



**Universität Stuttgart**

Institut für Steuerungstechnik  
der Werkzeugmaschinen und  
Fertigungseinrichtungen (ISW)



Forschungs- und Entwicklungspraxis

# **Konzeption und Implementierung einer Diagnoseschnittstelle zwischen Industrieroboter und Anlagensteuerung**

eingereicht von

*Lukas Schlotter*

aus Stuttgart

Studiengang

M. Sc. Mechatronik

Prüfer

Prof. Dr.-Ing. Alexander Verl

Betreuer

Dr.-Ing. Andreas Wolf, Dipl.-Ing. Daniel Knauss

Eingereicht am

5. Februar 2024

# Eigenständigkeitserklärung

Forschungs- und Entwicklungspraxis von Lukas Schlotter (M. Sc. Mechatronik)

Anschrift	Bläuering 16, 72160 Horb
Matrikelnummer	3668915
Deutscher Titel	<i>Konzeption und Implementierung einer Diagnoseschnittstelle zwischen Industrieroboter und Anlagensteuerung</i>
Englischer Titel	<i>Design and Implementation of a Diagnostic Interface between Industrial Robot and Control System</i>

Hiermit erkläre ich,

- dass ich die vorliegende Arbeit selbstständig verfasst habe,
- dass keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet sind,
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
- dass ich die Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe und
- dass ich mit der Arbeit keine Rechte Dritter verletze und die Universität von etwaigen Ansprüchen Dritter freistelle.

---

Stuttgart, den 5. Februar 2024

# Kurzfassung

Roboter spielen eine zentrale Rolle. Daten. Neue Geschäftsmodelle

**Stichwörter:** Hamster, Training, Anleitung

# **Abstract**

Add the english abstract here.

**Keywords:** hamster, training, guide

# Inhaltsverzeichnis

<b>Kurzfassung</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	1
1.3 Methodik und Vorgehensweise . . . . .	2
<b>2 Grundlagen und Stand der Technik</b>	<b>5</b>
2.1 Feldbusse . . . . .	5
2.1.1 Standard-Feldbusse . . . . .	6
2.1.2 Ethernet basierende Feldbusse . . . . .	7
2.2 TCP/IP . . . . .	10
2.3 OPC/UA . . . . .	13
2.4 Stäubli-Roboter . . . . .	14
2.4.1 Controller CS9 . . . . .	15
2.4.2 Programmiersprache VAL 3 . . . . .	16
2.5 Anlagensteuerung . . . . .	19
2.6 WPF-Anwendung . . . . .	20
2.7 Fazit . . . . .	22
<b>3 Anforderungsdefinition</b>	<b>23</b>
<b>4 Konzeptionierung</b>	<b>26</b>
4.1 Feldbus-Verbindung . . . . .	26
4.2 TCP/IP-Verbindung . . . . .	29
4.3 Datenverwertung . . . . .	34
4.4 Stäubli Roboterprogramm . . . . .	38
4.5 WPF-Anwendung . . . . .	41
<b>5 Design und Implementierung</b>	<b>47</b>
5.1 Stäubli-Roboter in VAL 3 . . . . .	47
5.1.1 EtherCAT . . . . .	47
5.1.2 TCP/IP . . . . .	48

## *Inhaltsverzeichnis*

5.2 WPF-Anwendung . . . . .	51
5.2.1 TCP/IP-Server . . . . .	51
5.2.2 Daten-Logger . . . . .	54
5.2.3 Diagramm . . . . .	55
5.2.4 Benutzeroberfläche . . . . .	56
<b>6 Validierung</b>	<b>59</b>
6.1 Funktionstests . . . . .	59
6.2 Fehlertests . . . . .	63
<b>7 Ausblick und Fazit</b>	<b>66</b>
<b>Abbildungsverzeichnis</b>	<b>67</b>
<b>Tabellenverzeichnis</b>	<b>69</b>
<b>Literatur</b>	<b>70</b>

## **Abkürzungsverzeichnis**

<b>SPS</b>	Speicherprogrammierbare Steuerung
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>TCP</b>	Transmission Control Protocol
<b>IP</b>	Internet Protocol
<b>WPF</b>	Windows Presentation Foundation
<b>LAN</b>	Local Area Network
<b>CSMA/CD</b>	Carrier Sense Multiple Access with Collision Detection
<b>FMMU</b>	Fieldbus Memory Management Unit
<b>UDP</b>	User Datagram Protocol
<b>CRC</b>	Cyclic Redundancy Check
<b>MAC</b>	Medium Access Control
<b>OPC UA</b>	Open Platform Communication Unified Architecture
<b>SRS</b>	Stäubli Robotic Suite
<b>IPC</b>	Indutrie-PC
<b>ADS</b>	Automation Device Specification
<b>XAML</b>	eXtensible Application Markup Language
<b>MVVM</b>	Model-View-ViewModel

*Inhaltsverzeichnis*

**MVC** Model-View-Controller

**UML** Unified Modeling Language

# 1 Einleitung

## 1.1 Motivation

Roboter sind in unserer Welt nicht mehr wegzudenken. In unterschiedlichsten Industrien verrichten sie zuverlässig ihre Arbeit und machen eine Produktion auch in Hochlohnländern wirtschaftlich. Ihre Arbeitsleistung ist konstant, fehlerfrei und präzise. In der Lebensmittel Industrie gelten besondere Hygiene-Anforderungen. Die „Stäubli-AG“ bietet mit ihrer HE-Serie Roboter an, die speziell für diese Anforderungen entwickelt wurden. Mit ihnen können beispielsweise Wiener-Würstchen verpackt werden. Um eine solche anspruchsvolle Aufgabe zu bewältigen, entwickelt die „robomotion GmbH“ komplexe Automatisierungslösungen ganz individuell nach den Kundenbedürfnissen. Eine solche Anlage setzt sich neben einem oder mehreren Robotern aus vielen weiteren Komponenten, wie z.B. der zentralen Steuerung durch die Speicherprogrammierbare Steuerung (SPS), einem Kamerasytem, Förderbändern und der Anlagenvisualisierung zusammen. Um ein gutes Zusammenspiel aller Komponenten zu gewährleisten, ist eine Kommunikation zwischen ihnen von großer Bedeutung. Dies ist besonders wichtig, da die Datenmenge in der heutigen Zeit kontinuierlich wächst und Kunden ein großes Interesse an Informationen über ihre Anlagen und Roboter haben.

## 1.2 Zielsetzung

In der „robomotion GmbH“ sollen Stäubli-Roboter eingeführt und in das bestehende Anlagensteuerungskonzept integriert werden. Hierzu soll eine bidirektionale Kommunikationsschnittstelle zwischen der Anlage und dem Roboter entworfen und implementiert werden.

Diese setzt sich aus zwei getrennten Einheiten zusammen. Eine Feldbus-Verbindung stellt den Signalaustausch zwischen Anlagen-SPS und Roboter sicher, um z.B. Befehle zur Bewegungsausführung zu übertragen.

Darüber hinaus müssen Fehlermeldungen und Prozessdaten des Roboters zur Anlage und Positionsdaten, generiert aus Kamerabildern, von der Anlage zum Roboter übertragen werden. Die Visualisierung und Verwertung dieser Daten erfolgen jedoch nicht in der SPS sondern in einer separaten Anlagenvisualisierung.

Aus diesem Grund ist neben dem Feldbus eine weitere Verbindung nötig. Im Rahmen dieser Arbeit muss daher neben dem Feldbus zwischen Roboter und SPS eine Kommunikationsverbindung zwischen Roboter und Anlagenvisualisierung realisiert werden.

## *1 Einleitung*

Diese ist mit Hilfe von Transmission Control Protocol/Internet Protocol (TCP/IP) zu konzeptionieren, entworfen und implementieren. Der Schwerpunkt dieser Arbeit liegt auf der TCP/IP-Verbindung und nicht auf der Feldbus-Verbindung, da der Einsatz der TCP/IP-Verbindung zur Roboterkommunikation unternehmensintern neu ist.

Ebenso muss die Anlagenvisualisierung mit Hilfe von Windows Presentation Foundation (WPF) für ein Windows-Betriebssystem konzeptioniert und programmiert werden. Diese WPF-Anwendung soll dem Bediener die Fehlermeldungen und Prozessdaten des Roboters anzeigen. Auf der anderen Seite muss das Roboterprogramm für den Stäubli-Roboter konzeptioniert und programmiert werden.

Ein Praxistest soll abschließend das entwickelte Gesamtsystem validieren. Darüber hinaus sollen die Potentiale, welche durch die zur Verfügung stehenden Daten entstehen, sowie mögliche Verwendungsszenarien aufgezeigt werden.

## **1.3 Methodik und Vorgehensweise**

Die Verwendung geeigneter Methoden und eines strukturierten Vorgehens sind entscheidend für eine erfolgreiche Umsetzung des Projektes. Durch ein Vorgehensmodell wird eine effiziente Arbeitsweise gefördert und durch eine Terminplanung eine zeitgerechte Fertigstellung sichergestellt. Methoden und Werkzeuge, wie bspw. ein morphologischer Kasten oder ein Harvey-Diagramm erleichtern es neue Lösungsansätze zu finden und faktenbasierte Entscheidungen zu treffen. [1] [2]

### **Vorgehensweise**

Die Vorgehensweise lehnt sich an Sommerville [3] und an den ISW-Leitfaden [2]. Da es sich in dieser Arbeit um kein reines Softwareprojekt handelt, sondern mechatronische Komponenten, wie der Roboter und die Kommunikationsschnittstelle enthalten sind erfolgt eine Anpassung der Vorgehensweise an die gegebenen Randbedingungen. Hieraus ergibt sich folgende Abfolge:

- **Recherche zu Grundlagen und Stand der Technik:** Zu Beginn soll zur Einarbeitung eine Recherche zu den Grundlagen von Feldbussen und TCP/IP erfolgen. Ebenso muss eine Recherche zum Stand der Technik bezüglich des Roboters und der Anlagensteuerung von „robomotion“ erfolgen. In der Literatur soll nach gängigen Umsetzungen von TCP/IP-Verbindungen im Kontext von Robotern gesucht werden.
- **Detaillierung der Anforderungen:** Die grobe Aufgabenstellung soll detailliert und in ein Lastenheft überführt werden.
- **Konzeptionierung:** Hier soll die Feldbus-Verbindung zwischen Roboter und SPS konzeptioniert werden. Vor allem ist die TCP/IP-Kommunikation zu konzeptuieren und der dazugehörige Nachrichtenaufbau und Kommunikationsablauf

## *1 Einleitung*

festzulegen. Ebenso sollen die zur Verfügung stehenden Daten hinsichtlich ihres Nutzens analysiert und die zu überragenden Daten gewählt werden. Abschließend soll das Roboterprogramm und die WPF-Anwendung konzeptioniert und die Softwarekomponenten spezifiziert werden.

- **Softwaredesign:** In diesem Rahmen soll ein Systementwurf der Software für den Stäubli-Roboter und vor allem der WPF-Anwendung erfolgen. Lösungsbausteine, wie Softwarebibliotheken müssen festgelegt und Lösungsansätze entworfen werden.
- **Softwareimplementierung:** Die konzeptionierte Software muss in Code umgesetzt werden. Bei der Programmierung ist auf verständliche Kommentierung zu achten und ein Source-Code-Verwaltungs-Programm ist einzusetzen.
- **Validierung:** Durch eine abschließende Validierung ist das System unter Realbedingungen zu testen. Dies soll ein Funktionstest beinhalten, um zu überprüfen, dass alle Funktionen ihre Aufgaben erfüllen. Fehlertests prüfen die Robustheit des Systems. Hierzu werden bewusst besondere Situationen oder Störungen herbeigeführt um die Fehlertoleranz des Systems zu überprüfen und sicherzustellen, dass das System nicht abstürzt. [2] [3]

Die schriftliche Ausarbeitung folgt angelehnt an diese Vorgehensweise. Eine vollständige Aufführung der Implementierung, welche sich durch den gesamten Quell-Code darstellt ist nicht sinnvoll. Aus diesem Grund wird die Implementierung mit in das Kapitel Design integriert und lediglich besonders erwähnenswerte und für das Verständnis wichtige kurze Code-Abschnitte aufgezeigt. Abgerundet wird die Arbeit durch ein Fazit und einem Ausblick, in dem Anwendungspotentiale der übertragenen Daten aufgezeigt werden sollen.

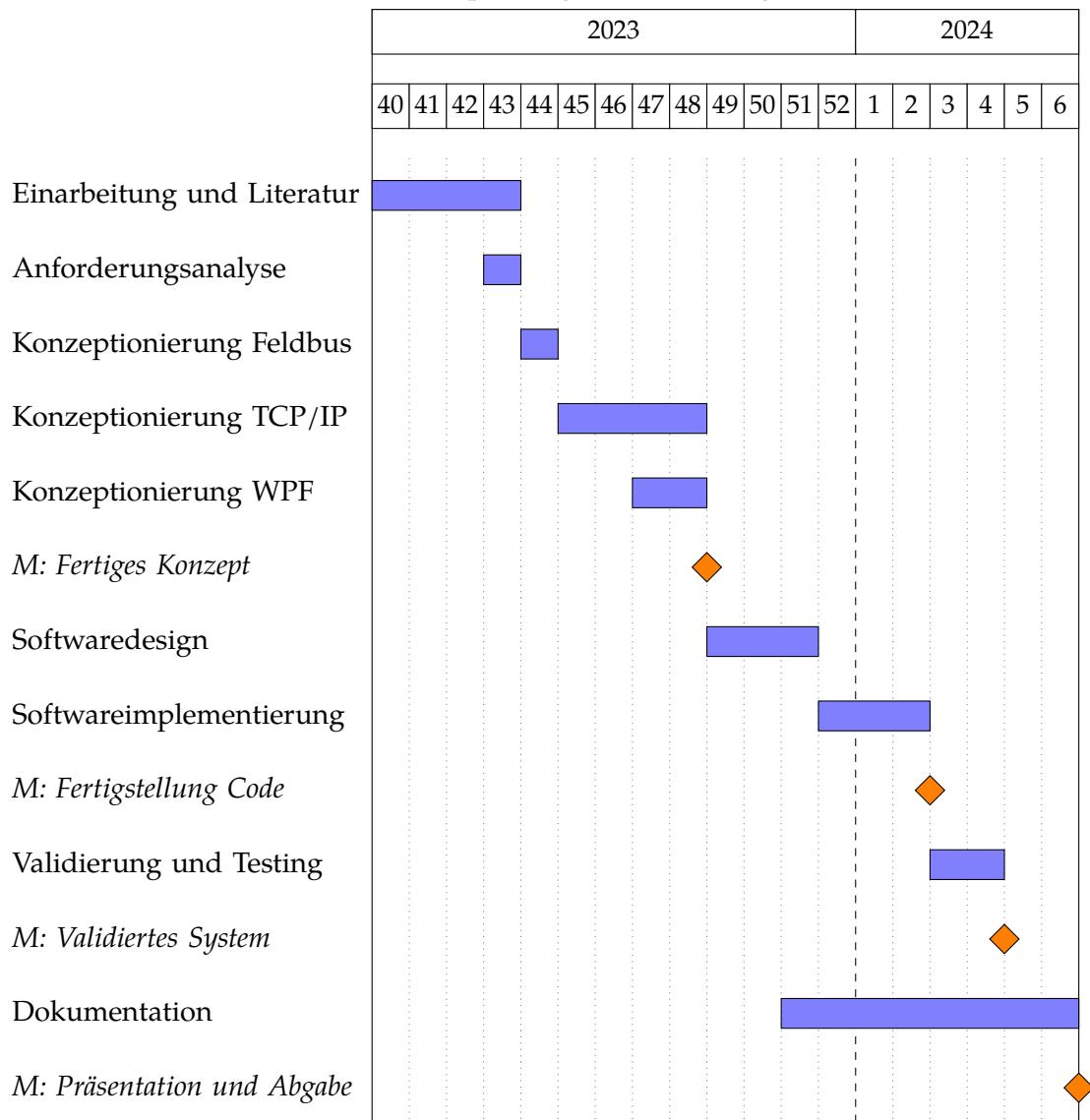
### **Terminplanung**

Für eine fristgerechte Fertigstellung und ein strukturiertes Vorgehen ist es wichtig zu Beginn eine Terminplanung durchzuführen. Hierzu kommt ein Gantt-Diagramm zum Einsatz (vgl. Tabelle 1.1). Die horizontalen Balken stellen die einzelnen Arbeitspakete und deren Zeitumfang dar. Meilensteine, sind wichtige Projekt-(Zwischen)-Ziele mit einer Zeidauer Null, welche mit einer Raute dargestellt werden. [2]

Um frühzeitig Umsetzungsschwierigkeiten zu erkennen und Lösungskonzepte zu testen soll parallel zu der Konzeptionierung ein Softwareprototyp mitgeführt werden. Ebenso soll die Datenübertragung möglichst früh im Projektverlauf an der Hardware getestet werden. Dadurch kann die entworfene Verbindung, sowie das Softwarekonzept zwischenzeitlich abgesichert werden. Dies verhindert, dass sich erst in der Implementierung herausstellt, dass grundlegende Funktionalitäten nicht funktionieren. Das parallel testen mit Hilfe von Prototypen ist im Gantt-Diagramm nicht aufgezeigt, da dieses über den gesamten Projektzeitraum in unterschiedlicher Intensität erfolgt.

## 1 Einleitung

Tabelle 1.1: Terminplanung mit Gantt-Diagramm



## 2 Grundlagen und Stand der Technik

### 2.1 Feldbusse

Maschinen und Anlagen verfügen über eine Vielzahl an Sensoren und Aktoren. Diese werden klassisch über Parallelverdrahtung an die Ein- und Ausgänge der SPS angeschlossen. Ein Feldbus ermöglicht es dagegen die Sensoren und Aktoren an einzelne aktive Verteilerboxen anzuschließen. Die Verteilerboxen werden wiederum mit einem Feldbus untereinander und mit der SPS verbunden. Dies ermöglicht eine Dezentralisierung, was den Schaltschrank vereinfacht, die Energieeffizienz steigert und die Flexibilität bezüglich Änderungen erhöht. Der Übergang von der Parallelverdrahtung zur Feldbusverdrahtung ist mit dem Zwischenschritt von passiven Verteilerboxen nachfolgend dargestellt (vgl. Abbildung 2.1). Bei der Feldbus Anschaltung wird meist, wie in der Abbildung dargestellt, eine Linien-Struktur verwendet. Alternativen sind die Stern-, Ring- oder Baum-Struktur. [4]

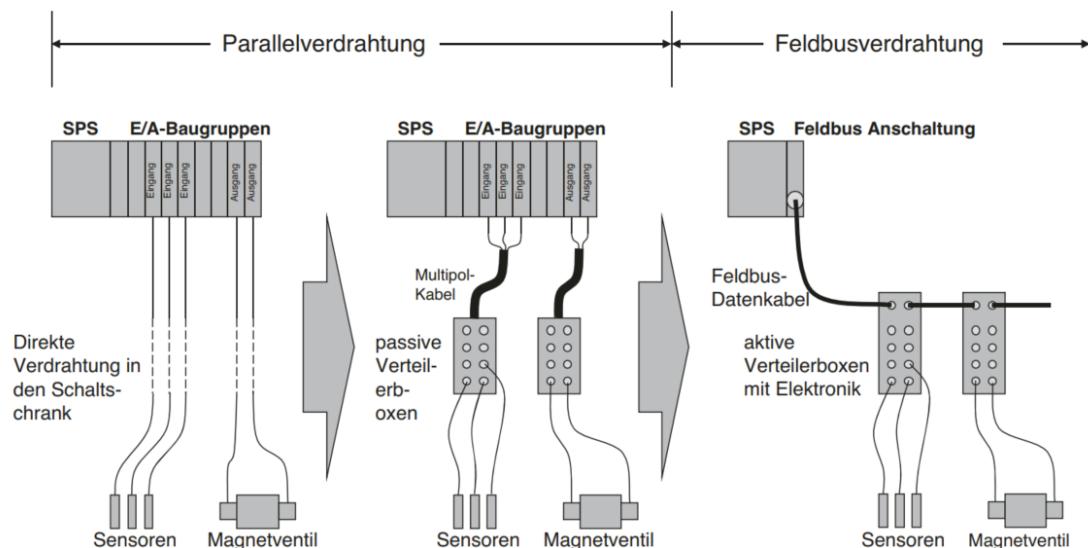


Abbildung 2.1: Übergang Parallelverdrahtung zu Feldbussen [4]

Es haben sich Feldbusse von vielen namhaften Herstellern etabliert. Mittlerweile werden diese Standard-Feldbusse immer weiter von Ethernet basierenden Feldbussen (auch Industrial Ethernet genannt) abgelöst, da diese vor allem Vorteile bezüglich der Übertragungsgeschwindigkeit aufweisen. Die Marktaufteilung aus dem Jahr 2021 ist nachfolgend

## 2 Grundlagen und Stand der Technik

abgebildet, wobei eine weitere Zunahme der Marktanteile von den Ethernet basierenden Feldbussen zu erwarten ist (vgl. Abbildung 2.2). [4] [5]

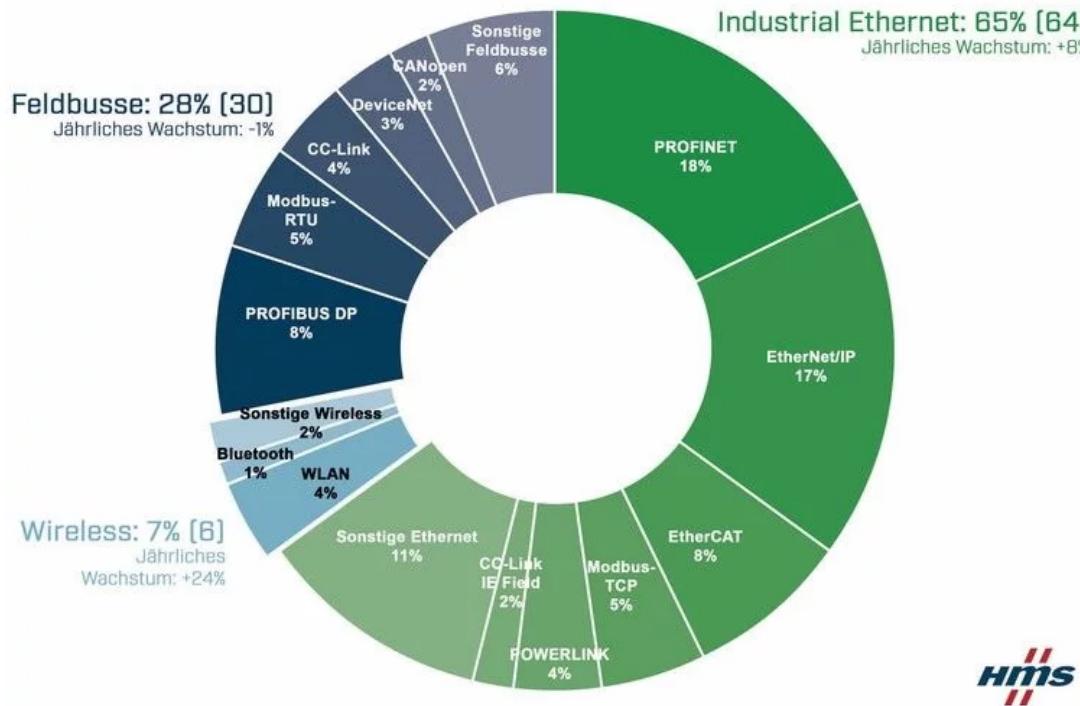


Abbildung 2.2: Marktanteile Feldbusse und Industrial Ethernet [5]

### 2.1.1 Standard-Feldbusse

In der Anlagentechnik ist die sofortige und zuverlässige Datenübertragung unter anderem aus Gründen der Sicherheit essentiell. Feldbusse sind daher echtzeitfähig. Die Echtzeitfähigkeit besagt, dass die Daten innerhalb einer sehr kurzen festgelegten Zeitspanne, unabhängig von äußereren Einflüssen, übertragen werden müssen. Durch die harte Echtzeit, ist im Gegensatz zur weichen Echtzeit zusätzlich definiert, dass die Daten mit einem absoluten Determinismus übertragen werden müssen. Bei der weichen Echtzeit verlieren die Daten lediglich an Nutzen bei verspätetem Eintreffen, bei der harten Echtzeit werden sie komplett nutzlos. Aus diesem Grund müssen bei der harten Echtzeit 100 % der Daten innerhalb der definierten Zeit übertragen werden, was insbesondere bei sicherheitsrelevanten Funktionen von Relevanz ist. [6] [7]

Im folgenden sollen ausgewählte gängige Feldbusse kurz erwähnt und erläutert werden.

## 2 Grundlagen und Stand der Technik

**Profibus** wird von dem Dachverband „Profibus & Profinet International“ verwaltet, um die Interessen der Nutzer einzubringen. Neben dem Profibus auf Feldbus-Ebene (auch Profibus-DP genannt) existiert eine weitere Variante auf Zellensteuerungs-Ebene und eine auf Prozess-Automatisierungs-Ebene. Als Übertragungstechnik dient der sogenannte RS485-Standard mit einer Zweidrahtleitung. Die Datentransferrate von Profibus beträgt bis zu 12 MBit/s. Speicherprogrammierbare Steuerungen von Siemens setzen häufig Profibus ein. [4]

**CAN-Bus** wurde ursprünglich für die Automobil-Industrie entwickelt und kommt dort heute noch zum Einsatz. Darüber hinaus, nahm die Verbreitung in der Anlagensteuerung zu. Möglich sind Datentransferraten von bis zu 1 MBit/s. Als Übertragungsmedium dient eine Zweidrahtleitung. CAN-Bus zeichnet sich durch eine besonders hohe Datensicherheit, also eine hohe Zuverlässigkeit bei der Datenübertragung aus, weshalb er in der Medizintechnik und Robotik hohen Zuspruch findet. [4]

**DeviceNet** basiert auf CAN und ist eine Entwicklung des nordamerikanischen Herstellers „Rockwell Automation“. Die Datenübertragungsrate beträgt bis zu 500 kBit/s und die Spannungsversorgung, sowie die Datenkommunikation erfolgen über ein Kabel. [4]

**CC-Link** stellt eine Entwicklung des Unternehmens „Mitsubishi“ dar. Verwaltet wird das Protokoll von einer Anwenderorganisation ähnlich zu Profibus. Die maximale Übertragungsrate beträgt 10 MBit/s und als Übertragungsmedium dient eine dreipolare Leitung. [4]

### 2.1.2 Ethernet basierende Feldbusse

Ethernet basierende Feldbusse, häufig auch als „Industrial Ethernet“ bezeichnet, ermöglichen deutlich höhere Übertragungsraten als Standard-Feldbusse. Daher nimmt deren Verbreitung ständig zu und sie lösen in vielen Bereichen den Standard-Feldbus ab. [4]

#### Ethernet-Technologie

Ethernet ist einer von mehreren Standards für die lokale Netzwerkverbindung mittels der LAN-Technologie (Local Area Network). Sie ist im IEEE 802.3- Standard (Institute of Electrical and Electronics Engineers) genormt. Der Aufbau eines Ethernet-Telegramms ist nachfolgend abgebildet (vgl. Abbildung 2.3). Die Adressierung erfolgt mittels sogenannter MAC-Adresse oder IP-Adresse (mehr hierzu in Kapitel 2.2). [8]

Verschiedene Geräte sind an das Übertragungsmedium angeschlossen und teilen sich dieses. Man spricht von einem Multi-Master-Bus, da jedes Gerät selbstständig die Initiative ergreifen kann, um Nachrichten zu senden. Damit jedes Gerät kommunizieren kann und dabei Sendekonflikte vermieden werden kommt bei Ethernet der CSMA/CD-Mechanismus zum Einsatz. CSMA/CD steht für „Carrier Sense Multiple Access with Collision Detection“ und ermöglicht einen mehrfachen Zugriff (Multiple Access) auf das Übertragungsmedium. [4] [8]

## 2 Grundlagen und Stand der Technik

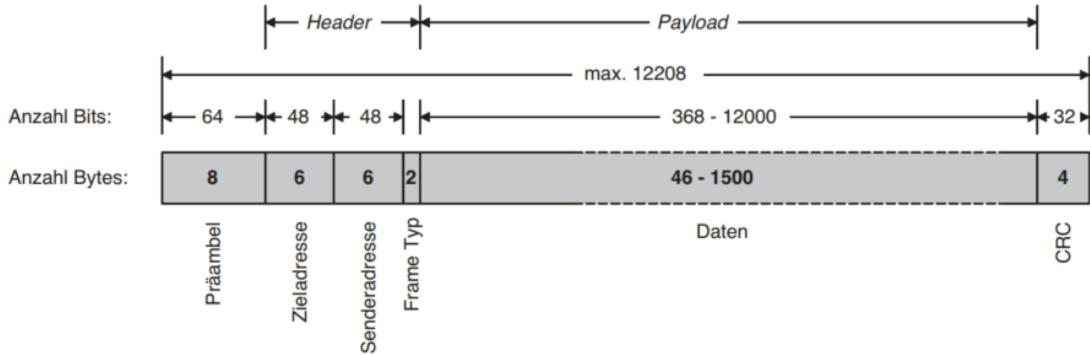


Abbildung 2.3: Telegramm-Aufbau Ethernet [4]

- **Carrier Sense:** Das 超ertragungsmedium wird 超berwacht, damit es nicht durch ein anderes Ger t belegt ist. Erst nachdem das Medium als frei erkannt wird, erfolgt nach einer kurzen Wartezeit die 超ertragung. W rend des Sendens, wird das Medium weiterhin auf Kollisionen abgeh rt.
- **Multiple Access:** Es besagt, dass alle Ger te gleichberechtigt auf das 超ertragungsmedium zugreifen k nnen.
- **Collision Detection:** Wenn mehrere Ger te zur gleichen Zeit das 超ertragungsmedium als frei erkennen kommt es zur 超lagerung, da mehrere Ger te gleichzeitig beginnen zu senden. Dies wird als Kollision bezeichnet und es kommt zur Signalverf lschung durch 超lagerung. Da jeder Sender das 超ertragungsmedium w hrend des Sendevorgangs 超berwacht wird die Kollision direkt erkannt. Dasjenige Ger t, welches die Kollision erkennt, informiert alle anderen Ger te mittels eines Signales 超ber die aufgetretene Kollision und fordert diese zur Unterbrechung jeglicher 超ertragung auf.  
Danach warten die Ger te eine zuf llige Zeitspanne ab und versuchen das Senden erneut. [8]

Durch die Kollision kommt es zu nicht vorhersehbaren Wartezeiten im Sendeprozess. Daher handelt es sich um ein nicht deterministisches Zugriffsverfahren. Ethernet mit CSMA/CD ist nicht echtzeitf hig, was wie bereits in Kapitel 2.1.1 erw hnt, zu Problem bei Automatisierungsanlagen f hren kann.

Um die Ethernet-Technologie dennoch in der Automatisierung zu nutzen, haben die Hersteller verschiedene Echtzeitprotokolle entwickelt, um Ethernet echtzeitf hig zu machen. Beispielsweise wird die Methodik CSMA/CD aufer Kraft gesetzt und stattdessen durch sogenanntes „Pooling“ oder ein Zeitscheibenverfahren ersetzt. Jeder Hersteller geht hier jedoch seinen eigenen Weg. Bekannte Ethernet basierende Feldbusse sind „Profinet“, „Ethernet/IP“, „EtherCAT“ und „Sercos III“. EtherCAT ist eine Technologie des Herstellers „Beckhoff“, dessen SPS im Rahmen dieses Projektes zum Einsatz kommt. Aus

## 2 Grundlagen und Stand der Technik

diesem Grund ist EtherCAT die bevorzugte Technologie und wird im Rahmen dieser Arbeit näher erläutert. [9] [8]

Weiterhin ist zu beachten, dass die Netzwerktopologie sich bei den Ethernet basierenden Feldbussen häufig von den Standard-Feldbussen unterscheidet. Während bei den Standard-Feldbussen die Ansteuerung vorwiegend in der Linienstruktur erfolgt, wird bei der Ethernet Anschaltung oft eine Sternstruktur mit einem Switch eingesetzt (vgl. Abbildung 2.4).

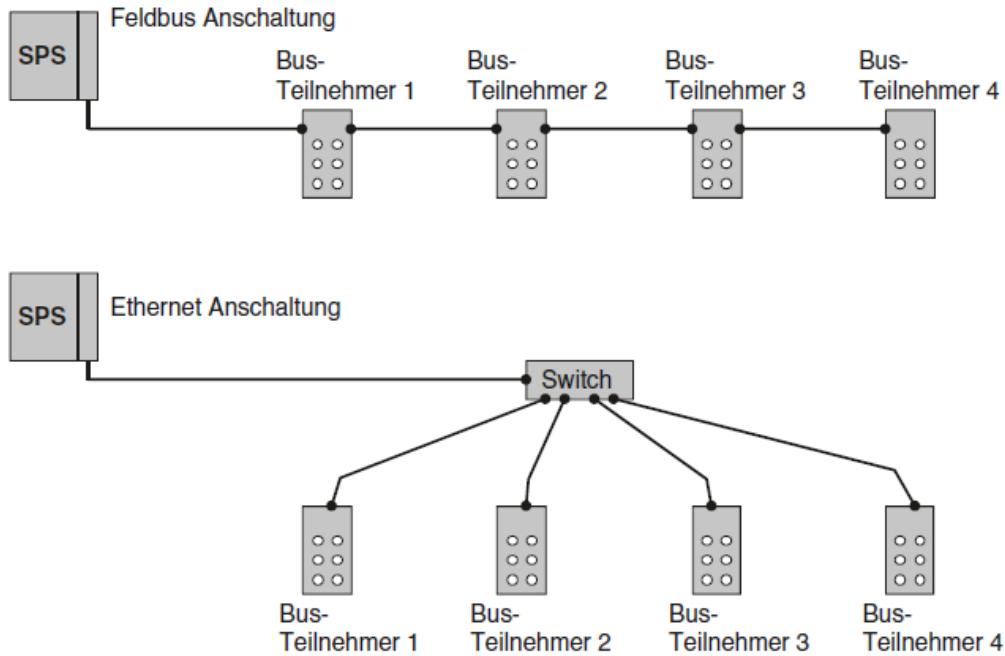


Abbildung 2.4: Topologie: oben Standard-Feldbus, unten Ethernet basierender Feldbus [4]

Der Ethernet Switching Hub, abgekürzt als Switch erlernt beim Einschalten an welchen Ports Teilnehmer angeschlossen sind. Der Switch leitet ein Datentelegramm eindeutig vom Sender zum Empfänger. Dadurch werden von dem Telegramm nicht betroffene Teilnehmer und Ethernet-Segmente nicht unnötig belastet und Kollisionen vermieden. Das Weiterschalten erfolgt auf Basis der Sende- und Ziel-Adresse, welche im Ethernet-Telegramm hinterlegt ist. [4]

### EtherCAT

Wie bereits erwähnt setzen viele Hersteller auf das Zeitscheibenverfahren oder Pooling, um Ethernet echtzeitfähig zu machen. Der Zeitverzug ist hierbei jedoch implementierungsabhängig und kann durch erforderliche technische Komponenten am Bus weiter steigen. Um dies zu verhindern setzt EtherCAT auf einen anderen Lösungsansatz. Im Gegensatz zu anderen Verfahren sollen Daten nicht mehr empfangen, interpretiert und

## 2 Grundlagen und Stand der Technik

dann wieder weiterversendet werden. Stattdessen setzt EtherCAT auf eine sogenannte FMMU (Fieldbus Memory Management Unit) in jeder E/A-Klemmen. Diese FMMU entnimmt nur die relevanten Daten vom Bus, sodass die Telegramme mit einer Verzögerung von nur wenigen Nanosekunden weiterlaufen können. Zum Versenden werden Daten in die durchlaufenden Telegramme eingefügt, sodass es auch hier zu kaum einer Verzögerung kommt. Dieses Prinzip ist nachfolgend dargestellt (vgl. Abbildung 2.5).

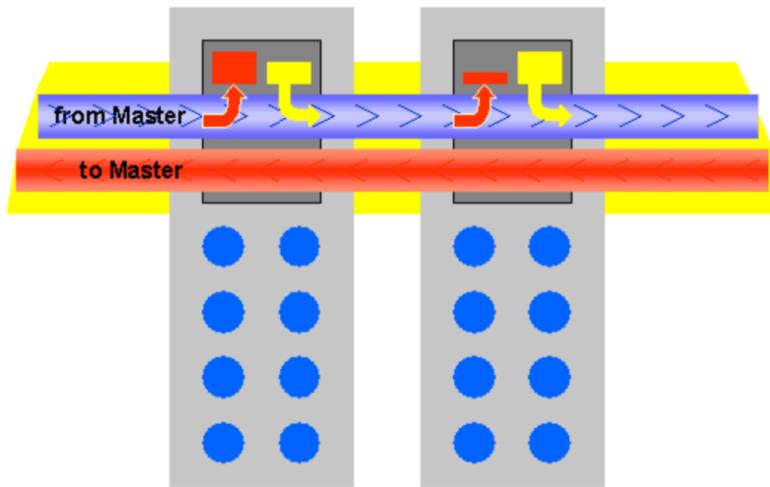


Abbildung 2.5: Telegrammbearbeitung [9]

Dabei wird auf dem gesamten Bus vom Ethernet-Protokoll nach IEE 802.3 nicht abgewichen. Bit-Fehler werden so durch die Prüfsumme erkannt. Neben der Stern-Topologie unterstützt EtherCAT auch eine Linien- oder Baum-Struktur und viele weitere. EtherCAT zeichnet sich durch geringe Laufzeitverzögerungen und eine hohe Datenrate aus. Beispielsweise können 1000 E/As in nur  $30 \mu\text{s}$  aktualisiert werden und auch eine Übertragungsrate im GBit-Bereich ist mit Erweiterung möglich. [9]

## 2.2 TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) ist das meist verwendete Netzwerkprotokoll weltweit und zudem frei zugänglich. Durch dieses Protokoll wird definiert, wie Daten durch Netzwerkkommunikationshardware versendet und empfangen werden können. Welches Übertragungsmedium verwendet wird ist nicht definiert, sodass sich neben Local Area Network (LAN) z.B. auch WLAN einsetzen lässt. [10] [11] TCP/IP ist nicht echtzeitfähig, stellt jedoch eine gute Ergänzung zu den Echtzeitprotokollen dar, um nicht zeitkritische Daten, wie z.B. zur Diagnose oder Visualisierung zu übertragen. Bei TCP/IP handelt es sich um eine sichere Datenübertragung, die Nachrichten

## 2 Grundlagen und Stand der Technik

ten in einen Bytestrom verpackt und entpackt. Vom Sender zum Empfänger durchlaufen die Daten vier Schichten (vgl. Abbildung 2.6). Für eine sichere fehlerfreie Übertragung werden die eigentlichen Daten ergänzt um die MAC-/IP- und TCP-Header. Das CRC-Feld stellt eine Art Prüfsumme, mit welcher die Korrektheit der übertragenen Daten überprüft wird. [4]

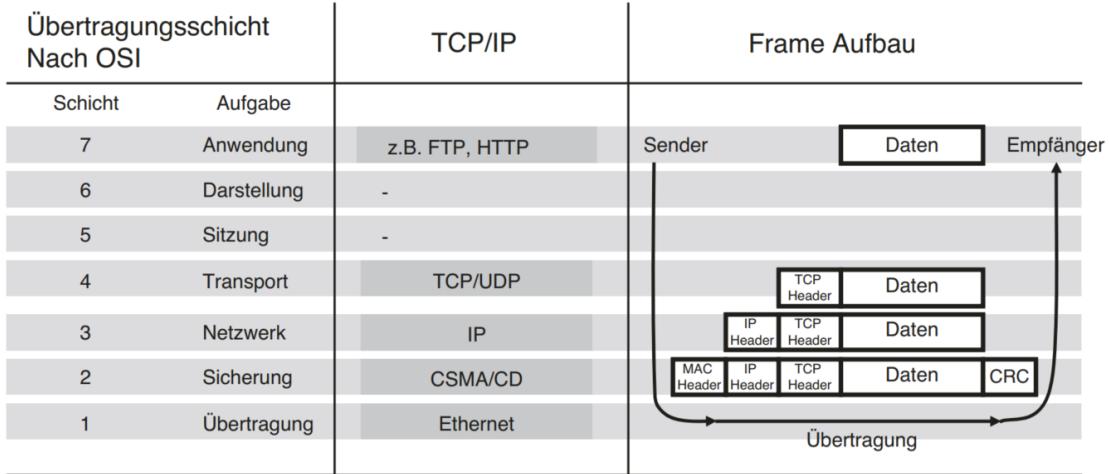


Abbildung 2.6: Schichtenmodell TCP/IP [4]

TCP/IP setzt sich aus den Protokollen TCP und IP zusammen.

### TCP

Das Ziel von TCP (Transmission Control Protocol) ist eine fehlerfreie Datenübertragung. Hierzu muss der Empfänger die korrekt erhaltenen Daten, über eine Nachricht, die an den Sender zugesendet wird, bestätigen. Die Überprüfung der Korrektheit erfolgt mit dem CRC-Segment. Erhält dieser Sender diese Bestätigung nicht wird erneut versucht die Daten zu versenden. Hierdurch ist eine fehlerfreie und lückenlose Datenübertragung, selbst bei Netzwerkproblemen garantiert, was jedoch die Prozesse verlangsamt. Eine Alternative zu TCP ist UDP (User Datagram Protocol) . Hierbei erhält der Absender keine Bestätigung, dass die Daten korrekt empfangen wurde. Der Sender fährt direkt mit der Versendung der nächsten Pakete durch. Eine fehlerfreie Übertragung ist nicht garantiert, jedoch ist sie im Vergleich zu TCP schneller. Sowohl TCP, als auch UDP bauen auf dem Internet Protocol (IP) auf. [10]

### IP

Das IP-Protokoll arbeitet auf der Internet bzw. IP-Schicht des TCP/IP-Protokolls, was der Netzwerkschicht des OSI-Modells entspricht. Die Aufgabe des IP-Protokolls ist, es Datenpakete vom Sender zum Empfänger zu übertragen. Eine Datenüberprüfung und Fehlerkorrektur erfolgt nicht, sodass Daten verloren gehen können oder fehlerbehaftet

## 2 Grundlagen und Stand der Technik

sind. Die Garantie für die korrekte Datenlieferung geschieht durch das TCP-Protokoll. Das IP-Protokoll verwendet IP-Adressen, um Netzwerknoten zu identifizieren. Mit Hilfe derer Adressen wird die Nachricht durch das Netz „geroutet“. Es wird ein idealer Weg zwischen Sender und Empfänger gesucht. [12] [8] Es gibt zwei Versionen des IP-Protokolls IPv4 und IPv6. Bei IPv4 besteht die IP-Adresse aus 32 Bit, bei IPv6 aus 128 Bit, was die Anzahl der eindeutigen Adressen erhöht. Aufgrund der zunehmenden Geräteanzahl wird in der Zukunft IPv6 der Standard werden. Heute dominiert jedoch IPv4. [8]

Eine MAC-Adresse (Medium Access Control) ermöglicht eine eindeutige Identifikation eines Gerätes im Netzwerk. Da allerdings meist herstellerübergreifende Netzwerke eingesetzt werden, ist die IP-Adresse für eine eindeutige Identifikation besser geeignet. Diese kann entweder manuell oder automatisch zugewiesen werden. Der Aufbau einer IP-Adresse nach IPv4 ist nachfolgend dargestellt (vgl. Abbildung 2.7). [4]

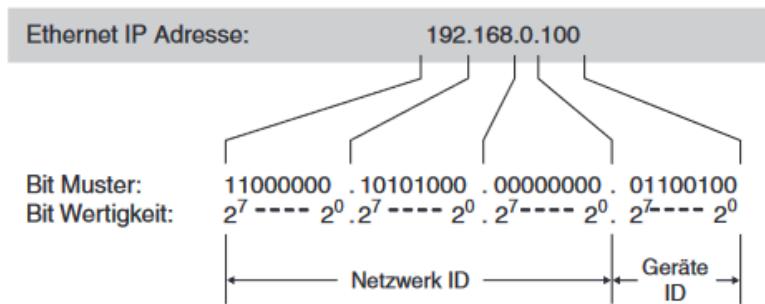


Abbildung 2.7: IP-Adresse Aufbau [4]

### Netzwerkarchitektur

Während Netzwerke in ihrer Topologie, wie z.B. Bus, oder Stern-Topologie unterschieden werden können, kann auch eine Aufteilung nach Architekturentyp erfolgen. Neben der monolithischen Architektur und der Client-Server-Architektur gibt es Cloud-, Edge- und Fog-Computing. In der monolithischen Architektur existiert nur ein zentraler Rechner. An diesen werden externe Geräte angebunden, die selbst keine Rechenleistung aufweisen. Bei Cloud-, Edge- und Fog-Computing wird ein Teil der Netzwerktechnik an externe Organisationen ausgelagert. Eine externe Organisation kann ein Provider sein, der ein Server bereitstellt.

Im nachfolgenden soll die Client-Server-Architektur näher erläutert werden, da diese von Relevanz für diese Arbeit ist. Bei der Client-Server-Architektur stellt der Client Anfragen an den Server, welche dieser beantwortet. Dieser Ablauf ist als Sequenzdiagramm in Abbildung 2.8 dargestellt.

Charakteristisch für den Server ist dessen vergleichsweise hohe Rechenleistung, was

## 2 Grundlagen und Stand der Technik

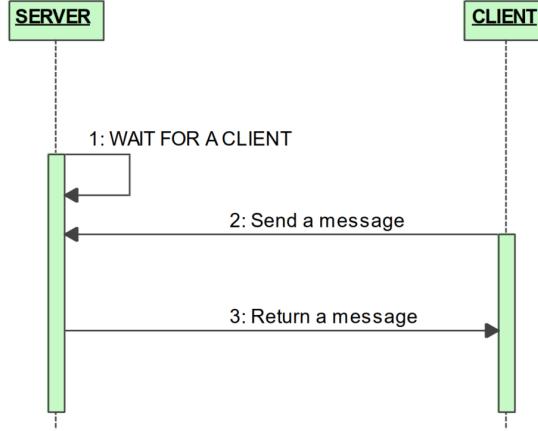


Abbildung 2.8: Sequenzdiagramm Server-Client-Architektur [10]

es ermöglicht, dass mehrere Clients zeitgleich auf den Server zugreifen können. Veranschaulichen lässt sich dieses Prinzip mit dem Webbrowser, wie beispielsweise Google Chrome auf dem privaten Rechner. Dieser stellt den Client dar, welcher Anfragen an einen Webserver stellt, um aktuelle Daten passend zu seiner Anfrage zu erhalten. Auf dem Webserver wird beispielsweise ein Online-Shop verwaltet und mehrere Clients können zeitgleich auf diesen zugreifen. Das oben erwähnte Cloud-Computing basiert auf der Client-Server-Architektur, wurde jedoch nutzerfreundlicher gestaltet. [13]

### 2.3 OPC/UA

OPC UA (Open Platform Communication Unified Architecture) ist ein Protokoll zur Maschine-zu-Maschine-Kommunikation, aber auch zur Kommunikation zwischen SPS oder Benutzerschnittstelle und Maschine. Es handelt sich hierbei um einen offenen und plattformunabhängigen Standard, welcher von der „Open Platform Foundation“ verwaltet wird. OPC UA basiert analog zu TCP/IP auf Ethernet und Internet. Es ist ebenso nicht echtzeitfähig. Das Kommunikationsprotokolls OPC UA ermöglicht eine vertikale und horizontale Kommunikation, da es die einzelnen Automatisierungsebenen verbindet. Somit kann von einem Roboter bis hin zur gesamten Fabriksteuerung alles verbunden werden, was OPC UA beliebt macht, um die Fabrik „Industrie 4.0“ fähig zu machen. [14] OPC UA basiert auf der Server-Client Architektur. Der OPC UA Server kann dann beispielsweise auf der SPS oder der Benutzerschnittstelle laufen und ein Roboter kann als OPC UA Client angebunden werden. Die Daten werden standardisiert und maschinenlesbar semantisch beschrieben, sodass eine herstellerübergreifende Kommunikation problemlos möglich ist. Sensordaten einer Maschine können dadurch an die Anlagensteuerung übertragen und visualisiert werden. OPC UA liefert dabei viele Vorteile: [15]

## 2 Grundlagen und Stand der Technik

- **Sicher:** IT-Sicherheit wurde bei der Entwicklung von OPC UA direkt mit eingeplant. Die Kommunikation und der Transport ist sicher und kann über Zugriffsrechte gesteuert werden.
- **Plattformunabhängig:** Neben der Herstellerunabhängigkeit ist OPC UA unabhängig von der Programmiersprache, was es ermöglicht verschiedene Anlagen einfach zu verknüpfen.
- **Einfach:** OPC UA ist auf Nutzerfreundlichkeit ausgelegt und das Festlegen von Kommunikationsprotokollen und der Aufbau von Nachrichten entfällt, sodass auch ohne Programmierkenntnisse eine Umsetzung möglich ist.
- **Big Data:** OPC UA ermöglicht einfaches Sammeln von Daten aus diversen Maschinen und von unterschiedlichsten Automatisierungsebenen. Hierdurch können sich umfangreiche Potentiale durch die Analyse der Daten ergeben und neue Geschäftsmodelle erschlossen werden.

Aufgrund dieser unterschiedlichsten Vorteile, wird OPC UA immer mehr im Bereich Industrie 4.0 eingesetzt. [15] [14]

## 2.4 Stäubli-Roboter

Stäubli ist ein in der Schweiz ansässiges Industrieunternehmen, welches in den Bereichen „Elektrische Verbindungen“, „Fluidische Verbindungen“, „Textil“ und „Robotik“ tätig ist. Das Produktpotfolio im Bereich Robotik umfasst Vier- und Sechsachs-Roboter, sowie kollaborative und mobile Roboter. [16]

Stäubli-Roboter sind nach IP 65 und IP 67 Staub- und Wasser- geschützt, wenn ein Arm mit Überdruckeinheit zum Einsatz kommt. Dies ermöglicht eine gründliche Reinigung aus Hygienezwecken und verhindert das Ansammeln von Staub im Roboter. Beispielsweise ist auch eine Reinigung mit Desinfektionsmittel oder Isopropylalkohol möglich. Zudem können die Roboter mit einem Lebensmittelverträglichem Öl eingesetzt werden. Stäubli-Roboter eignen sich daher insbesondere für Anwendungen in der Lebensmittel-, Pharma- und Medizin-Technik, in der viele andere Roboter die Hygieneanforderungen nicht erfüllen können. [17]

### Roboter TX2 - 90

Im Rahmen dieser Arbeit kommt der Sechsachs-Roboter TX2-90 zum Einsatz (vgl. Abbildung 2.9). Dieser weist eine Tragkraft von 12 kg und eine Reichweite von 1000 mm auf. Die Wiederholgenauigkeit beträgt 0,03 mm und die maximale kartesische Geschwindigkeit 10,9 m/s. Der Roboter wiegt 114 kg und wird von der Robotersteuerung CS9 mit 2kVA gesteuert. Der Roboter ist für verschiedene, teils auch aggressive Reinigungsmittel geeignet, was für die Lebensmittelindustrie wichtig ist. Die Achsen können maximale Drehmomente zwischen 11 Nm (Achse 6) und 318 Nm (Achse 1) aufbringen.

## 2 Grundlagen und Stand der Technik

Der Vorderarm des Roboters ist mit vier Magnetventilausgängen, von zwei 5/2-Wege-Magnetventilen versehen. Dies ermöglicht beispielsweise das Öffnen und Schließen eines Greifers.

Für den Roboter liegt ein Wartungsplan vor, welcher Wartungsarbeiten in unterschiedlichen Intervallen von monatlich bis 5-jährig vorsieht. Diese reichen von einfachen Sichtprüfungen und Bremstests über Ölwechsel und Dichtungstausch bis hin zum Auswechseln von Achsen. [18]



Abbildung 2.9: Stäubli TX2 - 90 **Staubli\_HE**

### 2.4.1 Controller CS9

Der CS9 Controller übernimmt die Steuerung und Versorgung des TX2 - 90 (vgl. Abbildung 2.10). Der Computer-Einschub (1) beinhaltet Karten, für die Sicherheitssteuerung, Bewegungssteuerung und die externe Kommunikation. Der Verstärker-Einschub (2) wandelt mit Hilfe von digitalen Achsverstärkern Bewegungsvorgaben in Motorströme um. Er beinhaltet zudem die Schnittstelle, an dem der Roboter angeschlossen wird. Der Stromversorgungs-Einschub (3) wandelt die Netzversorgungsspannung für die zuvor genannten Einschübe um und versorgt diese. Zudem existiert ein Anschluss für eine Antistatikmanschette (4), sowie Befestigungsmöglichkeiten (5). Der Computer-Einschub (vgl. Abbildung 2.11) besitzt eine Vielzahl an Kommunikationsschnittstellen, die in der Tabelle nachfolgend dargestellt sind (vgl. Tabelle 2.1).

Tabelle 2.1: CS9 Controller Computer-Einschub

Nr.	Bezeichnung
1	Feldbus, Option für PCIe-Karte
2	Anschluss für optionale externe 24 V Stromversorgung

## 2 Grundlagen und Stand der Technik

Tabelle 2.1: CS9 Controller Computer-Einschub

Nr.	Bezeichnung
3	Schnelle Ein-/Ausgänge
4	EtherCAT Port (Reserviert)
5	Anschluss Handbediengerät
6	USB-Ports
7	Echtzeit-Ethernet-Slave (EtherCAT, Sercos III, EtherNet/IP, PROFINET)
8	EtherCAT-Master
9	Ethernet-Ports (1000 MBits/s und 100 MBits/s)
10	Status LEDs
11	serielle Verbindung RS232
12	Benutzerschnittstelle

An den CS9 Controller kann ein Handbediengerät angeschlossen werden (5), was z.B. manuelles Verfahren oder einfachere Programmier- und Konfigurationsarbeiten ermöglicht. Hervorzuheben ist zudem der Echtzeit-Ethernet-Slave-Anschluss (7), durch welchen die Steuerung z.B. über EtherCAT mit einer SPS verbunden werden kann. Zusätzlich steht ein EtherCAT-Master-Anschluss (8) zur Verfügung. Die CS9 ermöglicht es somit den Roboter als EtherCAT-Slave oder als EtherCAT-Master zu betreiben.

Zudem stehen zwei Ethernet-Ports (9) zur Verfügung, was eine externe Kommunikation z.B. über TCP/IP ermöglicht. Die IP-Adresse der Ethernet-Ports kann frei konfiguriert oder automatisch vergeben werden. Als Dateiübertragungs-Serverprotokoll dient "FT-PS+FTP". FTP wird als Standardeinstellung verwendet, während FTPS eine sichere Authentifizierung ermöglicht.

Auf der CS9-Steuerung lassen sich sogenannte Sockets konfigurieren, welche eine TCP/IP oder eine UDP/IP - Kommunikation ermöglichen. Die Konfiguration kann dabei im Client oder Server-Modus erfolgen. [19] Ebenso kann die Steuerung für den Einsatz von OPC UA konfiguriert werden, hierbei kommen jedoch einmalige Lizenzkosten von ca. 500 € pro Roboter hinzu.

### 2.4.2 Programmiersprache VAL 3

Die Programmierung des Roboters kann auf dem Handbediengerät erfolgen oder bevorzugt in der Stäubli Robotic Suite (SRS). In der SRS kann neben der Programmierung eine Simulation des Roboters inklusive 3D-Darstellung erfolgen, sodass die Programme direkt getestet werden können.

„VAL 3“ ist eine höhere Programmiersprache und besitzt spezielle Steuerungsfunktionen für den Roboter zur Bewegungsteuerung und zur Ansteuerung von Ein-/Ausgängen. [20]

#### Programm

In einer Applikation können mehrere Programme angelegt werden. Diese verhalten

## 2 Grundlagen und Stand der Technik



Abbildung 2.10: Stäubli CS9 Controller [19]

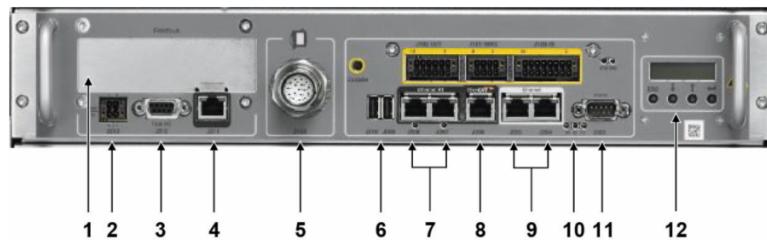


Abbildung 2.11: CS9 Controller Computer-Einschub [19]

sich wie Funktionen und lassen sich aufrufen. Zur Ausführung der VAL3-Applikation wird standardmäßig das Programm „start()“ gestartet, welches vergleichbar mit einer „main-Funktion“ ist. Von hier lassen sich wiederum andere Programme aufrufen. Bei Beendigung der Applikation wird das Programm „stop()“ einmalig aufgerufen, um z.B. von dort andere Programme zu beenden.

Programme lassen sich auch als Task ausführen. Bei einer Task handelt es sich um ein Programm welches zu einem gewissen Zeitpunkt ausgeführt wird. Üblicherweise wird beispielsweise die Bewegungssteuerung und die Kommunikation in eigene Tasks ausgelagert. Es wird ein möglichst gleichzeitiges Ausführen der Tasks angestrebt. Da die Steuerung nur einen Prozess besitzt, ist es jedoch nur möglich eine Task auszuführen. Die augenscheinliche Parallelität, wird dadurch erreicht, dass die Tasks sequentiell sehr schnell hintereinander ausgeführt werden (vgl. Abbildung 2.12). Dabei wird nicht die gesamte Task abgearbeitet, sondern einige wenige Anweisungen. Eine Task erhält zudem eine Priorität, die angibt, wie viele Programmzeilen ausgeführt werden bis die nächste Task abgearbeitet wird. Eine hohe Priorität sorgt daher für ein schnellstmögliches Abarbeiten dieser Task. Wird in einer Task eine Wartezeit („delay“) ausgeführt, pausiert diese Task und der Prozessor arbeitet nur die anderen Tasks ab. Neben den erwähnten

## 2 Grundlagen und Stand der Technik

asynchronen Tasks bietet VAL 3 ebenso synchrone Tasks.

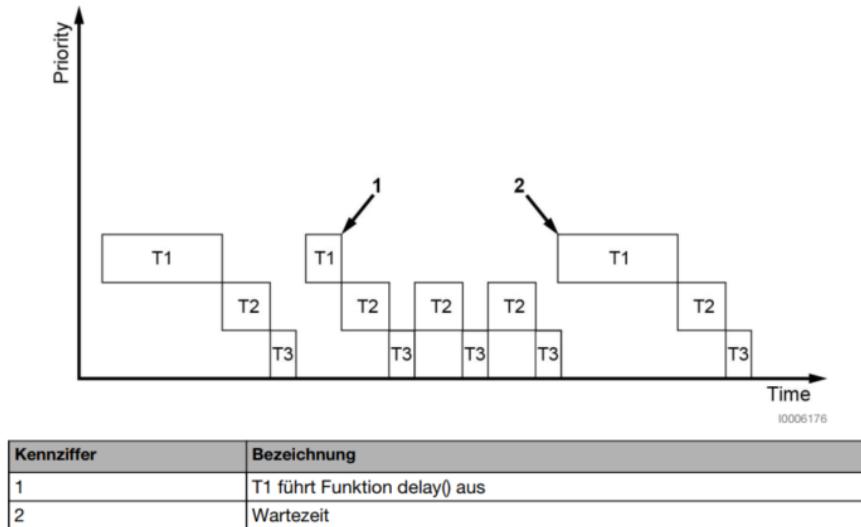


Abbildung 2.12: Sequentielle Anordnung von Tasks [20]

### Datentypen

In VAL 3 gibt es verschiedene einfache Datentypen, die nachfolgend kurz erläutert werden. Neben den einfachen Datentypen gibt es weitere Datentypen, die z.B. Punkte für die Bewegungssteuerung darstellen. Auf eine Erläuterung dieser wird verzichtet, da diese in der Kommunikation nicht benötigt werden.

- **BOOL:** Variablen und Konstanten von diesem Typ können „true“ und „false“ annehmen.
- **NUM:** Datentyp zur Darstellung von numerischen Werten mit 14 signifikanten Stellen. Im Gegensatz zu vielen gängigen Programmiersprachen wird in VAL 3 nicht zwischen verschiedenen Zahlenformaten unterschieden. Dieser Datentyp kann von ganzzahligen Werten bis hin zu Kommazählern alles abbilden.
- **BITFELD:** Dieser Datentyp ermöglicht es eine Bitfolge z.B. von booleschen Werten oder digitalen Ein- und Ausgängen kompakt abzuspeichern.
- **STRING:** In Variablen und Konstanten diesen Typs können Zeichenketten mit einer Länge von bis zu 256 Byte gespeichert werden. Die interne Zeichencodierung erfolgt mit Unicode UTF 8. Bei ASCII-Zeichen entspricht die Länge von 256 Bytes somit 234 Zeichen. Initialisiert werden Variablen dieses Datentyps standardmäßig mit einem leeren String ("").

## 2 Grundlagen und Stand der Technik

- **DIO:** Dieser Datentyp ermöglicht es aus dem Programm auf die digitalen Ein-/Ausgänge der Steuerung zuzugreifen. Hierzu speichert die DIO-Variable einen Link zu der entsprechenden Hardware in Form einer physischen Adresse.
- **AIO:** Mit Hilfe dieses Datentyps können die analogen Ein- und Ausgänge der Steuerung verknüpft werden.
- **SIO:** Dieser Datentyp ermöglicht es einen seriellen Port oder einen Ethernet Socket Anschluss zu verknüpfen. Dadurch kann beispielsweise eine TCP/IP - Nachricht geschrieben oder gelesen werden, in dem der Variable des Datentyps ein Wert zugewiesen wird oder ein Wert abgelesen wird.

### Robotersteuerung

Neben Funktionalitäten die klassische Programmiersprachen aufweisen, benötigt VAL 3 insbesondere Möglichkeiten, um den Roboterarm zu steuern. Exemplarisch werden einige Befehle kurz vorgestellt.

- **disable / enable Power:** Hiermit lässt sich die Armleistung ab bzw. einschalten. Für den lokalen, manuellen oder Testbetrieb hat diese Funktion keinen Einfluss.
- **getVersion:** Mit dieser Anweisung lässt sich die Version der Hardware- und Software - Komponenten des Roboterscontrollers abfragen.
- **movej:** Dieser Befehl führt eine Bewegung zu einem Punkt aus. Angegeben werden muss die Winkelposition der Achsen für den Zielpunkt (vom Datentyp „joint“), die Geometrie des Werkzeuges (Datentyp „tool“) und Bewegungsdaten, wie Geschwindigkeit und Beschleunigung (Datentyp „msdec“).
- **movel:** Dieser Befehl ermöglicht im Gegensatz zum Vorangegangen eine lineare Bewegung. Anstelle der Winkelpositionen als Ziel wird eine kartesische Zielkoordinate (Datentyp „point“) vorgegeben.

## 2.5 Anlagensteuerung

Ein Roboter ist meist nur ein Teil einer Automatisierungslösung. Dieser kann um Förderbänder, Sensoren und andere Aktoren ergänzt werden. Wie die Steuerung dieser Gesamtanlage umgesetzt wird unterscheidet sich von Anlagenautomatisierer zu Anlagenautomatisierer. Hier soll die bei „robomotion“ gängige Umsetzung aufgezeigt werden, da diese relevant für das Gesamtverständnis und die Konzeptionierung der Komponenten ist.

### SPS

Zentrales Element der Anlagenautomatisierung ist eine Beckhoff-SPS, welche auf einem

## 2 Grundlagen und Stand der Technik

Indutrie-PC (IPC) läuft. Diese ist für die Ablauflogik und „Intelligenz“ der Anlage zuständig. Auf anderen Bestandteilen, wie z.B. der Robotersteuerung ist hingegen nur eine geringe Komplexität abgebildet. In der Robotersteuerung werden meistens Kommandos mit Jobnummern angelegt, wie beispielsweise die Bewegung von Punkt A zu Punkt B. Durch den Aufruf der Jobnummer durch die SPS erhält der Roboter den Befehl zur Bewegungsausführung und meldet eine erfolgreich durchgeführte Bewegung der SPS zurück. Welche Bewegungen in welcher Reihenfolge und wann durchgeführt werden, ist dementsprechend nicht in der Robotersteuerung, sondern in der SPS programmiert.

### **roboTools**

Weiterer Bestandteil der Automatisierung ist ein Programm namens „roboTools“, welches zur Anlagenvisualisierung dient. Dieses läuft auf einem Windows-Betriebssystem in Form einer WPF-Anwendung (Windows Presentation Foundation), des „.NET-Frameworks“. Das Programm roboTools mit seiner grafischen Oberfläche enthält verschiedene Diagnosewerkzeuge und kann dem Anlagenbediener Informationen über bspw. Störungen oder der Anlagenauslastung ausgeben und visualisieren. Die Daten hierfür bezieht roboTools überwiegend von der SPS.

### **roboVision**

Häufig werden in Anlagen Industriekameras eingesetzt, um die Position von Objekten zu ermitteln oder eine automatisierte Qualitätskontrolle vorzunehmen. Zur Weiterverarbeitung der Kameradaten, um die Position oder die Qualitätsgüte zu bestimmen kommt das Programm „roboVision“ zum Einsatz. Wie roboTools handelt es sich um ein WPF-Anwendung, welche auf einem Windows-Betriebssystem läuft.

### **Hardware und Zusammenspiel**

Sowohl roboTools, als auch roboVision laufen auf einem IPC. Dies kann der selbe IPC sein, auf dem bereits die SPS läuft. Insbesondere für roboVision wird jedoch meist ein eigenständiger leistungsfähiger IPC eingesetzt, da für die Bildverarbeitung eine hohe Rechenleistung zur Verfügung stehen muss. Für die Umsetzung der Kommunikationschnittstelle, spielt dies jedoch keine große Rolle, da sich alle IPCs in einem Netzwerk befinden. Das Zusammenspiel dieser Komponenten ist nachfolgend dargestellt (vgl. Abbildung 2.13). Die Kommunikation zwischen Beckhoff-SPS und einer Robotersteuerung erfolgt bei robomotion mit dem Feldbus EtherCAT. Die Kommunikation zwischen der SPS, roboTools und roboVision über ein spezielles Kommunikationsprotokoll namens ADS (Automation Device Specification) von Beckhoff. ADS ermöglicht den Datenaustausch von verschiedenen Systemen über die Netzwerkverbindung, die bereits zwischen der SPS, roboTools und roboVision besteht. [21]

## 2.6 WPF-Anwendung

Da bereits roboTools und roboVision auf WPF beruhen, ist WPF von großer Relevanz für diese Arbeit. WPF (Windows Presentation Foundation) stellt den Nachfolger von

## 2 Grundlagen und Stand der Technik

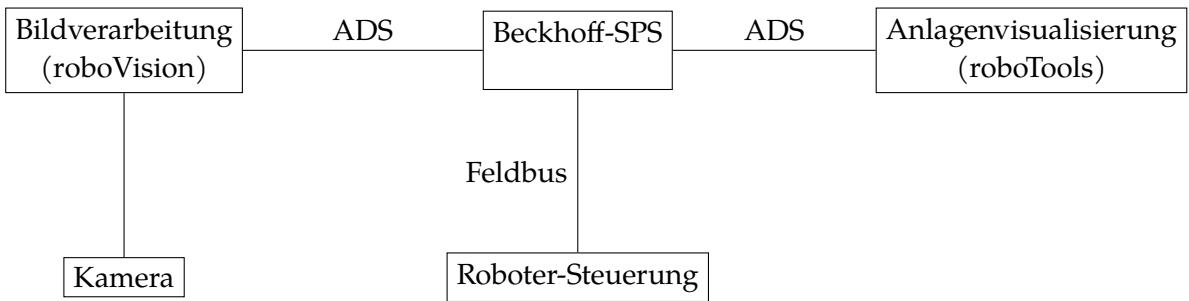


Abbildung 2.13: Anlagensteuerung Aufbau

Windows Forms dar. Es handelt sich um eine Art Framework und ermöglicht optisch anspruchsvolle Windows-Desktop-Anwendungen zu entwerfen. Die Oberflächenbeschreibung für eine WPF-Anwendung erfolgt mit der Beschreibungssprache XAML (eXtensible Application Markup Language) (vgl. nachfolgender Codeabschnitt). [22]

```
// XAML - Definition für einen Button
<Button Content="Test - Button" Height="74" Width="154" />
```

WPF basiert auf dem .NET - Framework.

### .NET - Framework

Das .NET Framework ist eine Plattform für unterschiedliche Programmiersprachen, wie C# oder Visual Basic mit denen verschiedene Applikationstypen, wie Desktop, Web oder Mobile entwickelt werden können.

Das .NET-Framework ist eine Entwicklung von Microsoft. Die meisten Anwendungen können jedoch plattformübergreifend eingesetzt werden. Lediglich Desktopanwendungen, wie z.B. WPF sind an Windows gebunden. [22] [23]

### C#

Das .NET-Framework ist offen für verschiedenste Programmiersprachen, jedoch ist die Verwendung von C# am geläufigsten und kommt in den vorhandenen Anwendungen von robomotion zum Einsatz. C# ist eine Entwicklung speziell für .NET und kombiniert Konzepte und Bestandteile von Java, JavaScript, C++ und Visual Basic. Dabei handelt es sich um eine vollwertige objektorientierte Sprache mit Eigenschaften, wie Vererbung und Kapselung. [22] [23]

### Programmierumgebung

Als Entwicklungsumgebung wird für WPF-Anwendungen vornehmlich Visual Studio eingesetzt. Die integrierte Entwicklungsumgebung Visual Studio wird von Microsoft angeboten und ist daher optimal auf die Entwicklung einer WPF-Anwendung zugeschnitten. Zum Projektzeitpunkt stellt Visual Studio 2022 die aktuellste Version dar. Die „Community Edition“ ist eine kostenfreie Version für private Projekte, sowie kommerzielle Projekte in Unternehmen mit bis zu 250 Mitarbeitern. [22] Als Source-Code-Verwaltungs-Programm wird unternehmensintern „Azure DevOps“ eingesetzt. Daher wird auch im

## *2 Grundlagen und Stand der Technik*

Rahmen dieser Arbeit auf Visual Studio 2022 in Kombination mit Azure DevOps zurückgegriffen.

### **2.7 Fazit**

Feldbusse ermöglichen eine echtzeitfähige Kommunikation, was für viele Funktionen von großer Bedeutung ist. Die klassischen Feldbusse werden immer mehr von Ethernet basierenden Feldbussen abgelöst. Unternehmensintern kommt neben der SPS der Firma Beckhoff der Ethernet basierende Feldbus EtherCAT zum Einsatz, welcher ebenso eine Entwicklung von Beckhoff darstellt.

Das TCP/IP- Protokoll stellt eine weitere Möglichkeit zur Datenübertragung dar, ist jedoch nicht echtzeitfähig. Das TCP-Protokoll stellt die korrekte Datenübertragung sicher, weshalb es in der Literatur häufig für Industrieanwendungen, die nicht echtzeitfähig sein müssen zum Einsatz kommt. In [24] kommt anstelle des TCP-Protokoll das Protokoll UDP zur Übertragung von Live-Kamera-Bildern zum Einsatz, da hier die Schnelligkeit über die Korrektheit zu stellen ist. Teilweise wird in der Literatur z.B. in [24] VPN zur IT-Sicherheit bei TCP/IP eingesetzt. OPC/UA stellt eine weitere Möglichkeit zur Datenübertragung dar. Sie überzeugt mit Einfachheit und Flexibilität, bringt aber Mehrkosten mit ohne in diesem konkreten Anwendungsfall mehr als TCP/IP leisten zu können. Zusammengefasst lässt sich sagen, dass TCP/IP für den Anwendungsfall des Übertragens von Fehlermeldungen und Prozessdaten eine gute Lösung darstellt, da hier keine Echtzeitfähigkeit gefordert ist. UDP hingegen soll im Rahmen dieser Arbeit nicht zum Einsatz kommen, da die Korrektheit der Daten über die Schnelligkeit zu stellen ist. Auf den Einsatz von VPN kann verzichtet werden, da die TCP/IP-Kommunikation ausschließlich zwischen Roboter und Anlagenvisualisierung erfolgt und keine Verbindung zum Internet aufweist.

Weiterer Teil des Projektes ist der Stäubli-Roboter TX2-90 mit dem Controller CS9. Dieser soll in der Programmiersprache VAL 3 in der Entwicklungsumgebung Stäubli Robotic Suite programmiert werden. Die Gesamtanlage von SPS und Roboter wird ergänzt um ein Visualisierungsprogramm namens roboTools, welches eine Benutzeroberfläche für den Bediener darstellt. roboTools ist eine WPF-Anwendung, die in C# programmiert ist. Daher soll die zu entwerfende Anwendung, welche die Diagnose und Prozessdaten des Stäubli-Roboters darstellt, ebenso als WPF-Anwendung entwickelt werden. Dies macht eine spätere Integration in roboTools einfach möglich.

### 3 Anforderungsdefinition

Die Aufgabenstellung und die grobe Anforderungsformulierung soll in diesem Kapitel mit Hilfe eines Lastenheftes in eine strukturierte Form gebracht werden. Das Lastenheft beschreibt, welche Funktionalitäten das zu entwerfende System aufweisen soll. Der Auftragnehmer beschreibt im Pflichtenheft durch welche Systemfunktionalitäten die Aufgaben erfüllt werden sollen und wie die Umsetzung technisch erfolgt. Das Lasten- und Pflichtenheft dient auch zur rechtlichen Absicherung der Vertragsparteien. Obwohl ein Lasten- und Pflichtenheft oft als Vertragsgrundlage entworfen wird und dies im Rahmen des Projektes nicht notwendig ist, soll ein Lastenheft eingesetzt werden. Dies ermöglicht es alle Anforderungen und Rahmenbedingungen zu berücksichtigen und diese im gesamten Projektverlauf im Blick zu behalten. Auf ein Pflichtenheft wird hingegen verzichtet. [1] Das Lastenheft wird in Muss- und Kann-Kriterien unterteilt. Während bei den Muss-Kriterien eine Erfüllung essentiell ist, stellen die Kann-Kriterien eine optionale Erweiterung dar, die dem System jedoch einen Mehrwert verschaffen.

Tabelle 3.1: Lastenheft

Nr.	Anforderung	Kann/Muss
1	Übertragung von Fehlermeldungen des Stäubli-Roboters an die Anlagenvisualisierung	Muss
2	Übertragung von Prozessdaten des Stäubli-Roboters an die Anlagenvisualisierung	Muss
3	Übertragung von Daten z.B. Kameradaten von der Anlage an den Stäubli-Roboter	Muss
4	Ausgabe der Roboterfehler auf der Anlagenvisualisierung in verständlicher Form (ausformuliert, Fehlercode genügt nicht)	Muss
5	Funktionierende Feldbusverbindung zwischen Stäubli-Roboter und SPS	Muss
6	Funktionierende TCP/IP Kommunikation zwischen Stäubli-Roboter und Anlagenvisualisierung	Muss
7	Abspeichern der Fehler und der Daten in log-File	Muss
8	Fehlerfreie und prozesssichere Kommunikation	Muss
9	Einsatz von mehreren Stäubli-Robotern zeitgleich	Kann
10	Vorgabe von der Anlagenvisualisierung, wie häufig Daten versendet werden sollen	Kann

### 3 Anforderungsdefinition

Tabelle 3.1: Lastenheft

Nr.	Anforderung	Kann/Muss
11	Visualisierung der Daten	Kann
12	Verwertung der Daten z.B. für Warnungen	Kann
13	Übersichtliche Darstellung aller Daten und Meldungen	Kann
14	Melden, dass Bahnpunkte umgeteacht wurden	Kann

Im Rahmen dieser Arbeit ist die Feldbus-Verbindung zwischen SPS und Roboter zu entwerfen. Der Fokus liegt jedoch auf der TCP/IP-Verbindung zwischen Roboter und Anlagenvisualisierung. Darüber hinaus ist das Programm für den Roboter zu konzeptionieren und zu implementieren.

Das andere Ende der TCP/IP-Verbindung stellt die Anlagenvisualisierung dar. Dort sollen die Fehlermeldungen ausgegeben und die Daten visualisiert werden. Als Anlagenvisualisierung dient bei robomotion roboTools. Die zu entwerfende WPF-Anwendung zur TCP/IP-Kommunikation und zur Datenvisualisierung kann nach Projektende in roboTools integriert werden. Dies soll jedoch nicht mehr Teil der Arbeit sein. Die zu entwerfende WPF-Anwendung wird im Rahmen dieser Arbeit vereinfacht als Anlagenvisualisierung bezeichnet.

Die bestehende Anwendung roboVision erstellt auf Basis von Kamerabildern Positionsdaten von zugreifenden Objekten. Diese Positionsdaten können über die Beckhoff-spezifische Verbindung ADS zu roboTools und somit auch zur entwerfenden WPF-Anwendung übertragen werden. Daher kann in dieser Arbeit angenommen werden, dass die Positionsdaten in der Anlagenvisualisierung vorhanden sind und von dort direkt versendet werden können.

Eine Konzeptionierung und Implementierung des SPS-Programms kann entfallen, da die Kommunikation mit einer Feldbus-Verbindung in der Beckhoff-SPS bei robomotion Stand der Technik ist und dadurch aus anderen Projekten übernommen werden kann. Das Zusammenspiel all dieser Komponenten ist nachfolgend dargestellt, wobei die im Rahmen dieser Arbeit zu entwerfenden Komponenten farbig markiert sind (vgl. Abbildung 3.1).

### 3 Anforderungsdefinition

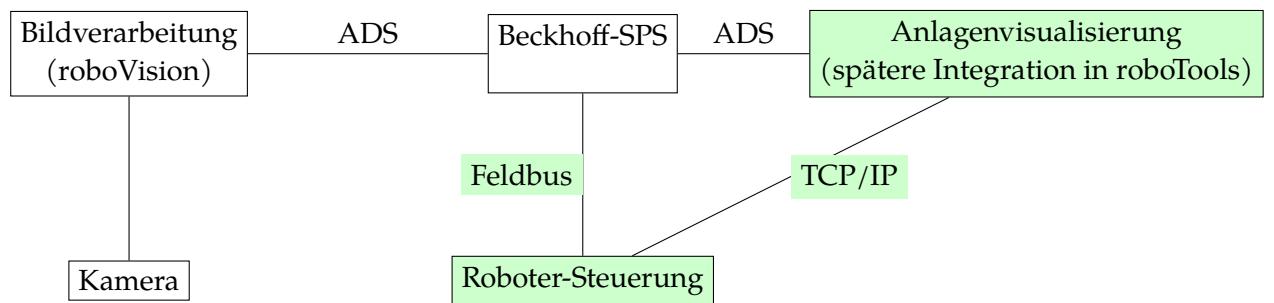


Abbildung 3.1: Gesamtanlage zu entwerfende Komponenten

# 4 Konzeptionierung

In diesem Kapitel sollen die Kommunikationsverbindungen, sowie das Roboterprogramm und die Anlagenvisualisierung konzeptioniert werden. Hierfür werden die Komponenten der Anlage benannt und ihre Funktionen grob skizziert. Dies erfolgt mit Hilfe eines UML-Klassendiagramms (vgl. Abbildung 4.1). UML (Unified Modeling Language) ermöglicht eine Systemmodellierung mit Hilfe von grafischer Notation [3]. Das UML-Diagramm zeigt die Komponenten SPS, Roboter und Anlagenvisualisierung auf. Die Kommunikation zwischen SPS und Roboter erfolgt mittels Feldbus, zwischen Anlagenvisualisierung und Roboter mittels TCP/IP. Daher soll im ersten Schritt der Feldbus und die TCP/IP-Verbindung konzeptioniert werden. Erst anschließend kann das Programm des Roboters und der Anlagenvisualisierung konzeptioniert werden, da dieses maßgeblich abhängig von der Wahl und dem Aufbau der Kommunikationsverbindung ist. Nachdem die Feldbus-Verbindung und die TCP/IP-Verbindung konzeptioniert wurden sollen die zu übertragenden Daten für die TCP/IP-Verbindung konzeptioniert werden. Anschließend erfolgt die Konzeptionierung der Software für den Stäubli-Roboter und die Anlagenvisualisierung.

## 4.1 Feldbus-Verbindung

Aufgabe der Feldbusverbindung ist es, die Kommunikation zwischen dem Stäubli-Roboter und der Beckhoff-SPS sicherzustellen. Hierzu werden einerseits Befehle, wie zur Bewegungsdurchführung zum Roboter übertragen, andererseits Informationen, wie z.B. eine Bestätigung der vollendeten Bewegung zurück an die SPS gesandt.

### Wahl des Feldbusses

In Kapitel 2.1 wurden die Feldbusse in Standard-Feldbusse und Ethernet basierende Feldbusse unterteilt. Die Wahl fällt hierbei auf einen Ethernet-basierenden Feldbus, weil dies der aktuelle Stand der Technik darstellt und zudem firmenintern ausnahmslos eingesetzt wird. Nach Kapitel 2.1.2 sind Profinet, Ethernet/IP, EtherCAT und Sercos III häufig eingesetzte Ethernet basierende Feldbusse. Eine Gegenüberstellung der Vor- und Nachteile der einzelnen Feldbusse und eine Auswahl anhand dieser Kriterien kann jedoch entfallen, da zwei Argumente für den Einsatz von EtherCAT, alle anderen Kriterien überwiegen.

Im Projekt kommt eine Beckhoff-SPS zum Einsatz. EtherCAT ist eine Entwicklung von Beckhoff und daher bestmöglichst auf die SPS von Beckhoff abgestimmt.

#### 4 Konzeptionierung

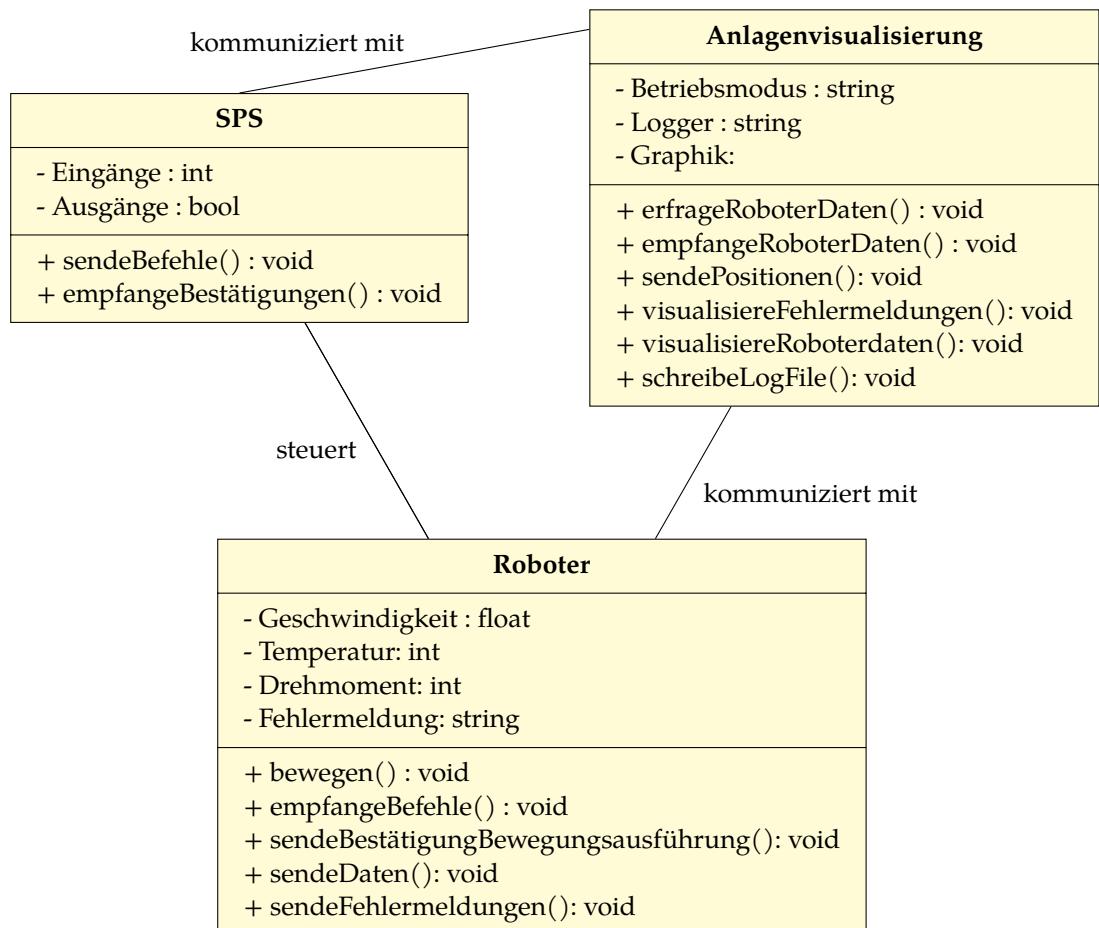


Abbildung 4.1: UML-Klassendiagramm Gesamte Anlage

## *4 Konzeptionierung*

Darüber hinaus, kommt unternehmensintern bereits in anderen Projekten EtherCAT zum Einsatz. Die Wahl von EtherCAT bringt daher folgende Vorteile mit sich

- **Know How:** Firmenintern hat sich durch den wiederholten Einsatz von EtherCAT bereits eine große Wissenbasis und Erfahrung aufgebaut. Bei schwerwiegenderen Problemen ermöglicht dies eine einfachere Fehlersuche.
- **Schulungszwecke:** Durch andere Projekte sind beispielsweise die Service-Techniker bereits in EtherCAT geschult. Die Buchung einer zeit- und kostenintensiven Schulung für einen anderen Feldbus kann entfallen.
- **Standardisierung:** Der Einsatz von vielen unterschiedlichen Systemen innerhalb des Unternehmens erhöht den Aufwand in Dokumentation und Einkauf. Daher wird versucht die Varianz möglichst gering zu halten und auf einen unternehmensinternen Standard, welcher bereits häufig verwendet wird zu setzen.

### **Master-Master vs. Master-Slave**

Standardmäßig wird im Auslieferungszustand sowohl die Beckhoff-SPS, als auch der Stäubli Roboter als Master eingesetzt. Jedoch ist EtherCAT rein für Master-Slave-Kommunikation ausgelegt, d.h. es kann nur ein Master existieren. [9] Beckhoff bietet jedoch eine spezielle Bridge-Klemme (EL6692) an, die es erlaubt EtherCAT-Stränge zu synchronisieren und ein Datenaustausch zwischen den einzelnen Strängen zu ermöglichen. Da jeder EtherCAT-Strang einen eigenen Master besitzt ermöglicht die Bridge den Einsatz von mehreren Mاستern. [25] Hierdurch entstehen jedoch zusätzliche Kosten und auch die Komplexität des Systems steigt. Da sowohl die Stäubli-Steuerung, als auch die SPS ebenso als Slave konfiguriert werden kann, fällt die Entscheidung auf die Master-Slave-Kommunikation, um die zusätzliche Master-Master-Klemme einzusparen.

### **Wahl des Masters**

Nachdem die Entscheidung für eine Master-Slave-Architektur gefallen ist, muss festgelegt werden, ob die Beckhoff-SPS oder der Stäubli-Roboter als Master konfiguriert wird. Da der Einsatz von mehreren Stäubli-Robotern in Automatisierungslösungen denkbar ist, könnte nicht jeder Stäubli-Roboter als Master eingesetzt werden. Folglich wäre ein Stäubli-Roboter der Master, die anderen Stäubli-Roboter, sowie die SPS Slaves. Um diese Unterschiede in den einzelnen Stäubli-Robotern zu verhindern, ist es sinnvoll die Beckhoff-SPS als Master zu konfigurieren, sodass alle Stäubli-Roboter einheitlich als Slaves eingesetzt werden. Darüber hinaus ist die SPS der zentrale Teil der Anlagenautomatisierung. Von ihr können weitere Komponenten, wie Roboter anderer Hersteller gesteuert werden, weshalb die Wahl der SPS als Master sinnvoll ist.

Die Gesamtentscheidung zur Konfiguration ist nachfolgend mit Hilfe eines morphologischen Kastens dargestellt.

## 4 Konzeptionierung

Tabelle 4.1: Morphologischer Kasten Feldbus

Parameters	Konfiguration			
	Option 1	Option 2	Option 3	Option 4
Bussystem	etherCAT	Profinet	Sercos III	Ethernet/IP
Architektur	Master-Master	Master-Slave		
Master	SPS	Stäubli		

## 4.2 TCP/IP-Verbindung

Nachdem die Feldbus-Verbindung konzeptioniert wurde, erfolgt die Auslegung der TCP/IP-Verbindung. Hierbei müssen viele Einzelentscheidungen getroffen werden, welche am Ende mit einem morphologischen Kasten zusammengefasst dargestellt werden.

### Wahl der Architektur

Wie in Kapitel 2.2 erläutert kommt für eine TCP/IP-Verbindung die Client-Server-Architektur zum Einsatz. Dabei kann entweder der Stäubli-Roboter oder die Anlagenvisualisierung als Server dienen. In der Literatur wird meist empfohlen, die Seite mit der höheren Rechenleistung als Server einzusetzen (vgl. Kapitel 2.2). Dies stellt in diesem Fall die Anlagenvisualisierung dar. Wenn hingegen der Roboter als Server dienen würde und es zum Einsatz von mehreren Robotern kommen würde, wäre der Einsatz von mehreren Client-Instanzen auf der Anlagenvisualisierung notwendig oder die anderen Roboter müssten als Client konfiguriert werden. Daher eignet sich die Anlagenvisualisierung besser als Server und die einzelnen Roboter werden als Client konfiguriert.

### Nachrichtentypen

Für folgende Daten, die übertragen werden müssen, muss festgelegt werden, ob und wie diese auf einzelne Nachrichten aufgeteilt werden.

- **Systemereignisse:** Erfasste Meldungen, wie z.B. Fehler müssen von dem Roboter zu der Anlagenvisualisierung übertragen werden.
- **Prozessdaten:** Vom Roboter erfasste Daten, wie Temperaturen oder digitale Werte, wie Fehler an den Ventilen müssen vom Roboter zur Anlagenvisualisierung übertragen werden.
- **Kameradaten:** Positionsdaten von zugreifenden Objekten, welche durch Kameradaten generiert werden, müssen von der Anlagenvisualisierung zum Roboter übertragen werden.

Da die Kameradaten in die entgegengesetzte Richtung, wie die anderen Daten übertragen werden, werden diese in eine eigene Nachricht verpackt. Die Systemereignisse und die Prozessdaten können in einer Nachricht oder in getrennten Nachrichten versandt werden.

## *4 Konzeptionierung*

Da die Prozessdaten durchgängig anfallen, die Systemereignisse jedoch azyklisch lohnt sich eine Trennung.

Die Prozessdaten wiederum können gebündelt oder getrennt in einzelne Nachrichten z.B. für die Temperaturen und die digitalen Werte, versendet werden. Da beispielsweise Temperaturen nicht besonders dynamisch sind könnte die Übertragungsfrequenz für diese Daten geringer sein, was die Auslastung der Verbindung reduzieren würde. Dies erhöht jedoch die Komplexität, da viele verschiedene Nachrichtentypen anfallen, was bei Wartungen oder Änderungen in späteren Jahren zum Problem werden kann. Da das zyklische Versenden aller Prozessdaten, die Übertragungsrate der TCP/IP-Verbindung nicht übersteigen wird, fällt die Entscheidung auf das Zusammenfassen aller Prozessdaten in einer Nachricht. Folglich gibt es drei Nachrichtentypen: Kameradaten, Systemereignisse und Prozessdaten.

### **Auslöser für Versand der Systemereignisse**

Die Systemereignisse können entweder zyklisch z.B. einmal pro Sekunde zusammengefasst, oder direkt bei Auftreten versendet werden. Die Entscheidung fällt auf ein direktes Versenden bei Auftreten, damit wichtige Fehlermeldungen schnellstmöglich ankommen. Zudem handelt es sich um ein azyklisches Ereignis, d.h. die Systemereignisse werden nur sporadisch versendet.

**Auslöser für Versand der Prozessdaten** Die Prozessdaten werden zyklisch benötigt und sind nicht abhängig von besonderen Ereignissen. Ein zyklischer Versand bietet sich hier an. Die Sendefrequenz muss dann in jedem Stäubli-Roboter einprogrammiert werden und ist dann fest eingestellt. Vorteilhaft wäre es wenn die Sendefrequenz von dem Server Anlagenvisualisierung vorgegeben werden kann. Dadurch kann die Sendefrequenz angepasst und z.B. zur Fehlerfindung erhöht werden. Darüber hinaus kann die Änderung an dieser zentralen Instanz erfolgen und muss nicht manuell bei jedem einzelnen Stäubli-Roboter durchgeführt werden. Dies ist insbesondere praktisch, wenn ein Zugriff über das Fernwartungssystem auf die Anlagenvisualisierung erfolgt. Daher soll ein Datenanforderungstelegramm zyklisch von der Anlagenvisualisierung an alle Clients (Stäubli-Roboter) gesandt werden. Diese sollen nach Erhalt der Anforderung ein Prozessdatentelegramm versenden. Der Einsatz von Datenanforderungstelegrammen bringt einen weiteren Vorteil mit sich. Durch die Zeitdifferenz zwischen Datenanforderungstelegramm und Erhalt des Datentelegramms kann überprüft werden, ob die Datenübertragung ausreichend stabil ist und ob der Client noch verbunden ist. Wenn innerhalb einer vorgegebenen Zeitspanne von keine Antwort in Form des Datentelegramms erfolgt kann eine Warnung ausgegeben werden, dass dieser Stäubli-Roboter nicht erreichbar ist.

### **Auslöser für Versand von Kameradaten**

Beim Anfallen von Positionsdaten, welche anhand von Kameradaten berechnet werden, sollen diese schnellstmöglich an den Roboter übertragen werden. Aus diesem Grund erfolgt der Versand dieser Daten Ereignisgetriggert, d.h. direkt nach Erhalt der Positionsdaten wird ein Telegramm an den entsprechenden Roboter gesandt.

## 4 Konzeptionierung

Tabelle 4.2: Morphologischer Kasten TCP/IP-Verbindung

Merkmal	Option 1	Option 2	Option 3
Architekturwahl	Anlage = Server	Roboter = Server	
Nachrichtentypen	Eine Nachricht für Alles	Prozessdaten unterteilt	Prozessdaten und Systemereignisse getrennt
Auslöser Versand Systemereignisse	Zyklisch	Ereignisbasiert	
Auslöser Versand Prozessdaten	Anforderungs- telegramm	Zyklisch	
Auslöser Versand Kameradaten	Zyklisch	Ereignisbasiert	

Die Entscheidungsfindung bei den eben genannten Auswahlmöglichkeiten ist in einem morphologischen Kasten (vgl. Tabelle 4.2) dargestellt.

### Nachrichtenaufbau Datentelegramm

Nachfolgend soll der Nachrichtenaufbau des Datentelegramms festgelegt werden. Durch Brainstorming und durch Inspiration von anderen Telegram-Aufbauten wie Ethernet (vgl. Kapitel 2.1.2) werden verschiedene Möglichkeiten des Telegrammaufbaus entworfen:

- **Konzept 1:** <Nachrichten-ID>,<Sender-ID>,<Uhrzeit>,<Daten>
- **Konzept 2:** <Nachrichten-ID>,<Sender-Name>,<Daten>
- **Konzept 3:** <Eindeutige ID (GUID)>,<Sender-ID> <Daten>
- **Konzept 4:** <Nachrichten-ID>,<Sender-ID>,<Daten>,<CRC>
- **Konzept 5:** <Nachrichten-ID>,<Sender-ID>,<Daten Gleitkomma, dynamisch>

In Konzept 1, 2, 4 und 5 existiert eine Nachrichten-ID am Anfang, damit der Empfänger weiß, dass es sich z.B. um ein Datentelegramm und kein Systemereignis handelt. Anschließend folgt die Sender-ID, damit der Server weiß von welchem Roboter die Daten stammen.

Bei Konzept 2 wird die Sender-ID, welche durch eine Zahl dargestellt wird, durch einen konfigurierbaren Namen ersetzt, was die Nutzerfreundlichkeit erhöht, da bei späterer Wartung ggf. unbekannt ist, welcher Roboter welche Nummer besitzt.

Bei Konzept 1 wird darüber hinaus die Uhrzeit mit versendet, dies ist für Dokumentationszwecke sinnvoll und ebenso kann eine Zeitdifferenz zwischen Roboter und Anlagenvisualisierung detektiert werden. Das Datum kann durch die Anlagenvisualisierung bei Erhalt der Nachricht bestimmt werden und muss daher nicht versendet werden.

Bei Konzept 4 wird zusätzlich ein CRC-Segment versendet. Im CRC-Segment wird eine

#### 4 Konzeptionierung

Prüfsumme übermittelt, welche vom Sender aus der zu übermittelnden Nachricht berechnet wird. Der Empfänger berechnet die Prüfsumme aus der erhaltenen Nachricht und vergleicht sie mit dem CRC-Segment, um zu überprüfen, ob die Nachricht korrekt übermittelt wurde. TCP/IP stellt jedoch bereits selbstständig eine fehlerfreie Übertragung durch ein CRC-Segment sicher. Durch eine zusätzliche Prüfsumme kann zwar die Sicherheit bezüglich des korrekten Einlesens der Daten erhöht werden, für die Übertragung selbst ist ein CRC-Segment jedoch nicht mehr notwendig.

Konzept 3 sieht den Einsatz eines einzigartigen Codes vor. Mit jedem Datenanforderungstelegramm wird ein einzigartiger Code generiert. Der Roboter antwortet mit der Rücksendung des selben Code und den angehängten Daten. Dadurch kann eine bessere Zuordnung der Datentelegramme zu den Aufforderungen erfolgen, um sicherzustellen, dass die Datentelegramme nicht hinterherhinken.

Die fünf entworfenen Konzepte zum Nachrichtenaufbau werden nachfolgend verglichen (vgl. Tabelle 4.3). Zum Einsatz kommen Harvey-Balls, die den Erfüllungsgrad des jeweiligen Kriteriums wiedergeben. Alle Kriterien haben eine ähnliche Wichtigkeit, mit der sich abschließend eine Gesamtwertung ermitteln lässt. 4 entspricht dabei der höchsten Wertung mit einem durchschnittlich komplett ausgefüllten Harvey-Ball.

Tabelle 4.3: Vergleich Nachrichtenaufbau

	1	2	3	4	5
Informationsgehalt	●	○	○	○	●
Übertragungssicherheit	●	●	●	●	●
Verständlichkeit/	●	●	○	○	○
Einfachheit					
Längenminimiert	●	●	●	●	○
Wertung (0-4)	3	3	2,5	2,5	2

**Legende:** ●: Hervorragend ○: Teils/Teils ○: Sehr schlecht

Konzept 1 und 2 erhalten insgesamt die höchste Wertung. Aus diesem Grund entfällt die Entscheidung auf Mischung von Konzept 1 und 2. Hierdurch kann der Vorteil des Datums mit dem Vorteil, dass der Sendername deutlich einfacher einem Roboter zugeordnet werden kann als eine Sender-ID, kombiniert werden. Der Nachrichtenaufbau stellt sich wie folgt dar.

Finales Konzept: <Nachrichten-ID>,<Sender-Name>,<Uhrzeit>,<Daten>

#### Nachrichtenaufbau Systemereignisse

Der Aufbau dieser Nachricht orientiert sich am Datentelegramm. Statt den Daten wird hier der Text des Systemereignisses übertragen. Ergänzt wird dies von der Schwere des Systemereignisses. Dies ist eine Ganzzahl zwischen 0 (Info) bis 4 (kritischer Fehler).

<Nachrichten-ID>,<Sender-Name>,<Uhrzeit>,<Schwere>,<Meldung>

#### Nachrichtenaufbau Datenanforderungstelegramm

#### 4 Konzeptionierung

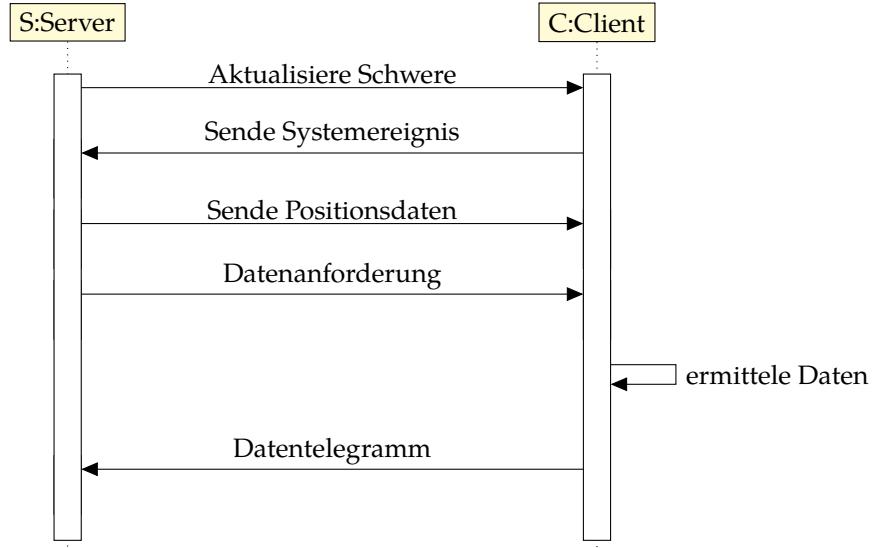


Abbildung 4.2: Sequenzdiagramm TCP/IP

Das Datenanforderungsdiagramm wird an alle Roboter versandt und ist wie folgt aufgebaut:

<Nachrichten-ID>,<Sender-Name>,<Uhrzeit>

Durch die Nachrichten-ID weiß der Roboter, dass es sich um eine Aufforderung für den Versand des Datentelegramms handelt.

Der Kommunikationsablauf lässt sich zusammenfassend in einem Sequenzdiagramm (vgl. Abbildung 4.2) darstellen.

Alle Nachrichten, die zwischen Roboter und Anlagenvisualisierung ausgetauscht werden, sind nachfolgend unter Angabe ihrer zugewiesenen Nachrichten-ID aufgezählt (vgl. Tabelle 4.4).

Tabelle 4.4: Übersicht Nachrichten TCP/IP

ID	Bezeichnung	Absender->Empfänger
1	Datenanforderungstelegramm	Anlagenvisualisierung->Roboter
2	Datentelegramm	Roboter -> Anlagenvisualisierung
3	Aktualisiere Systemereignis-Schwere	Anlagenvisualisierung->Roboter
4	Systemereignis	Roboter -> Anlagenvisualisierung
5	Positionsdaten	Anlagenvisualisierung->Roboter

### 4.3 Datenverwertung

Daten gewinnen immer mehr an Bedeutung. Durch die Verfügbarkeit von Roboterdaten ergeben sich Nutzungspotentiale, wie z.B.:

- **Überwachung und Alarm:** Bei Überschreiten von Schwellwerten oder bei Fehlermeldungen Alarm auslösen
- **Predictive Maintenance:** Vorhersagen treffen, wann Wartung erfolgen soll, um Ausfälle vorbeugend zu verhindern. Dies kann durch das Erkennen von Trends und Auffälligkeiten in den Daten erfolgen.
- **Datenarchivierung und Compliance:** Datenspeicherung für Schadensfall oder gesetzliche Gewährleistung

Der Stäubli-Roboter erfasst eine Vielzahl an Daten. Ausgewählte Daten werden nachfolgend dargestellt und auf deren Nutzen analysiert.

Tabelle 4.5: Verfügbare Daten

Daten	Zugriff	Verwendung
CPU battery test	D (CpuIO)	-
CPU overcurrent	D (CpuIO)	-
Fast memory state	D (CpuIO)	-
CPU temperature	A (CpuIO)	Alarm, PM
CPU board temperature	A (CpuIO)	Alarm, PM
CPU fan speed	A (CpuIO)	Alarm, PM
Free RAM	A (CpuIO)	-
CFast memory remaining lifetime	A (CpuIO)	-
System management CPU usage	A (CpuUsage)	-
Robot control CPU usage	A (CpuUsage)	-
Synchronous VAL 3 CPU usage	A (CpuUsage)	-
Available CPU usage for VAL 3 processing	A (CpuUsage)	-
User Fieldbuses CPU usage	A (CpuUsage)	-
HMI CPU usage	A (CpuUsage)	-
SRS connection CPU usage	A (CpuUsage)	-
OPC-UA CPU usage	A (CpuUsage)	-
CPU Load Score (higher is better)	A (CpuUsage)	-
CPU Load Score (min)	A (CpuUsage)	-
VAL 3 instructions per sequencing	A (CpuUsage)	-
VAL 3 synr inst. per cyle (high priority)	A (CpuUsage)	-
VAL 3 synr inst. per cyle (low priority)	A (CpuUsage)	-
Valve feedback (1.1, 1.2, 2.1, 2.2)	D (DsiIO)	-

#### 4 Konzeptionierung

Tabelle 4.5: Verfügbare Daten

Daten	Zugriff	Verwendung
Axis brake feedback (1-6)	D (DsIO)	-
Error on valves outputs	D (DsIO)	Alarm
Error on brakes outputs	D (DsIO)	Alarm
Error on safe digital inputs	D (DsIO)	Alarm
DSI non-reduced brake supply voltage undershoot	D (DsIO)	Alarm
DSI reduced brake supply voltage undershoot	D (DsIO)	Alarm
DSI logic supply voltage undershoot	D (DsIO)	Alarm
DSI overtemperature	D (DsIO)	Alarm
DSI board tempereature	A (DsIO)	Alarm, PM
Axis motor temperature (1-6)	A (DsIO)	Alarm, PM
Axis encoder temperature (1-6)	A (DsIO)	Alarm, PM
DSI state	A (DsIO)	-
DSI error code	A? (DsIO)	? Grau? Alarm
Arm operation counter	A (DsIO)	-
safe Input state (0-7)	D (DsIoSafe)	-
Fast Input (1-2)	D (FastIO)	-
Fast Output (1-2)	D (FastIO)	-
Power unit identification (bit 1-3)	D (PSIO)	-
Main power state	D (PSIO)	-
Internal bus voltage state	D (PSIO)	-
Power unit internal temperature state	D (PSIO)	-
24V state	D (PSIO)	-
Buckfull mode feedback	D (PSIO)	-
Memorize a power dropout	D (PSIO)	-
Brake test warning	D (Rsi9IO)	Alarm
Brake test successful	D (Rsi9IO)	-
Temperature of RSI board	A (Rsi9IO)	Alarm, PM
Error Code of RSI board	A? (Rsi9IO)	?Grau? Alarm
STARC board temperature	A (StarcIO)	Alarm, PM
Axis drive case temperature (1-6)	A (StarcIO)	Alarm, PM
Motor Winding Temperature (1-6)	A (StarcIO)	Alarm, PM
Axis drive junction temperature (1-6)	A (StarcIO)	Alarm, PM
Geschwindigkeit Endmanipulator	getSpeed(...)	-
Schleppfehler Achsen 1-6	getPositionErr()	-
Drehmoment Achsen 1-6	getJointForce(...)	-
Bewegungsauftrag und Fortschritt	getMoveld()	-

## 4 Konzeptionierung

Tabelle 4.5: Verfügbare Daten

Daten	Zugriff	Verwendung
Konfiguration des Roboters	getVersion(...)	-
Stromversorgung bei Stillstand abschalten	hibernateRobot()	-
Systemereignisse	getEvents(...)	-

Prinzipiell lassen sich alle verfügbaren Daten sammeln, visualisieren und auswerten. Bei einigen dieser Daten wie z.B. des freien RAM-Speichers ist der daraus entstehende Nutzen beschränkt. Mittels Brainstorming wurden verschiedene Anwendungsszenarien ausgedacht, im nachfolgenden sollen jedoch nur die sinnvollsten hiervon vorgestellt werden.

### Temperaturen

Neben der Temperatur von verschiedenen Computer-Chips und Platinen, wie z.B. CPU, CPU-Platine DSI-Platine, RSI-Platine und STARC-Platine sind verschiedene Temperaturwerte von den Antrieben abrufbar. Hier sind die Temperaturen der Motoren, Encoder, Antriebsgehäuse, Antriebswicklungen und Steuergeräte zu nennen. Diese Daten können sowohl zur Überwachung als auch zur Predictive Maintenance verwendet werden. Bei Überschreiten eines Grenzwertes kann ein Alarm bzw. eine Warnung ausgegeben werden. Da von Seiten des Herstellers keine zulässigen Grenzwerte vorgegeben sind, müssen die aufgezeichneten Messwerte unter Berücksichtigung von Schwankungen analysiert werden und Grenzwerte festgelegt werden, ab welchen Werten das Verhalten nicht mehr als „normal“ angesehen werden kann. Die Dynamik von Temperaturen ist im Allgemeinen relativ gering, da sich die Materialien aufgrund ihrer Wärmekapazität erst aufheizen müssen. Aus diesem Grund ist das Übertragen von den Temperaturwerten in vergleichsweise großen Abständen zulässig z.B. alle 15 Sekunden. Im Falle eines technischen Defektes wäre eine möglichst frühe Warnung wünschenswert, jedoch ist die Zeit bis ein Techniker den Alarm wahrnimmt und entsprechende Aktionen starten kann vergleichsweise groß, weshalb dieser Anwendungsfall kein höhere Übertragungsrate rechtfertigt. Da die Genauigkeit von vielen Temperatursensoren nur im Bereich von 0,5K liegt und sehr genaue Temperaturwerte keinen zusätzlichen Mehrwert bieten, ist eine Übertragung der Temperaturen ohne Nachkommastellen ausreichend. Dadurch lässt sich ein Temperaturwert problemlos mit einem einzelnen Byte (Werte zwischen 0 und 255°C) übertragen.

### Error-Meldungen

Von dem Stäubli-Roboter werden einige Fehlermeldungen erfasst, wie von den Ventil und Brems-Ausgängen, den digitalen Sicherheits-Eingängen, des DSI-Chip, RSI-Chip und Bremsentests. Ebenso werden "DSI non-reduced brake supply voltage undershoot", "DSI reduced brake supply voltage undershoot", "DSI logic supply voltage undershoot" und "DSI overtemperatureerfasst. Error-Meldungen wie diese sollen als Alarm bzw. Fehlermeldung ausgegeben werden. Ebenso ist ein Dokumentieren sinnvoll, um bei Ausfällen

## *4 Konzeptionierung*

oder Defekten die Ursachensuche zu erleichtern. Jeder Fehler kann als Wert 1 oder 0 dargestellt werden, weshalb je Fehlermeldung ein Bit genügt. Alle Error-Meldungen lassen sich somit über 2 Bytes abbilden.

### **Schleppfehler**

Der Schleppfehler entspricht der Abweichung zwischen Soll-Position und Ist-Position jeder einzelnen Achse. Ein herausstechend großer Schleppfehler kann auf Probleme mit der Steuerung, den Motoren, den Encoder oder den Achsen selbst hinweisen. Der Stäubli-Roboter überwacht den Schleppabstand bereits selbst und gibt bei einer zu großen Abweichung einen Fehler in Form eines Systemereignisses aus. Daher muss der Schleppfehler nicht zusätzlich über ein Datentelegramm übertragen werden, da er bereits über die Systemereignisse mit abgedeckt ist.

### **Drehmoment Achsen**

Das Drehmoment der einzelnen Achsen ändert sich kontinuierlich während der Bewegung. Durch das Vergleichen der Drehmomente mit einer Referenzfahrt können Auffälligkeiten entdeckt werden. Schleichend zunehmende Momente deuten z.B. auf fehlende Schmierung oder Lagerdefekte hin. Abrupt zunehmende Momente können durch ein Festhängen oder einen Crash verursacht werden. Eine Dokumentation der Abweichungen ist sinnvoll, um aus Gewährleistungsgründen unsachgemäße Verwendung nachzuweisen. Da sich die Momente hochdynamisch ändern, ist es nicht sinnvoll das aktuelle Moment zu übertragen, da minimale Zeitabweichungen zu großen Schwankungen führen können. Sinnvoller ist es, die Momente über einen kompletten Bewegungsablauf aufzuintegrieren und nur dieses Ergebnis zu übertragen.

### **Systemereignisse**

Ereignisse, die von der Steuerung festgestellt werden, wie z.B. Fehlermeldungen oder das Einschalten der Energieversorgung für die Achsen werden in einer system-log-Datei aufgezeichnet. Sie lassen sich in die Schwergrade „Info“, „Warning“, „Error“ und „Critical“ unterteilen. Das Auslesen der Systemereignisse kann mit dem Befehl „getEvents“ erfolgen. Diese sollen wie bereits bei der Konzeptionierung der TCP/IP-Verbindung festgelegt, beim Auftreten sofort über TCP/IP übertragen werden. Ein Systemereignis wird erzeugt, wenn Bahnpunkte am Roboter umgeteacht werden. Daher erfolgt automatisch eine Meldung an die Anlagenvisualisierung, wenn Änderungen an der Bewegungsbahn vorgenommen wurde. Dadurch ist diese Anforderung aus dem Lastenheft berücksichtigt.

### **Aufbau Datentelegramm**

Das Datentelegramm ist mit den ausgewählten Daten ist wie nachfolgend dargestellt (vgl. Abbildung 4.6) aufgebaut. Dadurch dass mehrere Werte aufgrund der Sechsachsen sechsfach vorkommen ist eine Anzahl von 6 Bytes notwendig. Für das Motormoment kommen je Achse 2 Bytes zum Einsatz, da die Werte zu groß für ein einzelnes Byte sind. In den 8 Bytes, gefüllt mit einzelnen binären Werten ist eine Vielzahl von Informationen, wie z.B. CPU overcurrent, Feedback der Ventile, Fehler der Bremsen oder das Ergebnis des Bremsentests gespeichert.

#### 4 Konzeptionierung

Tabelle 4.6: Aufbau Datentelegramm

Byte Inhalt	0 Nachrichten-ID	1-20 Sender-Name	21-23 Uhrzeit	24-31 Digitale Werte	32-37 Temperatur Motorspule 1-6
Byte Inhalt	38-43 Temperatur Antrieb-Antrieb Gehäuse-ID 1-6	44-49 Temperatur Antrieb Encoder 1-6	50-55 Temperatur Antrieb Verbindung 1-6	56 Temperatur Starc Board	57 Temperatur DSI Board
Byte Inhalt	58 Temperatur CPU	59 Temperatur CPU Board	60-70 Integriertes Motormoment 1-6		

## 4.4 Stäubli Roboterprogramm

Nachfolgend soll die Software und deren Komponenten für den Stäubli-Roboter konzeptioniert werden. Ein UML-Anwendungsdiagramm hilft die Anwendungsfälle zu definieren und die Software zu spezifizieren. Notwendige Bestandteile und Anwendungen ergeben sich aus der Anforderungsdefinition (vgl. Kapitel 3) und der Konzeptionierung der TCP/IP-Verbindung (vgl. Kapitel 4.2). Das Roboterprogramm erhält eine Nachricht von der Anlagenvisualisierung. Je nach Nachrichten-ID handelt es sich um eine Datenanforderungstelegramm (Nachrichten-ID: 1) zur Anforderung von Prozessdaten oder es handelt sich um die Übermittlung von Positionsdaten von der Kamera (Nachrichten-ID: 5). Alternativ kann über diese Nachricht auch die Schwere aktualisiert werden, ab welcher ein Systemereignis versandt werden soll (Nachrichten-ID: 3). Handelt es sich um ein Datenanforderungstelegramm müssen die Prozessdaten abgefragt und anschließend ein Datentelegramm versendet werden. Systemereignisse, wie z.B. Fehlermeldungen sollen an die Visualisierung übermittelt werden. Hierzu müssen diese abgefragt werden. Über die SPS kann ein Befehl zur Bewegungsausführung erfolgen. Dieser muss ausgeführt und die erfolgreiche Durchführung an die SPS zurückgemeldet werden.

Für diese Anwendungen bietet es sich an, insgesamt vier Programme anzulegen, die sich in VAL 3 wie Tasks verhalten.

- **Empfangen:** In diesem Programm wird die Nachricht von der Anlagenvisualisierung empfangen und interpretiert. Hier erfolgt ebenso das Auslesen und der Versand der Prozessdaten oder Alternativ die Verarbeitung der Positionsdaten oder das Aktualisieren der Schwere der Systemereignisse.

#### 4 Konzeptionierung

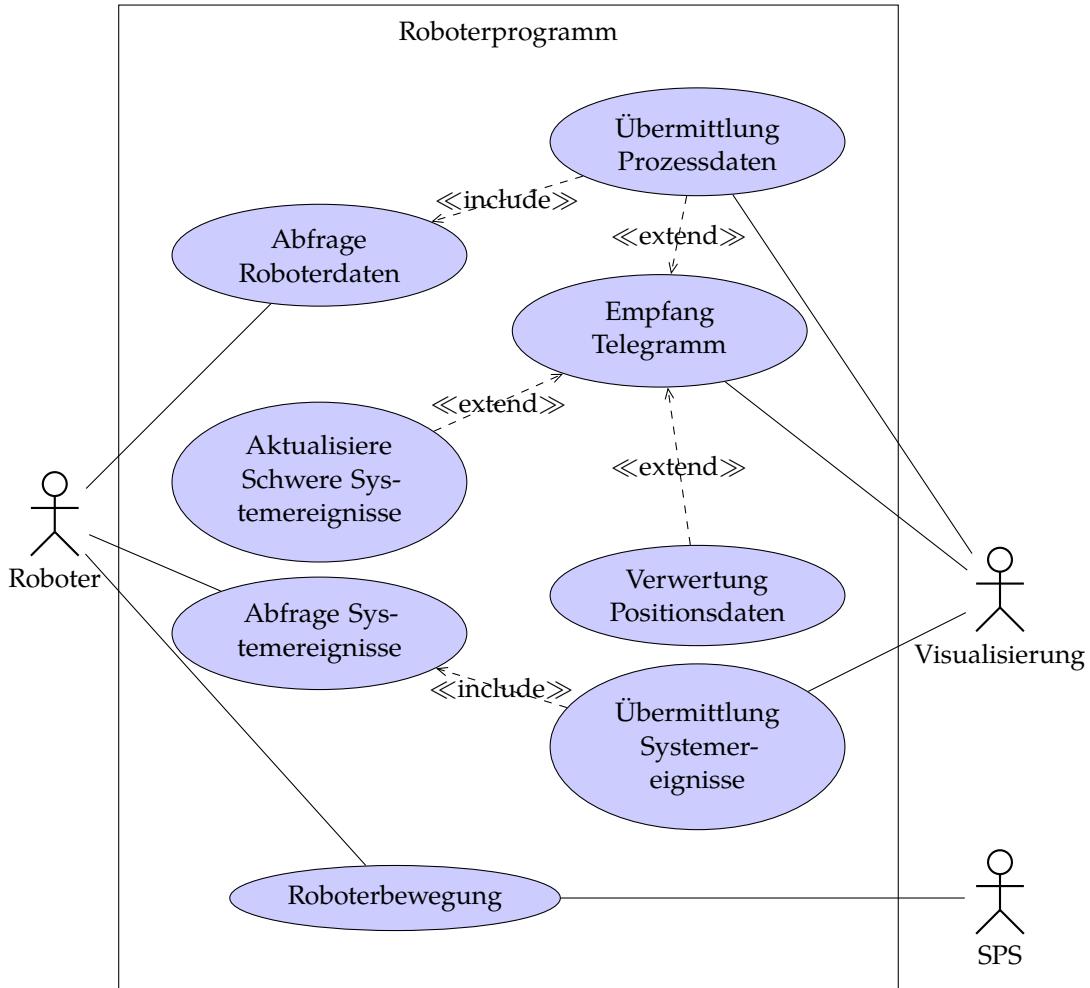


Abbildung 4.3: UML-Anwendungsdiagramm Roboterprogramm

- **Systemereignisse:** Hier erfolgt die Überprüfung ob neue Systemereignisse aufgetreten sind. In diesem Falle wird das Telegramm aufgebaut und das Systemereignis an die Anlagenvisualisierung versandt.
- **Bewegungssteuerung:** In diesem Programm erfolgt die Bewegungssteuerung des Roboters. Befehle, die von der SPS über die EtherCAT-Verbindung kommen werden ausgelesen. Bewegungsabläufe werden durchgeführt, sowie durchgeführte Bewegungen auf den EtherCAT geschrieben, um der SPS eine erfolgreiche Durchführung mitzuteilen.
- **Motormoment-Integrator:** Anstelle des aktuellen Motormomentes soll das über die gesamte Bewegung integrierte Motormoment übertragen werden. Es bietet sich an die Integration in ein externes Programm auszulagern. Nach erfolgreichem

#### *4 Konzeptionierung*

Abschluss eines kompletten Bewegungszyklus soll das Ergebnis in eine Variable gespeichert werden. Erst mit Abschluss des darauf folgenden Bewegungszyklus wird dieses aktualisiert. Übertragen wird das Moment, welches sich bei Datenabfrage in der Variable befindet.

Neben diesen vier Programmen werden in dem standardmäßigen „Start“-Programm, welches zum Programmstart ausgeführt wird, die oben genannten Tasks gestartet. Das „Stop“-Programm beendet wiederum die Tasks.

## 4.5 WPF-Anwendung

Die Anlagenvisualisierung soll wie im Kapitel Stand der Technik erläutert durch eine WPF-Anwendung umgesetzt werden. Um diese zu entwerfen muss zuerst eine Software-spezifikation erfolgen. Mit Hilfe des UML-Anwendungsdiagramms (vgl. Abbildung 4.4) sollen die Anwendungen identifiziert werden. Basis hierfür bildet die Anforderungsdefinition (vgl. Kapitel 3) und die Konzeptionierung der TCP/IP-Verbindung (vgl. Kapitel 4.2). Die WPF-Anwendung muss daher wie im UML-Anwendungsdiagramm beschrieben verschiedene Funktionalitäten umsetzen. Um Prozessdaten, wie z.B. Motortemperaturen graphisch darzustellen müssen diese durch ein Datenanforderungstelegramm von den Robotern angefordert werden. Erhält die WPF-Anwendung eine eingehende Nachricht muss diese anhand der Nachrichten ID interpretiert werden. Handelt es sich um ein Datentelegramm (Nachrichten-ID: 2) müssen die Daten strukturiert abgespeichert werden, damit eine graphische Darstellung der Daten zu einem späteren Zeitpunkt und über einen längeren Zeitraum möglich ist. Dies kann beispielsweise mit einer CSV-Datei erfolgen. Eine andere Anwendung der WPF-Anwendung ist der Empfang von Systemereignissen. Diese Nachrichten können anhand ihrer Nachrichten-ID (Nachrichten-ID: 4) erkannt werden. Die Systemereignisse, wie z.B. Fehler oder Informationsmeldungen sollen dem Nutzer dargestellt werden. Darüber hinaus sollen diese zu Archivierungs- und Dokumentationszwecken lokal in einer Datei (z.B. .txt) abgespeichert werden. Systemereignisse haben dabei unterschiedliche Ereignis-Schweren von Information bis hin zu fatalem Fehler. Die WPF-Anwendung soll steuern ab welcher Schwere ein Roboter ein Systemereignis versenden soll. Dadurch können je nach Anwendung beispielsweise die Informationsmeldungen entfallen, da diese vergleichsweise von geringer Bedeutung sind. Um die Schwere der zu sendeten Systemereignisse zu aktualisieren soll die WPF-Anwendung ein Telegramm (Nachrichten-ID = 3) mit Angabe der Soll-Schwere zu dem Roboter versenden. Darüber hinaus sollen Positionsdaten von zu greifenden Objekten an den Roboter versendet werden. Die Positionsdaten werden extern von der Anwendung „roboVision“ ermittelt und an die zu konfigurierende WPF-Anwendung weitergeleitet, die diese dann an den Roboter versenden kann (Nachrichten-ID: 5). Die zu entwerfende WPF-Anwendung soll den Programmablauf durch lokales Abspeichern (z.B. mit einer .txt-Datei) dokumentieren, wie z.B. das Versenden von Telegrammen, sowie interne Fehler oder besondere Ereignisse.

Im nächsten Schritt sollen die Softwarekomponenten, wie die Klassen der WPF-Anwendung definiert werden. Um eine effiziente und gute Struktur für die zu entwerfende Software zu erhalten wird in der Literatur auf sogenannte Muster zugegriffen. Ein Muster stellt eine bekannte und erprobte Lösungen dar [26]. Die Wiederverwendung dieser Lösungen durch Muster vermeidet Fehler und vereinfacht die Kommunikation zwischen Fachleuten. Ebenso reduzieren sich Entwicklungskosten und die Flexibilität des Entwurfes steigt [27].

#### 4 Konzeptionierung

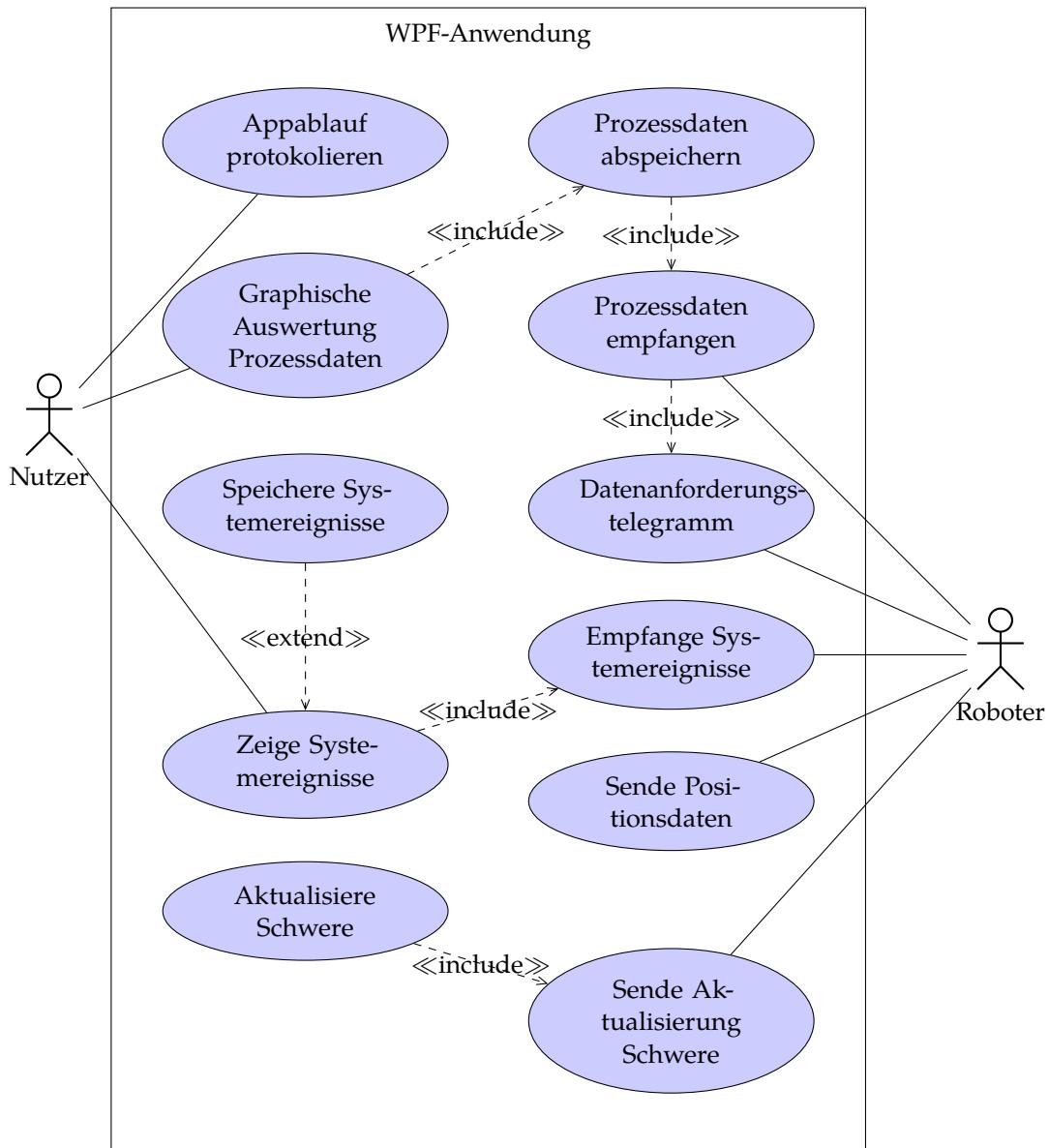


Abbildung 4.4: UML-Anwendungsdiagramm WPF-Anwendung

Nach Gollar, Koller und Watzko [27] lassen sich Muster im Entwurfsprozess in drei Gruppen einteilen:

- **Architekturnuster:** Behandelt auf hoher Ebene die Struktur und Architektur des gesamten Systems.
- **Entwurfsmuster:** Betrachtet eine Ebene unter der Architektur ein bestimmtes Problem eines Subsystems.

#### 4 Konzeptionierung

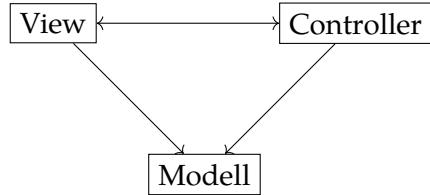


Abbildung 4.5: MVC-Muster nach [27]

- **Idiome:** Ein Idiom ist die Umsetzung eines abstrakten Musters in Form eines aus programmierten Entwurfsmustern. Hierbei handelt es sich um eine Lösung eines spezifischen Problems, weshalb Idiome hier in der Konzeptionierung der WPF-Anwendung keine Rolle spielen.

#### Architekturmuster

Es existieren eine Vielzahl von Architekturmustern, wie z.B. die Schichtenarchitektur, die Repository-Architektur, die Pipes-und-Filter-Architektur, sowie die MVC-Architektur. [3] Bei Benutzungsschnittstellen, was die zu konzeptionierende WPF-Anwendung darstellt, kommt gemäß Literatur ([3], [28], [27], [26]) übereinstimmend ein MVC-Muster zum Einsatz. MVC steht für Model-View-Controller und setzt sich aus den drei Klassen Model, View und Controller zusammen (vgl. Abbildung 4.5).

MVC ermöglicht es die Datenhaltung, die graphische Darstellung und die Interpretation der Benutzereingaben voneinander zu trennen. Durch die Entkoppelung wird die Flexibilität und die Wiederverwendbarkeit erhöht. Dadurch kann beispielsweise eine neue graphische Darstellung entwickelt werden ohne dass der restliche Code angepasst werden muss. [28] [27] Die drei Komponenten des MVC-Musters erfüllen dabei folgende Funktionen:

- **Model:** Hier werden Daten gehalten und verarbeitet.
- **View:** In der View erfolgt die graphische Darstellung für den Nutzer.
- **Controller:** Der Controller interpretiert die Eingabe des Nutzers, wie z.B. eine Nutzerfeld-Eingabe, was definierte Aktionen auslösen kann.

Das Architekturmuster MVVM ist eine Entwicklung von Microsoft und eignet sich besonders für WPF-Anwendungen. Model-View-ViewModel (MVVM) ist eine Variation von MVC, kommt dessen Prinzip jedoch sehr nahe. [29] [22] [23] Dass MVVM bereits in der Anwendung roboTools zum Einsatz kommt unterstreicht, dass MVVM das geeignete Architekturmuster für die zu entwerfende WPF-Anwendung ist. MVVM setzt sich dabei aus den Komponenten View, ViewModel und Model zusammen (vgl. Abbildung 4.6)

- **Model:** Hier werden Daten aus einer Datenquelle angefordert und bearbeitet.

## 4 Konzeptionierung

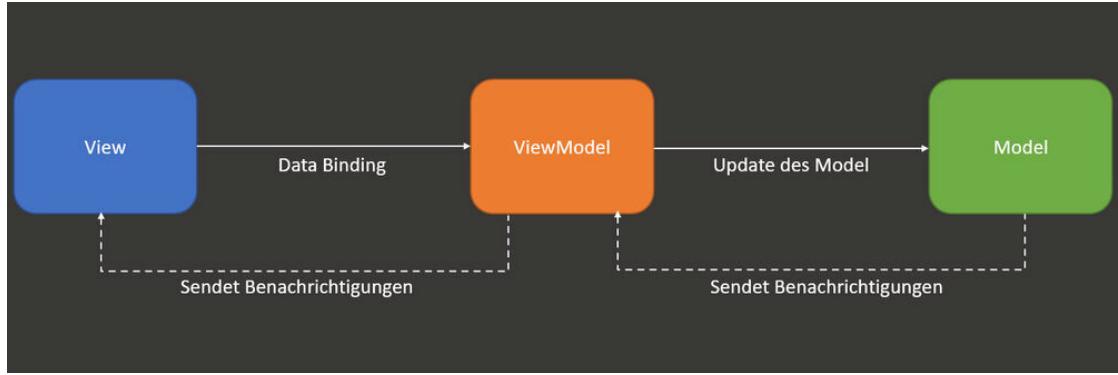


Abbildung 4.6: MVVM [29]

- **View:** In der View wird die Graphische Oberfläche definiert. Zur Darstellung von Daten nutzt sie ViewModel, welches Informationen für die View zur Verfügung stellt.
- **ViewModel:** Das ViewModel ähnelt dem Controller von MVC und stellt die Verbindung zwischen Model und View her. Hier wird die Logik, was bei Änderungen an der Benutzeroberfläche geschehen soll, implementiert. ViewModel holt sich von Model Daten, welche die View für Darstellungszwecke benötigt. [29] [23]

Auf die Verbindung zwischen den einzelnen Komponenten mit Datenbindung wird im folgenden unter Entwurfsmuster eingegangen.

### Entwurfsmuster

Nach Gollar, Koller und Watzko [27] bieten sich für die MVC-Architektur und somit auch für die MVVM-Architektur mehrere Entwurfsmuster an. Am geläufigsten ist das Beobachter-Muster (engl. Observer-Pattern). Weitere Entwurfsmuster sind das Strategie-Muster und das Kompositum-Muster [27]. Das Beobachter-Muster wird häufig in der Literatur empfohlen und kommt auch in der WPF-Anwendung roboTools umgesetzt durch sogenannte Datenbindung (engl. Data Binding) zum Einsatz. Daher fällt die Entscheidung bei der zu entwerfenden WPF-Anwendung ebenso auf MVVM mit Beobachter-Muster. Die beobachtende Komponente überwacht hierbei eine andere Komponenten. Ändert sich etwas an dem beobachtenden Objekt (wie z.B. ein Tastendruck) erkennt dies der Beobachter und kann in Folge dessen eine definierte Aktion ausführen. [27]

Die Verbindung zwischen View und ViewModel, sowie zwischen ViewModel und Model wird mit Hilfe von Datenbindung umgesetzt. Die Datenbindung gibt sozusagen Daten zwischen einzelnen Komponenten weiter.

### Adaption der Anwendung auf Architektur- und Entwurfsmuster

Um die Komponenten der zu entwerfenden Software zu definieren, müssen die Anwendungen, die im UML-Diagramm (vgl. Abbildung 4.4) definiert wurden auf das Entwurfsmuster aufgeteilt werden. Ebenso müssen die Klassen für die zu entwerfende

#### *4 Konzeptionierung*

WPF-Anwendung in C# definiert werden. Gemäß MVVM werden in der View (gemäß unternehmensinternem Standard als "MainWindow" bezeichnet) die Grafik des Programms festgelegt. Hier müssen sowohl der Protokollschreiber der Systemereignisse, sowie Diagramme zur Darstellung der Prozessdaten definiert werden. In der View soll jedoch keine Logik umgesetzt und der Code auf ein Minimum begrenzt werden. Im Model hingegen erfolgt gemäß MVVM die Datenhaltung und Datenverarbeitung. Hier werden Daten über die TCP/IP-Verbindung empfangen und gesendet, sowie die Daten abgespeichert. Das ViewModel (unternehmensintern als MainViewModel bezeichnet) verküpft die View und das Model. So soll auf Knopfdruck in der View, Prozessdaten aus dem Model ausgelesen und in der View dargestellt werden. Die Logik hierfür erfolgt in dem ViewModel (hier: MainViewModel). Für den Entwurf der WPF-Anwendung bietet sich daher folgende Klassenaufteilung an, welche nachfolgend in einem UML-Klassendiagramm dargestellt ist (vgl. Abbildung 4.7). Diese Klassen und ihre Funktionen sind in der Umsetzung mit C# zu implementieren.

#### 4 Konzeptionierung

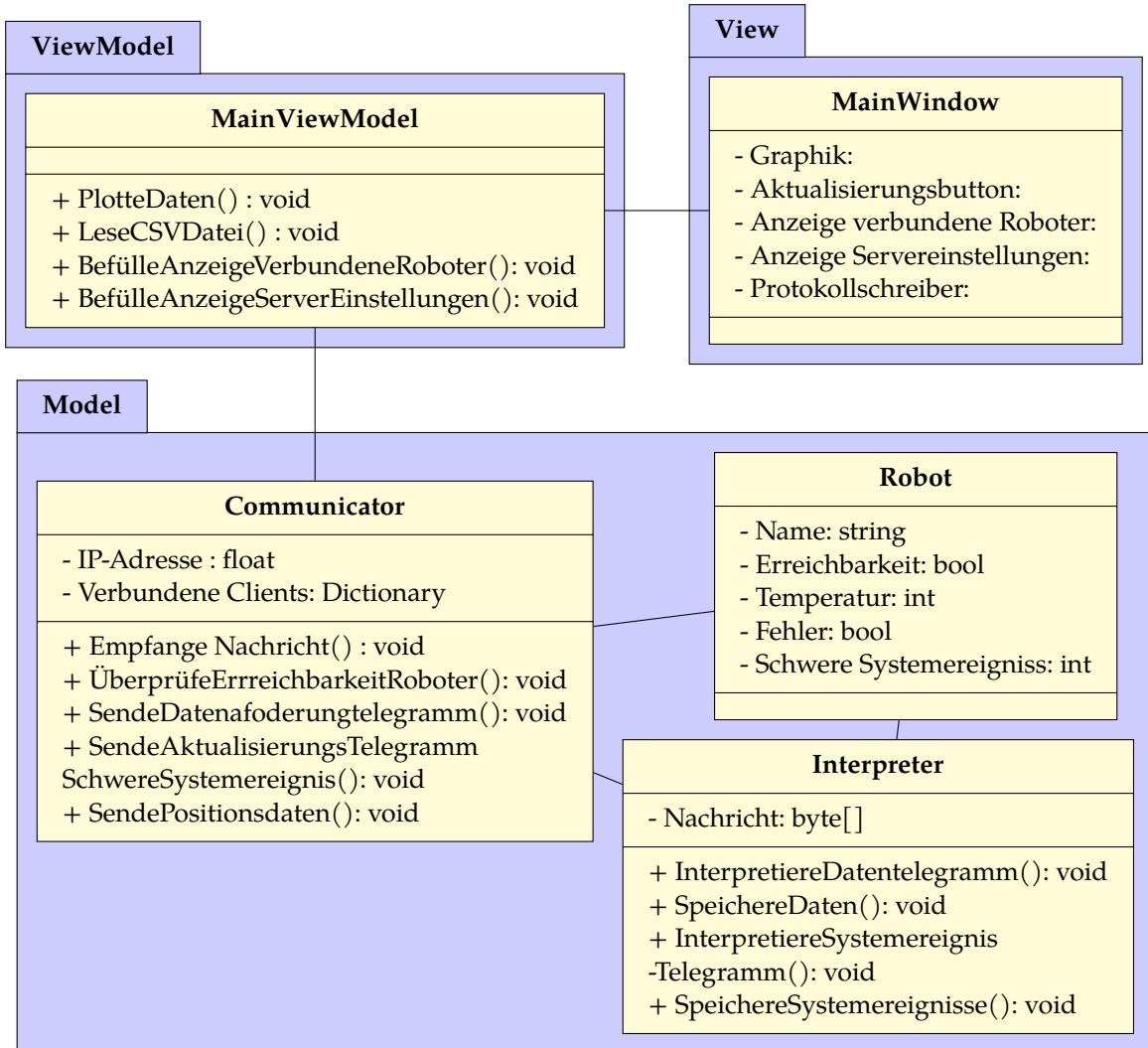


Abbildung 4.7: UML-Klassendiagramm WPF-Anwendung

# 5 Design und Implementierung

## 5.1 Stäubli-Roboter in VAL 3

### 5.1.1 EtherCAT

Nachdem die Feldbus-Verbindung in Kapitel 4.1 konzeptioniert wurde, wird nachfolgend die Umsetzung der gewählten Konfiguration der Feldbus-Verbindung kurz erläutert. Um eine EtherCAT Verbindung zwischen der SPS und der Stäubli Steuerung CS9 herzustellen muss zuerst eine physische Verbindung geschaffen werden. Ein LAN-Kabel verbindet den Master SPS mit dem Slave CS9. Anschließend erfolgt eine Konfiguration der EtherCAT-Verbindung in der Stäubli Robotic Suite. Hierzu dient eine in die SRS integrierte Anwendung namens „SYCON.net“ des Anbieters „Hilscher GmbH“. Hier wird unter anderem die zu übertragende Datenmenge festgelegt. Dabei erfolgt eine detaillierte Festlegung der zu übertragenden digitalen und analogen Werte. Es werden die zu übertragenden Daten, sowie deren Größe (z.B. 1 Byte oder ein Word, welches sich aus 2 Byte zusammensetzt) festgelegt. Von der Feldbus-Konfiguration kann eine sogenannte ESI-Datei erzeugt werden, welche in die SPS geladen wird, damit diese die EtherCAT-Konfiguration kennt. Nach der erfolgreichen Konfiguration muss in VAL 3 auf die EtherCAT-Verbindung zugegriffen werden, um Werte zu lesen oder zu beschreiben. Hierzu werden die einzelnen Daten der EtherCAT-Nachricht mit Variablen in der E/A-Verwaltung verknüpft. [19] [30]

Beispielsweise kann eine von der SPS gesendete Auftragsnummer über den Befehl „aioGet“ gelesen werden. Je nach Auftragsnummer wird ein gewisser Bewegungsablauf ausgeführt.

```
// Abfrage der Auftragsnummer von der EtherCAT-Schnittstelle
jobNumber = aioGet(aiJobNumber)
switch jobNumber
case 4
    // Aufruf zur Bewegungsausführung
    call moveToHome()
break
```

Ebenso können erfolgreich durchgeführte Bewegungen mit dem Befehl „aioSet“ bestätigt werden, in dem eine Übergabe an die EtherCAT-Verbindung erfolgt. [20]

## 5 Design und Implementierung

### 5.1.2 TCP/IP

Nach erfolgreicher Implementierung der EtherCAT-Verbindung soll die TCP/IP-Verbindung implementiert werden. Dabei sollen die in Kapitel 4.2 getroffenen Entscheidungen zur TCP/IP-Verbindung und dem Nachrichtenaufbau, wie dort konzeptioniert umgesetzt werden. Ebenso soll das Roboterprogramm in VAL 3 gemäß Kapitel 4.4 umgesetzt werden. Dies beinhaltet das Programm Empfangen, in welchem beispielsweise Datenanforderungstelegramme empfangen werden und mit einem Datentelegramm beantwortet werden. Ebenso sind die Programme zum Versand der Systemereignisse und zur Bewegungsausführung und Motormoment-Berechnung zu implementieren. Der Programmcode in VAL 3 wird hierbei nicht komplett aufgezeigt, sondern lediglich das Grundprinzip, sowie Besonderheiten aufgezeigt.

#### Implementierung TCP/IP-Verbindung

Für die Implementierung der TCP/IP-Verbindung auf dem Controller des Stäubli-Roboters muss in der SRS eine Socket-Verbindung angelegt werden. Hierzu wird in der E/A-Verwaltung ein Client angelegt, welcher die IP-Adresse und den Port des Servers zugewiesen bekommt. Darüber hinaus wird ein sogenannter Timeout von 0 s gesetzt. Bei einem Timeout von 0 wird auf den Vorgang, welcher ein Lesen oder Schreiben sein kann gewartet. Bei einem Timeout kleiner 0 wird hingegen nicht bis zur Ausführung des Vorgangs gewartet. Be einem Timeout größer 0 wird hingegen eine gewisse Zeit gewährt, bis zu dieser der Timeout durchgeführt werden kann. Die Nachricht soll in diesem Fall jedoch direkt gelesen oder geschrieben werden, weshalb kein Spielraum im Rahmen des Timeouts gewährt wird. [20] Die Socket-Verbindung wird als E/A-Verbindung in VAL3 betrachtet, weshalb eine globale Variable mit dem Namen des Clients angelegt werden kann und hierüber auch gelesen und beschrieben werden kann. Die Socket-Verbindung wird nur dann erstellt, wenn sie ihm Rahmen des Programmablaufs z.B. durch die Befehle sioSet und sioGet benötigt wird. Der Client versucht dann eine Verbindung zum Server aufzubauen. [20] [10]

#### Empfangen

In diesem Programm müssen zu Beginn eingehende Nachrichten der TCP/IP-Verbindung gelesen werden. Dies erfolgt mit folgendem Befehl:

```
num sioGet(sio siInput, num\& nData [])
```

Diese Funktion schreibt ein gelesenes Zeichen oder einen gelesenen Array von Zeichen von siInput in das Array nData. Als Rückgabewert dient die Anzahl der gelesenen Zeichen. Nachfolgend können die einzelnen Bytes der Nachricht interpretiert werden und somit entschieden werden, ob es sich um ein Datenanforderungstelegramm, eine Aktualisierung der Systemereigniss-Schwere oder Positionsdaten handelt. [20] [10]

Im Falle eines Datenanforderungstelegramm muss ein Datentelegramm gemäß der Konzeptionierung des Datentelegramms (vgl. Kapitel 4.2)) aufgebaut werden. Hierzu werden beispielsweise Temperaturen der Antriebe von den E/A-Boards der CS9 ausge-

## 5 Design und Implementierung

lesen. Diese eingelesenen numerischen Daten können durch folgenden Befehl in einen Byte-Strom gewandelt werden:

```
num toBinary(num nValue[], num nValueSize, string
sDataFormat, num\& nDataByte[])
```

Diese Funktion wandelt einen numerischen Wert, welcher das Datenformat sDataFormat besitzt in einen Byte-Strom und speichert diesen im Array nDataByte. Über das Datenformat wird beispielsweise angegeben ob es sich um einen Gleitkommawert handelt, ob ein Vorzeichen vorliegt und ob das Little-Endian oder das Big-Endian-Format angewandt wird. Mit nValueSize kann die Anzahl der zu kodierenden Zeichen beschränkt werden. Anschließend werden die gewandelten Bytes in den Byte-Array des zugesendeten Datentelegramms geschrieben. Das fertige Datentelegramm wird anschließend über folgenden Befehl versendet [20]:

```
num sioSet(sio siOutput, num\& nData[])
```

Mit dieser Funktion kann in VAL 3 die zu übermittelnde Nachricht nData versendet werden, indem der E/A-Verbindung siOutput die Nachricht zugewiesen wird. Zurückgegeben wird die Anzahl der geschriebenen Zeichen oder -1 im Falle des Timeouts.

Handelt es sich anstelle des Datenanforderungstelegramms um ein Telegramm zur Aktualisierung der zu übertragenden Systemereignis-Schwere, wird eine globale Variable, welche die zu übertragende Schwere des Roboters repräsentiert überschrieben. [20] [10]

### Systemereignisse

Um die Systemereignisse über TCP/IP an die Anlagenvisualisierung übertragen zu können müssen diese erst ausgelesen werden. Hierzu gibt es in VAL 3 zwei Möglichkeiten. Die eine Funktion ermöglicht die Ausgabe des gesamten Systemereignisses in ausformulierter Textform, die andere wiederum ermittelt nur den Ereigniscode. [20] Diese beiden Funktionen werden nachfolgend in einer Vergleichstabelle mit Hilfe von Harvey Balls verglichen (vgl. Tabelle 5.1). Der Füllheitsgrad des Harvey Balls entspricht dabei dem Grad der Zielerreichung. Für den Einsatz des Ereigniscodes spricht, dass der Um-

Tabelle 5.1: Vergleich Funktion zur Ermittlung Systemereignis

Ausformuliert	Ereigniscode
Datensparend	●
Verständlichkeit/Einfachheit	○
Flexibilität	○

**Legende:** ●: Hervorragend ○: Teils/Teils ○: Sehr schlecht

fang der zu übertragenden Daten viel geringer ist. Dafür muss bei dem Empfänger, der Anlagenvisualisierung, die Bedeutung der jeweiligen Ereigniscodes hinterlegt werden. Dies geht mit einem deutlichen Mehraufwand einher, vergrößert das Programm und

## 5 Design und Implementierung

reduziert die Verständlichkeit. Ebenso wird die Flexibilität deutlich reduziert, da bei Änderung der Ereigniscodes z.B. durch ein Software-Update oder eine Neubeschaffung des Stäubli-Roboters, eine manuelle Anpassung der Interpretation der Ereigniscodes erfolgen muss. Aus diesem Grund fällt die Entscheidung auf die Übertragung des Systemereignisses in ausformuliert Form. Hierzu kommt der VAL 3 spezifische Befehle zum Einsatz

```
l_nLastEvt = getEvents(l_nFirstEvt, l_nId, l_nType, l_sInfo,  
l_nSeverity)
```

Dieser ermittelt neben dem Systemereignis selbst und der Schwere, eine CS9-interne Nummer des letzten tatsächlich ausgelesenen Ereignisses (*l\_nLastEvt*). Bei der nächsten zyklischen Abfrage der Systemereignisse wird die Nummer des letzten ausgelesenen Ereignisses als Eingangsparameter für das erste auszulesende Ereignis (*l\_nFirstEvt*) verwendet. Dadurch wird vermieden, dass keine Systemereignisse mehrfach ausgelesen werden. Diese Funktion speichert zudem in *l\_nFirstEvt* die Nummer des tatsächlich zuerst ausgelesenen Ereignisses. Das System behält nur die letzten 20 Ereignisse gespeichert. Aus der Differenz des tatsächlich ersten ausgelesenen Ereignisses zur Vorgabe des ersten auszulesenden Ereignisses, lässt sich berechnen ob alle Ereignisse ausgelesen wurden oder ob aufgrund einer Überschreitung der Grenze von 20 Ereignissen, Ereignisse verloren gegangen sind. Die zyklische Abfrage der Ereignisse ist jedoch so dicht getaktet, dass ein Verlorengehen von Ereignissen im Betrieb quasi nicht auftreten kann. Nach dem Auslesen der Systemereignisse werden diese einzeln als Nachrichten über TCP/IP versendet. Hierbei wird der entworfene Telegrammaufbau von Kapitel 4.1) angewandt.

### Bewegungssteuerung

In dem Programm Bewegungssteuerung erfolgt die Steuerung der Roboterbewegung. Hierzu werden Punkte in der SRS angelegt, welche durch Teachen an dem realen Roboter auf die genaue geometrische Situation angepasst werden. Nachfolgend ist eine Verfahrbewegung exemplarisch aufgezeigt:

```
moveL(pZwischenPos, tTool, mLspeed)
```

Hierdurch erfolgt eine lineare Bewegung zu dem Punkt „*pZwischenPos*“. Die Bewegungsdaten wie Geschwindigkeit und Beschleunigung sind in „*mLspeed*“ abgespeichert. Die Geometrie des Robotergreifers ist in der Variablen „*tTool*“ hinterlegt.

Zu Beginn einer Bewegungsausführung wird zudem ein neuer Task gestartet, in dem das Drehmoment über die Zeit integriert werden soll. Nach Ablauf des kompletten Bewegungsablauf wird dieser Task wieder beendet. Das aktuelle Ergebnis der Integration wird abgefragt und entspricht dem integrierten Moment eines kompletten Bewegungsablaufes.

### Motormoment-Integrator

Dieses Programm, welches von dem Programm Bewegungssteuerung gestartet und

## 5 Design und Implementierung

beendet wird führt die Integration der Motormomente aller sechs Achsen über die Zeit durch. Mit dem Aufruf des Programms wird das integrierte Motormoment auf Null gesetzt, da es sich um den Beginn eines Bewegungsablaufes handelt. Die Integration wird durch das Bilden einer Summe mit einer Abtastperiode von 0,1 Sekunden realisiert. Dieser Zusammenhang ist in nachfolgender Formel dargestellt:

$$\int_0^T M(t) dt \approx \sum_{i=0}^n 0,1 \cdot M_i \quad (5.1)$$

In einer Endlos-Schleife mit einer Wartezeit von 0,1 Sekunden wird jeweils der Absolutwert des aktuellen Motor-Momentes multipliziert mit 0,1 Sekunden zu dem aktuellen integrierten Motormoment addiert. Das aktuelle integrierte Motormoment kann am Ende eines Bewegungsablaufes abgefragt werden und entspricht somit dem integrierten Motormoment über die Zeit des gesamten Bewegungsablaufes.

## 5.2 WPF-Anwendung

Wie in Stand der Technik (vgl. Kapitel 2.6) analysiert, soll für die Implementierung der WPF-Anwendung C# zum Einsatz kommen. Als Entwicklungsumgebung dient Visual Studio und als Source-Code-Verwaltungs-Programm wird „Azure DevOps“ eingesetzt. In Kapitel 4.5 wurde MVVM als Architekturmuster gewählt. Mit Hilfe eines Klassendiagramms wurde festgelegt, dass neben den Klassen MainWindow und MainViewModel, die Klassen Communicator, Robot und Interpreter, welche zusammen das Model bilden, existieren sollen. Im Klassendiagramm wurden zudem die Methoden der jeweiligen Klassen definiert. Auf eine detailreiche Darstellung der Implementierung wird in diesem Kapitel verzichtet. Stattdessen werden richtungsweisende Entscheidungen, sowie Grundprinzipien und wichtige Funktionen erläutert.

### 5.2.1 TCP/IP-Server

In der Klasse Communicator muss ein TCP/IP-Server umgesetzt werden, der zudem Nachrichten sendet und empfängt. Hierzu wird in einem ersten Schritt eine Instanz der Klasse TcpListener unter Angabe der IP-Adresse und des Ports erstellt. [31]

```
tcpListener = new TcpListener(localAddr, port);
```

Nachdem der Server erstellt wird muss auf Clients gewartet werden, die sich verbinden möchten. Dies wird in einer Endlos-Schleife realisiert, die nur durch ein Schließen des Programmes gesteuert unterbrochen wird. Dies ermöglicht es, dass mehrere Clients eine Verbindung zu dem Server aufbauen können. Ebenso erhält ein Client, der eine Verbindung abgebrochen hat, die Chance eine erneute Verbindung aufzubauen. Der Verbindungsaufbau wird durch folgenden Befehl realisiert:

## 5 Design und Implementierung

```
var tcpClient = await tcpListener.AcceptTcpClientAsync();
```

Nachfolgend wird noch innerhalb der endlos-Schleife eine asynchrone Methode aufgerufen, die es ermöglicht Nachrichten von der Verbindung zu lesen. Bei Aufruf der asynchronen Methode, wird nicht auf das Ende der Methode gewartet, sondern die Ausführung wird direkt nach Aufruf fortgeführt [22]. Dies ermöglicht es in einem erneuten Schleifen-Durchlauf direkt eine Verbindung zu einem neuen Client aufzubauen. Das Prinzip der asynchronen Methoden ermöglicht daher, eine Kommunikation mit mehreren Robotern sicherzustellen, was einer Kann-Forderung aus dem Lastenheft entspricht. Zudem kann diese Methode mehrfach gleichzeitig für jeden einzelnen Client ausgeführt werden. Bei Aufruf der asynchronen Methode muss der zuvor verbundene Client übergeben werden. Innerhalb dieser Methode wird ein sogenannter Netzwerkstream aufgerufen und als Objekt zurückgegeben.

```
var stream = client.GetStream();
```

Netzwerkstream ist eine Klasse in C# und stellt Methoden zum Senden und Empfangen von Nachrichten. Nachfolgend wird über die Methode ReadAsync der Klasse NetworkStream von dem Netzwerkstream gelesen und in einen Byte-Array namens bufferIn geschrieben.

```
int bytesRead = await stream.ReadAsync(bufferIn, 0, bufferIn.Length);
```

Der await-Operator ermöglicht es, dass das Ende dieser Methode (ReadAsync) trotz des asynchronen Charakters abgewartet wird [22].

### Nachrichteninterpretation

Anschließend muss der gelesene Byte-Array interpretiert werden. Hierfür wird gemäß der Konzeptionierung der WPF-Anwendung (vgl. Kapitel 4.5) eine eigene Klasse namens Interpreter erstellt. Nach dem Lesen einer TCP-IP Nachricht kann eine Methode dieser Klasse entweder ein Datentelegramm oder ein Systemereignis-Telegramm entschlüsseln bzw. interpretieren, in dem dieser Methode der Byte-Array übergeben wird. Dort werden dann die einzelnen Bytes der Nachricht gelesen und somit z.B. die Temperatur der CPU ermittelt. Diese ermittelten Eigenschaften werden in eine Instanz der Klasse Robot, welche somit einen einzelnen konkreten Roboter darstellt, geschrieben.

### Verbundene Roboter

Zusätzlich soll die Klasse Communicator ermitteln, welche Roboter aktuell eine Verbindung mit dem Server aufgebaut haben. Diese Information kann über das Fenster, dem Nutzer angezeigt werden. Die Schwierigkeit besteht darin, dass über die aufgebauten Verbindungen zwar einfach dokumentiert werden kann, ob Clients verbunden sind, jedoch keine Information darüber besteht, um welchen Roboter es sich dabei handelt. Die Identifikation des Roboters kann erst mit der ersten Nachricht des Roboters erfolgen, in welcher der Robotername enthalten ist. Zur Speicherung der verbunden Clients und

## 5 Design und Implementierung

den damit verknüpften Robotern ergeben sich mehrere Möglichkeiten. Beispielsweise kann eine Liste von Tupeln angelegt werden, als Schlüssel kann der Client dienen und als Wert der Roboter. Eine andere Alternative stellt eine Liste oder ein Array aus benutzerdefinierten Objekten dar. Das Objekt besteht in diesem Fall aus dem Client und dem Roboter. Eine weitere Möglichkeit ist ein Dictionary. Dies ermöglicht es Mengen von Schlüssel-Werte-Paaren zu speichern. Gegen den Einsatzes einer Liste mit Objekten spricht, dass dies den Aufwand und die Komplexität erhöht, da diese Objekte erst kreiert werden müssen. Die Besonderheit von Listen mit Tupeln hingegen ist, dass die Elemente in einer vorgegeben Sequenz angeordnet sind und ein Zugriff über die Reihenfolge erfolgen kann. In diesem Anwendungsfall ist die Sequenz der Elemente jedoch ohne Bedeutung, weshalb sich ein Dictionary besonders für die Anwendung eignet. Dieses ermöglicht einen effizienten Zugriff mit Hilfe der Schlüsselbasierten Suche. [23] [22] Die Initialisierung des Dictionarys ist nachfolgend dargestellt.

```
private ConcurrentDictionary<TcpClient, Robot>
    dictionaryConnectedClients = new
        ConcurrentDictionary<TcpClient, Robot>();
```

Der TcpClient wird in diesem Fall als Schlüssel eingesetzt und das Objekt der Klasse Robot dient als Wert. Ein ConcurrentDictionary unterscheidet sich von einem normalen Dictionary durch seine Fähigkeit, sicher von mehreren Threads gleichzeitig bearbeitet zu werden, ohne dass externe Synchronisierung notwendig ist. Da mehrere Clients verbunden sein können, kann es dazukommen, dass mehrere asynchronen Methoden zeitgleich das Dictionary beschreiben möchten. Wird erstmalig eine Verbindung zu einem Client aufgebaut wird dieser in das Dictionary aufgenommen, bricht eine Verbindung ab, wird dieser wieder aus dem Dictionary entfernt. Wird erstmalig ein Roboter durch seinen Namen identifiziert, wird dieser dem entsprechenden Client als Wert im Dictionary zugewiesen.

**appsettings.json** Darüberhinaus ist eine appsettings.json-Datei in die WPF-Anwendung implementiert, in der wichtige Einstellungen hinterlegt werden. Der Bediener oder der Anlagenautomatisierer kann in diese Datei wichtige Einstellungen hineinlegen. Hier wird z.B. neben der IP-Adresse und des Ports, eine Liste der installierten Roboter abgelegt. Durch ein Vergleich dieser mit den tatsächlich verbundenen Robotern im Dictionary, kann eine Warnung ausgegeben werden, dass ein gewisser Roboter nicht verbunden ist. Zudem ist in dieser Datei hinterlegt, ab welcher Schwere Systemereignisse übertragen werden sollen. Gemäß Konzeptionierung der WPF-Anwendung (vgl. Kapitel 4.5) muss die WPF-Anwendung den Robotern mitteilen, ab welcher Schwere diese die Systemereignisse versenden sollen. Wird erstmalig eine Verbindung zu einem Roboter aufgebaut oder nach einem Abbruch der Verbindung erfolgt ein erneuter Aufbau, wird ein Telegramm an diesen Roboter versandt, welches diese Soll-Schwere enthält. Liegt beispielsweise eine Störung oder Auffälligkeit an einem Roboter vor kann ein Servicetechniker die

## 5 Design und Implementierung

appsettings.json-Datei anpassen (ggf. auch per Fernzugriff) und es dadurch ermöglichen, dass alle Systemereignisse, wie z.B. auch reine Informationen übertragen werden.

### 5.2.2 Daten-Logger

Um die Prozessdaten, vor allem aber die Systemereignisse zu protokollieren gibt es mehrere Möglichkeiten. Einerseits kann eine entsprechende Funktion selbst implementiert werden, alternativ kann auf eine Logging-Bibliothek zurückgegriffen werden. Hierfür existieren eine Vielzahl an verschiedenen Anbieter von NLog bis hin zu ELMAH. [32] Einige Lösungen werden nachfolgend kurz vorgestellt

- **NLog:** Diese Logging-Bibliothek zählt zu den weit verbreitetsten und performantesten Lösungen. Es ermöglicht ebenso lokale Logging-Files zu erzeugen. Dieses Framework wurde bereits in der Vergangenheit bei robomotion eingesetzt. NLog steht kostenfrei zur Verfügung.
- **Log4NET:** Der Aufbau und der Funktionsumfang ähnelt stark NLog und es erhebt ebenso keine Lizenzkosten.
- **ELMAH:** Diese Bibliothek ist spezialisiert auf das Protokollieren von Fehlern. Es erhebt keine Gebühren. [32]

Die vorgestellten Lösung einschließlich einer manuellen Implementierung werden nachfolgend gegenübergestellt (vgl. Tabelle 5.2). Der Funktionsumfang ist bei der manuellen

Tabelle 5.2: Vergleich Logging-Möglichkeiten

	<b>NLog</b>	<b>Log4NET</b>	<b>ELMAH</b>	<b>Manuell</b>
Umfang	●	●	○	●
Implementierungsfreundlichkeit	●	○	○	○
Standardisierung	●	○	○	○
Kostengünstig	●	●	●	●

**Legende:** ●: Hervorragend ○: Teils/Teils ○: Sehr schlecht

Lösungen am höchsten zu bewerten, da jede beliebige Funktion implementiert werden können. ELMAH hingegen ist etwas spezialisierter als NLog und Log4NET, weshalb hier der Funktionsumfang etwas geringer ausfällt. Bei der Implementierungsfreundlichkeit sind die Bibliotheken klar der manuellen Lösung überlegen. NLog stellt hier die einfachste Lösung dar, da bei Bedarf auf die vorherige Erfahrung aus früheren Projekten zurückgegriffen werden kann. Ebenso ist bei NLog die Standardisierung am höchsten einzustufen, da es bereits in anderen Anwendungen zum Einsatz kommt. Alle vier Möglichkeiten sind zudem kostenfrei, weshalb alle die maximal mögliche Wertung erhalten.

## 5 Design und Implementierung

Die Gesamtentscheidung fällt auf Basis der vorangegangenen Punkten klar auf NLog. Bei der Implementierung von NLog kann zudem ein lokaler Ablageort als sogenanntes TargetFile angegeben werden. Dies ermöglicht es, die Systemereignisse und die Prozessdaten als CSV- bzw. Text-Datei abzuspeichern. Ebenso ist es möglich eine Ausgabe durch einen sogenannten NlogViewer zu realisieren. Dadurch kann eine Darstellung für den Nutzer erfolgen. Zum Einsatz kommt die „Sentinel.NLogViewer“-Bibliothek.

NLog bietet verschiedene „Logging-Levels“ wie z.B. Info oder Fatal an [33]. Bei den übertragenen Systemereignissen wird die Schwere in der Nachricht übertragen. Nach Interpretation dieser Schwere, kann das Systemereignis über NLog mit dem selben Level ausgegeben werden, wie nachfolgend exemplarisch dargestellt ist:

```
_EventLogger.Error(senderName + ":" + messageContent);
```

Es werden drei verschiedene Logger-Instanzen angelegt. Eine für das Protokollieren von Systemereignissen, Einen für die Prozessdaten und einen für den Ablauf der WPF-Anwendung, z.B. für wichtige Ereignisse oder Fehler.

### 5.2.3 Diagramm

Die erfassten und gespeicherten Prozessdaten, wie z.B. die Temperatur der CPU soll als graphischer Verlauf über die Zeit dargestellt werden. Für die Erstellung von Diagrammen gibt es in C# eine Vielzahl von Bibliotheken. Bekannte Bibliotheken sind Oxyplot, LiveCharts und ScottPlot. Aufgrund des besonders positiven optischen Erscheinens von OxyPlot fällt die Wahl auf diese Bibliothek. Zudem sind die Anpassungsmöglichkeiten bei LiveCharts und ScottPlot geringer. Prinzipiell lässt sich für die Umsetzung jedoch jede Bibliothek einsetzen. Für das Erstellen eines Diagrammes mit OxyPlot müssen in einem ersten Schritt die Prozessdaten aus der CSV-Datei gelesen und in einer Liste von Tuples abgespeichert werden. Für die Darstellung der x-Achse dient die Uhrzeit, für die y-Achse der eigentliche Datenwert. Abschließend wird ein Diagramm dieser Daten mit Hilfe von OxyPlot erstellt. Darüberhinaus wird realisiert, dass es möglich ist mehrere Daten, wie z.B. die Temperaturen verschiedener Achsen in einem Diagramm darzustellen. Hierzu werden mehrere Spalten aus der CSV-Datei ausgelesen und für die einzelnen Datenreihen verwendet. Eine Legende mit einer Beschriftung der Datenreihen komplettiert das Diagramm.

Da eine Vielzahl von Robotern in einer Anlage eingesetzt werden, wird ein Dropdown-Menü angelegt, um zwischen den einzelnen Robotern in der Darstellung wechseln zu können. Das Dropdown-Menü bezieht die Roboternamen automatisiert aus der appsettings.json-Datei, in der alle installierten Roboter hinterlegt sind. Es wird bewusst diese Liste und nicht das Dictionary der tatsächlich verbundenen Roboter als Basis für das DropDown-Menü verwendet. Dadurch ist es auch noch nach Ausschalten eines Roboters und eines einhergehenden Verbindungsabbruch möglich, Daten dieses Roboters aus der CSV-Datei darzustellen. Die Diagramme werden bei Programmstart einmalig automatisiert erstellt, ebenso wie bei einer Neuauswahl im DropDown-Menü. Jedoch

## 5 Design und Implementierung

werden sie nicht zyklisch automatisiert aktualisiert, um Rechenkapazität zu sparen und da sich die Daten meist nicht hochdynamisch ändern. Ein Aktualisierungsschalt-Feld erlaubt ein nachträgliches Aktualisieren.

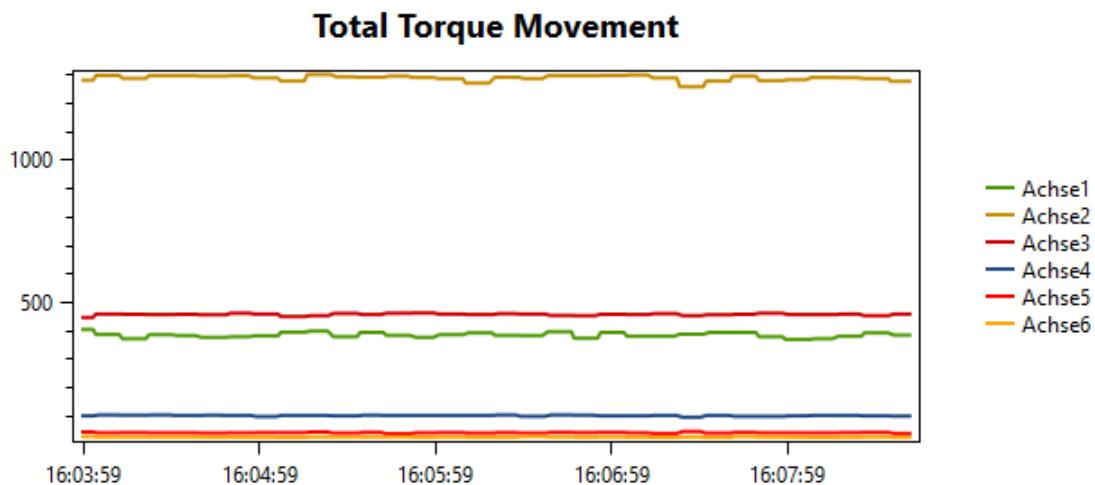


Abbildung 5.1: OxyPlot - Diagramm

### 5.2.4 Benutzeroberfläche

Für die Darstellung der gesammelten Informationen, wie die Systemereignisse oder die Prozessdaten in Diagrammform muss eine Benutzeroberfläche implementiert werden. Gemäß gewähltem MVVM-Muster (vgl. Kapitel 4.5) erfolgt die Definition der Benutzeroberfläche in der Klasse MainWindow. Mittels XAML werden einzelne Elemente, wie z.B. Buttons oder OxyPlot-Diagramme angelegt und mittels DatenBindung mit der Klasse MainViewModel verknüpft.

Alle die Elemente sollen möglichst übersichtlich dargestellt werden und es sollen möglichst viele Informationen integriert werden. Dadurch, dass die Anwendung später in roboTools integriert wird, ist das visuelle Erscheinungsbild nicht von höchster Bedeutung. Einzelne Elemente, wie z.B. die Ausgabe der Systemereignisse oder die Diagramme mit den Prozessdaten werden in roboTools integriert, wobei das optische Erscheinungsbild, wie z.B. das Unternehmenslogo und die Farbgestaltung durch roboTools bereits vorgegeben ist. Eine Integration ist durch den Einsatz von Datenbindung besonders einfach möglich (vgl. Kapitel 4.5). Die Anordnung der einzelnen Elemente ist nachfolgend dargestellt (vgl. Abbildung 5.2 und 5.3). Dabei erfolgt eine Trennung der Protokoll-Ausgaben und der Graphen in zwei verschiedene Registerkarten.

## 5 Design und Implementierung

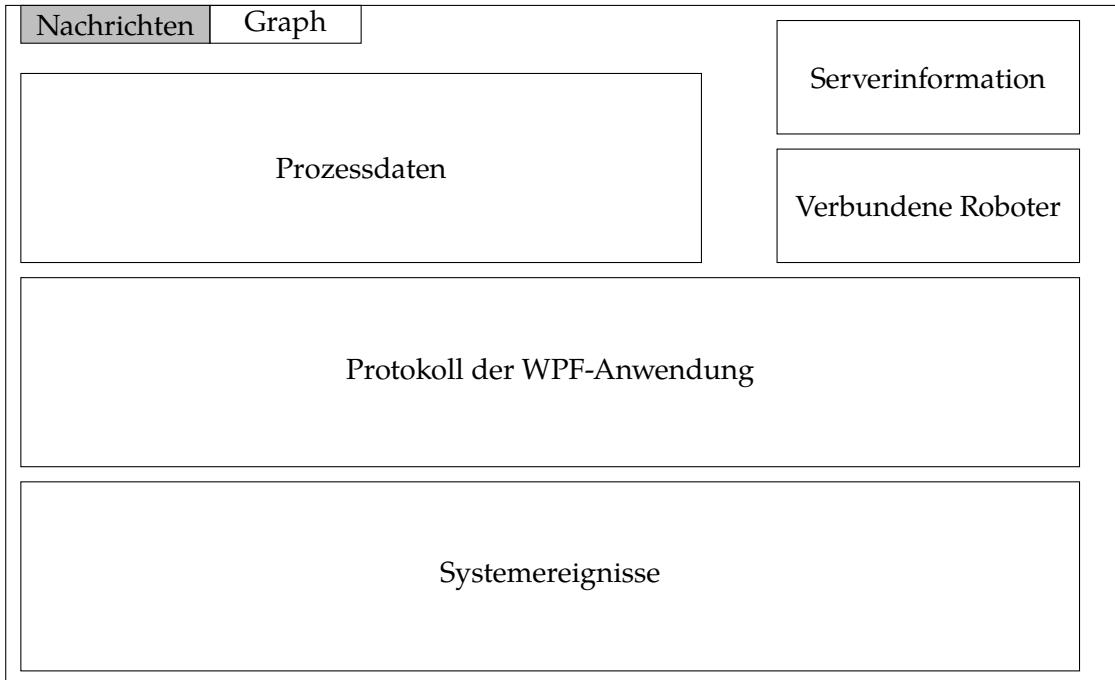


Abbildung 5.2: WPF-Anwendung Benutzeroberfläche Registerkarte Nachrichten

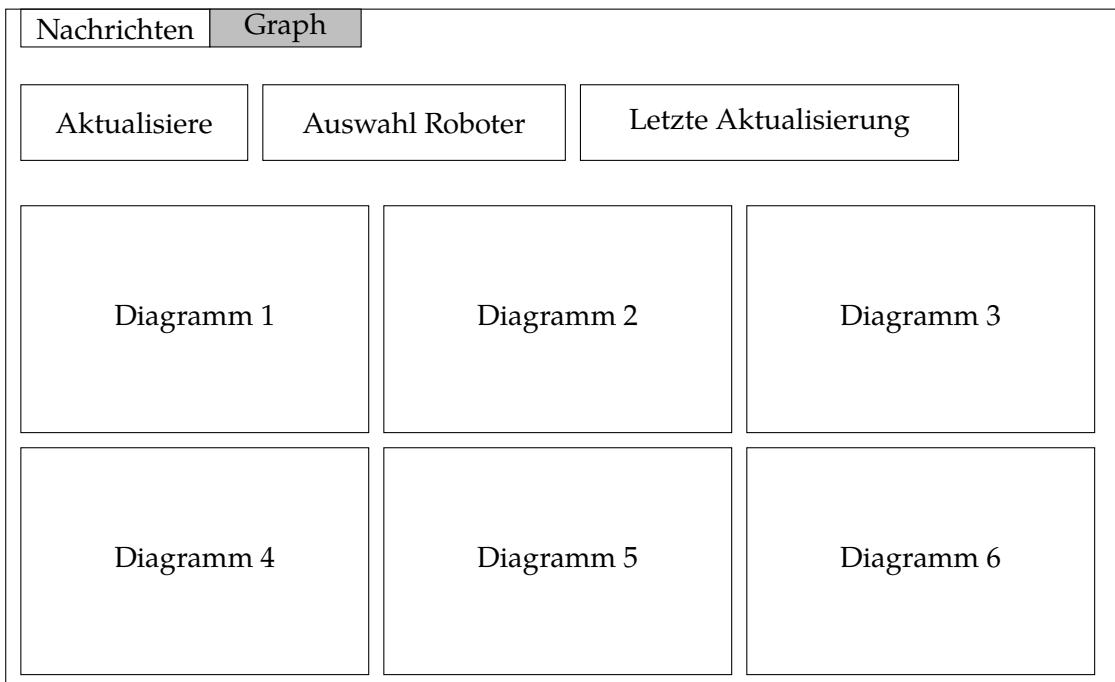


Abbildung 5.3: WPF-Anwendung Benutzeroberfläche Registerkarte Graph

## *5 Design und Implementierung*

Hierbei wurde versucht alle Informationen zu integrieren. Beispielhaft ist die Protokollierung des Programmablaufes enthalten, auch wenn eine Integration dieser Ausgabe für den Endnutzer in roboTools nicht vorgesehen ist, sondern lediglich eine lokale Speicherung. Zu Debugging-Zwecken oder zur Fehlerfindung an dem Roboter bei Reparaturarbeiten ist eine Ausgabe aller Daten jedoch hilfreich.

# 6 Validierung

Die korrekte und zuverlässige Funktionsweise des entworfenen Systems und insbesondere der Software ist von zentraler Bedeutung [34]. Ein Test wird dazu genutzt, um einen Nachweis zu erzielen, dass die Funktionsweise gegeben ist. Hierbei kann in Funktions- tests und Fehlertests unterschieden werden. Funktionstests dienen dazu, zu prüfen, ob die vorgegebenen Funktion erfüllt sind. Fehlertests hingegen prüfen die Fehlertoleranz des Systems. [2] [3]

Beide Testarten sollen in diesem Rahmen zur Validierung zum Einsatz kommen.

## 6.1 Funktionstests

Um die Funktionalität des entworfenen Systems nachzuweisen, soll das System unter realen Bedingungen getestet werden. Hierzu wird an den Stäubli-Roboter TX2-90 ein Greifer für Wiener-Würstchen montiert. Ebenso wird ein Windows-PC, auf welchem die WPF-Anwendung läuft mit dem Steuergerät CS9 des Roboters über LAN-Kabel verbunden (vgl. Abbildung 6.1).

Folgende Funktionalitäten werden nachfolgend am Gesamtsystem getestet:

- **1. Übertragung Warnungen/Fehlermeldungen/...:** Zum Testen, ob beispielsweise Fehlermeldungen wie gewünscht übertragen werden, erfolgt ein Provozieren von Fehlern oder Warnungen durch den Versuch von manuellem Verfahren im Betrieb oder durch das Auslösen des Not-Halts. Dies lässt der Roboter nicht zu und generiert ein Systemereignis.
- **2. Übertragen bei Umteachen:** Es wird ein Punkt der Roboterbahn umgeteacht, um zu überprüfen, ob diese Meldung an die WPF-Anwendung übertragen wird.
- **3. Änderung Ereignisschwere:** Die Schwere, ab der der Roboter die Systemereignisse übertragen soll, wird in der appsettings.json-Datei der WPF-Anwendung angepasst. Abschließend kann überprüft werden, ob der Roboter nur die Ereignisse gemäß den Vorgaben überträgt.
- **4. Anzeige der Prozessdaten:** Es wird überprüft, ob Prozessdaten, wie Temperaturen korrekt übertragen werden.

## 6 Validierung

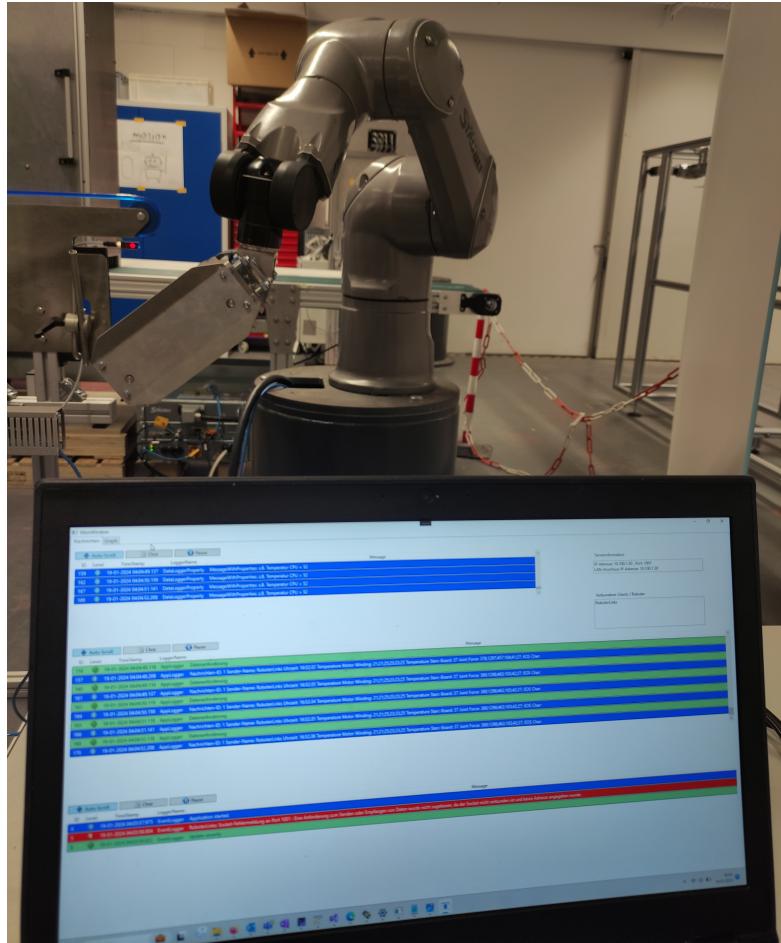


Abbildung 6.1: Roboteraufbau mit Greifer und WPF-Anwendung

- **5. Integration der Motormomente:** Die Motormomente werden über einen gesamten Bewegungsablauf aufgezeichnet. Durch das Aufbringen einer externen Kraft, soll der Einfluss auf das integrierte Moment untersucht werden.
- **6. Binäre Werte zur Fehlerüberwachung:** Es wird eine Vielzahl von binären-Werten im Datentelegramm übertragen, wie z.B. die Informationen ob ein CPU-Überstrom vorliegt. Zum Testen, wird der Wert manuell am Stäubli-Roboter manipuliert.
- **7. Verbindung mehrerer Roboter:** Da als Hardware nur ein Roboter zur Verfügung steht, kann die Kommunikation mit mehreren Robotern nicht unter realen Bedingungen getestet werden. Um dies dennoch möglichst praxisnah zu validieren wird ein physischer Roboter über das LAN-Kabel mit dem Windows-Rechner verbunden. Ein weiterer Roboter, wird durch ein TCP/IP-Client-Terminal simuliert. In dieser

## 6 Validierung

virtuellen Client-Anwendung werden Nachrichten an den Roboter gesendet, so wie Nachrichten von dem Roboter empfangen.

Die Ergebnisse dieser Funktionstests sollen nachfolgend kurz dargestellt werden. Dabei wird die Funktion durch Bildschirmfotos der Benutzeroberfläche unterstrichen, um einen Eindruck über die entworfene Anwendung zu vermitteln.

### 1. Übertragung Warnungen/Fehlermeldungen/...

Die Systemereignisse, die am Stäubli-Roboter entstanden sind, wie einfache Infomeldungen oder vorliegende Fehler wurden erfolgreich an die WPF-Anwendung übertragen. Dort wurden sie wie folgt ausschnitthaft dargestellt ausgegeben (vgl. Abbildung 6.2):

ID	Level	TimeStamp	LoggerName	Message
1101	!	19-01-2024 03:39:55.674	EventLogger	RoboterLinks: Abschalten der Antriebe: Betriebsart wurde geändert.
1103	!	19-01-2024 03:39:56.260	EventLogger	RoboterLinks: Antriebe werden abgeschaltet ...
1108	!	19-01-2024 03:39:56.696	EventLogger	RoboterLinks: Der Status des Sicherheitsstopps wurde auf '2' (SS1) geändert.
1110	!	19-01-2024 03:39:56.924	EventLogger	RoboterLinks: Bremsen geschlossen.
1112	!	19-01-2024 03:39:57.150	EventLogger	RoboterLinks: Sequenz zum Abschalten der Antriebe beendet.
1114	!	19-01-2024 03:39:57.386	EventLogger	RoboterLinks: Safety Restart (z.B. blaue Taste an WMS9) wird benötigt

Abbildung 6.2: Anzeige Systemereignisse

### 2. Übertragung bei Umteachen

Ebenso wurde beim Umteachen von Bewegungspunkten erfolgreich eine Meldung in der WPF-Anwendung ausgegeben (vgl. Abbildung 6.3).

1253	!	19-01-2024 03:40:29.689	EventLogger	RoboterLinks: joint jHome[0] geteacht.
------	---	-------------------------	-------------	--

Abbildung 6.3: Meldung bei Umteachen von Bewegungspunkten

### 3. Änderung Ereignisschwere

Nach Anpassung der Ereigniss-Schwere in der WPF-Anwendung wurden wie gewünscht nur noch die Systemereignisse aber dieser Schwere übertragen. Bei Setzen der Ereigniss-Schwere auf 2 wurden die einfachen Informationsmeldungen nicht mehr übertragen, sondern lediglich Warnungen oder noch schwerwiegender Ereignisse.

### 4. Anzeige der Prozessdaten

Die Prozessdaten des Roboters werden wie gewünscht in der WPF-Anwendung angezeigt (vgl. Abbildung 6.4). Ebenso ist es möglich zwischen potentiell mehreren Robotern über ein Drop-Down-Menü zu wechseln. Über die Aktualisierungs-Schaltfläche können die Grafiken aktualisiert werden.

### 5. Integration der Motormomente

Eine Besonderheit stellen die Motormomente dar, da diese nicht wie andere Prozessdaten unverarbeitet übertragen werden, sondern über einen gesamten Bewegungsablauf über

## 6 Validierung

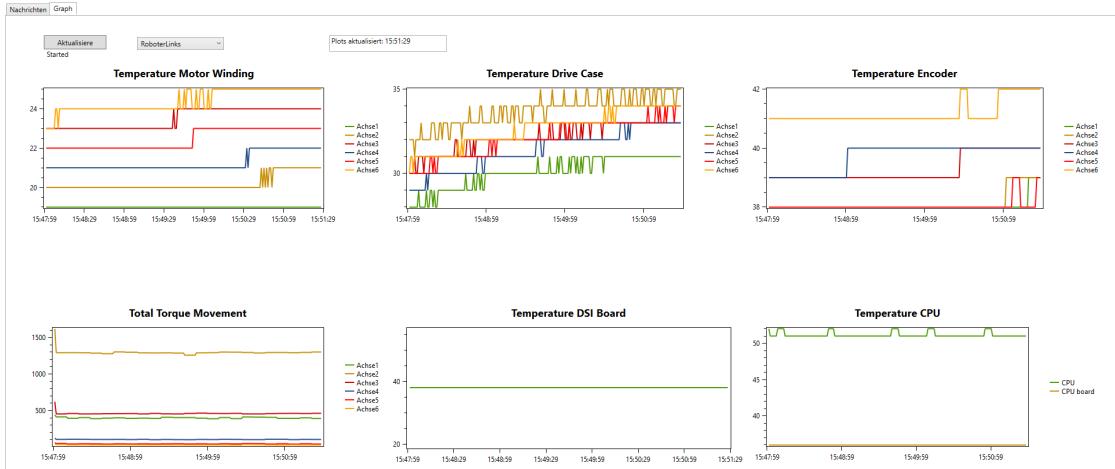


Abbildung 6.4: Visualisierung der Prozessdaten

die Zeit integriert werden. Im Testfall wurde auf den Greifer zeitweise eine externe Kraft aufgebracht, um den Einfluss der Last auf das Motormoment zu erkennen. In dem Drehmomentenverlauf ist ein deutlicher Anstieg der Momente, in den Bewegungsabläufen zu sehen, in denen eine Kraft aufgeprägt war (vgl. Abbildung 6.5). Für die Motormomente, ergeben sich vielfältige Nutzungspotentiale. Ein Anstieg der Motormomente kann darauf hindeuten, dass die Achse geschmiert werden soll oder das ein Defekt z.B. am Lager vorliegt. Bei Überschreiten von Grenzwerten, die für jeden Anwendungsfall individuell bestimmt werden müssen, kann der Bediener gewarnt werden. Im Idealfall kann bereits durch das Erkennen von Tendenzen im Momentenverlauf, eine Wartung vorgenommen werden, bevor eine Schädigung überhaupt eintritt. Hierbei spricht man von vorausschauender Wartung, was die Ausfallzeiten der Anlage deutlich reduzieren kann.

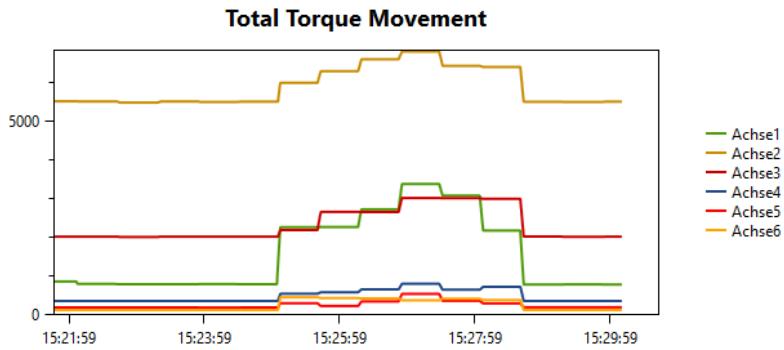


Abbildung 6.5: Visualisierung der integrierten Motormomente

## 6. Binäre Werte zur Fehlerüberwachung

## 6 Validierung

Von dem Roboter werden eine Vielzahl an binären Werten zu der WPF-Anwendung übertragen. Manche von ihnen sind ein Fehlerindikator. Wird z.B. der CPU Überstrom auf true bei dem Roboter manipuliert, wird dies von der WPF-Anwendung in der Fehleranalyse wie gewünscht entdeckt und ausgegeben (vgl. Abbildung 6.6).

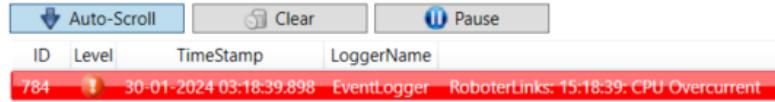


Abbildung 6.6: Binäre Werte zur Fehlerüberwachung

### 7. Verbindung mehrerer Roboter

Durch eine virtuelle Client-Anwendung wird der Einsatz von mehreren Robotern nachgebildet. Die Kommunikation der WPF-Anwendung mit mehreren Clients erfolgt wie gewünscht und es erfolgt eine Ausgabe der verbundenen Clients (Roboter) im dafür eingerichteten Ausgabefelder der WPF-Anwendung (vgl. Abbildung 6.7).



Abbildung 6.7: Anzeige der verbundenen Roboter

## 6.2 Fehlertests

Ein Fehler ist dadurch definiert, dass das Ergebnis nicht den Anforderungen entspricht und es zu einem Fehlverhalten kommt. [35]

Fehler können sehr vielfältiger Natur sein. Insbesondere solche, die nur in speziellen Ausnahmefällen auftreten, sind schwer zu entdecken. Ein Ermitteln und Testen aller Ausnahmefällen ist daher schlicht unmöglich. Ziel ist es daher, möglichst viele spezielle Anforderungen an das System zu identifizieren, welche zu Fehlern führen können und diese anschließend zu testen. Für eine systematische Analyse von potentiellen Fehlerursachen sollen ein Fehlerbäume eingesetzt werden. Diese ermöglichen es Zusammenhänge zwischen Ursache und Wirkung darzustellen. [34]

Hierzu werden in einem ersten Schritt mögliche Fehler, wie z.B. ein Programm-Absturz durch Brainstorming identifiziert. In einem zweiten Schritt werden mögliche Ursachen

## 6 Validierung

ermittelt, welche zu den genannten Fehlern führen könnten. Ein Testen durch bewusste Provokieren dieser Fehlerursachen ermöglicht es potentielle Fehler zu entdecken und die Fehlertoleranz des Systems zu validieren. Die entworfenen Fehlerbäume für mögliche Fehler sind nachfolgend dargestellt (vgl. Abbildung 6.8, 6.9, 6.10 und 6.11).

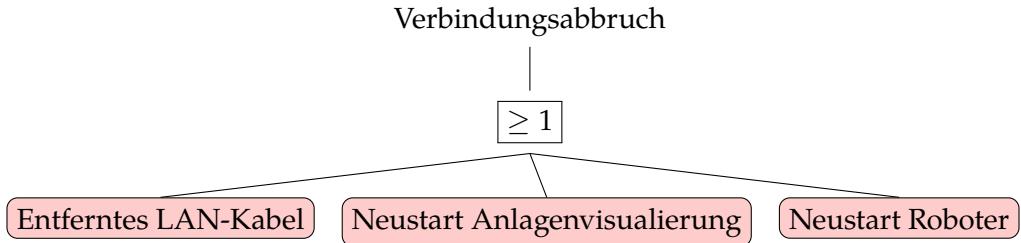


Abbildung 6.8: Fehlerbaumdiagramm Verbindungsabbruch

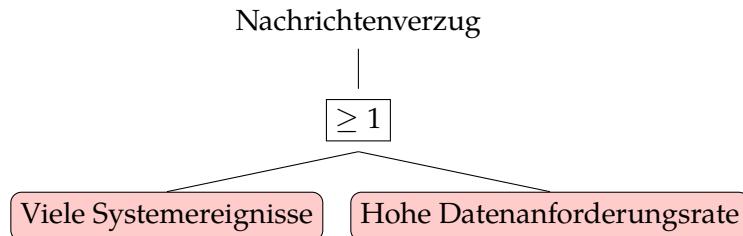


Abbildung 6.9: Fehlerbaumdiagramm Nachrichtenverzug

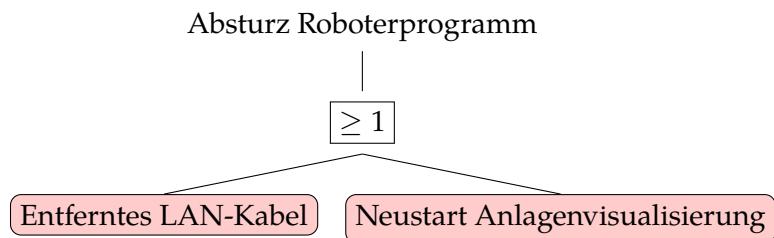


Abbildung 6.10: Fehlerbaumdiagramm Absturz Roboterprogramm

Alle ermittelten möglichen Fehlerursachen, wie z.B. das Entfernen des LAN-Kabels, wurden zur Validierung getestet, um auszuschließen, dass es zu einem Fehler kommt. Nach kleineren Optimierungen an der WPF-Anwendung ist es dem System nun möglich, auf alle potentiellen Fehlerursachen zu reagieren und eine robuste und fehlerfreie Fortführung des Systems zu gewährleisten. Beispielsweise wird beim Neustart des Roboters eine Meldung in der WPF-Anwendung ausgegeben, dass die Verbindung verloren gegangen ist. Nach dem Wiederhochfahren baut die Verbindung jedoch selbst wieder auf und die

## 6 Validierung

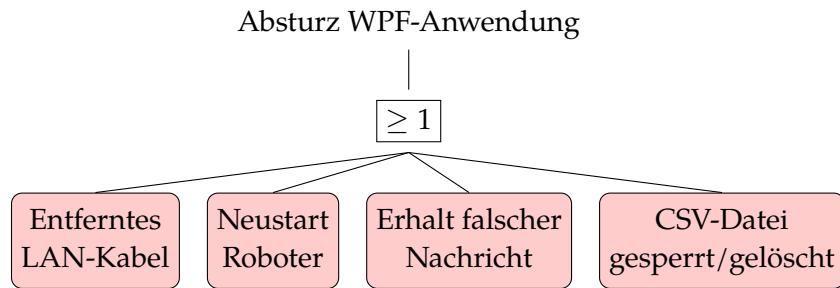


Abbildung 6.11: Fehlerbaumdiagramm Absturz WPF-Anwendung

Kommunikation fährt automatisch fort. Ebenso verursacht ein Löschen der CSV-Datei keinen Fehler, da in diesem Fall das System selbst eine neue Datei anlegt. Wird die CSV-Datei durch ein anderes System, wie z.B. Excel blockiert, hindert dies die Funktionsweise nicht, da sich die WPF-Anwendung eine Kopie der CSV-Datei anlegt. Eine Validierung des Systems ist hiermit nach bestem Gewissen sichergestellt. Dennoch ist darauf hinzuweisen, dass besondere unvorhergesehene Ereignisse nicht ausgeschlossen werden können. Die Robustheit von Software kann jedoch durch längerfristige Benutzung im Betrieb weiter verbessert werden.

## **7 Ausblick und Fazit**

Übertrag auf andere Hersteller

# Abbildungsverzeichnis

2.1	Übergang Parallelverdrahtung zu Feldbussen [4] . . . . .	5
2.2	Marktanteile Feldbusse und Industrial Ethernet [5] . . . . .	6
2.3	Telegramm-Aufbau Ethernet [4] . . . . .	8
2.4	Topologie: oben Standard-Feldbus, unten Ethernet basierender Feldbus [4]	9
2.5	Telegrabbearbeitung [9] . . . . .	10
2.6	Schichtenmodell TCP/IP [4] . . . . .	11
2.7	IP-Adresse Aufbau [4] . . . . .	12
2.8	Sequenzdiagramm Server-Client-Architektur [10] . . . . .	13
2.9	Stäubli TX2 - 90 <b>Staubli_HE</b> . . . . .	15
2.10	Stäubli CS9 Controller [19] . . . . .	17
2.11	CS9 Controller Computer-Einschub [19] . . . . .	17
2.12	Sequentielle Anordnung von Tasks [20] . . . . .	18
2.13	Anlagensteuerung Aufbau . . . . .	21
3.1	Gesamtanlage zu entwerfende Komponenten . . . . .	25
4.1	UML-Klassendiagramm Gesamte Anlage . . . . .	27
4.2	Sequenzdiagramm TCP/IP . . . . .	33
4.3	UML-Anwendungsdiagramm Roboterprogramm . . . . .	39
4.4	UML-Anwendungsdiagramm WPF-Anwendung . . . . .	42
4.5	MVC-Muster nach [27] . . . . .	43
4.6	MVVM [29] . . . . .	44
4.7	UML-Klassendiagramm WPF-Anwendung . . . . .	46
5.1	OxyPlot - Diagramm . . . . .	56
5.2	WPF-Anwendung Benutzeroberfläche Registerkarte Nachrichten . . . . .	57
5.3	WPF-Anwendung Benutzeroberfläche Registerkarte Graph . . . . .	57
6.1	Roboteraufbau mit Greifer und WPF-Anwendung . . . . .	60
6.2	Anzeige Systemereignisse . . . . .	61
6.3	Meldung bei Umteachen von Bewegungspunkten . . . . .	61
6.4	Visualisierung der Prozessdaten . . . . .	62
6.5	Visualisierung der integrierten Motormomente . . . . .	62
6.6	Binäre Werte zur Fehlerüberwachung . . . . .	63
6.7	Anzeige der verbundenen Roboter . . . . .	63
6.8	Fehlerbaumdiagramm Verbindungsabbruch . . . . .	64

*Abbildungsverzeichnis*

6.9 Fehlerbaumdiagramm Nachrichtenverzug . . . . .	64
6.10 Fehlerbaumdiagramm Absturz Roboterprogramm . . . . .	64
6.11 Fehlerbaumdiagramm Absturz WPF-Anwendung . . . . .	65

# **Tabellenverzeichnis**

1.1	Terminplanung mit Gantt-Diagramm . . . . .	4
2.1	CS9 Controller Computer-Einschub . . . . .	15
2.1	CS9 Controller Computer-Einschub . . . . .	16
3.1	Lastenheft . . . . .	23
3.1	Lastenheft . . . . .	24
4.1	Morphologischer Kasten Feldbus . . . . .	29
4.2	Morphologischer Kasten TCP/IP-Verbindung . . . . .	31
4.3	Vergleich Nachrichtenaufbau . . . . .	32
4.4	Übersicht Nachrichten TCP/IP . . . . .	33
4.5	Verfügbare Daten . . . . .	34
4.5	Verfügbare Daten . . . . .	35
4.5	Verfügbare Daten . . . . .	36
4.6	Aufbau Datentelegramm . . . . .	38
5.1	Vergleich Funktion zur Ermittlung Systemereignis . . . . .	49
5.2	Vergleich Logging-Möglichkeiten . . . . .	54

# Literatur

- [1] M. Broy und M. Kuhrmann, *Einführung in die Softwaretechnik*. Springer, 2021.
- [2] P. D.-I. A. Verl und P. D.-I. O. Riedel, *Leitfaden für die Anfertigung studentischer Arbeiten am ISW*. ISW Stuttgart, 2020.
- [3] I. Sommerville, *Software Engineering*. Pearson, 2018.
- [4] E. Hering, R. Martin, J. Gutekunst und J. Kempkes, *Elektrotechnik und Elektronik für Maschinenbauer*. Springer, 2017.
- [5] H. Networks, *Profinet jetzt vor EtherNet/IP - Industrial Ethernet legt weiter zu*, [Online, aufgerufen am 01. Dezember 2023], 2021.
- [6] F. Dopatka, „Ein Framework für echtzeitfähige Ethernet-Netzwerke in der Automatisierungstechnik mit variabler Kompatibilität zu Standard-Ethernet“, 2008.
- [7] K. GmbH, *Echtzeit-Ethernet-Kommunikation*, [Online, aufgerufen am 01. Dezember 2023], -.
- [8] W. Riggert, „Rechnernetze“, *Fachbuchverlag Leipzig*, 2002.
- [9] Beckhoff, „EtherCAT System-Dokumentation Version 5.6“, *Beckhoff Automation GmbH & Co. KG*, 2002.
- [10] Stäubli, *socket TCP-IP Technical documentation*. Stäubli International AG, 2019.
- [11] D. W. Kim, H.-D. Lee, C. W. de Silva und J.-W. Park, „Service-provider intelligent humanoid robot using TCP/IP and CORBA“, *International Journal of Control, Automation and Systems*, Jg. 14, S. 608–615, 2016.
- [12] C. Harnisch, *Netzwerktechnik*. mitp Verlags GmbH & Co. KG, 2009.
- [13] S. Wendzel, „Einführung in die Netzwerksicherheit“, in *IT-Sicherheit für TCP/IP- und IoT-Netzwerke: Grundlagen, Konzepte, Protokolle, Härtung*, Springer, 2018.
- [14] W. Babel, „Internet of Things und Industrie 4.0“, in *Internet of Things und Industrie 4.0*, Springer, 2023.
- [15] H. Stefan und J. Schreier, *Was ist OPC UA? Definition, Architektur und Anwendung*, [Online, aufgerufen am 18. Dezember 2023], 2019.
- [16] S. I. AG, *Stäubli - Über uns*, [Online, aufgerufen am 19. Dezember 2023].
- [17] S. I. AG, *Stäubli - Produktübersicht*, [Online, aufgerufen am 19. Dezember 2023].
- [18] Stäubli, *Roboterarm - Baureihe TX 90 Betriebsanleitung*. Stäubli International AG, 2019.

## Literatur

- [19] Stäubli, *Controller CS9 Betriebsanleitung*. Stäubli International AG, 2022.
- [20] Stäubli, *VAL 3 - Handbuch*. Stäubli International AG, 2022.
- [21] B. A. G.  
bibinitperiod C. KG, TC1000, *TwinCAT 3 ADS*, [Online, aufgerufen am 13. Januar 2023], -.
- [22] J. Kotz und C. Wenz, *C# und .NET 6–Grundlagen, Profiwissen und Rezepte*. Carl Hanser Verlag GmbH Co KG, 2022.
- [23] A. Troelsen und P. Japikse, *Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming*. Springer, 2022.
- [24] B. Groza und T.-L. Dragomir, „Using a cryptographic authentication protocol for the secure control of a robot over TCP/IP“, in *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, IEEE, Bd. 1, 2008, S. 184–189.
- [25] Beckhoff, „EtherCAT Bridge Terminal Documentation - Version 3.4“, *Beckhoff Automation GmbH & Co. KG*, 2002.
- [26] K.-H. Rau und T. Schuster, *Agile objektorientierte Software-Entwicklung*. Springer, 2015.
- [27] J. Goll, M. Koller und M. Watzko, *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer, 2023.
- [28] E. Gamma, *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2004.
- [29] Kraus, *Was bedeutet MVVM?*, [Online, aufgerufen am 13. Januar 2024], 2022.
- [30] Stäubli, *CS9 Fieldbus Technical Documentation*. Stäubli International AG, 2018.
- [31] N. Schonning, D. Pine und A. I. Aksoy, *Microsoft Übersicht über TCP*, [Online, aufgerufen am 26. Januar 2023], 2023.
- [32] S. Dugan, *C# logging: Best practices in 2023 with examples and tools*, [Online, aufgerufen am 26. Januar 2023], 2023.
- [33] NLog, *NLog*, [Online, aufgerufen am 26. Januar 2023], -.
- [34] P. Liggesmeyer, *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Springer Science & Business Media, 2009.
- [35] L. Pilorget und L. Pilorget, „Test-Definitionen & Begriffe“, *Testen von Informationssystemen: Integriertes und prozessorientiertes Testen*, S. 13–63, 2012.