drei Taster Start, Stop und Quit, sowie der Auswahlschalter für die Anwahl des Hand- oder Automatikbetriebs. Da die Anlage weitestgehend automatisch arbeiten soll, wird auf die Verwendung weiterer Schalter vorerst verzichtet. Diese können allerdings bei Bedarf mit Funktionen belegt werden.

Die Bedieneinheit verfügt zusätzlich über Leuchtelemente, die den Zustand der Anlage visualisieren sollen. Welcher Zustand welche Leuchtmeldungen hervorruft, ist in Tabelle 3.1 dokumentiert.

Anlagenzustand	Leuchtmelder
Referenzfahrt	S6 blinkt
Anlage Bereit	Start blinkt
Hand/Automatik Betrieb	Start leuchtet
Stop	Stop leuchtet
Stop / Quittierbereit	Stop / Quit leuchten

Tabelle 3.1: Übersicht der Leuchtmelder

3.3.4 Prozessvariablen

Da für die Modellanlage keine Dokumentation verfügbar ist, werden alle Prozessvariablen der Anlage empirisch ermittelt und in den folgenden Tabellen festgehalten. Die angegebenen Adressen beziehen sich auf die in CODESYS angegeben Hardwareadressen nach Einbindung der EtherCAT-Koppler in die Entwicklungsumgebung.

Eingangsvariable	Adresse	Datentyp
Not-Aus (low active)	IX4.0	BOOL
Stop (low active)	IX4.1	BOOL
Hand	IX4.2	BOOL
Quit	IX4.3	BOOL
Auto	IX4.4	BOOL
S6	IX4.5	BOOL
Start	IX4.6	BOOL
Not-Aus	IX4.7	BOOL
Schalter S1	IX5.0	BOOL
Schalter S5	IX5.1	BOOL
Schalter S2	IX5.2	BOOL
Ausschub Magazin Boden ausgefahren B1	IX5.3	BOOL
Schalter S3	IX5.4	BOOL
Ausschub Magazin Boden eingefahren B2	IX5.5	BOOL
Schalter S4	IX5.6	BOOL
Magazin Boden Teil vorhanden	IX5.7	BOOL
Ausschub Magazin Deckel ausgefahren B1	IX6.0	BOOL
Prüfeinheit B2	IX6.1	BOOL
Ausschub Magazin Deckel eingefahren B2	IX6.2	BOOL
Lichtschranke B3	IX6.3	BOOL
Magazin Deckel Teil vorhanden	IX6.4	BOOL
Induktiver Sensor B4	IX6.5	BOOL
Kapazitiver Sensor B1	IX6.6	BOOL
Endlage Förderband	IX6.7	BOOL

Tabelle 3.2: Eingänge Magazin und Band

Ausgangsvariable	Adresse	Datentyp
Beleuchtung Taster Start	QX4.0	BOOL
Ausschub Magazin Boden einfahren	QX4.1	BOOL
Beleuchtung Taster Stop	QX4.2	BOOL
Ausschub Magazin Boden ausfahren	QX4.3	BOOL
Beleuchtung Taster Quit	QX4.4	BOOL
Ausschub Magazin Deckel einfahren	QX4.5	BOOL
Beleuchtung Taster S6	QX4.6	BOOL
Ausschub Magazin Deckel ausfahren	QX4.7	BOOL
Prüfeinheit ausfahren	QX5.0	BOOL
Prüfeinheit einfahren	QX5.2	BOOL
Band Linkslauf	QX5.4	BOOL
Band Rechtslauf	QX5.6	BOOL

Tabelle 3.3: Ausgänge Magazin und Band

3.4 Schwenkarm und Presse

Der Schwenkarm und die Presse sind in Abbildung 3.5 dargestellt. Die Presse presst jeweils ein Bodenteil und einen Deckel zu einem Werkstück zusammen, welches dann bereit zum Einlagern ist. Der Schwenkarm ist das Bindeglied zwischen dem Transportband, der Presse und dem Hochregallager.

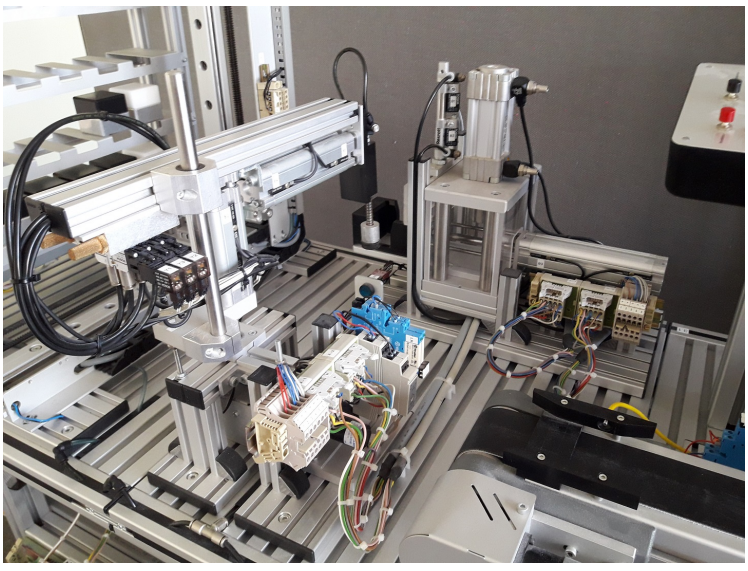

Abbildung 3.5: Schwenkarm und Presse

3.4.1 Schwenkarm

Der Schwenkarm ist mit einer drehbaren Achse ausgestattet, die von einem DC-Getriebemotor angetrieben wird. Diese Achse ermöglicht das Anfahren der drei Ablage- und Entnahmepositionen. Um die Position zu kontrollieren, ist an jeder Station ein Gewindestift montiert, der einen induktiven Sensor am Schwenkarm aktiviert. Da lediglich ein Sensor vorhanden ist, kann anhand des Signales nicht entschieden werden, an welcher Station sich der Schwenkarm befindet. Neben der Drehachse verfügt der Schwenkarm über zwei einfachwirkende Pneumatikzylinder. Im Gegensatz zum doppeltwirkenden Pneumatikzylinder können einfachwirkende lediglich Arbeit in eine Richtung verrichten. Bei Wegnahme des Ansteuerungssignals wird der Zylinder durch eine Rückstellfeder wieder in die Ruheposition befördert. Beim Schwenkarm dient ein Zylinder dem vertikalen Anheben und Senken der Entnahmevorrichtung, weshalb dieser als Vertikalzylinder bezeichnet wird. Der andere dient der horizontalen Bewegung der Entnahmevorrichtung. Da die Postionen der beiden Zylinder für einen sicheren Betrieb der Anlage von Bedeutung sind, sind beide mit Endlagensensoren ausgestattet.
Um die Werkstücke befördern zu können, ist an der Entnahmevorrichtung ein Vakuumsauger installiert. Dieser kann über ein 2/2-Wegeventil aktiviert und deaktiviert werden.

3.4.2 Presse

Die Presse verfügt über eine Ausschubvorrichtung, in der ein Bodenteil und ein Deckel platziert werden können. Sie dient der Führung der Werkstücke und ermöglicht einen sicheren Pressvorgang. Bewegt wird die Ausschubvorrichtung durch einen doppeltwirkenden Pneumatikzylinder. Ist der Ausschub eingefahren, kann eine Sicherheitstür, die ebenfalls durch einen doppeltwirkenden Pneumatikzylinder bewegt wird, geschlossen werden. Sie verhindert ein Eingreifen in den Gefahrenbereich während des Pressvorgangs. Der letzte Aktor der Presse ist der Stempel, der den eigentlichen Pressvorgang realisiert. Der Stempel ist vertikal angebracht und drückt die beiden Werkstücke zusammen. Die Sicherheitstür und die Ausschubvorrichtung sind beide mit Endlagensensoren ausgestattet. Lediglich beim Stempel wurde auf diese verzichtet.

3.4.3 Prozessvariablen

Das Ergebnis der empirischen Ermittlung aller Prozessvariablen sind in Tabelle 3.4 und 3.5 festgehalten.

Eingangsvariable	Adresse	Datentyp
Positionssensor Drehachse	IX4.0	BOOL
Horizontalzylinder ausgefahren	IX4.1	BOOL
Vertikalzylinder unten	IX4.2	BOOL
Vertikalzylinder oben	IX4.4	BOOL
Horizontalzylinder eingefahren	IX4.6	BOOL
Sicherheitstür eingefahren	IX4.7	BOOL
Sicherheitstür ausgefahren	IX5.0	BOOL
Taster Presse S6	IX5.1	BOOL
Ausschub ausgefahren	IX5.2	BOOL
Ausschub eingefahren	IX5.4	BOOL
Taster Presse S5	IX5.6	BOOL

Tabelle 3.4: Eingänge Schwenkarm und Presse

Ausgangsvariable	Adresse	Datentyp
Vertikalzylinder ausfahren	QX4.0	BOOL
Schwenkarm Linkslauf	QX4.1	BOOL
Sauger einschalten	QX4.2	BOOL
Sicherheitstür öffnen	QX4.3	BOOL
Horizontalzylinder ausfahren	QX4.4	BOOL
Sicherheitstür schließen	QX4.5	BOOL
Schwenkarm Rechtslauf	QX4.6	BOOL
Ausschub ausfahren	QX4.7	BOOL
Ausschub einfahren	QX5.0	BOOL
Stempel einfahren	QX5.1	BOOL
Beleuchtung Taster S5	QX5.2	BOOL
Beleuchtung Taster S6	QX5.4	BOOL
Stempel ausfahren	QX5.6	BOOL

Tabelle 3.5: Ausgänge Schwenkarm und Presse

3.5 Hochregal

Das Hochregal bildet das letzte Modul des hier vorgestellten Prozesses. Die fertig gepressten Werkstücke werden einzeln in das in Abbildung 3.6 zu sehende Hochregallager eingelagert.

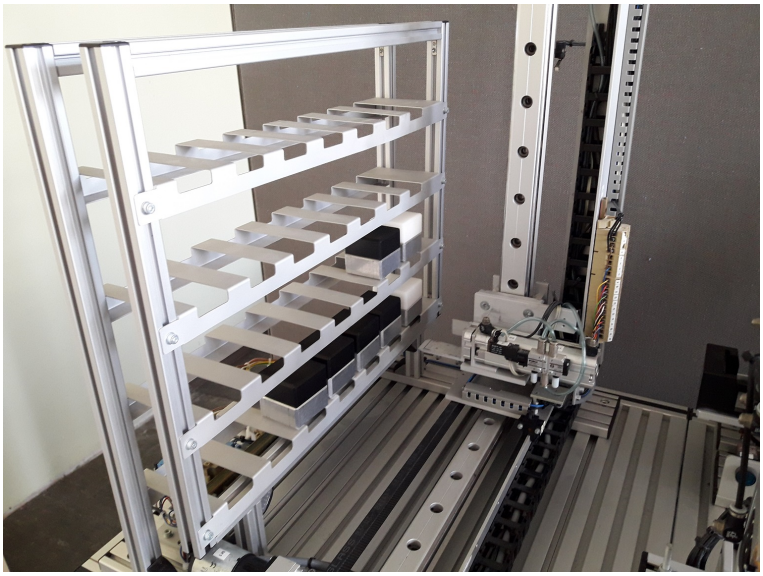

Abbildung 3.6: Hochregal

3.5.1 Verfahreinheit

Für die Positionierung des Ausschubs ist die Verfahreinheit mit einem DC-Getriebemotor für die horizontale Bewegung und einem für die vertikale Bewegung ausgestattet. Die vertikalen und horizontalen Endlagen sind mit Positionsschaltern ausgestattet, die ein Überschreiten der Endlagen verhindern.

Um die genaue Position zu bestimmen, sind zwei Gabellichtschranken am beweglichen Teil der Verfahreinheit angebracht (siehe Abbildung 3.7).

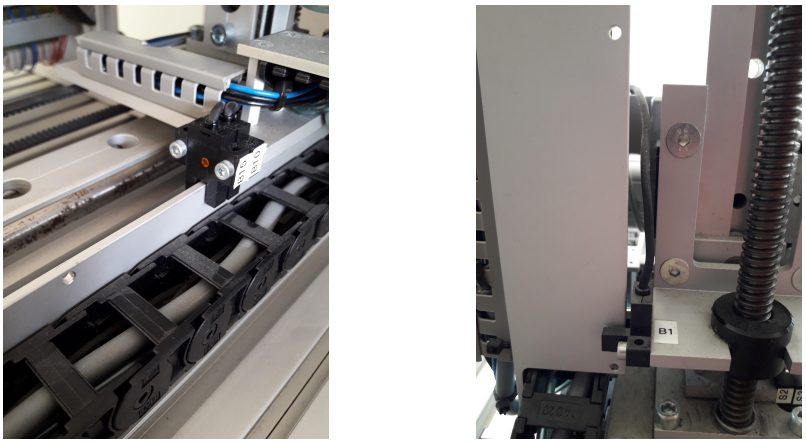

Abbildung 3.7: Horizontale und vertikale Gabellichtschranken

Die Bohrungen, die in die fest montierten Schienen gebohrt sind, werden durch die Gabellichtschranken detektiert und zeigen die Lagerplätze an. In horizontaler Richtung sind sieben Lagerplätze vorhanden und damit sieben Bohrungen in der Schiene. In vertikaler Richtung sind vier Etagen verfügbar. Der sechste Lagerplatz ist gleichzeitig die Übergabestelle für die Werkstücke vom Schwenkarm.

3.5.2 Ausschub

Der Ausschubzylinder dient als Ablagestelle des vom Schwenkarm kommenden Werkstücks. Ist die gewünschte Lagerposition angefahren, fährt der Ausschubzylinder aus. Erst wenn die Verfahreinheit die vertikale Achse absenkt und das Werkstück damit im Lager aufliegt, kann der Zylinder wieder eingefahren werden und die Einlagerung ist abgeschlossen. Der Ausschubzylinder ist ein einfachwirkender Zylinder mit Endlagenüberwachung.

3.5.3 Prozessvariablen

Auch für das letzte Modul wurden die Prozessvariablen empirisch ermittelt.

Eingangsvariable	Adresse	Datentyp
Positionssensor Regal vertikal	IX4.0	BOOL
Ausschub eingefahren	IX4.1	BOOL
Endlage vertikal - unten	IX4.2	BOOL
Positionssensor Regal horizontal B10	IX4.3	BOOL
Endlage vertikal - oben	IX4.4	BOOL
Endlage horizontal - Übergabe	IX4.5	BOOL
Ausschub ausgefahren	IX4.6	BOOL
Endlage horizontal - Grundposition	IX4.7	BOOL
Positionssensor Regal horizontal B15	IX5.0	BOOL

Tabelle 3.6: Eingänge Hochregal

Ausgangsvariable	Adresse	Datentyp
Vertikalachse aufwärts	QX4.0	BOOL
Horizontalachse Übergabe	QX4.1	BOOL
Vertikalachse abwärts	QX4.2	BOOL
Ausschub ausfahren	QX4.4	BOOL
Horizontalachse Grundstellung	QX4.6	BOOL

Tabelle 3.7: Ausgänge Hochregal

4 Analyse der Anforderungen

Die Anforderungsanalyse ist ein wichtiger Prozess der Systementwicklung. Durch diese Analyse werden vorab Anforderungen definiert, strukturiert sowie dokumentiert. Die folgenden zwei Abschnitte gliedern die Anforderungsanalyse in funktionale und nicht-funktionale Anforderungen.

4.1 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen legen fest, wie die Automatisierungslösung umgesetzt werden soll.

- Das Steuerungsprogramm ist mit CODESYS Version 3.5 SP11 und der zugehörigen CODESYS UML Erweiterung zu realisieren.
- Die erstellte Software soll auf drei Raspberry Pis implementiert werden.
- Während des Anlagenbetriebs im Automatikmodus ist kein Eingreifen eines Bedieners notwendig.

4.2 Funktionale Anforderungen

In den funktionalen Anforderungen wird festgehalten, was die Anlage leisten soll. Nach dem Start der Anlage wird automatisch eine Initialisierungsfahrt von allen Komponenten durchgeführt, bei der sie ihre vorgesehene Grundstellung einnehmen. Lediglich der Schwenkarm muss manuell durch den Bediener zur Warteposition an der Presse gefahren werden. Hierfür dient ein fest an der Anlage montiertes Bedienpanel, das neben dem Schwenkarm auch alle anderen Aktoren der Modellanlage steuern kann.

Sind alle Anlagenmodule initialisiert, kann zwischen dem Hand- und Automatikbetrieb gewählt werden. Während beim Handbetrieb in der Software alleinig darauf geachtet werden muss, dass kein Aktor aktiv von der Steuerung angesprochen wird und somit die Anlage über

das Bedienpanel kontrolliert werden kann, gelten für den Automatikbetrieb Anforderungen, die in den nachfolgenden Abschnitten dokumentiert sind.

4.2.1 Magazin und Transportband

Dieses Anlagenmodul verarbeitet alle Eingaben des Bedieners und dient der Hauptsteuerung der Anlage. Des Weiteren werden hier die aus den Magazinen kommenden Werkstücke an den Prozess geführt. Für die Überprüfung der Werkstücke gelten die nachstehenden Prüfkriterien.

- Die Reihenfolge von Bodenteil und Deckel muss eingehalten werden.
- Die Bodenteile müssen aus Metall sein.
- Die Deckel sind nur aus Kunststoff zulässig.
- Werden Teile erkannt, die nicht den Prüfkriterien entsprechen, müssen diese automatisch vom Band befördert werden.

4.2.2 Schwenkarm und Presse

Der Schwenkarm reagiert auf die Signalisierung eines abholbereiten Teils vom Transportband und befördert dieses zur Presse. Nach dem Pressvorgang wird das fertige Werkstück durch den Schwenkarm zum Hochregal transportiert. Für diesen Prozess lassen sich zwei wichtige Anforderungen definieren.

- Ist die Presse belegt, dürfen keine weiteren Teile mehr vom Transportband abgeholt werden.
- Bevor der Schwenkarm ein Werkstück an das Hochregal übergeben darf, muss eine Freigabe erteilt werden, um Kollisionen zu vermeiden.

4.2.3 Hochregal

Das Hochregal nimmt die fertig gepressten Werkstücke an und lagert diese ein. Die Grundposition der Verfahreinheit ist in der Horizontale in der hinteren Endlage und in der Vertikale in der unteren Endlage definiert.

- Nach einem ausgeführten Befehl verweilt die Verfahreinheit in der Grundposition.
- Das Hochregallager soll erst die unterste horizontale Reihe mit Werkstücken belegen und dann in der nächsthöhere Ebene fortfahren.
- Bevor die Verfahreinheit in die Position zur Aufnahme eines Werkstücks fährt, muss sie, um eine Kollision zu vermeiden, eine Freigabe vom Schwenkarm erhalten.

4.2.4 Sicherheitsrelevantes Verhalten

Wird der Not-Aus-Schalter der Anlage betätigt oder die Anlage während des Automatikbetriebs gestoppt, muss das Verhalten der Anlage definiert sein. Der aktuelle Zustand soll dabei gespeichert werden, sodass die Anlage beim Wiedereinschalten die Aktionen fortsetzt. Die folgenden Punkte bestimmen das Verhalten der Aktoren für die gesamte Anlage.

- Doppeltwirkende Pneumatikzylinder werden nicht mehr mit Druckluft beaufschlagt.
- Einfachwirkende Pneumatikzylinder behalten ihre aktuelle Position bei.
- Rotierende Teile, die durch einen Motor angetrieben werden, stoppen.

5 Hardwarekonfiguration

Bevor mit der eigentlichen Erstellung der Software begonnen werden kann, müssen der Aufbau des Netzwerks und die Hardware bekannt sein. Dieses Kapitel erläutert das Zusammenwirken der verschiedenen Hardware-Komponenten und deren Anbindung in die CODESYS Entwicklungsumgebung.

5.1 Netzwerkaufbau

Wie bereits in Abschnitt 3.2 beschrieben, sind alle Signale der drei Anlagenmodule mit jeweils einem Feldbuskoppler der Firma WAGO Kontakttechnik verbunden. Für jedes Modul ist ein Raspberry Pi vorgesehen, auf dem eine Soft-SPS die Steuerungsaufgabe übernimmt.

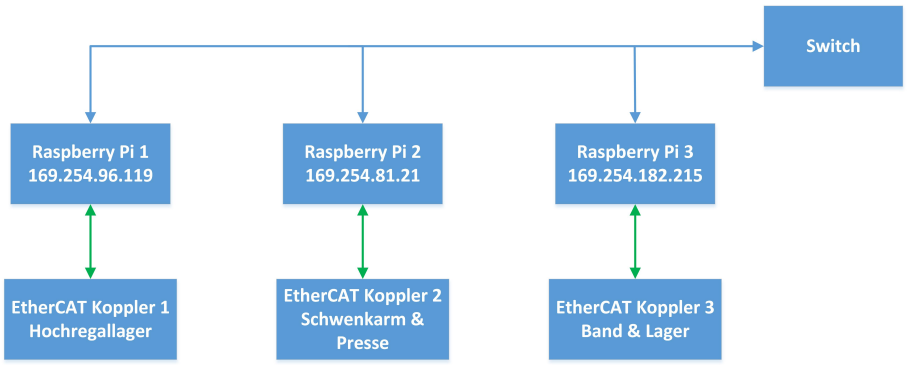

Abbildung 5.1: Netzwerkplan

Die Kommunikation zwischen den Feldbuskopplern und den Raspberry Pis (grüne Verbindungslinien, vlg. Abbildung 5.1) wird mit Hilfe von EtherCAT realisiert. Die blauen Verbindungen, die die einzelnen Raspberry Pis miteinander konnektieren, stellen eine Ethernet Verbindung dar. Über die freien Slots am Switch kann der Entwicklungs-PC in das Netzwerk eingebracht werden.

5.2 Installation der Feldbuskoppler

Der Feldbuskoppler 750-354 verbindet das feldbusunabhängige WAGO-I/O-SYSTEM 750 mit dem Feldbussystem EtherCAT. Der Feldbuskoppler kann durch die Anreihung von Busklemmen um digitale und analoge Ein- und Ausgänge sowie um Sonderfunktionen erweitert werden. Alle Busklemmen werden in einem Knoten zusammengefasst und ausgewertet. Das entstehende Prozessabbild wird in ein Prozessabbild der Eingänge (PAE) und ein Prozessabbild der Ausgänge (PAA) unterschieden. Über den Feldbusknoten werden die erzeugten Daten an die übergeordnete Steuerung gegeben und dort ausgewertet. Für den Anschluss an das Feldbussystem verfügt die EtherCAT-Baugruppe über zwei RJ45 Ports, wovon in diesem Anwendungsfall nur einer notwendig ist, da der Feldbuskoppler neben der Steuerung den einzigen Teilnehmer im Netzwerk darstellt. [17]

Um auf diese Baugruppe innerhalb der Entwicklungsumgebung zugreifen zu können, muss sie der Geräte-Repository von CODESYS hinzugefügt werden. Dafür wird die Gerätebeschreibungsdatei benötigt, welche auf der Herstellerseite heruntergeladen werden kann. [16]

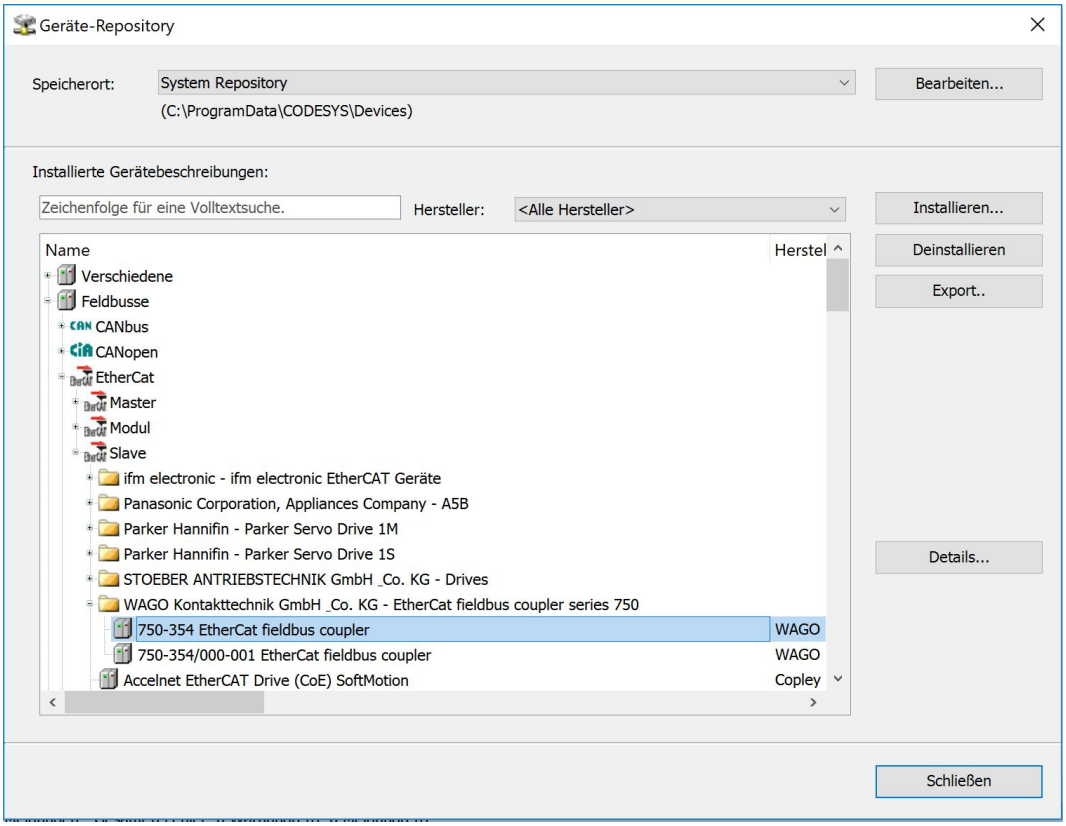

Abbildung 5.2: Geräte-Repository

Die Gerätebeschreibungsdatei enthält das „WAGO_750_354.xml“ File, welches bei der Installation auszuwählen ist. Ist der 750-354 EtherCat fieldbus coupler in der Geräte-Repository verfügbar (siehe Abbildung 5.2), war die Installation erfolgreich.

5.3 Installation der Raspberry Pis

Damit der Raspberry Pi als Steuerung verwendet werden kann, müssen zwei Schritte durchgeführt werden. Der erste Schritt beinhaltet die Installation der „CODESYS Control for Raspberry Pi SL“. Dabei handelt es sich um ein an den Raspberry Pi angepasstes CODESYS Control Laufzeitsystem, das es ermöglicht, den Raspberry Pi als SPS zu programmieren. Der Download der Installationsdateien erfolgt über die Homepage von CODESYS. [2] Wurde die Installation in der Geräte-Repository erfolgreich durchgeführt, wird „CODESYS Control for Raspberry Pi“ zu den Steuerungen hinzugefügt und ist für zukünftige Projekte auswählbar.

Im zweiten Schritt muss die CODESYS Runtime auf dem Raspberry Pi installiert werden. Dies ist direkt aus der CODESYS Entwicklungsumgebung möglich. Unter *„Tools“* $\rightarrow$ *„Update Raspberry Pi“* wird das in Abbildung 5.3 zu sehende Fenster geöffnet.

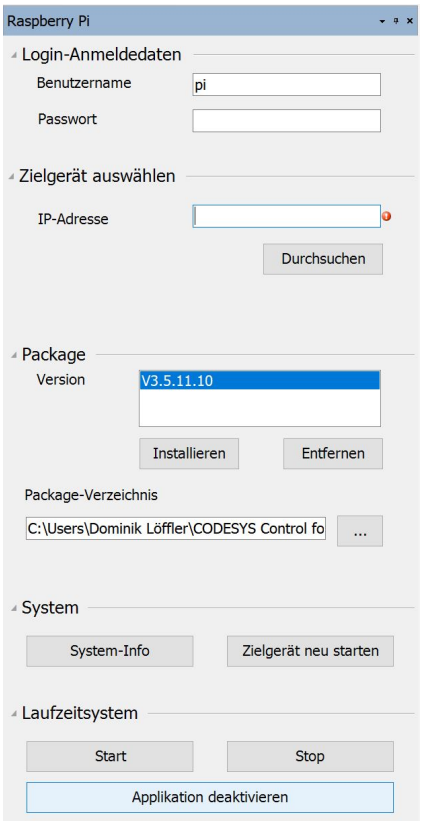

Abbildung 5.3: Update Raspberry Pi

Es ist darauf zu achten, dass der Raspberry Pi eingeschaltet ist und eine Ethernet-Verbindung zum PC besteht. Trägt man den Benutzernamen, das Passwort sowie die IP-Adresse des Raspberry Pi ein, kann die CODESYS Control for Raspberry Pi Runtime installiert werden. Erst nach der Installation wird der Raspberry Pi als Steuerung erkannt und ist im Netzwerk sichtbar.

5.4 Einbinden der Hardware in CODESYS

Nach der Installation aller Hardware-Komponenten in der Entwicklungsumgebung erfolgt die Erstellung des eigentlichen Projekts. Ausgangspunkt ist dabei ein leeres Projekt in CODESYS.

5.4.1 Hinzufügen der Steuerungen

Die drei Raspberry Pi werden in der leeren Geräteansicht durch *Rechtsklick* → *„Gerät anhängen“* und der Auswahl der zuvor installierten CODESYS Control for Raspberry Pi hinzugefügt.

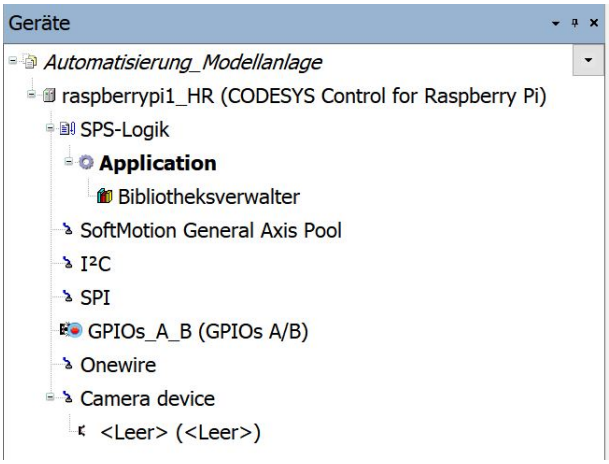

Abbildung 5.4: Raspberry Pi in der Geräteansicht

Abbildung 5.4 zeigt den hinzugefügten Raspberry Pi in der Geräteansicht. Unter *„Application“* können alle Objekte zur Erstellung des Steuerprogramms hinzugefügt werden. Dies wird in Kapitel 6 beschrieben. Die vergebenen Namen der Steuerungen für die Module lauten:

- raspberrypi1_HR: Steuerung für das Hochregal
- raspberrypi2_PR: Steuerung für den Schwenkarm und die Presse
- raspberrypi3_MA: Steuerung für das Transportband und das Magazin

5.4.2 Einbinden der Feldbuskoppler

Damit die Feldbuskoppler in das Projekt eingebunden werden können, muss vorerst zu jeder Steuerung ein EtherCAT Master hinzugefügt werden. Durch einen *Rechtsklick* auf die Steuerung kann unter *„Gerät anhängen“* der EtherCAT Master angehängt werden. Automatisch wird zu dem EtherCAT Master ein entsprechender Task erstellt, der die zyklische Kommunikation sicherstellt. Die Einstellungen können bei den Standardeinstellungen belassen werden.

Da die Steuerung nun über einen EtherCAT Master verfügt, kann der Feldbuskoppler mit den zugehörigen Busklemmen als Slave angehängt werden. Dabei besteht die Möglichkeit, die Geräte wie bekannt einzeln aus dem Geräteverzeichnis hinzuzufügen oder sie automatisch auslesen zu lassen. Für die Suche der Geräte im Netzwerk, die über einen *Rechtsklick* auf den EtherCAT Master gestartet werden kann, muss eine Ethernet-Verbindung zum Feldbuskoppler bestehen.

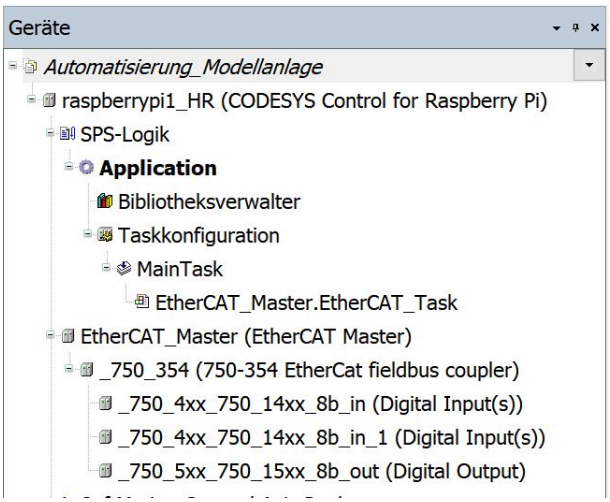

Abbildung 5.5: Feldbuskoppler des Hochregals in der Geräteansicht

Die Ansicht des Raspberry Pis mit eingebundenen EtherCAT Master ist exemplarisch für das Hochregal in Abbildung 5.5 dokumentiert. Für die Steuerungen der beiden verbleibenden Module wird die Konfiguration auf dieselbe Weise durchgeführt.

5.4.3 Kommunikation zwischen den Raspberry Pis

Im letzten Schritt der Konfiguration des Projekts soll die Kommunikation zwischen den Raspberry Pis vorbereitet werden. CODESYS bietet für dieses Vorhaben Netzwerkvariablenlisten an, die einen unidirektionalen Datenaustausch ermöglichen. Die Netzwerkvariablenlisten müssen dabei immer paarweise als Empfänger- und Senderliste in der Geräteansicht eingefügt werden. Beispielhaft soll die Erstellung der Variablenlisten für den Datenaustausch vom Hochregallager zum Transportband und Magazin durchgeführt werden.

Hinzufügen einer Sender-Netzwerkvariablenliste

Die Variablenlisten können in der Geräteansicht zur Application der jeweiligen Steuerung durch *Rechtsklick* → *„Objekt hinzufügen“* → *„Netzwerkvariablenliste (Sender)“* hinzugefügt werden. Es öffnet sich der in Abbildung 5.6 dargestellte Dialog.

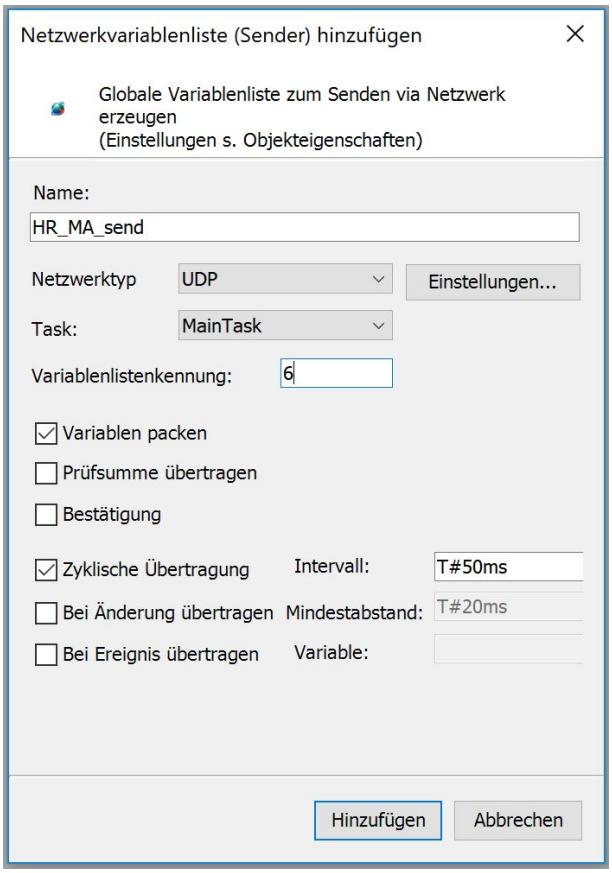

Abbildung 5.6: Netzwerkvariablenliste (Sender) Dialog

In dem Fenster werden alle Konfigurationsmöglichkeiten für die Kommunikation angezeigt. Die wichtigsten Einstellungen sind die Festlegung des Netzwerktyps auf UDP und die Vergabe einer Variablenlistenkennung, die nur einmal im Projekt vorkommen darf. Die weiteren Einstellungen können bei den Standardeinstellungen belassen werden. Deren Funktion soll jedoch kurz erläutert werden.

Es empfiehlt sich den Haken bei *„Variablen packen“* gesetzt zu lassen, da so die Daten in einem gemeinsamen Paket zusammengefasst werden. Anderenfalls wird für jede Variable ein eigenes Paket erstellt, was zu einer hohen Kommunikationsdichte führt. Des Weiteren besteht die Möglichkeit zur Übertragung einer Prüfsumme. Mit dieser kann der Empfänger

kontrollieren, ob das Telegramm vollständig ist. Stimmen die Prüfsummen nicht überein, wird das Datenpaket verworfen. Durch das Aktivieren der Bestätigung fordert der Sender für jedes gesendete Datenpaket eine Bestätigung vom Empfänger an. Dies ist empfehlenswert, wenn eine sichere Übertragung notwendig ist. Das Senden der Variablenlisten kann wahlweise zyklisch, bei Änderung einer Variable oder nach Eintreten eines Ereignisses gestartet werden.

Da als Steuerungssysteme drei Raspberry Pis verwendet werden, gilt es hier eine Besonderheit zu beachten. Die UDP-Datenpakete werden typischerweise als Broadcast-Nachrichten an alle Teilnehmer des Netzwerks gesendet. Bei der Durchführung dieser Arbeit fiel auf, dass diese Nachrichten blockiert werden. Eine Anfrage beim CODESYS-Support lieferte den Hinweis, dass die Broadcast-Adresse bei Linux-basierten Systemen auf das tatsächliche Subnetz angepasst werden sollte. Den Dialog zum Anpassen der Broadcast-Adresse wird durch Klicken des Buttons *„Einstellungen...“* beim Netzwerktyp geöffnet.

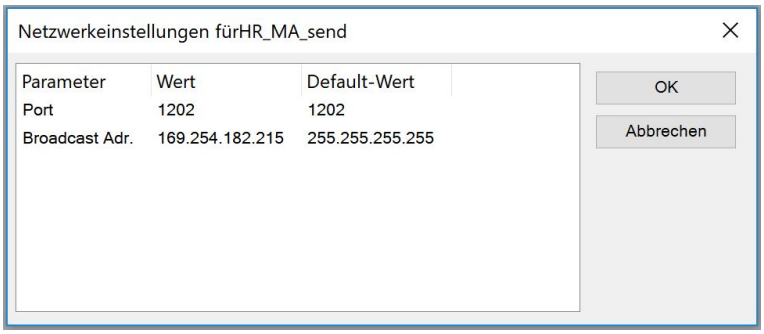

Abbildung 5.7: Netzwerkeinstellungen Dialog

Als Broadcast-Adresse wird die IP-Adresse des Empfängers eingetragen. Die Konfiguration für das Senden der Daten vom Hochregal zum Transportband und Magazin ist in Abbildung 5.7 zu sehen.

Hinzufügen einer Empfänger-Netzwerkvariablenliste

Die Empfänger-Variablenliste wird, wie die Sender-Variablenliste, über die Geräteansicht zur Applikation hinzugefügt. Abbildung 5.8 zeigt den sich öffnenden Dialog. Es kann direkt der Sender ausgewählt werden, sodass keine weitere Konfiguration der Empfänger-Variablenliste notwendig ist.

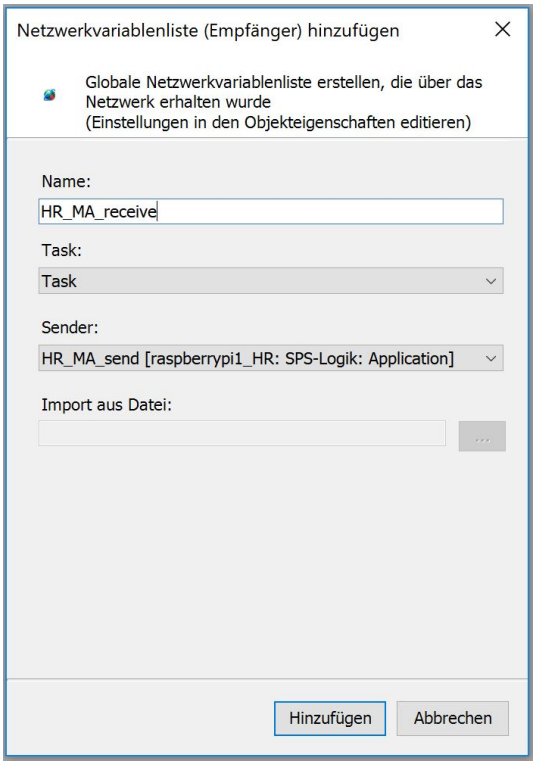

Abbildung 5.8: Netzwerkvariablenliste (Empfänger) Dialog

Die Kommunikation zwischen den anderen Modulen wird auf die gleiche Weise realisiert. Zur Übersichtlichkeit werden alle Netzwerkvariablenlisten mit den zugehörigen Variablenlistenkennungen in Tabelle 5.1 aufgelistet.

Sender	Empfängner	Kommunikation zwischen	Kennung
MA_PR_send	MA_PR_receive	raspberrypi3_MA → raspberrypi2_PR	1
MA_HR_send	MA_HR_receive	raspberrypi3_MA → raspberrypi1_HR	2
PR_MA_send	PR_MA_receive	raspberrypi2_PR → raspberrypi3_MA	3
PR_HR_send	PR_HR_receive	raspberrypi2_PR → raspberrypi1_HR	4
HR_MA_send	HR_MA_receive	raspberrypi1_HR → raspberrypi3_MA	6
HR_PR_send	HR_PR_receive	raspberrypi1_HR → raspberrypi2_PR	5

Tabelle 5.1: Netzwerkvariablenlisten

6 Programmentwurf und Implementierung

In diesem Kapitel wird der Entwurf und die Implementierung der Steuerungssoftware erläutert. Es soll an dieser Stelle darauf hingewiesen werden, dass eine ausführliche Herleitung aller implementierten Module den Rahmen dieser Bachelorarbeit überschreiten würde. Aus diesem Grund werden lediglich ausgewählte Teile der Implementierung beschrieben.

6.1 Identifikation der Anlagenmodule

Für die Automatisierung von Maschinen und Anlagen wird in der Regel eine große Anzahl von Sensoren und Aktoren eingesetzt. Die damit verbundene Komplexität soll durch den Einsatz objektorientierter Programmierung verringert werden.

Die Einteilung der Anlage in funktionell abgrenzbare Einheiten (Module) fördert die Verständlichkeit und ermöglicht die Wiederverwendung von bereits getesteten Einheiten. Besonders in der Automatisierungstechnik lässt sich die Modularisierung gewinnbringend einsetzen, da sich die maschinenbauliche Struktur einer Anlage oftmals direkt auf die Software-Struktur abbilden lässt. [15]

Für die Umsetzung der Software bedeutet dies, dass möglichst viele Feldgeräte, die sich in ihrer Funktion gleichen, in einer Klasse zusammengefasst werden sollen. Grundsätzlich gibt es zwei mögliche Vorgehensweisen bei der Erstellung modularer Software. Die erste Variante beschreibt die Dekomposition[1] der Gesamtanlage nach dem Top-Down-Prinzip. Dabei werden die Anlagenteile von oben nach unten in ihre Bestandteile zerlegt und strukturiert. Die andere Variante stellt den umgekehrten Weg, also das Zusammensetzen der Anlage aus ihren Teilkomponenten dar. Hierbei wird nach dem Bottom-up-Prinzip vorgegangen und die Gesamtanlage beginnend mit den kleinsten Teilkomponenten zusammengesetzt.

Ein Kriterium bei der Zerlegung der Modellanlage ist die Funktion der jeweiligen Einheit. Bereits bei der Beschreibung der Modellanlage wurden in Abschnitt 3.2 fünf Module genannt, die unabhängige Funktionen für den Betrieb der Anlage bieten. Die Module lauten:

[1] Auflösen von etwas in seine Bestandteile

- Magazin
- Transportband
- Schwenkarm
- Presse
- Hochregal

„Wird die Modularisierung weiter nach funktionalen Gesichtspunkten vorgenommen, ist die Bearbeitung des Werkstückes als Anlagenfunktionalität weiter zu unterteilen.“[14] Letztendlich führt die Dekomposition der Anlage zu den Modulen der untersten Ebene, die auch als Elementarmodule bezeichnet werden. Diese zeichnen sich dadurch aus, dass sie nicht weiter zerlegt werden können und eine oder mehrere eindeutige Funktionen zu den entsprechenden Sensoren/Aktoren kapseln.

6.1.1 Elementarmodule der Modellanlage

Bei der Modellanlage fällt der häufige Einsatz von binären Sensoren innerhalb verschiedener Funktionen auf, was ein weiteres Kriterium für die Auswahl von Modulen ist. Binäre Sensoren zeichnen sich dadurch aus, dass sie lediglich zwei Zustände annehmen können. Sie sind an der Anlage als Reedschalter, Positionsschalter, Lichtschranken sowie in Form eines induktiven und eines kapazitiven Sensors vorhanden. Alle Sensoren sollen in der Klasse „Sensor“ zusammen gefasst werden und die Methode zur Auswertung ihres aktuellen Status bereitstellen.

In gleicher Weise lässt sich das Modul „Motor“ herleiten. Das Transportband, der Schwenkarm und die Verfahreinheit werden von DC-Getriebemotoren desselben Typs angetrieben. Die Motoren sollen allesamt dieselben Funktionen liefern, sodass sie sich sehr gut als einheitliches Modul darstellen lassen.

Eine weitere wiederkehrende Komponente im Anlagenprozess ist der Pneumatikzylinder. Er liefert die Funktionalitäten „Zylinder einfahren“, „Zylinder ausfahren“ und „Zylinder drucklos schalten“. Neben der Unterscheidung in einfach- und doppeltwirkende Zylinder, ist für die Umsetzung der Module die Endlagenüberwachung von Bedeutung.

Aus diesem Grund werden die Zylinder in drei Module geteilt. Der einfachwirkende Zylinder mit Endlagenüberwachung wird durch die Klasse „Zylinder_1x“ abgebildet, der doppeltwirkende Zylinder mit Endlagenüberwachung durch die Klasse „Zylinder_2x“ und der doppeltwirkende Zylinder ohne Endlagenüberwachung hat die Klasse „Zylinder_2x_ohne_Sensor“.

6.1.2 Übersicht der Anlagenmodule

Das Magazin lässt sich aus zwei Pneumatikzylindern, welche die Werkstücke auf das Transportband befördern und zwei Sensoren, die die Werkstücke im Lager erkennen, zusammensetzen.

Das Transportband besteht aus einem Motor, welcher das Band antreibt, den vier Sensoren zur Werkstücküberprüfung, einem Endlagensensor und einem Pneumatikzylinder als Prüfzylinder.

Der Schwenkarm setzt sich aus zwei Pneumatikzylindern für die horizontale und vertikale Bewegung, einem Motor für die Drehbewegung und einem Sensor für die Positionserkennung zusammen.

Die Presse besteht aus drei Pneumatikzylindern, wovon einer die Auf- und Abwärtsbewegung des Stempels steuert. Der zweite Zylinder bewegt die Sicherheitstür und der dritte ist für das Verfahren der Ausschubvorrichtung zuständig.

Das Hochregal lässt sich aus zwei Motoren, die die Verfahreinheit in vertikaler und horizontaler Position verfahren können, vier Endlagensensoren für die Absicherung der Verfahreinheit und zwei Positionssensoren zur Erkennung der Lagerpositionen zusammensetzen.

Damit ist die Modellanlage vollständig beschrieben. Die hier angewandte Vorgehensweise lässt sich auch auf größere Anlagen mit mehreren Abstraktionsebenen übertragen.

6.2 Schnittstellen der Module

Üblicherweise koordinieren die Module der höheren Ebenen die ihnen unterlagerten Module durch den Aufruf von Methoden sowie durch die Übergabe von Parametern. Neben den Funktionen, die von außerhalb aufgerufen werden können, gibt es Funktionen, die die eigenen Arbeitsabläufe eines Moduls kapseln. Werden Module im Projekt wiederverwendet, ist eine Definition der Schnittstellen von hoher Bedeutung. So lässt sich über Module, die die selbe Schnittstelle implementieren sagen, dass sie auf die gleiche Weise verwendet werden können. Dabei kann die tatsächliche Realisierung der Methoden einer Schnittstelle innerhalb der Module unterschiedlich sein.

Ein Beispiel für die unterschiedliche Implementierung der Methoden einer Schnittstelle sind die Pneumatikzylinder der Modellanlage. Sie sollen alle die Methoden für das Einfahren, Ausfahren und Stillsetzen des Kolbens besitzen. Innerhalb dieser Methoden muss jedoch die unterschiedliche Ansteuerung der Ventile für den einfachwirkenden und doppeltwirkenden Zylinder berücksichtigt werden. Die genaue Implementierung der Module wird unter 6.7.3

beschrieben. In Abbildung 6.1 sind die Schnittstellen der Modellanlage im Klasseneditor von CODESYS zu sehen.

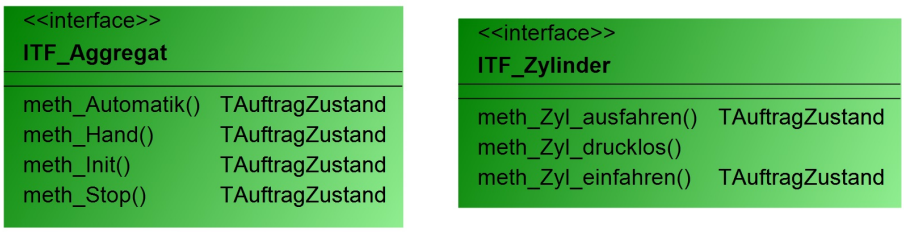

Abbildung 6.1: Schnittstellen

Die andere abgebildete Schnittstelle *ITF_Aggregat* fasst die Methoden innerhalb der Betriebsarten zusammen. Jedes der fünf Hauptmodule der Anlage (Magazin, Transportband, Schwenkarm, Presse, Hochregal) soll über die Methoden *meth_Automatik*, *meth_Hand*, *meth_Init* und *meth_Stop* aufrufbar sein. Die Schnittstelle beschreibt wiederum nur die zu erstellenden Methoden, die in jedem implementierenden Modul individuell programmiert werden müssen.

6.3 Enumerationen

Enumerationen sind benutzerdefinierte Datentypen, die die Lesbarkeit der Programmierung erhöhen sollen. In der Modellanlage werden vier verschiedene Datentypen verwendet, die in der folgenden Auflistung kurz erläutert werden.

TAnlageZustand
: *TAnlageZustand* beinhaltet alle möglichen Zustände der Anlage und wird in jedem Aggregat verwendet.

TAuftragZustand
: Der Datentyp *TAuftragZustand* wird bei den meisten Methoden als Rückgabewert verwendet und liefert die Zustände *Bereit*, *Belegt* und *Erledigt*.

TZustandTeil
: *TZustandTeil* wird in der Methode zur Prüfung der Werkstücke auf dem Transportband verwendet und liefert die Zustände *Teil_wird_geprueft*, *Teil_OK* und *Teil_ausschleusen*.

TZylinderZustand
: Jeder Zylinder enthält die Enumeration *TZylinderZustand*, die die möglichen Zylinderzustände *stehend*, *einfahren* und *ausfahren* enthält.

Abbildung 6.2: Benutzerdefinierte Datentypen der Modellanlage

6.4 Implementierung der Klassen

In diesem Abschnitt wird die Implementierung der zuvor identifizierten Module mit Hilfe des CODESYS Klasseneditors beschrieben. Dabei soll die Beschreibung der Beziehungen zwischen den Modulen im Vordergrund stehen und nicht die Implementierung der Methoden und Aktionen. Diese werden im nächsten Abschnitt unter 6.7 beschrieben.

Die Klassen, die im Klasseneditor erstellt werden, werden gleichzeitig als Funktionsbausteine in der Ansicht der POUs (Programm-Organisations-Einheiten) hinzugefügt. Auch die Beziehung zwischen den Klassen wie Generalisierung oder Realisierung werden automatisch hinzugefügt.

6.4.1 FB_Sensor

Die Klasse Sensor dient der Rückmeldung des aktuellen Sensorstatus an die fragende Klasse. Dafür besitzt sie das Sensorsignal als eingehendes Attribut und eine Methode, die als Rückgabewert das boolesche Signal des Sensors zurückliefert.

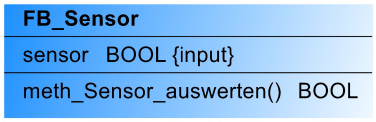

Abbildung 6.3: FB_Sensor

6.4.2 FB_Motor

Die DC-Getriebemotoren der Modellanlage werden in der Klasse *FB_Motor* zusammengefasst. Die an die Motoren gehenden Ausgangswerte für den rechtsdrehenden und linksdrehenden Betrieb werden der Klasse als Attribute hinzugefügt. Des Weiteren bietet die Klasse

drei Methoden zur Steuerung des Antriebs, die durch externe Objekte aufgerufen werden können.

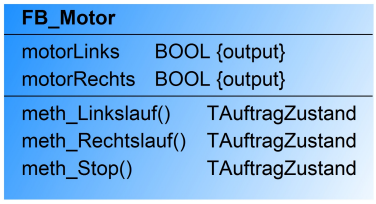

Abbildung 6.4: FB_Motor

6.4.3 FB_Timer

Für den Anlagenbetrieb werden mehrere Verzögerungszeiten benötigt. Um nicht zu jedem Modul ein TON-Glied (Funktionsbaustein für eine Einschaltverzögerung) zuweisen zu müssen, wird diese Klasse entworfen. Sie beinhaltet eine Einschaltverzögerung und eine Methode, die als Aufrufparameter die abzuwartende Zeit erhält. Ist die Zeit abgelaufen, liefert die Methode einen benutzerdefinierten Datentyp zurück, der unter 6.3 erläutert wird.

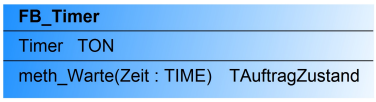

Abbildung 6.5: FB_Timer

6.4.4 Pneumatikzylinder

Bisher wurden lediglich Methoden zu den Klassen hinzugefügt, da diese sich aufgrund ihrer Charakteristik als Schnittstelle zu anderen Funktionen eignen. Beim Pneumatikzylinder werden nun zusätzlich interne Funktionalitäten benötigt, die beispielsweise das Ausfahren des Zylinders steuern. Für diese Funktionen werden Aktionen verwendet.

Der Aufruf von externen Klassen und die Umsetzung der geforderten Aktion soll am Beispiel des einfachwirkenden Zylinders verdeutlicht werden. Die Aktion *act_Zyl_ausfahren()* legt den Vorgang zum Ausfahren des Zylinders fest, während die Methode *meth_Zyl_ausfahren()* eingesetzt wird, um diese Aktion aufzurufen.

Für die Erstellung der Klassen kann das Prinzip der Vererbung sinnvoll eingesetzt werden. Die Klasse *FB_Zylinder_1x* implementiert das Interface *ITF_Zylinder*, wodurch sichergestellt wird, dass die Methoden der Schnittstelle innerhalb der Klasse vorhanden sind. Betrachtet

man die einfach- und doppeltwirkenden Zylinder, fällt schnell auf, dass beide Klassen über einige gemeinsame Attribute, wie z.B. die Variablen für die Endlagensensoren, verfügen. Aus diesem Grund erbt die Klasse *FB_Zylinder_2x* von der Basisklasse des einfachwirkenden Zylinders. Durch die Vererbung werden alle Attribute und Methoden von der Basisklasse übernommen und können in der abgeleiteten Klasse verwendet werden. Für den doppeltwirkenden Zylinder wird das Attribut *einfahren* zum Einfahren des Zylinders ergänzt und die Aktionen zur Steuerung werden überschrieben. Die Beziehung der beiden Klassen wird im Klassendiagramm durch eine Generalisierung dargestellt (siehe Abbildung 6.6).

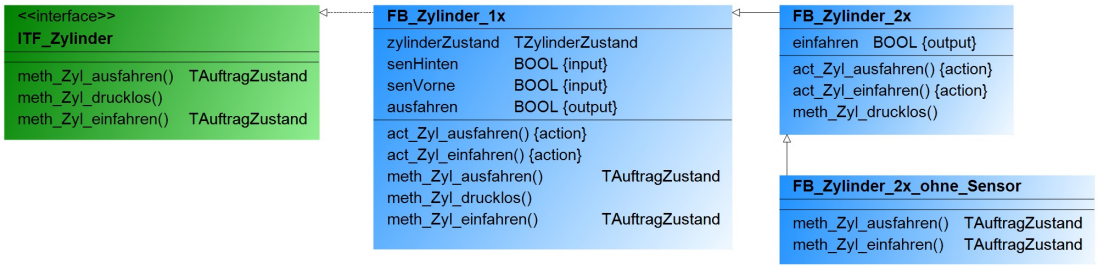

Abbildung 6.6: Klassen der Pneumatikzylinder

Der doppeltwirkende Pneumatikzylinder ohne Endlagenüberwachung erbt von der Klasse *FB_Zylinder_2x*. So müssen lediglich die Methoden der Basisklasse neu definiert werden, um die fehlende Überwachung der Zylinder zu übergehen. Mit der Implementierung dieser drei Klassen können alle an der Anlage vorhandenen Pneumatikzylinder beschrieben und gesteuert werden.

In der ersten Zeile der Variablendeklaration des Funktionsbausteins werden die Beziehungen zwischen den Klassen deklariert. Durch das Zeichnen der Verbindungen im Klasseneditor werden die folgenden Codezeilen automatisch in dem jeweiligen Funktionsbaustein der Klasse ergänzt.

```
FUNCTION_BLOCK FB_Zylinder_1x IMPLEMENTS ITF_Zylinder

FUNCTION_BLOCK FB_Zylinder_2x EXTENDS FB_Zylinder_1x

FUNCTION_BLOCK FB_Zylinder_2x_ohne_Sensor EXTENDS FB_Zylinder_2x
```

Listing 6.1: Schlüsselwörter der Vererbung

6.4.5 FB_Aggregat

Die Klasse *FB_Aggregat* ist die Basisklasse der fünf Hauptmodule der Anlage. Sie umfasst alle gemeinsamen Eigenschaften und Funktionalitäten. Jedes Aggregat wird innerhalb der Betriebszustände der Modellanlage aufgerufen und bearbeitet die dazugehörige Methode. Befindet sich die Anlage beispielsweise im Automatikbetrieb, wird die Aktion *act_automatik()* aller Aggregate ausgeführt. Die Klasse ist in Abbildung 6.7 zu sehen.

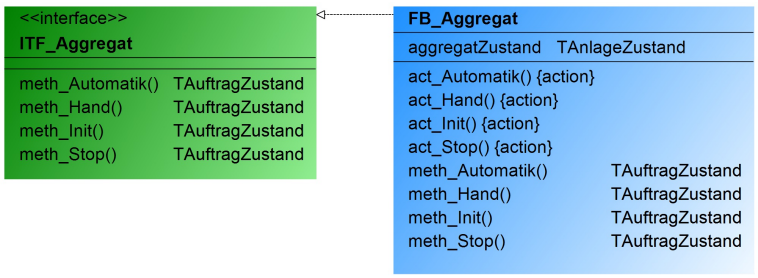

Abbildung 6.7: FB_Aggregat

Sie implementiert außerdem die Schnittstelle *ITF_Aggregat* und beinhaltet damit zwingend die Methoden für jeden Betriebszustand der Modellanlage.

6.4.6 FB_Lager

Das Lager hat die Aufgabe die Werkstücke dem Prozess zuzuführen. Dafür steht die Methode *meth_Werkstueck_ausschieben()* zur Verfügung, die eine Rückmeldung liefert, sobald der Vorgang abgeschlossen ist. Für den Ausschiebevorgang des Werkstücks bildet das Lager eine Instanz der Klasse *FB_Zylinder_2x* und kann den Zylinder über die entsprechenden Methoden steuern. Das Lager erbt von der Klasse *FB_Aggregat* und überschreibt die Aktionen in den verschiedenen Anlagenzuständen.

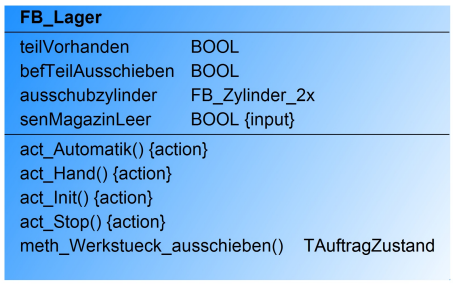

Abbildung 6.8: FB_Lager

Das Enthalten einer anderen Klasse entspricht der Kompositionsverbindung im Klassenediter. Dabei wird im Variablendeklarationsteil des Funktionsbausteins eine Instanz der enthaltenen Klasse erzeugt.

```
1  FUNCTION_BLOCK FB_Lager EXTENDS FB_Aggregat
2  VAR_INPUT
3      senMagazinLeer : BOOL;        // low active
4  END_VAR
5  VAR_OUTPUT
6  END_VAR
7  VAR
8      teilVorhanden: BOOL;
9      befTeilAusschieben : BOOL;
10     ausschubzylinder : FB_Zylinder_2x;
11 END_VAR
```

Listing 6.2: Variablendeklaration des FB_Lager

Im Codeausschnitt 6.2 ist die Variablendeklaration des *FB_Lager* abgebildet. In Zeile 10 wird die Variable *ausschubzylinder* erzeugt, die eine Instanz der Klasse *FB_Zylinder_2x*. Anstelle des Begriffs Instanz wird hier auch häufiger die Bezeichnung Objekt verwendet.

6.4.7 FB_Band

Das Band ist für die Überprüfung der Werkstücke und die Übergabe an den Schwenkarm verantwortlich. Für die Kontrolle der Werkstücke werden die Methoden *meth_pruefe_Boden()* und *meth_pruefe_Deckel()* verwendet. Sie liefern einen Rückgabewert vom Datentyp *TZustandTeil*. Das Band enthält außerdem fünf Instanzen der Klasse *FB_Sensor*, eine der Klasse *FB_Motor* sowie einen Timer und einen Pneumatikzylinder ohne Endlagenüberwachung.

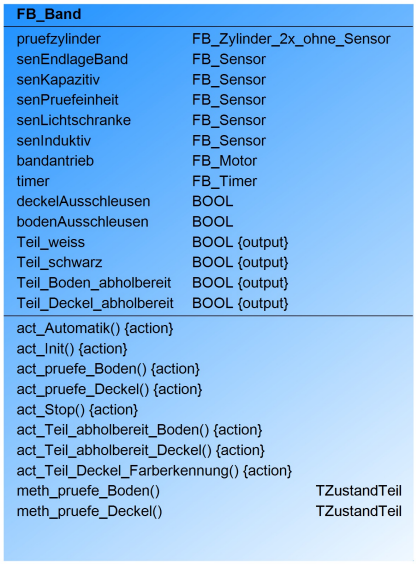

Abbildung 6.9: FB_Band

6.4.8 FB_Schwenkarm

Der Schwenkarm dient als Bindeglied zwischen dem Transportband, der Presse und dem Hochregallager. Er implementiert die Methoden zur Aufnahme und Abgabe eines Werkstücks. Zusammengesetzt wird das Modul durch die Elementarmodule *FB_Sensor*, *FB_Motor*, *FB_Timer* sowie zwei Objekten der Klasse *FB_Zylinder_2x*. Neben den für den Prozessablauf relevanten Variablen besitzt diese Klasse die Variable *sauger*, die den Vakuumsauger für das Festhalten des Werkstücks steuert.

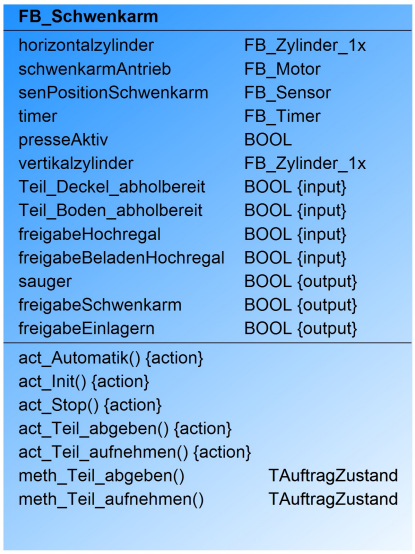

Abbildung 6.10: FB_Schwenkarm

6.4.9 FB_Presse

Die Aktoren der Presse lassen sich in zwei Klassen einteilen. Die Zylinder der Sicherheitstür und der Ausschubvorrichtung werden als Objekte der Klasse *FB_Zylinder_2x* erzeugt. Der Zylinder für den Stempel ist ein Objekt der Klasse *FB_Zylinder_2x_ohne_Sensor*. Durch die Methode *meth_Pressvorgang_starten()* wird der Pressvorgang gestartet. Nach Ablauf des Prozesses wird der Status als benutzerdefinierter Datentyp *TAuftragZustand* zurückgegeben.

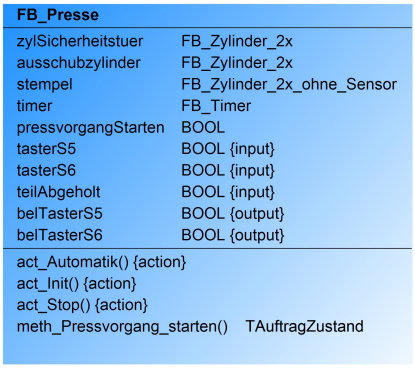

Abbildung 6.11: FB_Presse

6.4.10 FB_Hochregal

Die verbleibende Klasse setzt das Hochregal in einem Softwaremodul um. Die enthaltenen Objekte anderer Klassen sind der Abbildung 6.12 zu entnehmen.

Die verfügbaren Lagerplätze werden durch das zweidimensionale Array *hochregal* abgebildet, wobei das erste Feld die fünf Lagerplätze in horizontaler Lage beschreibt und das zweite Feld die vier Etagen des Hochregals darstellt. Beim initialisieren des Arrays wird für alle Koordinaten der boolesche Ausdruck *FALSE* eingetragen. Wird ein Lagerplatz mit einem Werkstück besetzt, ändert sich der Wert zu *TRUE*.

Um zu erkennen an welcher Position sich die Verfahreinheit befindet, werden zwei CTU-Zähler (Vorwärtszähler) genutzt, die die Positionssensoren *senPositionB10* für die Horizontale und *senPositionVertikal* für die Vertikale auswerten. Die aktuellen Zählerwerte werden in die Variablen *posHorizontal* und *posVertikal* geschrieben.

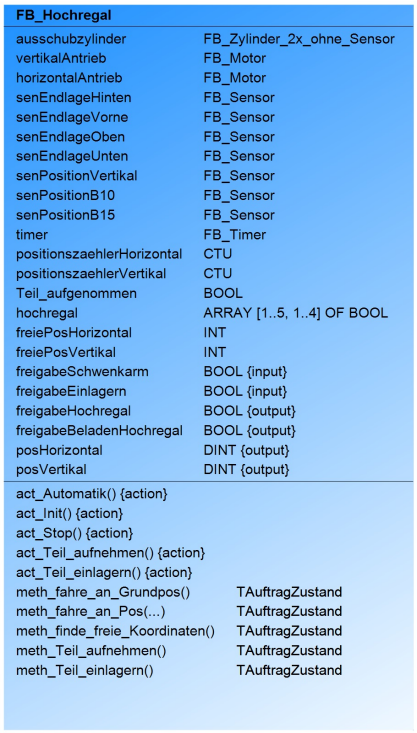

Abbildung 6.12: FB_Hochregal

Das Hochregal besitzt die Methode *meth_Teil_aufnehmen()*, die intern aufgerufen wird und den Prozess zur Aufnahme eines Werkstücks vom Schwenkarm einleitet. Die Methode

meth_Teil_einlagern() startet den Einlagerungsprozess. Zum Positionieren der Verfahreinheit werden die Methoden *meth_fahre_an_Grundpos()* und *meth_fahre_an_Pos(...)* verwendet. Deren Funktionen werden unter Abschnitt 6.7.6 detailliert erläutert. Die fünfte Methode *meth_finde_freie_Koordinaten()* ermittelt die Koordinaten der nächsten freien Lagerposition.

6.5 Resultierende Klassendiagramme

Um die Beziehungen zwischen den Klassen noch einmal grafisch darzustellen, sind nachfolgend drei Klassendiagramme abgebildet. Dabei werden die Module entsprechend ihrer Implementierung auf den Steuerungen gegliedert. Das bedeutet, dass das Transportband und das Magazin sowie die Presse und der Schwenkarm in einer Darstellung zusammengefasst werden.

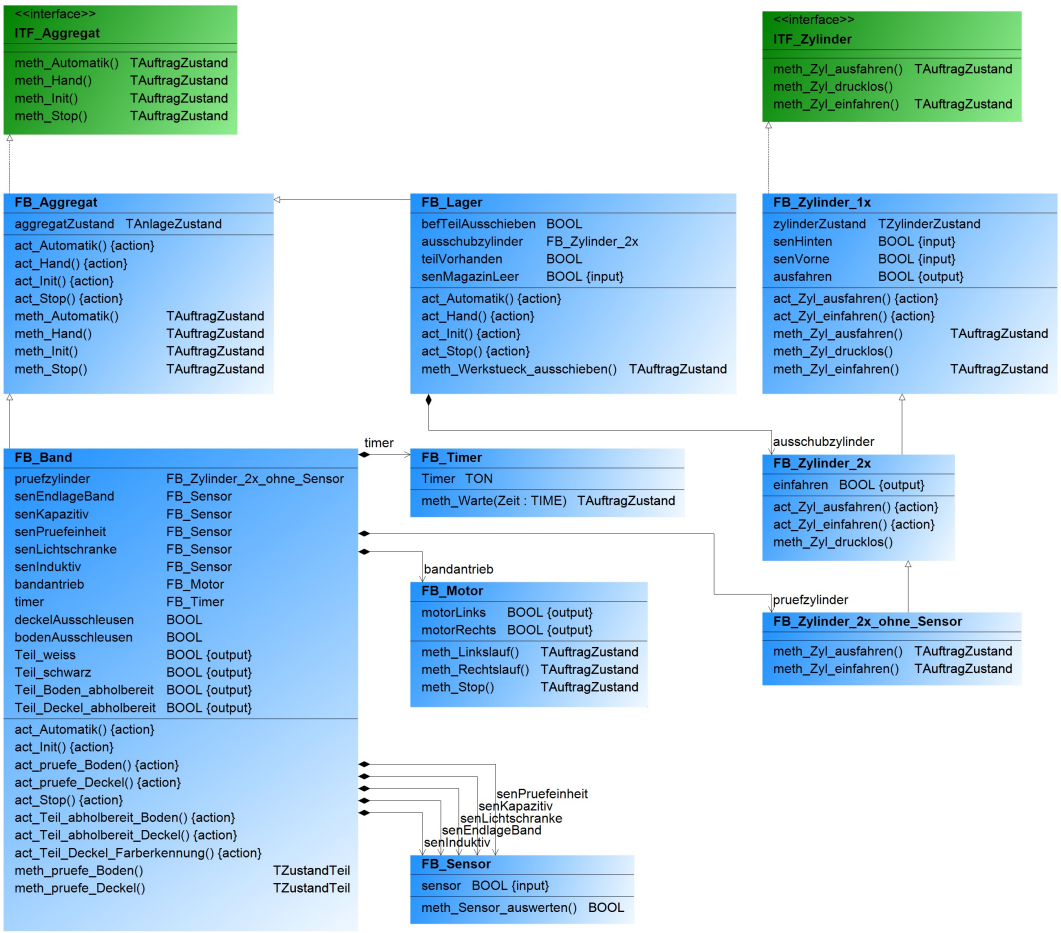

Abbildung 6.13: Klassendiagramm für das Transportband und das Magazin

Anhand der Verbindungslinien, deren Funktionen unter 2.4.2 beschrieben wurden, lässt sich schnell ein Überblick über das Zusammenwirken aller Komponenten verschaffen. Auch hier ist noch einmal zu erkennen, dass sich die fünf Hauptmodule aus den Elementarmodulen zusammensetzen lassen.

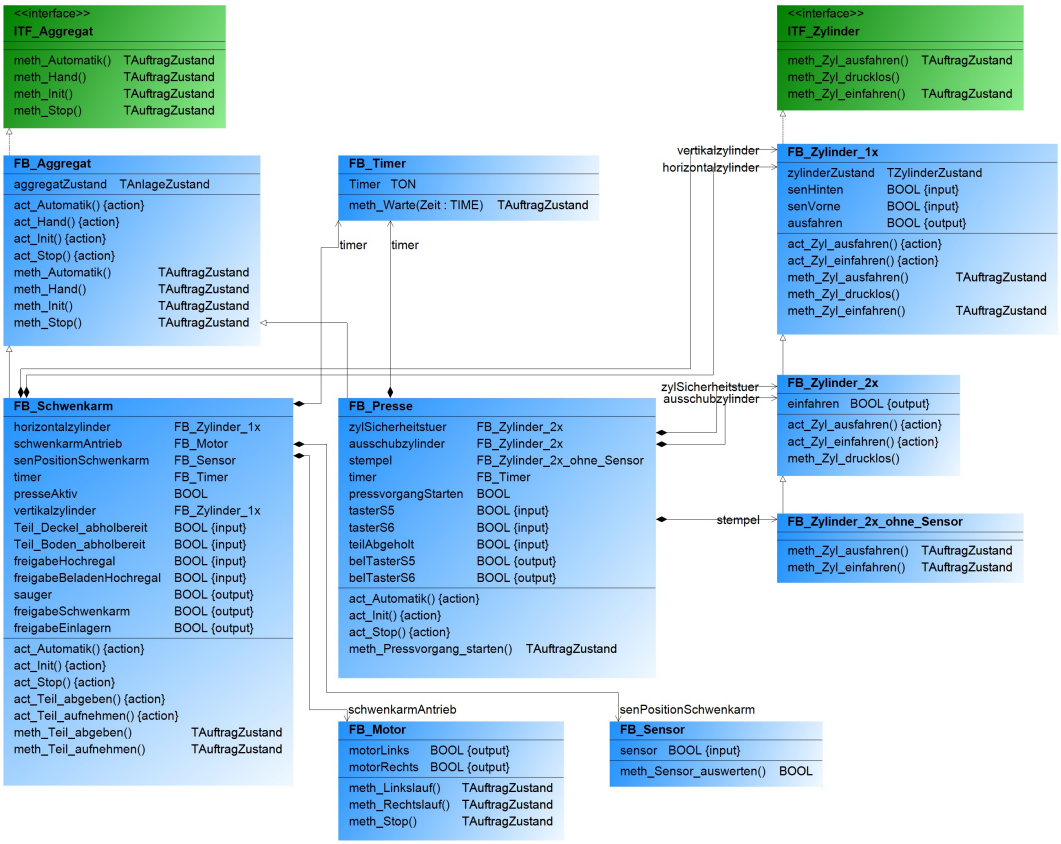

Abbildung 6.14: Klassendiagramm für den Schwenkarm und die Presse

Klassendiagramme dienen in der Softwareentwicklung als Diskussionsbasis zur Strukturierung des Programms. Durch die Integration des Klassendiagrammeditors in die Entwicklungsumgebung müssen die Diagramme nicht mehr manuell in Steuerungssoftware umgesetzt werden, da dies nun automatisch durchgeführt wird. Als Beispiel hierfür soll die Klasse *FB_Presse* aus Abbildung 6.14 dienen. Die Erstellung der Klasse im Diagramm erzeugt automatisch einen Funktionsbaustein. Dabei werden auch die Methoden und Aktionen der Klasse vom Funktionsbaustein übernommen. Die Abbildung 6.15 zeigt den Funktionsbaustein in der POU-Ansicht.

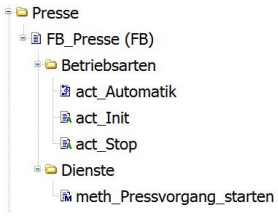

Abbildung 6.15: FB_Presse in der POU Ansicht

Des Weiteren werden auch die Attribute der Klasse in die Variablendeklaration des Funktionsbausteins übernommen. Damit wird die gesamte Struktur aus den Klassendiagrammen in POUs umgewandelt und lediglich die Implementierungsteile, Methoden und Aktionen müssen programmiert werden.

Um das Klassendiagramm des Hochregals übersichtlicher zu gestalten, werden die Klassen *FB_Zylinder_1x* und *FB_Zylinder_2x* sowie die Schnittstelle *ITF_Zylinder* nicht mit dargestellt.

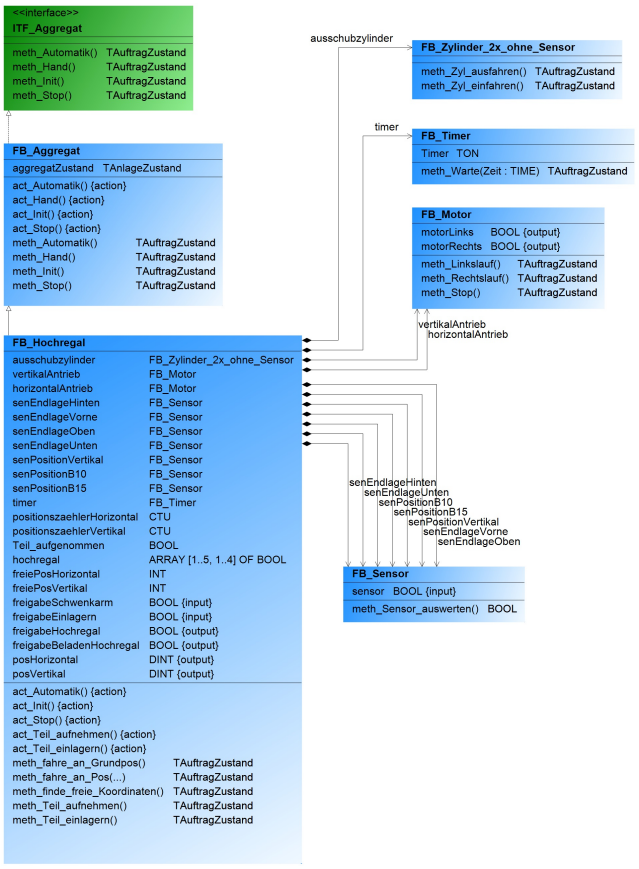

Abbildung 6.16: Klassendiagramm für das Hochregal

6.6 Globale Variablenlisten

Innerhalb einer globalen Variablenliste (GVL) können Variablen deklariert werden. Wird die GVL einer Applikation zugeordnet, sind die Variablen nur für alle Objekte derselben Applikation sichtbar. Da alle Steuerungen mit den gleichen Variablenlisten arbeiten sollen, werden sie in der Ansicht der POUs hinzugefügt und sind für alle Objekte sichtbar.

Für die Modellanlage werden diese drei Variablenlisten angelegt:

GVL_Aggregate
Innerhalb dieser Variablenliste sind alle Instanzen der fünf Hauptmodule definiert. Die Ansicht einer Variablenliste im Klassendiagramm ist in Abbildung 6.17 dargestellt.

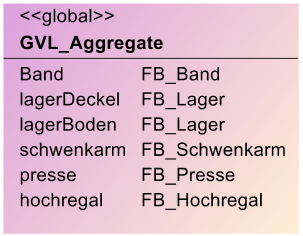

Abbildung 6.17: GVL_Aggregate

GVL_Anlagenzustand
In GVL_Anlagenzustand werden die aktuellen Anlagenzustände des Hochregals, der Presse und des Magazins dokumentiert. Sie sind vom Typ *TAnlageZustand*.

GVL_Input
Alle vom Bedienpanel eingehenden Steuersignale der Anlage, wie beispielsweise der Start Taster oder der Not-Aus Schalter, werden in dieser globalen Variablenliste zusammengefasst.

Da die Variablenlisten in die Ansicht der Programmorganisationseinheiten eingebunden werden, werden diese auf jeder Steuerung gespeichert. Dabei muss beachtet werden, dass die Änderung einer Variable nur die Variablenliste der eigenen Steuerung betrifft. Dies soll anhand des zweidimensionalen Arrays des Hochregals verdeutlicht werden. Das Array wird von der Steuerung *raspberrypi1_HR* beschrieben und die belegten Lagerplätze mit dem booleschen Ausdruck *TRUE* gekennzeichnet. Diese Änderungen werden nicht in den Variablenlisten der beiden übrigen Steuerungen übernommen.

Variablen, die für alle Steuerungen sichtbar sein müssen, müssen über Netzwerkvariablenlisten verschickt werden. Alle Kommunikationen zwischen den Steuerungen der Modellanlage werden in Kapitel 6.8.1 erläutert.

6.7 Implementierung der Module

Da somit alle Klassen, Methoden und Aktionen der Anlage durch das Klassendiagramm definiert und generiert sind, werden die entsprechenden Programmorganisationseinheiten programmiert. Für die Umsetzung wird die IEC-Programmiersprache strukturierter Text (ST) und die Programmiersprache Statechart (SC), die mit der UML Erweiterung in die Entwicklungsumgebung integriert wurde, verwendet. Die nachstehenden Abschnitte erläutern die Programmierung der zuvor implementierten Klassen. Um den Umfang im Rahmen zu halten, werden nicht alle Klassen im Detail und mit ihren UML-Diagrammen beschrieben. Diese können jedoch in dem CODESYS-Projekt, welches sich auf der CD im Anhang befindet, angesehen werden.

6.7.1 Hauptprogramm

Das *Hauptprogramm_Band* ist das Kernstück der Anlagensteuerung und führt den Aufruf der Funktionsbausteine sowie deren Funktionen aus. Es dient auch als Schnittstelle zum Bedienpanel, das die Signale für die Bedienung der Anlage enthält. Das Hauptprogramm wird vom *raspberrypi3_MA* aufgerufen, da hier die Signale des Bedienpanels ankommen.

Die anderen beiden Steuerungen müssen ebenfalls ein Hauptprogramm für den Ablauf der Anlage besitzen. Das *Hauptprogramm_Presse_Schwenkarm* dient der Steuerung der Aggregate Presse und Schwenkarm und das *Hauptprogramm_Hochregal* dient der Steuerung des Hochregals. Da diese keinen direkten Zugriff auf die Eingaben des Bedieners haben, wird der aktuelle Anlagenzustand durch das Hauptprogramm des *raspberrypi3_MA* bestimmt.

Bei der Erstellung der Hauptprogramme wird darauf geachtet, diese möglichst gleich zu halten. Daher besitzen sie dieselben Zustände. Auch die Transitionsbedingungen zum Wechsel der Zustände sind - bis auf den Schaltvorgang in den *Init*-Zustand - identisch. In Abbildung 6.18 ist das Zustandsdiagramm des Hauptprogramms dargestellt.

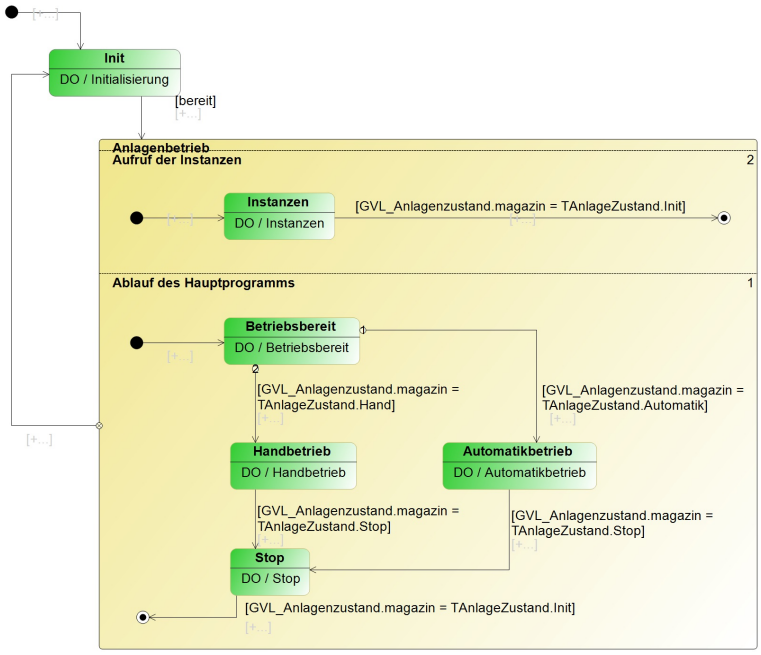

Abbildung 6.18: Hauptprogramm *raspberrypi3_MA*

Innerhalb der Zustände werden die entsprechenden Methoden *meth_Init()*, *meth_Automatik()*, *meth_Hand()* oder *meth_Stop()*, die aufgrund der Schnittstelle *ITF_Aggregat* von allen Aggregaten implementiert werden müssen, aufgerufen. Der Codeausschnitt 6.3 zeigt diesen Aufruf exemplarisch für das *Hauptprogramm_Band*.

```
1  // Aufruf der Methoden des Automatikbetriebs
2  GVL_Aggregate.Band.meth_Automatik();
3  GVL_Aggregate.lagerBoden.meth_Automatik();
4  GVL_Aggregate.lagerDeckel.meth_Automatik();
5
6  IF NOT GVL_Input.Stop THEN
7      GVL_Anlagenzustand.magazin := TAnlageZustand.Stop;
8  END_IF
9
10 IF GVL_Input.Not_aus THEN
11     GVL_Anlagenzustand.magazin := TAnlageZustand.Stop;
12 END_IF
```

Listing 6.3: Aktion *Automatikbetrieb*

Anhand der Signale vom Bedienpanel, die in der globalen Variablenliste *GVL_Input* enthalten sind, wird der Anlagenzustand verändert. Dieser wird als Transitionsbedingung zum Schalten zwischen den Zuständen verwendet.

Da die Signale des Bedienpanels nur im *Hauptprogramm_Band* verfügbar sind, vereinfacht sich der Code der verbleibenden Hauptprogramme auf den Aufruf der Automatikmethode.

Für den Übergang vom Initialisierungs- in den Betriebsbereitzustand ist zusätzlich anzumerken, dass dem *Hauptprogramm_Band* gemeldet wird, ob alle Aggregate der anderen Steuerungen die Initialisierung abgeschlossen haben. Erst dann wird in den Betriebsbereitzustand gewechselt. Gleiches gilt für den Wechsel vom Stop-Zustand in den Initialisierungszustand.

Es wird ein zusammengesetzter Zustand verwendet, damit die erzeugten Objekte parallel zum Prozessablauf aufgerufen werden können. Werden die Objekte nicht zyklisch aufgerufen, wird der Code im Implementierungsteil nicht aufgeführt. Der Inhalt der Aktion *Instanzen* des *Hauptprogramm_Band* ist im Codeausschnitt 6.4 dargestellt.

```
1  // Aktualisierung der Instanzen
2  GVL_Aggregate.Band();
3  GVL_Aggregate.lagerDeckel();
4  GVL_Aggregate.lagerBoden();
```

Listing 6.4: Aktion *Instanzen*

6.7.2 FB_Timer

Die Methode *meth_Warte(...)* der Klasse *FB_Timer* soll nach einer vorgegebenen Zeit den Rückgabewert *TAuftragZustand.Erledigt* liefern. Die Implementierung der Methode lautet wie folgt.

```
1  meth_Warte := TAuftragZustand.Belegt;
2
3  IF Timer.IN = FALSE THEN
4      meth_Warte := TAuftragZustand.Bereit;
5      Timer(IN := TRUE, PT := Zeit);
6  ELSE
7      Timer.IN := FALSE;
8  END_IF
9
10 IF Timer.Q THEN
11     Timer(IN := FALSE); // Zurücksetzen des Timers
12     meth_Warte := TAuftragZustand.Erledigt;
13 END_IF
```

Listing 6.5: *meth_Warte(...)*

Der Zeitwert, der abgewartet werden soll, wird in der Methode als Eingangsparameter deklariert. Das Zurücksetzen des Timers nach Ablauf der Zeit ist notwendig, damit die Methode mehrfach hintereinander von einer Klasse aufgerufen werden kann.

6.7.3 Pneumatikzylinder

Die Pneumatikzylinder der Modellanlage sind in drei Klassen gegliedert. Da der einfachwirkende Zylinder die Basisklasse darstellt, wird die Implementierung als erstes erläutert.

FB_Zylinder_1x

Der Implementierungsteil des einfachwirkenden Zylinders dient der Überwachung der Zylinderzustände. Neben dem Initialisierungszustand kann der Zylinder die Zustände *Eingefahren*, *Ausgefahren*, *Einfahren* sowie *Ausfahren* annehmen. Das Zustandsdiagramm ist in Abbildung 6.19 zu sehen.

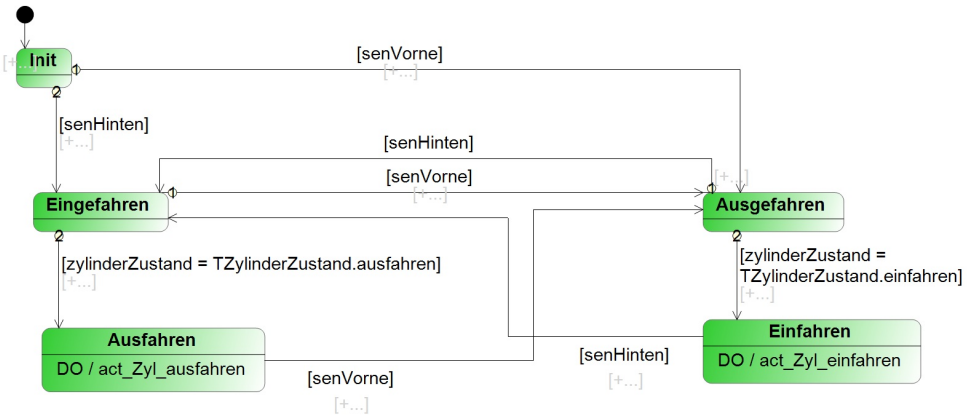

Abbildung 6.19: Implementierungsteil FB_Zylinder_1x

Die Zustandswechsel zwischen dem eingefahrenen Zustand und dem ausgefahrenen Zustand werden durch die Methoden der Klasse angestoßen. Die Aktionen steuern lediglich die Ausgangsvariablen für den Zylinder an.

Die Implementierung der Methode *meth_Zyl_ausfahren()* soll anhand des Codeausschnitts 6.6 erklärt werden.

```
1  meth_Zyl_ausfahren := TAuftragZustand.Belegt;
2
3  // States[5] für den State "Eingefahren"
4  IF UML_SC_FB_Zylinder_1x.States[5].Active THEN
5      meth_Zyl_ausfahren := TAuftragZustand.Bereit;
```

```
6       zylinderZustand := TZylinderZustand.ausfahren;
7   END_IF
8
9   // States[3] für den State "Ausgefahren"
10  IF UML_SC_FB_Zylinder_1x.States[3].Active THEN
11      meth_Zyl_ausfahren := TAuftragZustand.Erledigt;
12  END_IF
```

Listing 6.6: *meth_Zyl_ausfahren()*

Mit Hilfe der impliziten Variablen des Zustandsdiagramms kann geprüft werden, ob ein Zustand gerade aktiv ist. Wenn der Zylinder gerade eingefahren ist, wird der Zylinderzustand auf *TZylinderZustand.ausfahren* geändert und die Transitionsbedingung zum Schalten in den nächsten Zustand wird erfüllt. Befindet sich der Zylinder bereits im ausgefahrenen Zustand, wird der Auftrag direkt als erledigt zurückgemeldet.

Um zu prüfen, ob ein Zustand gerade aktiv ist, muss auf das *States* Array innerhalb der impliziten Variablen zugegriffen werden. In diesem Array werden alle States des Zustandsdiagramms in alphabetischer Reihenfolge hinterlegt. Da zu den selbst erstellten Zuständen noch automatisch generierte Zustände hinzukommen, ist die Identifikation des gesuchten Zustands nur während der Online-Verbindung zur Steuerung möglich. Bekanntlich besteht bei der Entwicklung der Software nicht immer die Möglichkeit die Variablen online zu betrachten, sodass die Positionen im Array unklar bleiben. An dieser Stelle besteht bei der CODESYS UML Erweiterung noch Verbesserungspotential.

FB_Zylinder_2x

Der doppeltwirkende Zylinder kann durch das gleiche Zustandsdiagramm wie der einfachwirkende Zylinder beschrieben werden. Da die Klasse *FB_Zylinder_1x* die Basisklasse ist, kann mit dem *SUPER*-Operator auf die Implementierungen dieser zugegriffen werden. Auf diese Weise kann der Programmieraufwand für den doppeltwirkenden Zylinder auf die Neudefinition der Ein- und Ausfahraktionen minimiert werden.

FB_Zylinder_2x_ohne_Sensor

Ohne Endlagenüberwachung ist das Zustandsdiagramm des einfachwirkenden Zylinders hinfällig. Aus diesem Grund werden die Methoden zum Ein- und Ausfahren des Zylinders überschrieben, sodass sie direkt die Ausgangsvariablen der Klasse steuern.

6.7.4 FB_Lager

Das Lager dient der Zuführung von Werkstücken an den Prozess. Um zu prüfen, ob sich ein Werkstück im Lager befindet, wird der Implementierungsteil um einige Zeilen Code erwei-

tert. Außerdem muss die Instanz des Ausschubzylinders zyklisch aufgerufen werden. Der gesamte Implementierungsteil ist im Codeausschnitt 6.7 zu sehen.

```
1  // Prüfen, ob ein Teil im Magazin vorhanden ist
2  IF NOT senMagazinLeer THEN
3      teilVorhanden := 1;
4  END_IF
5
6  IF ausschubzylinder.senVorne AND senMagazinLeer THEN
7      teilVorhanden := 0;
8  END_IF
9
10 // Aufruf der Instanz des Zylinders
11 ausschubzylinder();
12
13 SUPER^();
```

Listing 6.7: Implementierungsteil *FB_Lager*

Während der Initialisierung soll der Ausschubzylinder eingefahren werden. Im Automatikbetrieb wird das folgende Zustandsdiagramm ausgeführt.

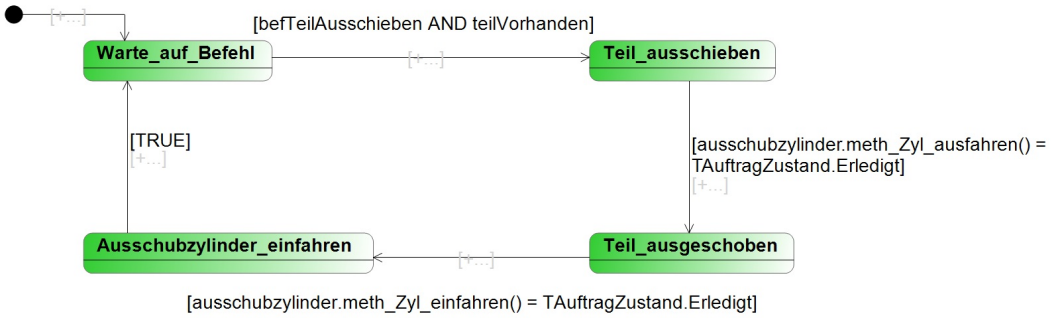

Abbildung 6.20: FB_Lager *act_Automatik()*

Durch die Methode *meth_Werkstueck_ausschieben()* wird der Ausschiebevorgang eines Werkstücks aus dem Magazin gestartet. Wird der Zustand *Ausschubzylinder_eingefahren* erreicht, wird der Befehl zum Ausschieben eines Werkstücks zurückgesetzt und der Auftrag als erledigt zurückgemeldet. Innerhalb der Stop-Aktion wird die Methode *meth_Zyl_drucklos()* des Ausschubzylinders aufgerufen.

Mit den beiden Funktionsbausteinen *FB_Band* und *FB_Schwenkarm* sollen noch zwei komplexere Funktionen beschrieben werden. Die Klassen für den Schwenkarm und der Presse werden hier nicht erläutert, da diese im wesentlichen aus Abläufen bestehen, die sich aus den Erläuterungen zur Klasse *Band* ableiten lassen.

6.7.5 FB_Band

Der Funktionsbaustein *FB_Band* steuert die beiden Lager und prüft die ausgeschobenen Werkstücke, bevor diese an den Schwenkarm übergeben werden. Die im Automatikbetrieb aufgerufene Aktion *act_Automatik()* wird als Zustandsdiagramm programmiert und ist in Abbildung 6.21 zu sehen.

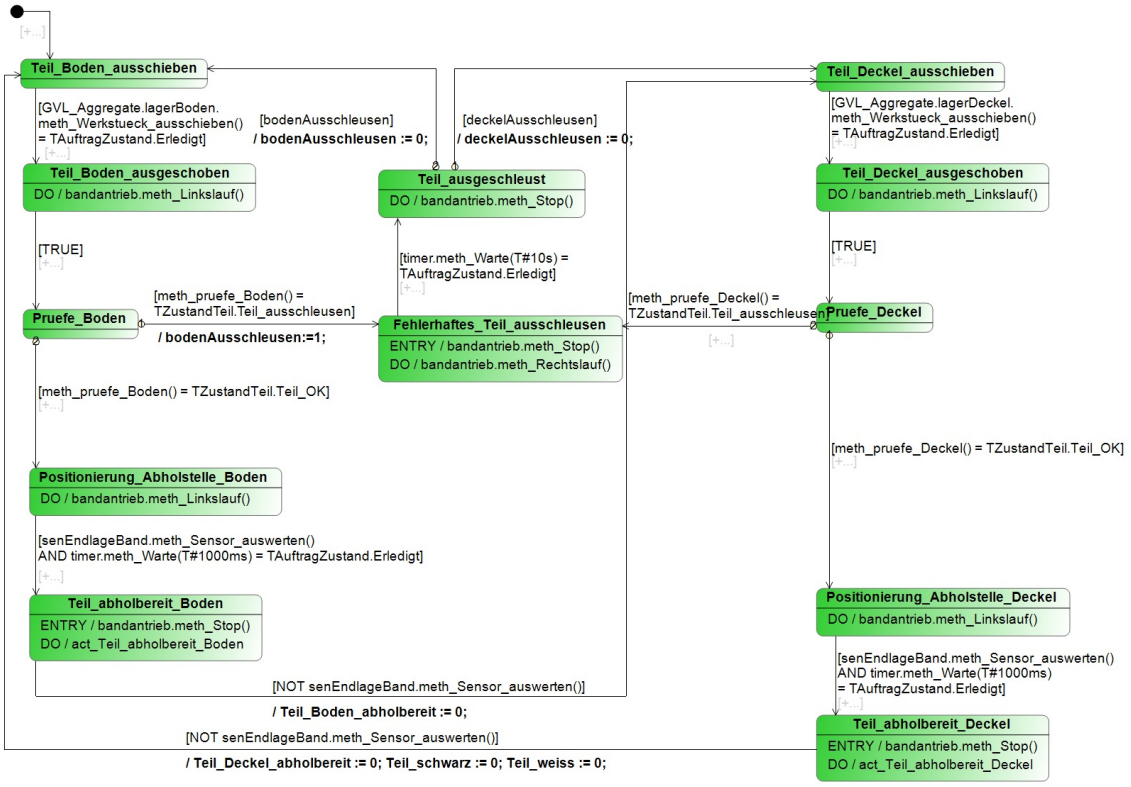

Abbildung 6.21: FB_Band *act_Automatik()*

In der linken Hälfte der Abbildung befinden sich die Zustände für die Überprüfung eines Bodenteils. Erwähnenswert ist die erste Transitionsbedingung, die die Methode zum Ausschieben eines Werkstücks des Objekts *lagerBoden* aufruft. Erst wenn die Methode erfolgreich abgeschlossen ist, kann die Transition schalten. Als nächstes wird die Methode zum Prüfen eines Bodenteils aufgerufen. Wird der Wert *Teil_ausschleusen* zurückgeliefert, muss das Teil vom Band befördert werden und ein neues Werkstück vom Lager angefordert werden. Wird der Wert *Teil_OK* zurückgegeben, wird das Teil an der Endlage des Transportbands positioniert und die Aktion *act_Teil_abholbereit_Boden* signalisiert dem Schwenkarm das fertig geprüfte Bodenteil. Befindet sich das Teil nicht mehr in der Endlage, wird der Vorgang für

den Deckel wiederholt. Das Rücksetzen der Variable, die ein abholbereites Teil meldet, wird beim Transitionsübergang durchgeführt.

Bei dem soeben beschriebenen Prozess findet ein stetiger Austausch zwischen verschiedenen Objekten statt. Um das Zusammenwirken der Objekte besser überschauen zu können, visualisiert das folgende Sequenzdiagramm diese Kommunikation.

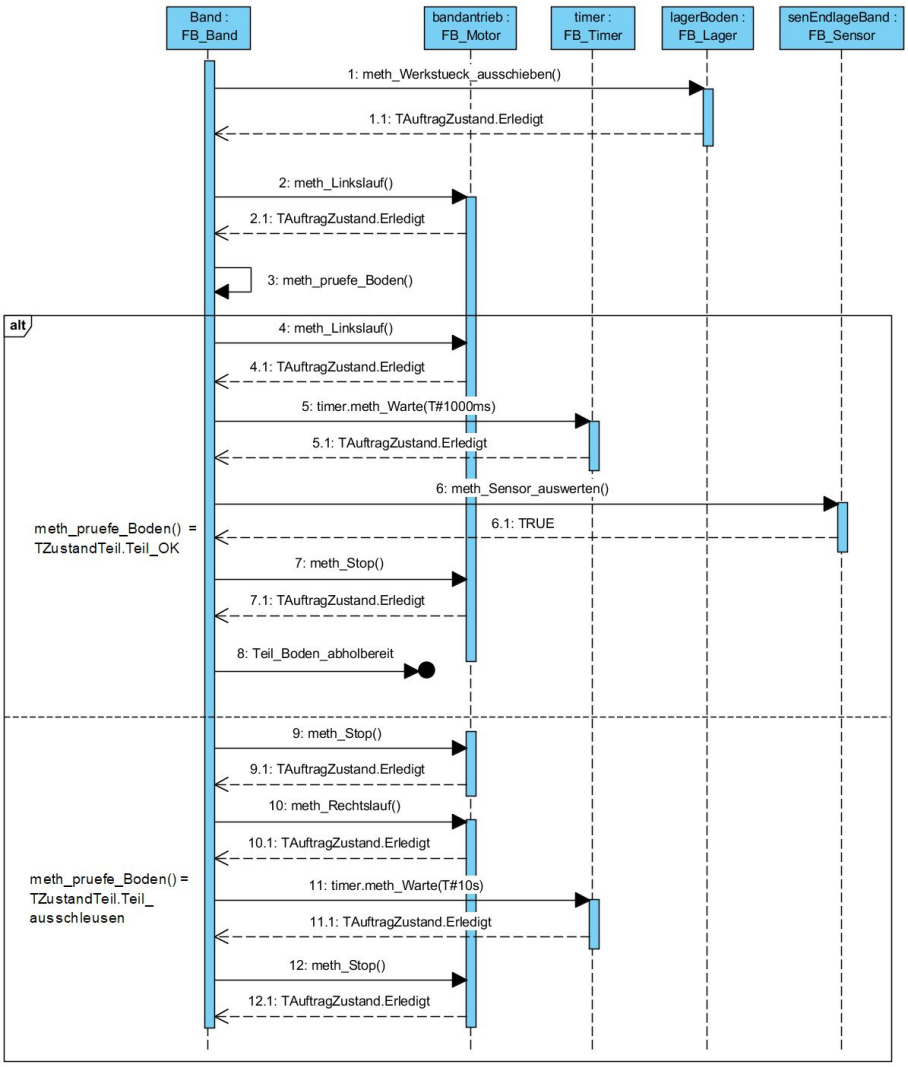

Abbildung 6.22: Sequenzdiagramm FB_Band: *act_Automatik*

In der oberen Zeile sind alle Objekte, zwischen denen eine Kommunikation stattfindet, angereiht. Die meisten Methoden arbeiten auf den drei grundlegenden Zuständen *belegt*, *bereit*

und *erledigt* des Datentyps und geben eine Antwort in Form des Rückgabewerts. Das kombinierte Fragment entspricht der Überprüfung des Werkstücks aus dem Zustandsdiagramm und unterscheidet den weiteren Ablauf des Prozesses in Abhängigkeit von dem Ergebnis.

Die Methode *meth_pruefe_Boden()* ruft die Aktion *act_pruefe_Boden* auf, die ebenfalls als Statechart ausformuliert ist. Anhand des Zustandsdiagramms in Abbildung 6.23 wird die genaue Vorgehensweise der Überprüfung eines Bodenteils erklärt.

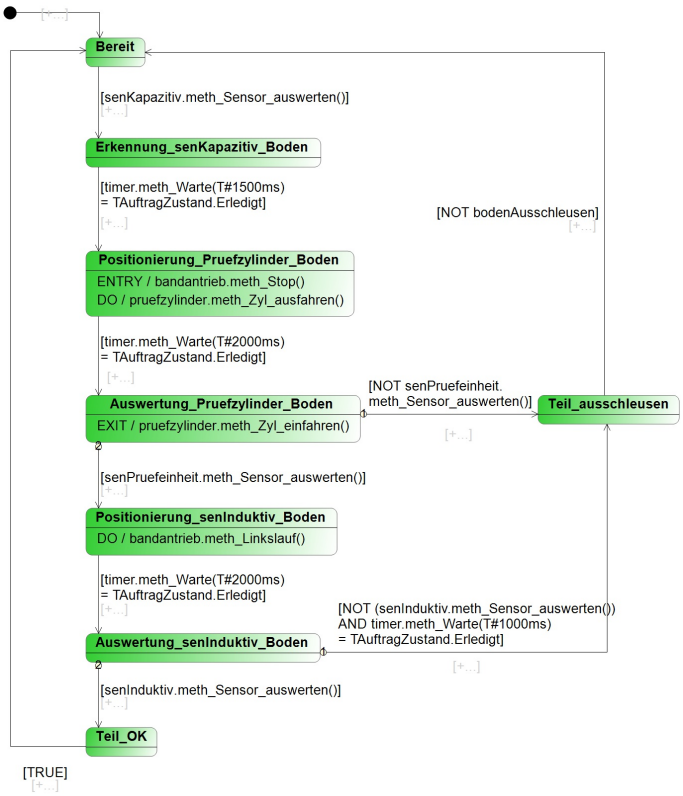

Abbildung 6.23: FB_Band *act_pruefe_Boden*

Wird das Bodenteil vom kapazitiven Sensor erfasst, läuft eine Zeit bis zum stoppen des Bandes ab. Das Werkstück befindet sich dann unterhalb der Prüfeinheit, die feststellt, ob das Teil richtig herum auf dem Transportband liegt. Fällt diese Prüfung negativ aus, liefert die Methode den Wert *Teil_ausschleusen* zurück. Ansonsten wird das Teil durch Einschalten des Motors weiter transportiert. Erkennt der induktive Sensor das Werkstück vor Ablauf der Zeit, hat das Werkstück die Überprüfung bestanden und die Methode gibt den Wert *Teil_OK* zurück.

Auch für dieses Zustandsdiagramm wurde ein Sequenzdiagramm zur Verdeutlichung der Kommunikation zwischen den Objekten erstellt (siehe Abbildung 6.24). Der Aufruf der Me-

thode wird als gefundene Nachricht an das Objekt *Band* modelliert. Die Rückgabewerte der Methode werden dementsprechend als verlorene Nachrichten im Sequenzdiagramm dargestellt. Auch hier werden kombinierte Fragmente für die möglichen Ergebnisse der Sensorauswertungen verwendet.

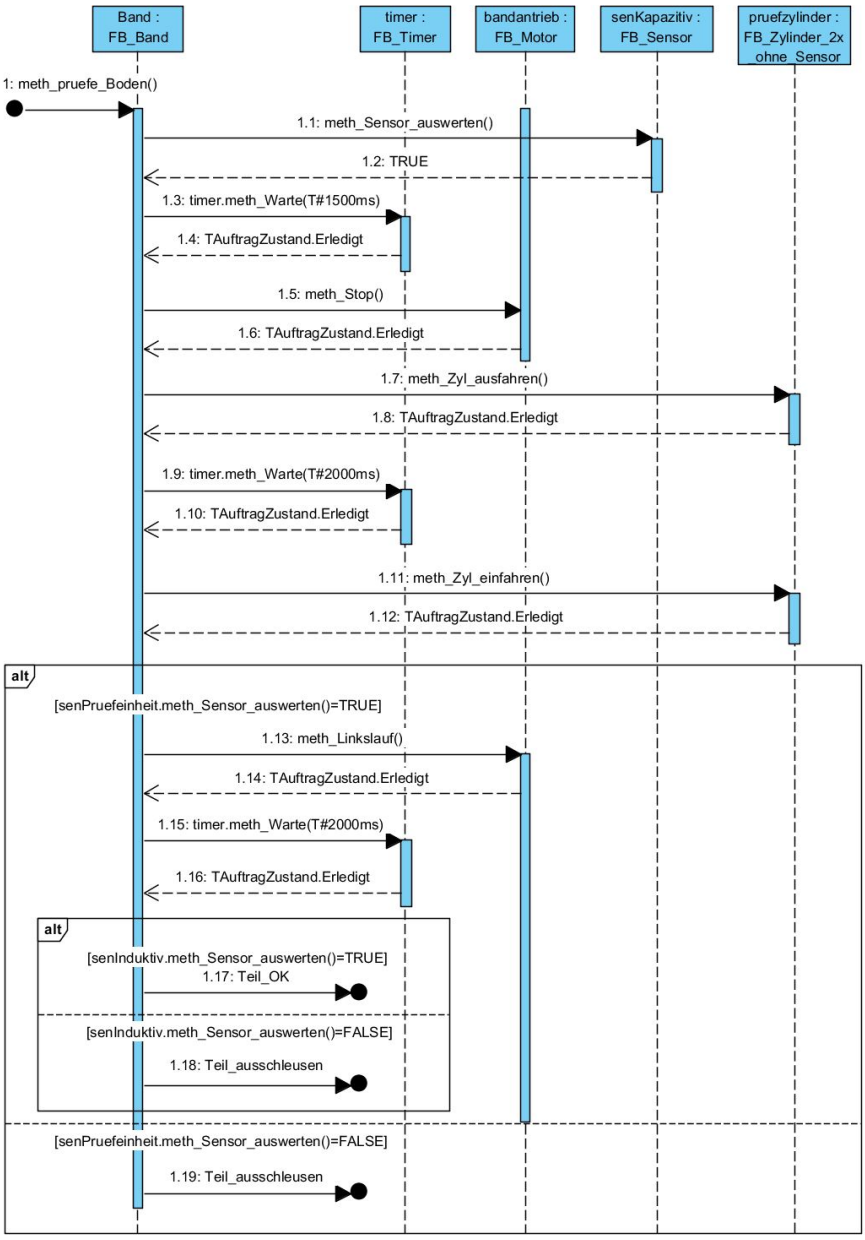

Abbildung 6.24: Sequenzdiagramm Prüfung Bodenteil

6.7.6 FB_Hochregal

Das Zustandsdiagramm der Aktion des Hochregals im Automatikbetrieb ist in der Grafik 6.26 abgebildet. Welche Freigaben zu welchem Zeitpunkt zwischen dem Hochregal und dem Schwenkarm ausgetauscht werden müssen, wird mit Hilfe des nachstehenden Sequenzdiagramms erklärt.

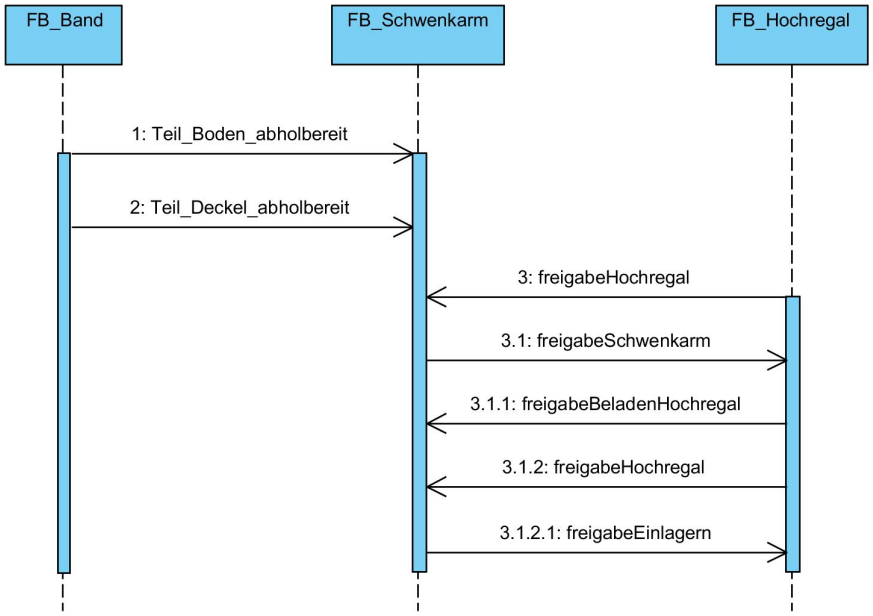

Abbildung 6.25: Sequenzdiagramm zur Darstellung der Kommunikation

Das Diagramm enthält die Signale, die zwischen den drei Raspberry Pis ausgetauscht werden und den Prozessablauf koordinieren. Im Funktionsbaustein *FB_Band* werden Signale erzeugt, die den Abholvorgang des Schwenkarms auslösen. Da es beim Verfahren des Schwenkarms und der Verfahreinheit des Hochregals zu Kollisionen kommen kann, müssen hier gleich mehrere Signale ausgetauscht werden.

Bevor das in der Presse befindliche Werkstück zum Hochregal transportiert werden kann, muss sichergestellt werden, dass sich dieses in der Grundposition befindet. Ist die Variable *freigabeHochregal* gesetzt, nimmt der Schwenkarm das Teil auf und bewegt sich zur Aufnahmeposition des Hochregals. Ist der Schwenkarm positioniert, kann die Verfahreinheit in die Aufnahmeposition fahren. Erteilt das Hochregal die Freigabe zum Beladen (3.1.1) wird der Sauger des Schwenkarms deaktiviert und das Werkstück liegt auf der Verfahreinheit auf. Nach diesem Schritt fährt die Verfahreinheit in die Grundposition. Sobald die Grundposition erreicht ist, wird mit der Variable *freigabeHochregal* der Schwenkarm zur Presse bewegt. Das Hochregal wartet auf die Freigabe zum Einlagern des Werkstücks, bis der Schwenkarm

die Presse erreicht hat und somit der Kollisionsbereich verlassen wurde. Die Variablen werden als asynchrone Nachrichten zwischen den Modulen ausgetauscht. Das bedeutet, dass der angestoßene Prozess sofort startet und keine Rückmeldung an den Sender erfolgt.

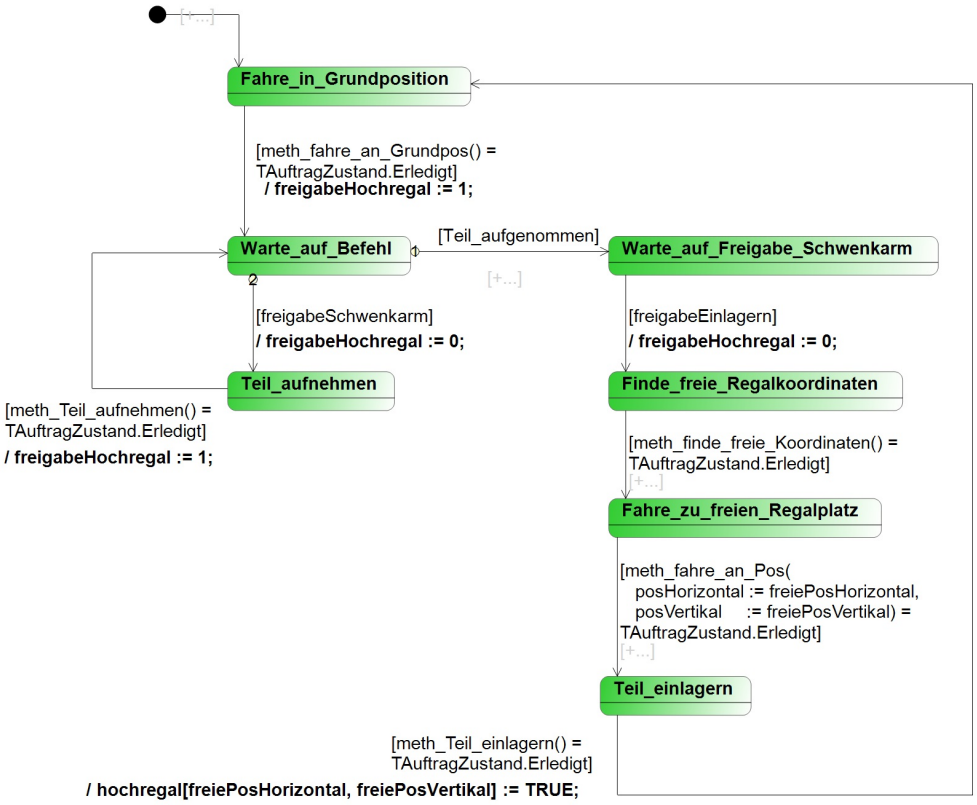

Abbildung 6.26: FB_Hochregal *act_Automatik*

Die Methoden *meth_fahre_an_Pos(...)* und *meth_fahre_an_Grundpos()* sind in ST programmiert und dienen zur Positionierung der Verfahreinheit. Während die Methode zum Fahren in die Grundposition lediglich abfragt, ob die Endlagen erreicht wurden, nutzt die Methode *meth_fahre_an_Pos(...)* die aktuellen Zählerstände zur Positionierung. Die anzufahrende Position kann als (X,Y)-Koordinate an die Methode übergeben werden.

Um zu ermitteln, welche die nächste freie Position im Hochregal ist, kann die Methode *meth_finde_freie_Koordinaten()* aufgerufen werden. Sie iteriert über das zweidimensionale Lagerarray und speichert die Koordinaten einer freien Position in den Attributen der Klasse. Die Methoden zum Aufnehmen und Einlagern eines Werkstücks rufen ihre unterlagerten Aktionen auf, die in SC programmiert sind. Da in den Zuständen dieser beiden Aktionen nur bereits bekannte Methodenaufrufe implementiert sind, wird auf eine Darstellung der Zustandsdiagramme verzichtet.

6.8 Prozessanbindung in CODESYS

Nachdem das Projekt samt seiner Funktionsbausteine und Methoden realisiert wurde, sind noch zwei weitere Schritte notwendig. Als erstes sollen die Variablen, die zwischen den Steuerungen ausgetauscht werden müssen, zu den Netzwerkvariablenlisten hinzugefügt werden. Dieser Vorgang wird unter 6.8.1 erläutert. Im letzten Schritt müssen die Prozessvariablen, die durch die Feldbuskoppler eingelesen werden, mit den Variablen der Applikationen verknüpft werden.

6.8.1 Kommunikation zwischen den Raspberry Pis

Um Daten über das Netzwerk auszutauschen, müssen die in Tabelle 5.1 definierten Netzwerkvariablenlisten verwendet werden. Die Variablen, die für die Koordinierung der Prozesse zuständig sind, können dem Sequenzdiagramm aus Abbildung 6.25 entnommen werden.

Beispielhaft wird die Netzwerkvariablenliste *HR_PR_send*, die die beiden Variablen *freigabeHochregal* und *freigabeBeladenHochregal* an die Presse schicken soll, erläutert. Die Variablen können wie im Variablendeklarationsteil einer Programmorganisationseinheit hinzugefügt werden. Der Codeausschnitt 6.8 zeigt die Definition der beiden Variablen innerhalb der Liste *HR_PR_send*.

```
1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3     freigabeHochregal : BOOL;
4     freigabeBeladenHochregal : BOOL;
5 END_VAR
```

Listing 6.8: NVL *HR_PR_send*

Die dazugehörige Empfänger Variablenliste wird automatisch um alle eingetragenen Variablen ergänzt. Um die Variablen den Attributen der Klasse zuzuordnen, werden in den Applikationen der drei Steuerungen das Programm *Netzwerkkommunikation* hinzugefügt. In diesem Programm werden alle eingehenden und versendeten Daten mit Hilfe der globalen Variablenlisten den entsprechenden Variablen zugeordnet. Der Codeausschnit 6.9 zeigt das Netzwerkkommunikationsprogramm des Hochregals.

```
1 // HR -> PR
2 HR_PR_send.freigabeHochregal := GVL_Aggregate.hochregal.
      freigabeHochregal;
3 HR_PR_send.freigabeBeladenHochregal := GVL_Aggregate.hochregal.
      freigabeBeladenHochregal;
4
```

```
5  // HR -> MA
6  HR_MA_send.anlageZustand := GVL_Anlagenzustand.hochregal;
7
8  // MA -> HR
9  GVL_Anlagenzustand.magazin := MA_HR_receive.anlageZustand;
10
11 // PR -> HR
12 GVL_Aggregate.hochregal.freigabeSchwenkarm := PR_HR_receive.
       freigabeSchwenkarm;
13 GVL_Aggregate.hochregal.freigabeEinlagern := PR_HR_receive.
       freigabeEinlagern;
```

Listing 6.9: Hochregal Netzwerkkommunikation

6.8.2 Variablenadressierung

Da die Feldbuskoppler die Schnittstelle zur Anlage darstellen, müssen die Eingangs- und Ausgangsmodule mit den Variablen der Anlage gekoppelt werden. Das I/O-Abbild der Module kann durch *Klicken* auf diese in der Geräte-Ansicht geöffnet werden. In Abbildung 6.27 ist das I/O-Abbild für ein Eingangsmodul der Steuerung *raspberrypi3_MA* dargestellt.

Variable	Mapping	Kanal	Adresse	Typ
Application.GVL_Aggregate.lagerDeckel.ausschubzylinder.senVorne		Channel 1 Data	%IX6.0	BIT
Application.GVL_Aggregate.Band.senPruefeinheit.sensor		Channel 2 Data	%IX6.1	BIT
Application.GVL_Aggregate.lagerDeckel.ausschubzylinder.senHinten		Channel 3 Data	%IX6.2	BIT
Application.GVL_Aggregate.Band.senLichtschranke.sensor		Channel 4 Data	%IX6.3	BIT
Application.GVL_Aggregate.lagerDeckel.senMagazinLeer		Channel 5 Data	%IX6.4	BIT
Application.GVL_Aggregate.Band.senInduktiv.sensor		Channel 6 Data	%IX6.5	BIT
Application.GVL_Aggregate.Band.senKapazitiv.sensor		Channel 7 Data	%IX6.6	BIT
Application.GVL_Aggregate.Band.senEndlageBand.sensor		Channel 8 Data	%IX6.7	BIT

Abbildung 6.27: Zuweisung der Prozessvariablen

Es empfiehlt sich die Anbindung der Prozessvariablen direkt an die Attribute der Klasse, denn so werden die Attribute zyklisch aktualisiert und beinhalten immer den aktuellen Status. Werden die Variablen beispielsweise in einer globalen Variablenliste zwischengespeichert, müssten sie beim Aufruf der Instanzen (vgl. Codeausschnitt 6.4) mit angegeben werden, da sonst der Status der Attribute nicht aktualisiert wird.

7 Funktionstest

Bei dem Entwurf und der Programmierung von Automatisierungssystemen besteht aufgrund der Vielzahl potentieller Fehlerquellen die Notwendigkeit das System nach dem Fertigstellen der Software zu testen. Die Herausforderung besteht darin, die Anzahl der Fehler möglichst gering zu halten und einen sicheren Betrieb der Anlage zu gewährleisten.

Die Modellanlage wurde anhand der im Kapitel 4 festgelegten Anforderungen getestet und erfüllt diese. Eine mögliche Option zur Dokumentation der Ergebnisse ist die in CODESYS implementierte Trace-Funktion. Die folgenden Abschnitte dokumentieren drei Testszenarien, die kurz beschrieben werden.

7.1 Trace Visualisierung

Die Trace-Funktionalität erlaubt die Aufzeichnung von Werteverläufen der Variablen zur Laufzeit. Eine Trace-Aufzeichnung kann einer Applikation als Objekt hinzugefügt werden. Es können mehrere Diagramme erstellt werden, denen einzeln Variablen zugeordnet werden.

7.1.1 Detektion eines falschen Werkstücks

Das erste Testszenario ist die Detektierung eines falschen Werkstücks auf dem Transportband. Dabei wird ein Bodenteil aus dem Magazin ausgestoßen, das aus Kunststoff ist und vom Prozess aussortiert werden soll.

Die Legende innerhalb der Diagramme gibt die aufgezeichneten Variablen an. Die Diagramme werden für die Beschreibung von oben nach unten durch nummeriert.

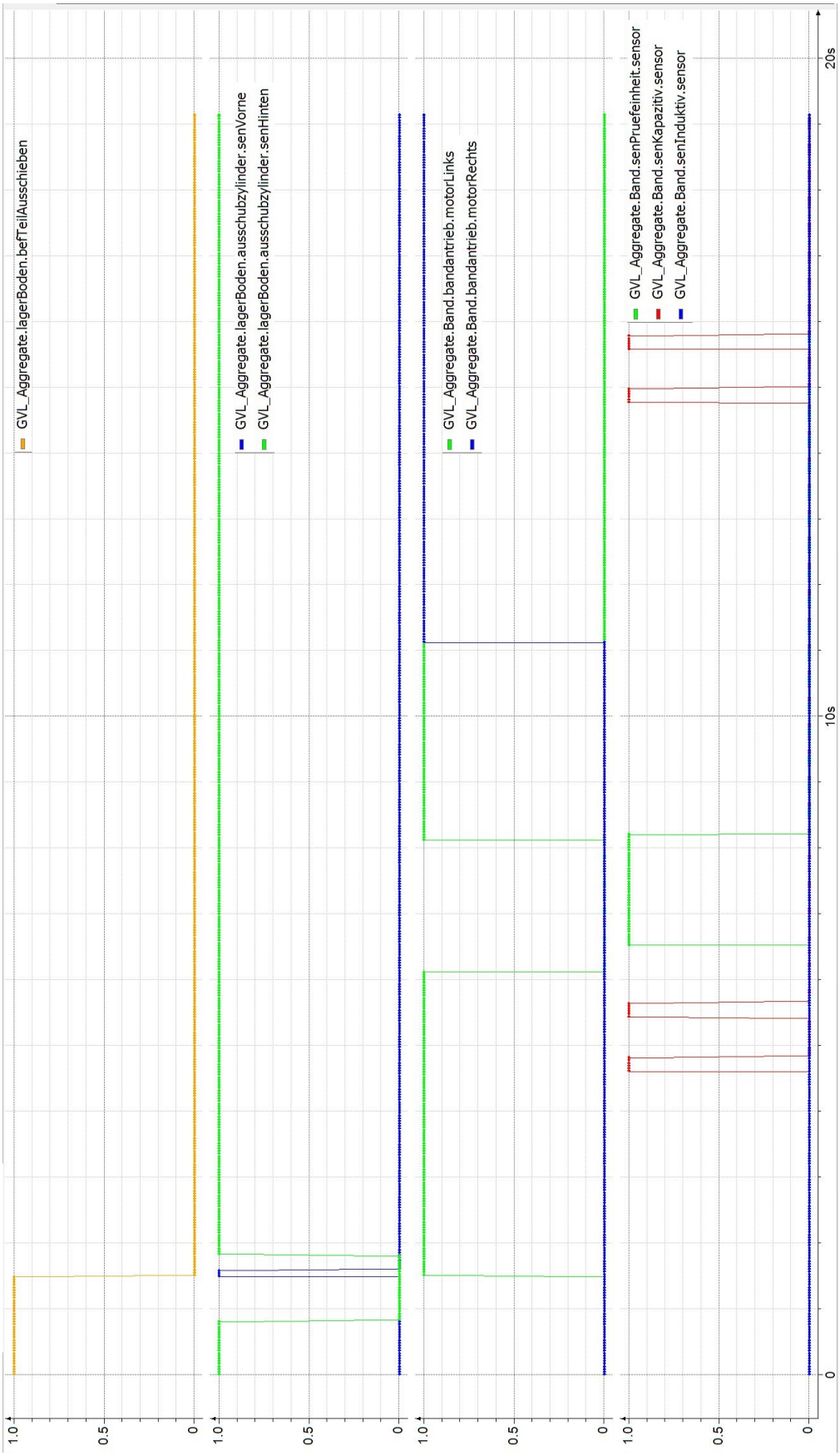

Abbildung 7.1: Trace Transportband

Der Befehl zum Ausschieben eines Bodenteils startet den aufgezeichneten Prozess. Im zweiten Diagramm werden die Signale der Endlagensensoren des Ausschubzylinders gezeigt. Ist der Ausschubzylinder ausgefahren, wird auch der Befehl zum Ausschieben eines Teils zurückgesetzt und das Band im Linkslauf gestartet. Das vierte Diagramm zeigt die aktuellen Werte des kapazitiven Sensors, der Prüfeinheit und des induktiven Sensors. Sobald der kapazitive Sensor (rot) das Werkstück detektiert, läuft eine fest eingestellte Zeit ab. Nach dieser Zeit stoppt das Band und der Prüfzylinder fährt aus. Nach der Auswertung der Prüfeinheit wird das Band wieder im Linkslauf gestartet.

Da der induktive Sensor kein Metall detektiert, wird nach circa elf Sekunden das Band in den Rechtslauf geschaltet und das Werkstück vom Band befördert.

7.1.2 Betätigung des Not-Aus Schalters bei Betrieb des Schwenkarms

Um die Funktion des Not-Aus Schalters zu überprüfen, werden die wichtigsten Variablen des Schwenkarms aufgezeichnet. Nach der Betätigung des Not-Aus Schalters muss der Sauger aktiviert bleiben und die einfachwirkenden Zylinder sollen in ihrer Position bleiben.

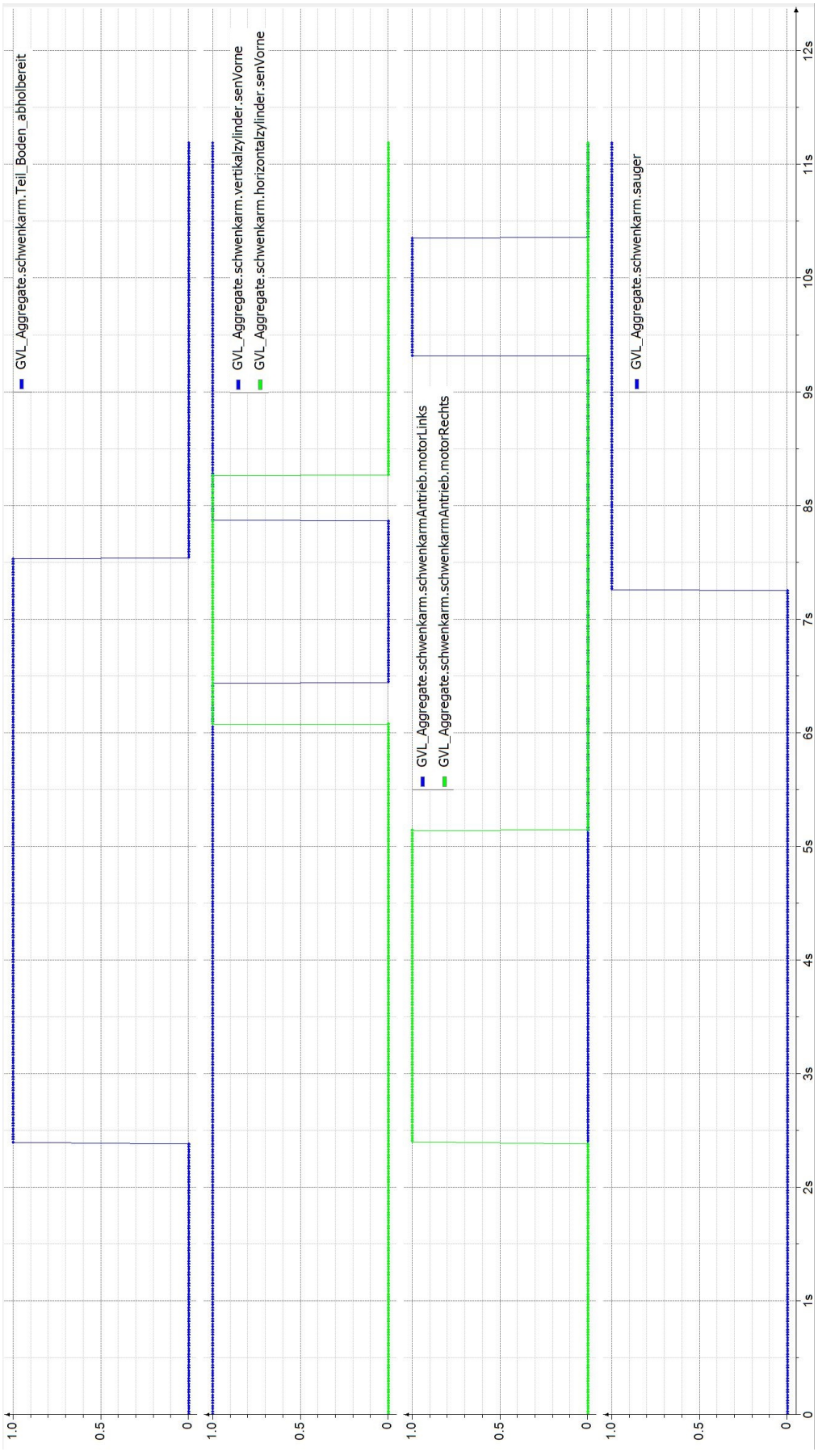

Abbildung 7.2: Trace Schwenkarm

Sobald das Transportband ein abholbereites Teil signalisiert, wird der Schwenkarm zum Band bewegt. Das zweite Diagramm zeigt die ausgefahrenen Zustände des Horizontal- und Vertikalzylinders. Im Zeitraum von sechs bis sieben Sekunden ist der Aufnahmevorgang des Schwenkarms dokumentiert. Sobald dieser abgeschlossen ist, wird der Motor im Linkslauf aktiviert. Während des Transports zur Presse wird der Not-Aus Schalter betätigt. Bei etwa 10.5 Sekunden stoppt das Band dadurch sofort. Der Vakuumsauger bleibt, wie gefordert, auch nach Aktivierung des Not-Aus Schalters eingeschaltet.

7.1.3 Einlagerungsprozess des Hochregals

Der dritte aufgezeichnete Prozess ist die Einlagerung eines Werkstücks im Hochregal (Abbildung 7.3). Durch die erteilte Freigabe vom Schwenkarm (orange) wird der Prozess zur Aufnahme des Werkstücks gestartet. Die Sensoren für die vertikalen und horizontalen Positionen melden die Bewegung beim Verfahren an die Zähler des Hochregals. Die Aufnahmeposition eines Werkstücks entspricht den Koordinaten (6,1). Ist diese Position erreicht, wird der Motor deaktiviert. Um das Teil vom Schwenkarm aufnehmen zu können, wird bei circa 8 Sekunden die vertikale Achse an die Höhe des Schwenkarms angepasst. Der nächste Schritt ist das Fahren in die Grundposition.

In der Zeit zwischen 16 und 20 Sekunden finden keine Aktionen statt, da hier auf die Freigabe zum Einlagern des Werkstücks gewartet werden muss. Das Werkstück wird an der Position (2,1) des Hochregals eingelagert. Das fünfte Diagramm zeigt den ausgefahrenen Zustand des Ausschubzylinders an. Sobald der Zylinder wieder eingefahren ist, fährt die Verfahreinheit zurück in die Grundposition und wartet auf den Befehl für das nächste Werkstück.

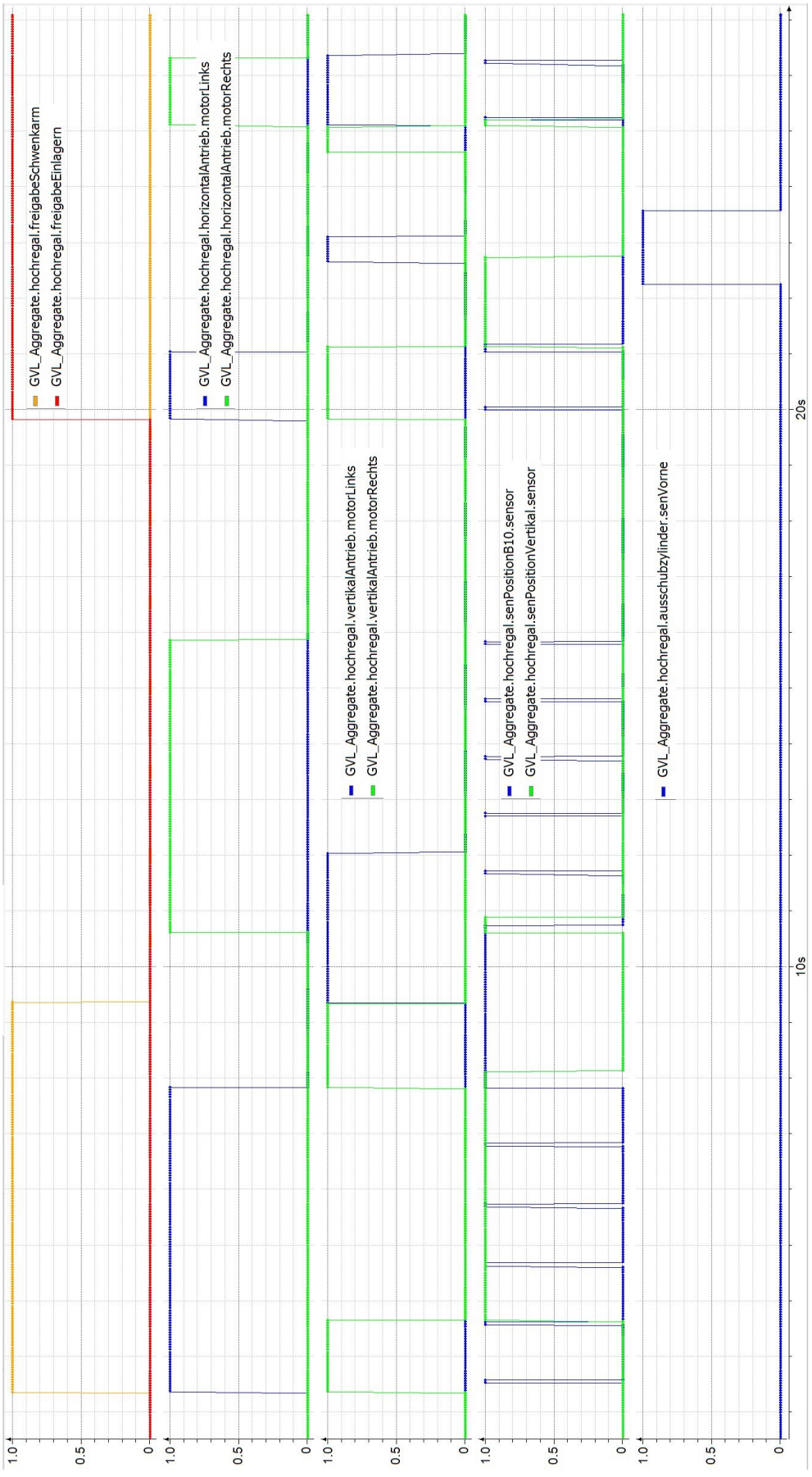

Abbildung 7.3: Trace Hochregal

8 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde der Programmentwurf und die Implementierung einer Automatisierungslösung für eine an der HAW Hamburg befindlichen Modellanlage beschrieben. Der Fokus lag dabei auf der objektorientierten Softwarestruktur und der Nutzung der CODESYS UML Erweiterung. Als Laufzeitsysteme wurden drei Raspberry Pis erfolgreich in die CODESYS Entwicklungsumgebung eingebunden und die notwendige Kommunikation zwischen den Teilnehmern hergestellt. Eine genaue Analyse der Anlage liefert den Grundstein für die Modularisierung und damit der Identifikation der zu definierenden Klassen. Im ersten Schritt der Implementierung wurde mit Hilfe des Klasseneditors von CODESYS die Struktur der Anlage in die Software übertragen. Der zweite Schritt sah die Implementierung der unterlagerten Funktionen vor. Die Vorgehensweise bei der Programmierung, besonders im Umgang mit der Programmiersprache Statechart (SC), wurde anhand der Implementierung mehrerer Klassen dargestellt. Die erstellte Software wurde auf die Modellanlage übertragen und entsprechend der definierten Anforderungen getestet. Zusätzlich wurden drei Testszenarien durch die Trace-Funktionalität von CODESYS dokumentiert und vorgestellt.

Eine besondere Herausforderung stellte die Kommunikation zwischen den einzelnen Raspberry Pi dar. Erst durch die Hilfe des CODESYS Supports konnte das Problem gelöst werden. Wie unter Abschnitt 5.4.3 beschrieben, musste in den Netzwerkeinstellungen der Sender-Netzwerkvariablenliste die Broadcast Adresse auf die tatsächliche IP-Adresse des Empfängers geändert werden.

Bei der Durchführung der Arbeit stellt sich besonders die genaue Beschreibung der Modellanlage und die Modularisierung als wertvoll heraus. Nur wenn die Module klar definiert sind, lassen sich diese sinnvoll in Klassen übertragen. Eine genaue Definition der Schnittstellen vermittelt früh das Zusammenwirken der einzelnen Komponenten im Gesamtprojekt. Durch die sinnvolle Wahl der Module, konnten Klassen, deren Funktionsweise bereits getestet wurde, wiederverwendet werden. Dies führt besonders in der späteren Phase der Programmierung zu einer erheblichen Reduzierung des Entwicklungsaufwands.

Bei der Realisierung des Automatisierungsprozesses wurde der Fokus auf die Modularisierung der Modellanlage und dem objektorientierten Softwareentwurf gelegt. Aus diesem Grund bietet die Anlage noch einige mögliche Verbesserungs- bzw. Optimierungsoptionen, die in dieser Arbeit nicht umgesetzt wurden. Denkbar wäre eine Visualisierung der Anlage oder die Option bereits eingelagerte Werkstücke wieder auslagern zu können.

Die UML-Erweiterung von CODESYS erwies sich als praktikabel. Besonders der integrierte Klasseneditor eignet sich sehr gut um die Struktur eines Systems direkt in die Programmstruktur zu übertragen. Auch die Nutzung der Programmiersprache Statechart (SC) konnte im Projekt gewinnbringend eingesetzt werden. Hier ist jedoch zu erwähnen, dass grafische Fehler, besonders während der Online-Betrachtung von Bausteinen, vorkommen können. Auch besteht noch Verbesserungspotential bei der Nutzung der impliziten Variablen, da der Zugriff auf Zustände innerhalb eines Diagrammes nur über ein Array, dessen Inhalt nur im Online-Modus eingesehen werden kann, möglich ist. Ein Zugriff über die Namen der jeweiligen Zustände würde die Lesbarkeit des Programmcodes fördern.

Literaturverzeichnis

[1] BATHELT, Jens: *Entwicklungsmethodik für SPS-gesteuerte mechatronische Systeme*. Bd. 405. ETH Zurich, 2007

[2] CODESYS: *CODESYS Control for Raspberry Pi SL*. https://store.codesys.com/codesys-control-for-raspberry-pi-sl.html. – Eingesehen am 20.12.2017

[3] CODESYS: *CODESYS Online Help*. https://help.codesys.com/. – Eingesehen am 14.01.2018

[4] CODESYS: *CODESYS UML Produktdatenblatt*. – PDF-Datei befindet sich im Anhang

[5] ELEKTRONIK KOMPENDIUM: *UDP - User Datagram Protocol*. https://www.elektronik-kompendium.de/sites/net/0812281.htm. – Eingesehen am 28.12.2017

[6] ETHERCAT: *EtherCAT Einführung*. www.ethercat.org/pdf/german/Ethercat_einfuehrung_dt.pdf. – Eingesehen am 27.12.2017

[7] IGLER, Bodo ; KRÜMMEL, Nadja ; RIED, Malte ; RENZ, Burkhardt: *Kurzanleitung UML*. https://homepages.thm.de/~hg11260/mat/uml.pdf. – Eingesehen am 20.01.2018

[8] KECHER, Christoph: *UML 2: das umfassende Handbuch;[aktuell zu UML 2.3 und 2.4; UML lernen und effektiv in Projekten anwenden; alle Diagramme und Notationselemente im Überblick; mit zahlreichen Praxisbeispielen in C# und Java; inkl. CD-ROM mit UML-Tools und DIN-A2-Poster mit allen Diagrammen]*. Galileo Press, 2011

[9] LAHRES, Bernhard ; STRICH, Stefan ; RAYMAN, Gregor: Objektorientierte Programmierung. In: *Rheinwerk Computing* 3 (2016)

[10] RASPBERRY PI: *Raspberry Pi 3 Model B*. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. – Eingesehen am 08.01.2018

[11] SCHMITT, Karl: *SPS-Programmierung mit ST: nach IEC 61 131 mit CoDeSys und mit Hinweisen zu STEP 7 V11;[Tools, Tests und Lösungen auf CD-ROM]*. Vogel, 2012

[12] SCHULZ, Gerd ; GRAF, Klemens: *Regelungstechnik 1: lineare und nichtlineare Regelung, rechnergestützter Reglerentwurf*. Walter de Gruyter GmbH & Co KG, 2015

[13] SEITZ, Matthias: *Speicherprogrammierbare Steuerungen für die Fabrik-und Prozessautomation: Strukturierte und objektorientierte SPS-Programmierung, Motion Control, Sicherheit, vertikale Integration*. Carl Hanser Verlag GmbH Co KG, 2015

[14] VOGEL-HEUSER, Birgit: *Automation & embedded systems*. Oldenbourg Industrieverlag, 2008

[15] VOGEL-HEUSER, Birgit: *Automation and Embedded Systems: Effizienzsteigerung Im Engineering*. Bd. 1. kassel university press GmbH, 2009

[16] WAGO KONTAKTTECHNIK GMBH & CO. KG: *Feldbuskoppler EtherCAT*. `https://www.wago.com/de/io-systeme/feldbuskoppler-ethercat/p/750-354`. – Eingesehen am 20.02.2018

[17] WAGO KONTAKTTECHNIK GMBH & CO. KG: *EtherCAT Feldbuskoppler 100 Mbit/s; digitale und analoge Signale*, 2015

[18] WELLENREUTHER, Günter ; ZASTROW, Dieter: *Automatisieren mit SPS: Theorie und Praxis*. Springer-Verlag, 2005

A Anhang

Der Anhang zur Arbeit befindet sich auf CD und ist einzusehen bei Prof. Dr.-Ing. Ulfert Meiners oder Prof. Dr. Marc Hensel.

A.1 Produktdatenblatt CODESYS UML

Datenblatt CODESYS UML

Als Teil der CODESYS Professional Developer Edition erweitert CODESYS UML das CODESYS Development System (32/64 Bit) um zwei Sprachen der Unified Modeling Language: das Klassendiagramm und das Zustandsdiagramm.

Produktbeschreibung

Die UML (Unified Modeling Language) ist eine grafische Sprache für die Spezifikation, das Design und die Dokumentation von objektorientierter Software. Sie bietet eine allgemein verständliche Diskussionsbasis zwischen Programmierung und anderen Fachbereichen innerhalb der Systementwicklung zu schaffen.

Die UML besteht aus 14 verschiedenen Diagramm-Typen in zwei Hauptkategorien: Strukturdiagramme und Verhaltensdiagramme. Strukturdiagramme stellen die Architektur der Software schematisch dar und dienen im Wesentlichen der Modellierung und Analyse (z.B. Projektdesign, Spezifikation von Systemanforderungen, Dokumentation). Verhaltensdiagramme sind ausführbare Modelle mit eindeutiger Syntax und Semantik, aus denen direkt Anwendungscode generiert werden kann („model driven architecture“).

CODESYS UML erweitert das CODESYS Development System um zwei Sprachen der Unified Modeling Language: das Klassendiagramm und das Zustandsdiagramm, unabhängig davon ob es sich um die 32- Bit oder 64 Bit-Variante des CODESYS Development System handelt.

Funktionsumfang

Klassendiagramm

Das Klassendiagramm gehört der Gruppe der UML-Strukturdiagramme an. Mit dem zusätzlichen graphischen Editor kann die objektorientierte Struktur von CODESYS-Projekten abgebildet oder designt werden. Die verschiedenen Objektklassen (z.B. Funktionsblöcke oder Schnittstellen) inklusive der darin verwendeten Variablen und Methoden und deren Beziehungen werden übersichtlich dargestellt (Bild 1).

Die bestehende Projektstruktur kann bei Neuanlage eines Klassendiagramms direkt aus dem Gerätebaum importiert werden. Eine Projektstruktur kann aber auch mit Hilfe der zur Verfügung stehenden Klassen- und Beziehungselemente neu aufgebaut werden:

- Klasse (POU)
- Schnittstelle
- Variablendeklaration
- Eigenschaft
- Methode
- Generalisierung (EXTENDS)
- Realisierungsbeziehung (IMPLEMENTS)
- Assoziation (POINTER)
- Komposition (VAR)

Neue Objekte im Klassendiagrammeditor werden automatisch auch im Gerätebaum eingefügt.

Zustandsdiagramm

Das Zustandsdiagramm gehört der Gruppe der UML-Verhaltensdiagramme an. Es stellt eine grafische Sprache dar, um den Ablauf ereignisdiskreter Systeme zu spezifizieren und zu designen (Bild 2). Im Gegensatz zum Klassendiagramm wird beim Kompilieren eines Zustandsdiagramms ausführbarer Applikationscode erzeugt. Der Zustandsdiagrammeditor beinhaltet eine Auswahl von Schritt- und Übergangselementen:

- Startzustand
- Endzustand
- Zustand
- Zusammengesetzter Zustand
- Gabelung / Verbindung

- Auswahl
- Transition
- Abschlusstransition
- Ausnahmetransition

Bei laufender Applikation wird das Zustandsdiagramm in Abhängigkeit des Taktzyklus geschaltet. Darüber hinaus kann ein unabhängiges Schaltverhalten über sogenannte zyklusinterne Zustandsdiagramme realisiert werden. Im Online-Modus wird das Zustandsdiagramm animiert, so dass der aktuelle Status des Ablaufs jederzeit verfolgt werden kann.

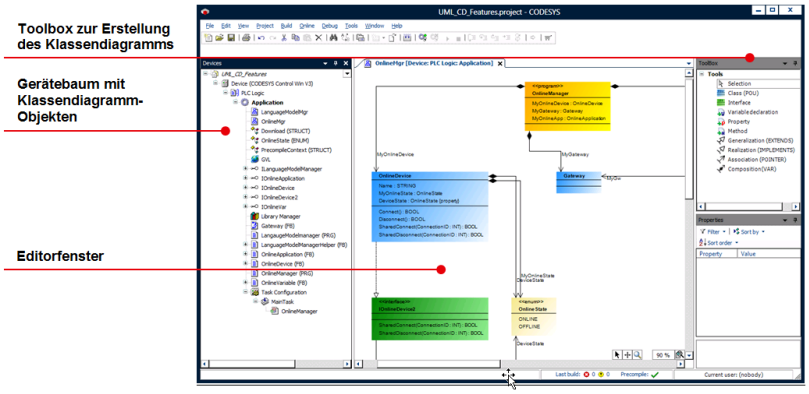

Bild 1: CODESYS Development System mit integriertem Klassendiagramm-Editor

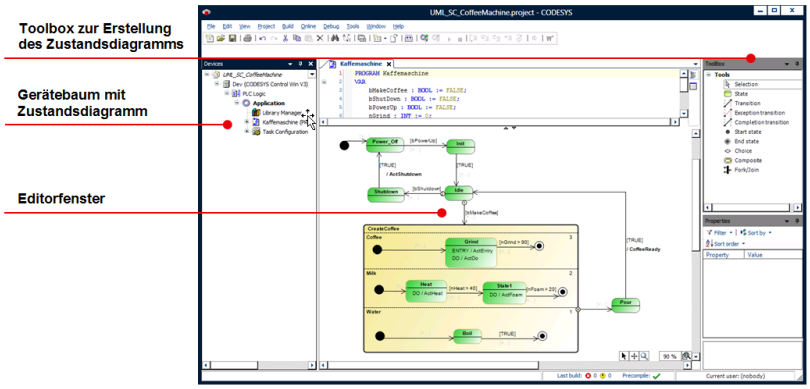

Bild 2: CODESYS Development System mit integriertem Zustandsdiagramm-Editor

Allgemeine Informationen

Herstelleradresse:

3S-Smart Software Solutions GmbH
Memminger Straße 151
87439 Kempten
Deutschland

Support: https://support.codesys.com

Dieses Produkt beinhaltet ein CODESYS Support Ticket, d.h. 1 Stunde Support von CODESYS. Mehr Details dazu finden Sie im CODESYS Store Produkt: CODESYS Support Ticket.

Artikelname: CODESYS UML

Artikelnummer: 2101000001

Vertrieb: CODESYS Store
https://store.codesys.com

Lieferumfang:

- Package für das CODESYS Development System inklusive Lizenzvereinbarung und Online-Hilfe
- Lizenzschlüssel

Sytemvoraussetzungen und Einschränkungen

Programmiersystem	CODESYS Development System V3.5.3.0 oder höher
Laufzeitsystem	CODESYS Control Version 3.5.3.0 oder höher
Unterstützte Plattformen/ Geräte	-
Zusätzliche Anforderungen	-
Einschränkungen	64 Bit Unterstützung ab Version 4.0.3.0 und höher
Lizensierung	Lizenzaktivierung auf CODESYS Security Key
Erforderliches Zubehör	CODESYS Security Key: http://store.codesys.com/accessories.html

Bitte beachten Sie: Nicht alle CODESYS-Funktionen sind in allen Ländern verfügbar. Weitere Informationen zu diesen länderspezifischen Einschränkungen erhalten Sie unter sales@codesys.com.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 12. März 2018	
Ort, Datum	Unterschrift