

**Universität Stuttgart**

Institut für Steuerungstechnik  
der Werkzeugmaschinen und  
Fertigungseinrichtungen (ISW)



Studienarbeit

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

eingereicht von

*Lukas Schlotter*

aus Stuttgart

Studiengang

Prüfer

Betreuer

Eingereicht am

M. Sc. Mechatronik

Prof. Dr.-Ing. Alexander Verl

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

19. Januar 2024

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Zielsetzung / Aufgabenstellung . . . . .	1
1.3	Methodik und Vorgehensweise . . . . .	2
<b>2</b>	<b>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 VAL3 . . . . .	17
2.5	Anlagensteuerung . . . . .	19
2.6	WPF-Anwendung . . . . .	21
2.7	Fazit . . . . .	22
<b>3</b>	<b>Anforderungsdefinition</b>	<b>23</b>
<b>4</b>	<b>Konzeptionierung</b>	<b>26</b>
4.1	Feldbus-Verbindung . . . . .	26
4.2	TCP/IP-Verbindung . . . . .	29
4.3	Datenverwertung . . . . .	33
4.4	Stäubli Roboterprogramm . . . . .	38
4.5	WPF-Anwendung . . . . .	40
<b>5</b>	<b>Design und Implementierung</b>	<b>46</b>
5.1	Stäubli-Roboter in VAL3 . . . . .	46
5.1.1	EtherCAT . . . . .	46
5.1.2	TCP/IP . . . . .	46
5.2	WPF-Anwendung . . . . .	47
5.2.1	Architektur . . . . .	47
5.2.2	TCP/IP-Server . . . . .	47
5.2.3	Daten-Logger . . . . .	47

## *Inhaltsverzeichnis*

5.2.4	Graphen . . . . .	47
<b>6</b>	<b>Validierung</b>	<b>48</b>
<b>7</b>	<b>Ausblick und Fazit</b>	<b>49</b>
	<b>Abbildungsverzeichnis</b>	<b>50</b>
	<b>Tabellenverzeichnis</b>	<b>51</b>
	<b>Literatur</b>	<b>52</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>IEE</b>	Institute of Electrical and Electronics Engineers
<b>CSMA/CD</b>	Carrier Sense Multiple Access with Collision Detection
<b>FMMU</b>	Fieldbus Memory Management Unit

# 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, denen Roboter der „Stäubli-AG“ gerecht werden. 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. einem Förderband, der zentralen Steuerung durch die Speicherprogrammierbare Steuerung (SPS), sowie einer 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 / Aufgabenstellung

In der „robomotion GmbH“ sollen Stäubli-Roboter neu 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 muss Befehle von der Anlagen-SPS zum Roboter 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, sowie die Bereitstellung der Positionsdaten 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. Diese ist mit Hilfe von Transmission Control Protocol/Internet Protocol (TCP/IP) zu

## 1 Einleitung

konzeptionieren, entwerfen 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 erfolgend. 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 konzeptionieren und der dazugehörige Nachrichtenaufbau und Kommunikationsablauf festzulegen. Ebenso sollen die zur Verfügung stehenden Daten hinsichtlich ihres

## 1 Einleitung

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 vorallem der WPF-Anwendung erfolgen. Lösungsbausteine, wie Softwarebibliotheken müssen festgelegt und Lösungsansätze entworfen werden.
- **Softwareimplementierung:** Die zuvor konzeptionierte und entworfene Software muss in einen Source-Code überführt werden. Bei der Programmierung ist auf verständliche Kommentierung zu achten und eine Verwaltung soll mit Hilfe eines Source-Code-Verwaltungs-Programm erfolgen.
- **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.

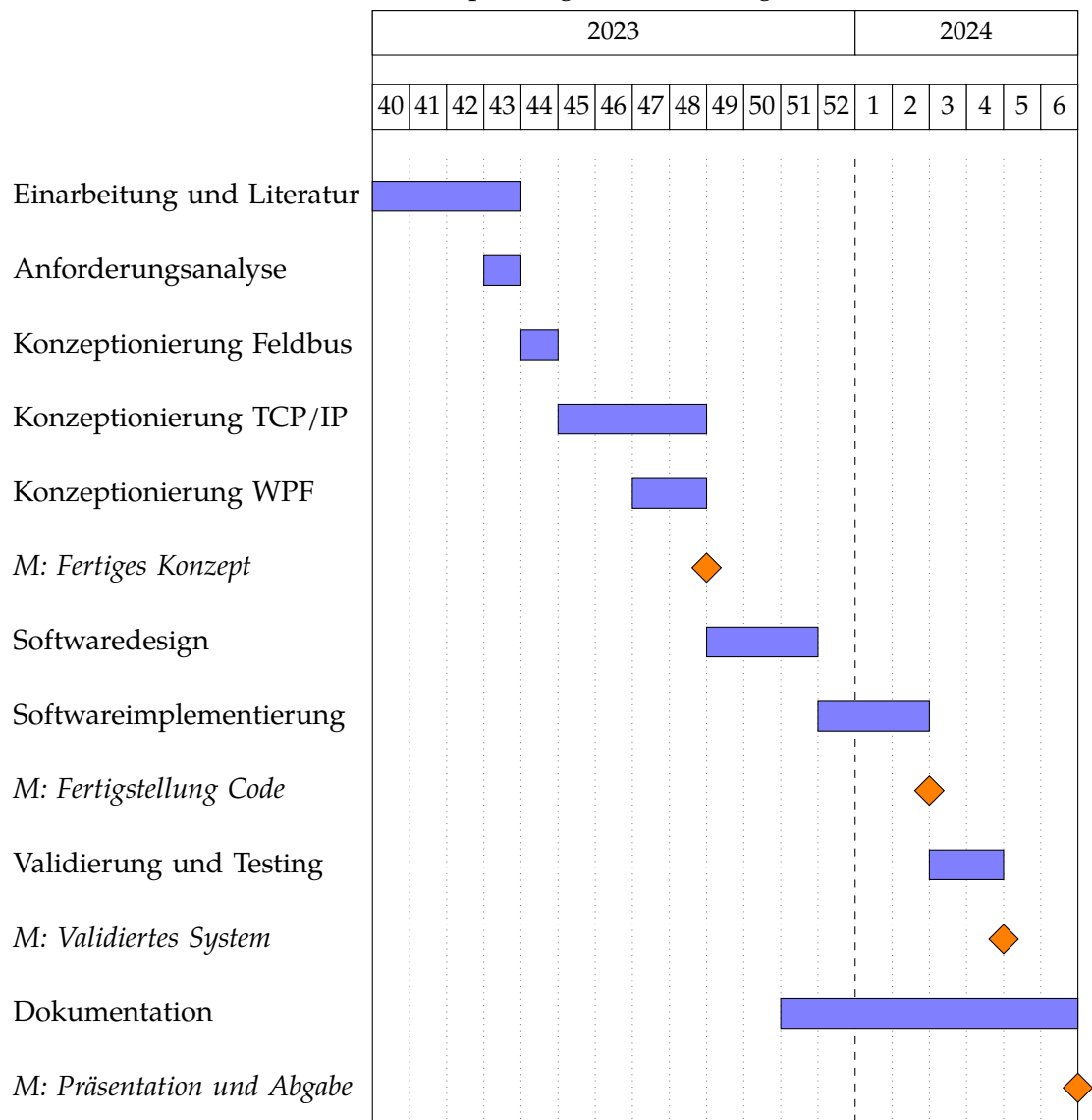
### Terminplanung

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 Zeitdauer 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]

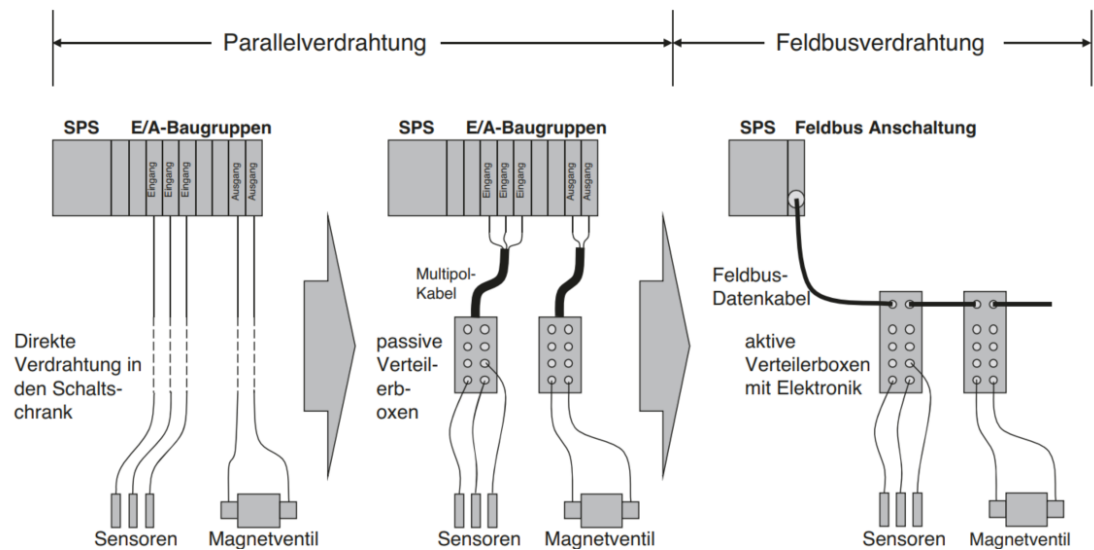


Abbildung 2.1: Übergang Parallelverdrahtung zu Feldbussen [4]

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]

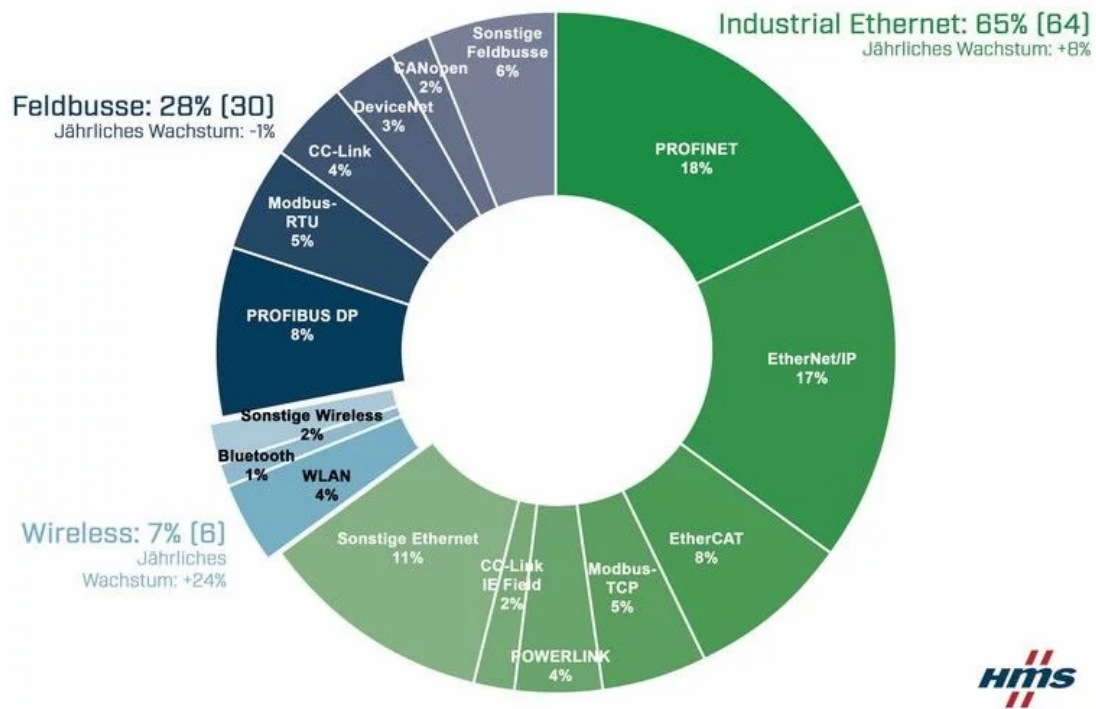


Abbildung 2.2: Marktanteile Feldbusse und Industrial Ethernet [5]

### 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 äußeren 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.

**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 Unternehmen „Mitsubishi“ dar. Verwaltet wird das Protokoll von einer Anwenderorganisation ähnlich zu Profibus. Die maximale Übertragungsrate beträgt 10 MBit/s und als Übertragungsmedium dient eine dreidrahtige 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 Institute of Electrical and Electronics Engineers 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 selbständig 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]

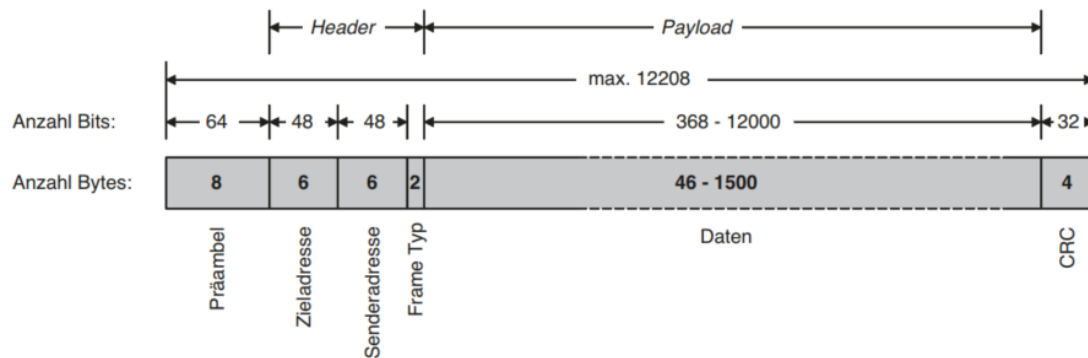


Abbildung 2.3: Telegramm-Aufbau Ethernet [4]

- **Carrier Sense:** Das Übertragungsmedium wird überwacht, damit es nicht durch ein anderes Gerät belegt ist. Erst nachdem das Medium als frei erkannt wird, erfolgt nach einer kurzen Wartezeit die Übertragung. Während des Sendens, wird das Medium weiterhin auf Kollisionen abgehört.
- **Multiple Access:** Es besagt, dass alle Geräte gleichberechtigt auf das Übertragungsmedium zugreifen können.
- **Collision Detection:** Wenn mehrere Geräte zur gleichen Zeit das Übertragungsmedium als frei erkennen kommt es zur Überlagerung, da mehrere Geräte gleichzeitig beginnen zu senden. Dies wird als Kollision bezeichnet und es kommt zur Signalverfälschung durch Überlagerung. Da jeder Sender das Übertragungsmedium während des Sendevorgangs überwacht wird die Kollision direkt erkannt. Dasjenige Gerät, welches die Kollision erkennt, informiert alle anderen Geräte mittels eines Signales über die aufgetretene Kollision und fordert diese zur Unterbrechung jeglicher Übertragung auf.  
Danach warten die Geräte eine zufällige 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 echtzeitfähig, was wie bereits in Kapitel 2.1.1 erwähnt, zu Problem bei Automatisierungsanlagen führen kann.

Um die Ethernet-Technologie dennoch in der Automatisierung zu nutzen, haben die Hersteller verschiedene Echtzeitprotokolle entwickelt, um Ethernet echtzeitfähig zu machen. Beispielsweise wird die Methodik CSMA/CD außer 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

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).

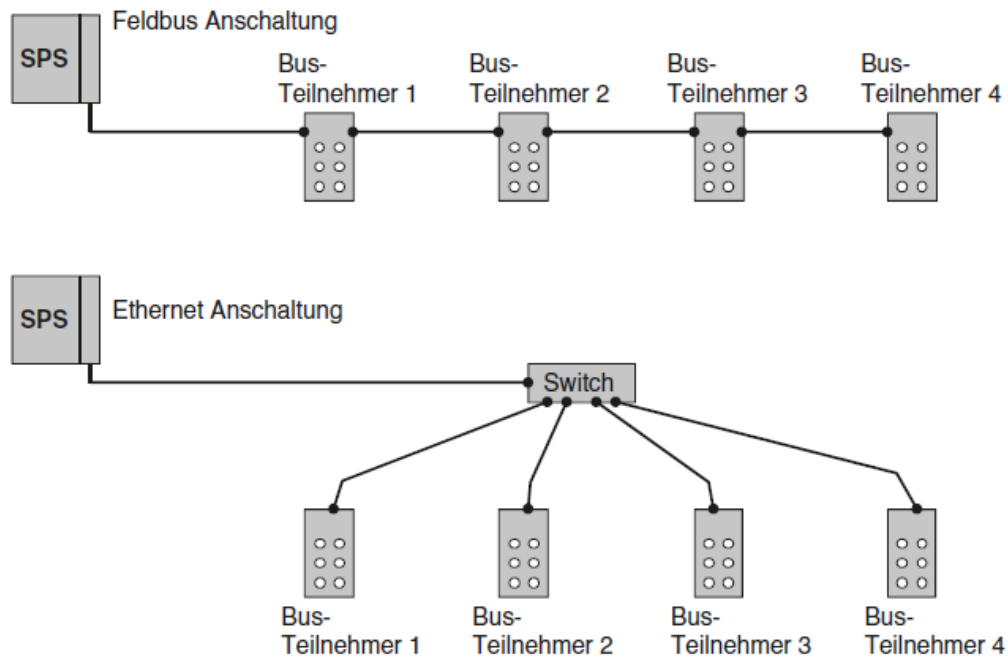


Abbildung 2.4: Topologie: oben Standard-Feldbus, unten Ethernet basierender Feldbus [4]

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

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).

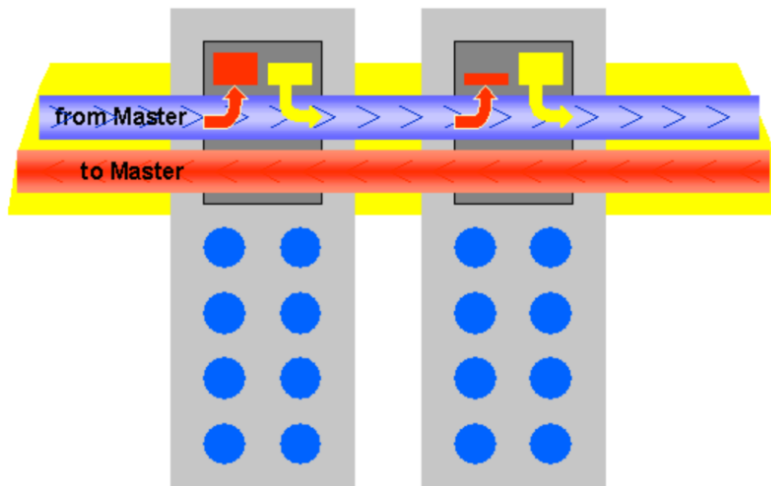


Abbildung 2.5: Telegrammbearbeitung [9]

Dabei wird auf dem gesamten Bus vom Ethernet-Protokoll nach Institute of Electrical and Electronics Engineers (IEEE) 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  $\mu$ 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 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]

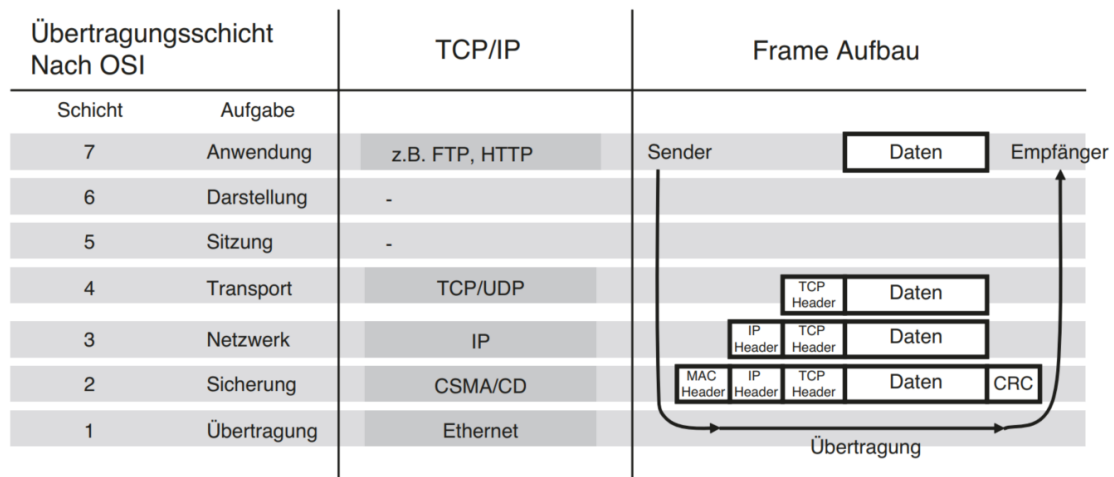


Abbildung 2.6: Schichtenmodell TCP/IP [4]

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 Internetprotokoll 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 Ausgabe 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 sind. Die Garantie für die korrekte Datenlieferung geschieht durch das TCP-Protokoll. Das IP-Protokoll verwendet IP-Adressen, um Netzwerkknoten 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]

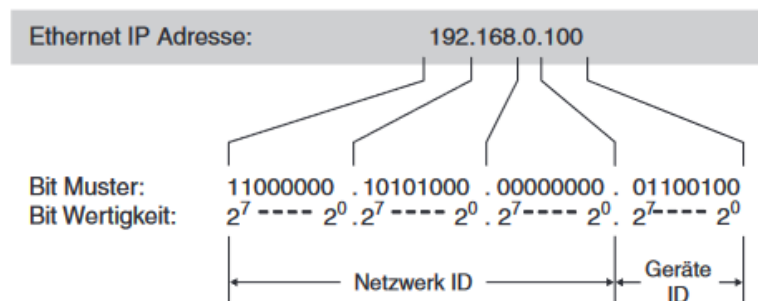


Abbildung 2.7: IP-Adresse Aufbau [4]

### Netzwerkarchitektur

Während Netzwerke in ihrer Topologie, wie z.B. Bus, oder Stern-Topologie unterschieden werden können, kann auch eine Aufteilung nach Architekturtyp 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

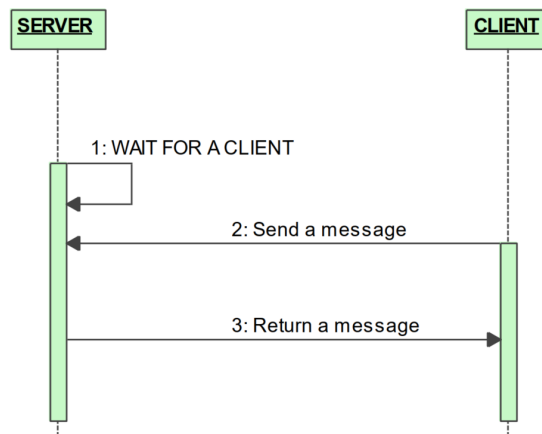


Abbildung 2.8: Sequenzdiagramm Server-Client-Architektur [10]

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 HMI (Human Interface) und Maschine. Es handelt sich hierbei um einen offenen und plattformunabhängigen Standard, welcher von der Open Platform Foundation (OPC) verwaltet wird. OPC UA basiert analog zu TCP/IP auf Ethernet und Internet. Es ist ebenso nicht echtzeitfähig. Das Kommunikationsprotokoll 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, 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 dem HMI (Human Interface) laufen und ein Roboter kann als OPC UA Client angebunden werden. Die Daten werden standardisiert und maschinenlesbar semantisch beschrieben, sodass herstellerübergreifende Kommunikation problemlos möglich ist. Sensordaten einer Maschine können dadurch an die Anlagensteuerung übertragen und visualisiert werden. OPC UA liefert dabei viele Vorteile: [15]



- **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 IoT und 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 Produktportfolio im Bereich Robotik umfasst Vier- und Sechachs-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 6 Achs-Roboter TX2-90 zum Einsatz (vgl. Abbildung 2.12). 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.

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: Staubli TX2 - 90 [18]

### 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).

## 2 Grundlagen und Stand der Technik



Abbildung 2.10: Stäubli CS9 Controller [19]

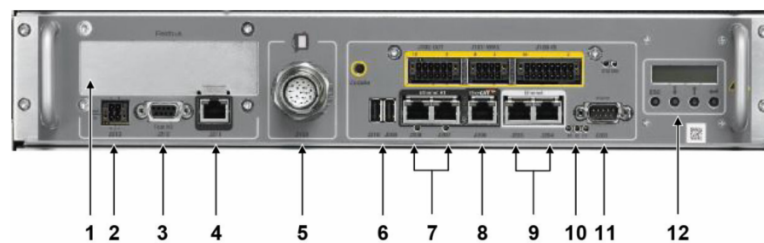


Abbildung 2.11: CS9 Controller Computer-Einschub [19]

Tabelle 2.1: CS9 Controller Computer-Einschub

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

### 2.4.2 Programmiersprache VAL3

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 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 Tasks 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 jedoch nur einen Prozess besitzt, ist es jedoch nur möglich eine Task auszuführen. Die augenscheinliche Parallität, wird dadurch erreicht, dass die Tasks sequentiell sehr schnell hintereinander ausgeführt werden. 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 asynchronen Tasks bietet VAL 3 ebenso synchrone Tasks.

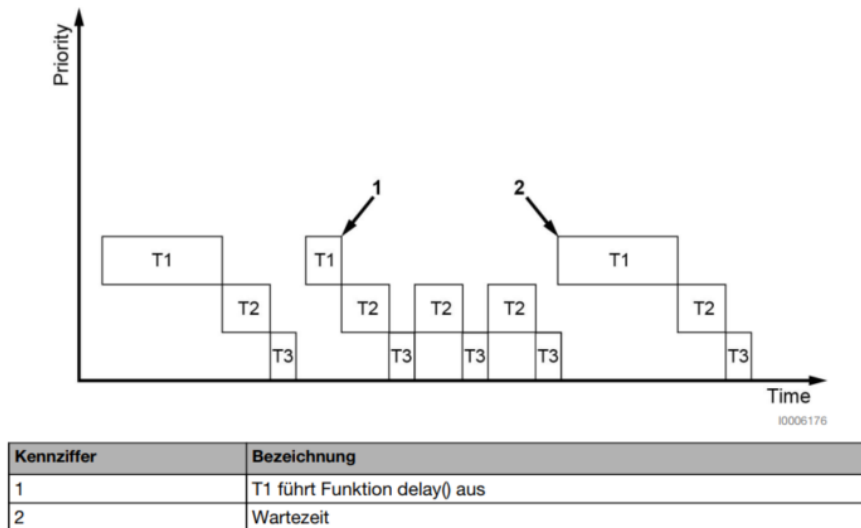


Abbildung 2.12: Sequentielle Anordnung von Tasks [20]

### 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 Bewegungsteuerung darstellen. Auf eine Erläuterung dieser wird verzichtet, da diese 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 Kommazahlen 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 diesen Datentyps standardmäßig mit einem leeren String ().

- **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 diesen 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. Zum Einsatz

- **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 lassen sich die Version der Hardware- und Software - Komponenten des Robotercontrollers 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 Vorangegangenen 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 jeder Anlagenautomatisierung ist eine Beckhoff-SPS, welche auf einem IPC läuft. Diese ist für die Ablauflogik und "Intelligenz" der Anlage zuständig. Auf anderen Bestandteilen, wie z.B. in einer 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-Steuerung 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 (Windows Presentation Foundation), des ".NETFrameworks. 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 Kommunikationsschnittstelle, spielt dies jedoch keine große Rolle, da sich alle IPCs in einem Netzwerk befinden. Da es sich um getrennte Programme handelt können diese alle als eigenständige Einheiten behandelt werden, auch wenn sie auf dem selben IPC laufen. Das Zusammenspiel dieser Komponenten ist nachfolgend dargestellt (vgl. Abbildung 2.13). Die Kommunikation zwischen Beckhoff-SPS und einer Robotersteuerung erfolgt bei robomotion über 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]

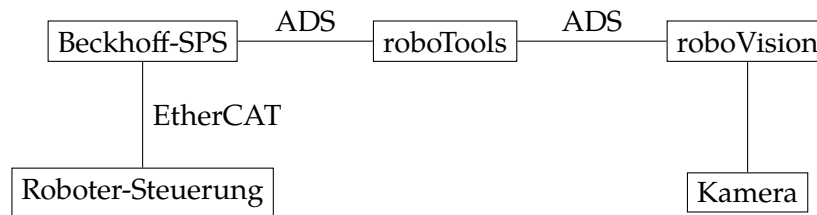


Abbildung 2.13: Anlagensteuerung Aufbau

## 2.6 WPF-Anwendung

Da bereits roboTools und roboVision auf WPF beruhen, ist WPF von großer Relevanz für diese Arbeit. Windows Presentation Foundation (WPF) stellt die Nachfolger von 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). [28]

```
// 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. [28] [29]

### C#

Das .NET-Framework ist offen für verschiedenste Programmiersprachen, jedoch ist die Verwendung von C# am häufigsten und kommt in den vorhandenen Anwendungen von robomotion zum Einsatz. C# ist eine Entwicklung speziell für .NET und kombinierte 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. [28] [29]

### Programmierungsumgebung

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. [28] Als Source-Code-Verwaltungs-Prorgamm wird unternehmensintern Äzure DevOpsëingesetzt. Daher wird auch im Rahmen dieser Arbeit auf Visual Studio 2022 in Kombination mit Azure DevOps zurückgegriffen.

### 2.7 Fazit

Feldbusse ermöglichen eine echtzeitfähige Kommunikation, was für sicherheitsrelevante 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 Industrienanwendungen, die nicht echtzeitfähig sein müssen zum Einsatz kommt. In [22] 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 [22] 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 Anlagensvisualisierung 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 VAL3 in der Entwicklungsumgebung SStäubli 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. Darüber hinaus kann das Lastenheft Rahmenbedingungen enthalten, wie z.B. zulässige Reaktionszeiten oder weitere Einschränkungen. Das Lastenheft wird in aller Regel vom Auftraggeber erstellt. 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 (K) / Muss (M)
1	Übertragung von Fehlermeldungen des Stäubli-Roboters an die Anlagenvisualisierung	M
2	Übertragung von beliebigen Daten des Stäubli-Roboters an die Anlagenvisualisierung	M
3	Übertragung von Daten z.B. Kameradaten von der Anlage an den Stäubli-Roboter	M
4	Ausgabe der Roboterfehler auf der Anlagenvisualisierung in verständlicher Form (ausformuliert, Fehlercode genügt nicht)	M
5	Funktionierende Feldbusverbindung zwischen Stäubli-Roboter und SPS	M
6	Funktionierende TCP/Ip Kommunikation zwischen Stäubli-Roboter und Anlagenvisualisierung	M

### 3 Anforderungsdefinition

Tabelle 3.1: Lastenheft

Nr.	Anforderung	Kann (K) / Muss (M)
7	Abspeichern der Fehler und der Daten in log-File	M
8	Fehlerfreie und prozesssichere Kommunikation	M
9	Einsatz von mehreren Stäubli-Robotern zeitgleich soll möglich sein	K
10	Vorgabe von der Anlagenvisualisierung, wie häufig Daten versendet werden sollen	K
11	Visualisierung der Daten	K
12	Verwertung der Daten, um Warnungen oder ähnliches auszugeben	K

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 auch 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 namens ADS zu robotools und somit auch zur entwerfenden WPF-Anwendung übertragen werden. Die SPS ist mit roboTools ebenso über ADS verbunden, was für diese Arbeit jedoch keine Relevanz hat. 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

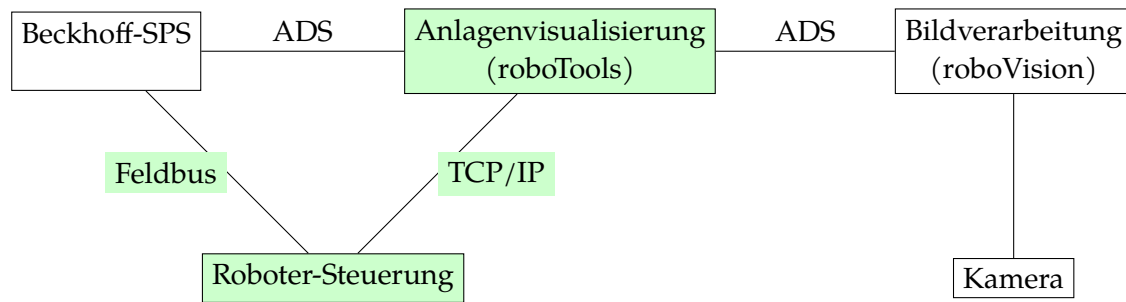


Abbildung 3.1: Gesamtanlage zu entwerfende Komponenten

## 4 Konzeptionierung

In diesem Kapitel soll das Gesamtsystem und seine Bestandteile 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) [3]. !UML-Diagramm. Anwendungsfälle definieren, Rahmenbedingungen identifizieren, Anforderungen aus Lastenheft verfeinern. Softwarekomponenten definieren und im Komponentendiagramm darstellen gemäß UML.

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 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.

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

#### 4 Konzeptionierung

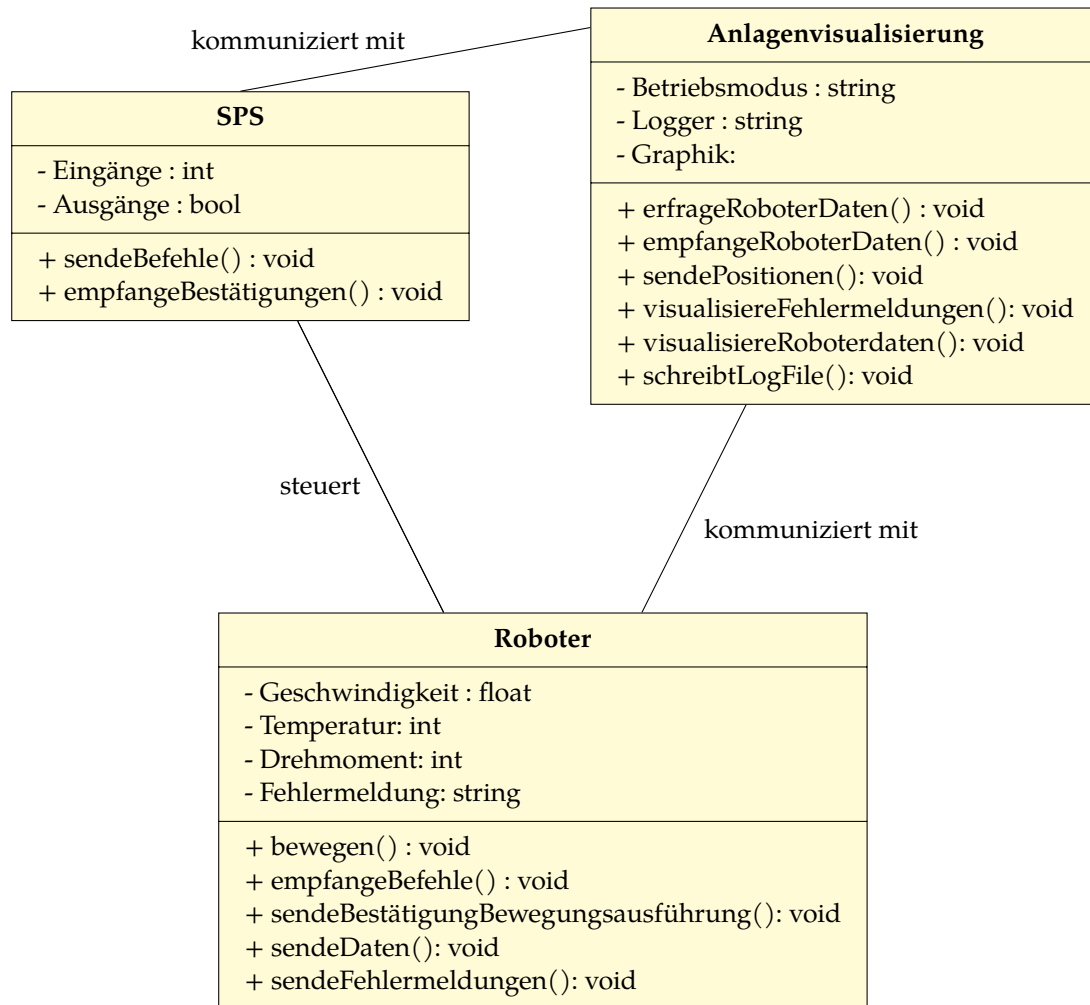


Abbildung 4.1: UML-Klassendiagramm Gesamte Anlage

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 Wissensbasis und Erfahrung aufgebaut. Bei schwerwiegenden 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 erhöht den Aufwand in Dokumentation und Einkauf. Daher wird versucht die Varianz möglichst gering zu halten und auf einen Standard, welcher bereits eingesetzt und verbreitet ist zu setzen.

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

Standardmäßig wird jedoch 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 einen Datenaustausch zwischen den einzelnen Strängen zu ermöglichen. Da jeder EtherCAT-Strang einen eigenen Master besitzt ermöglicht die Bridge den Einsatz von mehreren Masters. [23] 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, entfällt 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 ein Master, die anderen Stäubli-Roboter, wie auch 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 welcher weitere Komponenten, wie Roboter anderer Hersteller gesteuert werden können, weshalb die Wahl der SPS als Master sinnvoll ist.

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

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 Roboter 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. Da die Prozessdaten durchgängig anfallen, die Systemereignisse jedoch azyklisch lohnt sich eine Trennung. Die Prozessdaten wiederum können gebündelt gesendet werden oder getrennt werden in einzelne Nachrichten z.B. für die Temperaturen und die digitalen Werte. 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 versendet werden oder direkt bei Auftreten. 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 z.B. einmal pro Sekunde von der Anlagenvisualisierung an alle Clients (Stäubli-Roboter) gesandt werden. Diese sollen nach Erhalt der Anforderung ein Prozessdatentelegramm versenden. Der Einsatz von Datenanforderungstelegrammen bringt ein 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 vorgegeben Zeitspanne von x Milisekunden 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, sollten diese schnellstmöglichst an den Roboter übertragen werden. Aus diesem Grund erfolgt der Versand dieser Daten Ereignisstriggert, d.h. direkt nach Erhalt der Positionsdaten wird ein Telegramm an den entsprechenden Roboter gesandt. Die Entscheidungsfindung bei den eben genannten Auswahlmöglichkeiten ist nachfolgend in einem morphologischen Kasten dargestellt.

Tabelle 4.2: Morphologischer Kasten TCP/IP-Verbindung

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

### 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:** <Einzigartiger-Code>,<Sender-ID> <Daten>
- **Konzept 4:** <Nachrichten-ID>,<Sender-ID>,<Daten>,<CRC>
- **Konzept 5:** <Nachrichten-ID>,<Sender-ID>,<Daten Gleitkomma, dynamisch>

In Konzept 1,2,4,5 kommt eine Nachrichten-ID am Anfang, damit der Empfänger weiß, dass z.B. ein Datentelegramm und keine Systemereignisse versendet werden. 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 oder unterschiedliche Einstellungen in der Uhrzeit zwischen Roboter und Anlagenvisualisierung detektiert werden. Das Datum kann durch die Anlagenvisualisierung bei Erhalt der Nachricht bestimmt werden und muss daher nicht mitversendet werden.

#### 4 Konzeptionierung

Bei Konzept 4 wird zusätzlich ein CRC-Segment (Cyclic Redundancy Check) versendet. Im CRC-Segment wird eine Prüfsumme übermittelt, welche vom Sender aus der zu übermittelnden Nachricht berechnet. Der Empfänger berechnet die Prüfsumme aus der erhaltenen Nachricht und vergleicht sie mit dem CRC-Segment, um zu überprüfen, ob die Nachricht korrekt übermittelt wird. TCP/IP stellt jedoch bereits selbständig 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 einzartiger 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.

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

Tabelle 4.3: Vergleich Nachrichtenaufbau

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**

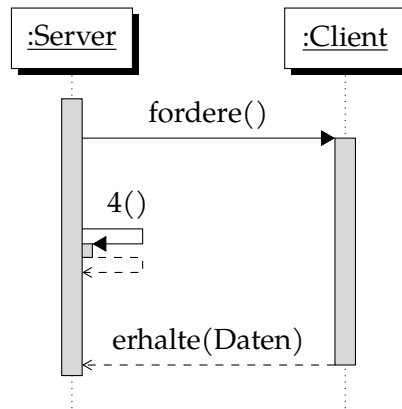
Der Aufbau dieser Nachricht orientiert sich am Datentelegramm. Statt den Daten wird hier der Text des Systemereignisses übertragen. Ergänzt wird dies von der Schwere des Systemereignisses. Dies ist eine Ganzzahl zwischen 0 (Info) bis 4 (kritischer Fehler).

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

##### **Nachrichtenaufbau Datenanforderungstelegramm**

Das Datenanforderungsdiagramm wird an alle Roboter versandt und ist wie folgt aufge-

#### 4 Konzeptionierung



baut:

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

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

Der Kommunikationsablauf lässt sich zusammenfassend in einem Sequenzdiagramm darstellen.

Das Zusammenspiel von Roboter, SPS und Windows-PC ist nachfolgend in einem UML-

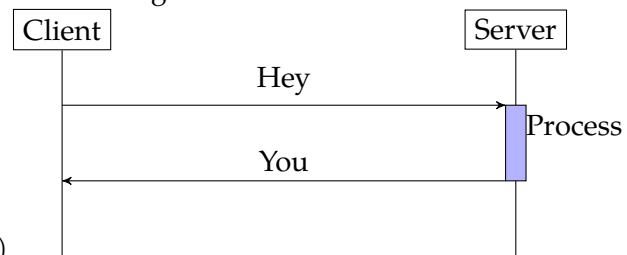


Diagramm dargestellt. (am besten vor Konzeptionierung)

### 4.3 Datenverwertung

Tabelle 4.4: 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)	-

#### 4 Konzeptionierung

Tabelle 4.4: Verfügbare Daten

Daten	Zugriff	Verwendung
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)	-
Axis brake feedback (1-6)	D (DsiIO)	-
Error on valves outputs	D (DsiIO)	Alarm
Error on brakes outputs	D (DsiIO)	Alarm
Error on safe digital inputs	D (DsiIO)	Alarm
DSI non-reduced brake supply voltage undershoot	D (DsiIO)	Alarm
DSI reduced brake supply voltage undershoot	D (DsiIO)	Alarm
DSI logic supply voltage undershoot	D (DsiIO)	Alarm
DSI overtemperature	D (DsiIO)	Alarm
DSI board tempereature	A (DsiIO)	Alarm, PM
Axis motor temperature (1-6)	A (DsiIO)	Alarm, PM
Axis encoder temperature (1-6)	A (DsiIO)	Alarm, PM
DSI state	A (DsiIO)	-
DSI error code	A? (DsiIO)	? Grau? Alarm
Arm operation counter	A (DsiIO)	-
safe Input state (0-7)	D (DsiIoSafe)	-
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)	-

Tabelle 4.4: Verfügbare Daten

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

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.
- **Datenarchivierung und Compliance:** Datenspeicherung für Schadensfall oder gesetzliche Gewährleistung
- **Trendanalyse:** Muster und Tendenzen in den Daten erkennen

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 "normallängesehen 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 overtemperature" erfasst. Error-Meldungen wie diese sollen als Alarm bzw. Fehlermeldung ausgegeben werden. Ebenso ist ein Dokumentieren sinnvoll, um bei Ausfällen 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. Eine genaue Eingrenzung ist auf Basis der gegebenen Daten nicht möglich, jedoch ist eine Warnung sinnvoll. Zur Ermittlung der Grenze, bei deren Überschreiten eine Warnung ausgegeben werden soll, können aufgezeichnete Daten unter Berücksichtigung von Schwankungen herangezogen werden. Der Schleppfehler muss sehr regelmäßig erfasst werden, um die kontinuierliche Bewegung best möglichst abzubilden.

##### **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

#### 4 Konzeptionierung

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 z.B. durch einen Crash nachzuweisen. Analog zum Schleppfehler müssen auch die Drehmomente sehr kontinuierlich erfasst werden.

##### **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 Schweregrade "Info", "Warning", "Error" und "Critical" unterteilen. Das Auslesen der Systemereignisse kann mit dem Befehl "getEvent" erfolgen. Diese sollen wie bereits bei der Konzeptionierung der TCP/IP-Verbindung festgelegt, beim Auftreten sofort über TCP/IP übertragen werden, .



## 4.4 Stäubli Roboterprogramm

Nachfolgend soll das 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

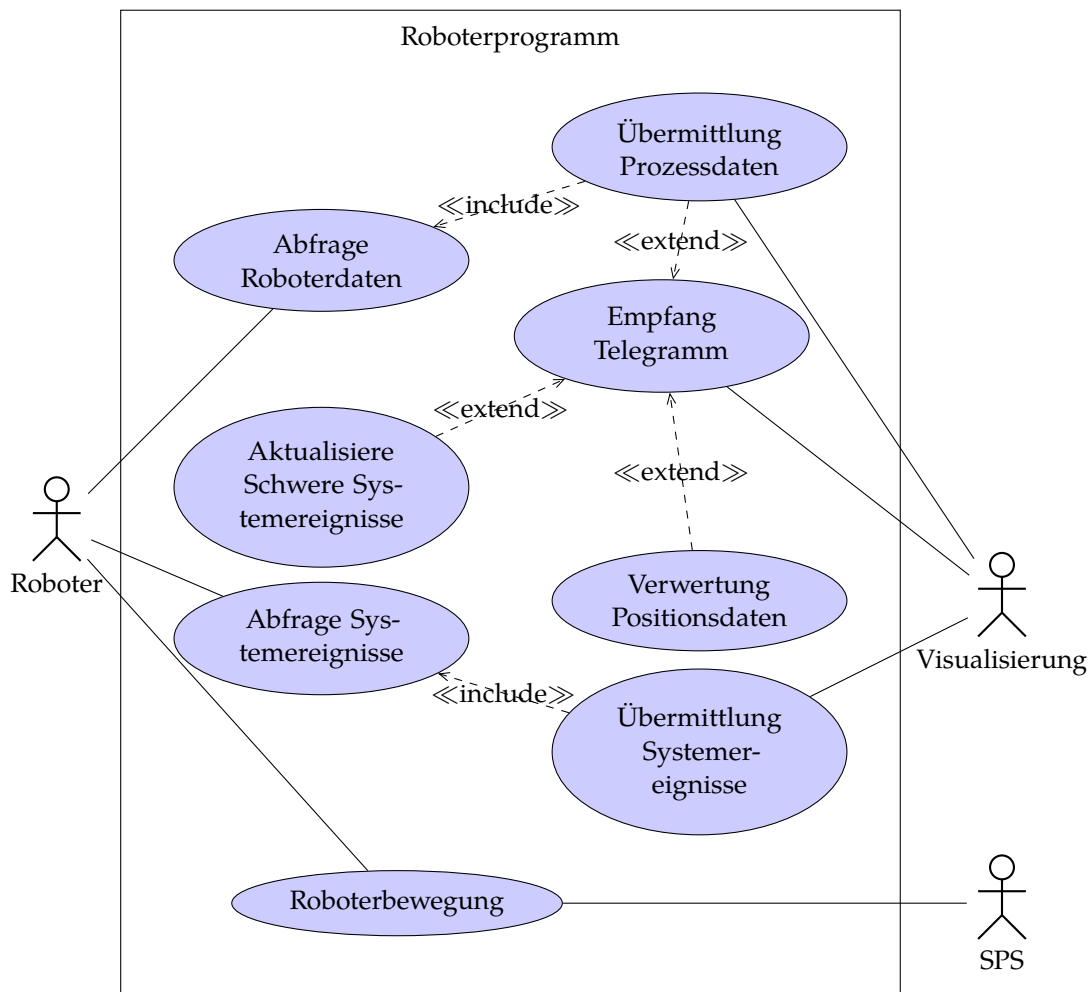


Abbildung 4.2: UML-Anwendungsdiagramm Roboterprogramm

von der Anlagenvisualisierung. Je nach Nachrichten-ID handelt es sich um eine Datenanforderungstelegramm (ID: 1) zur Anforderung von Prozessdaten oder es handelt sich um die Übermittlung von Positionsdaten von der Kamera (ID: 2). Alternativ kann über diese Nachricht auch die Schwere aktualisiert werden, ab welcher ein Systemereignis versandt werden soll (ID: 50). 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 3 Programme anzulegen, die sich in der Stäubli-spezifischen Programmiersprache 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.
- **Systemereignisse:** Hier erfolgt die Überprüfung ob neue Systemereignisse aufgetreten sind. In diesem Falle wird das Telegramm aufgebaut und das Systemereignis an die Anlagenvisualisierung versandt.
- **Bewegung:** 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.

Neben diesen 3 Programmen werden in dem standardmäßigen SStartProgramm, welches zum Programmstart ausgeführt wird, die oben genannten Tasks gestartet. Das SStop-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 4.3) 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 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=0) 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 (ID=8) 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 mit der Nachrichten-ID 50 unter 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 "robo-Vision"ermittelt und an die zu konfigurierende WPF-Anwendung weitergeleitet, die diese dann mit der Nachrichten-ID 3 versenden werden. 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 [24]. Die Wiederverwendung dieser Lösungen durch Muster vermeidet Fehler und vereinfacht die Kommunikation zwischen Fachleuten. Ebenso reduzieren sich Entwicklungskosten und die Flexibilität des Entwurfes steigt

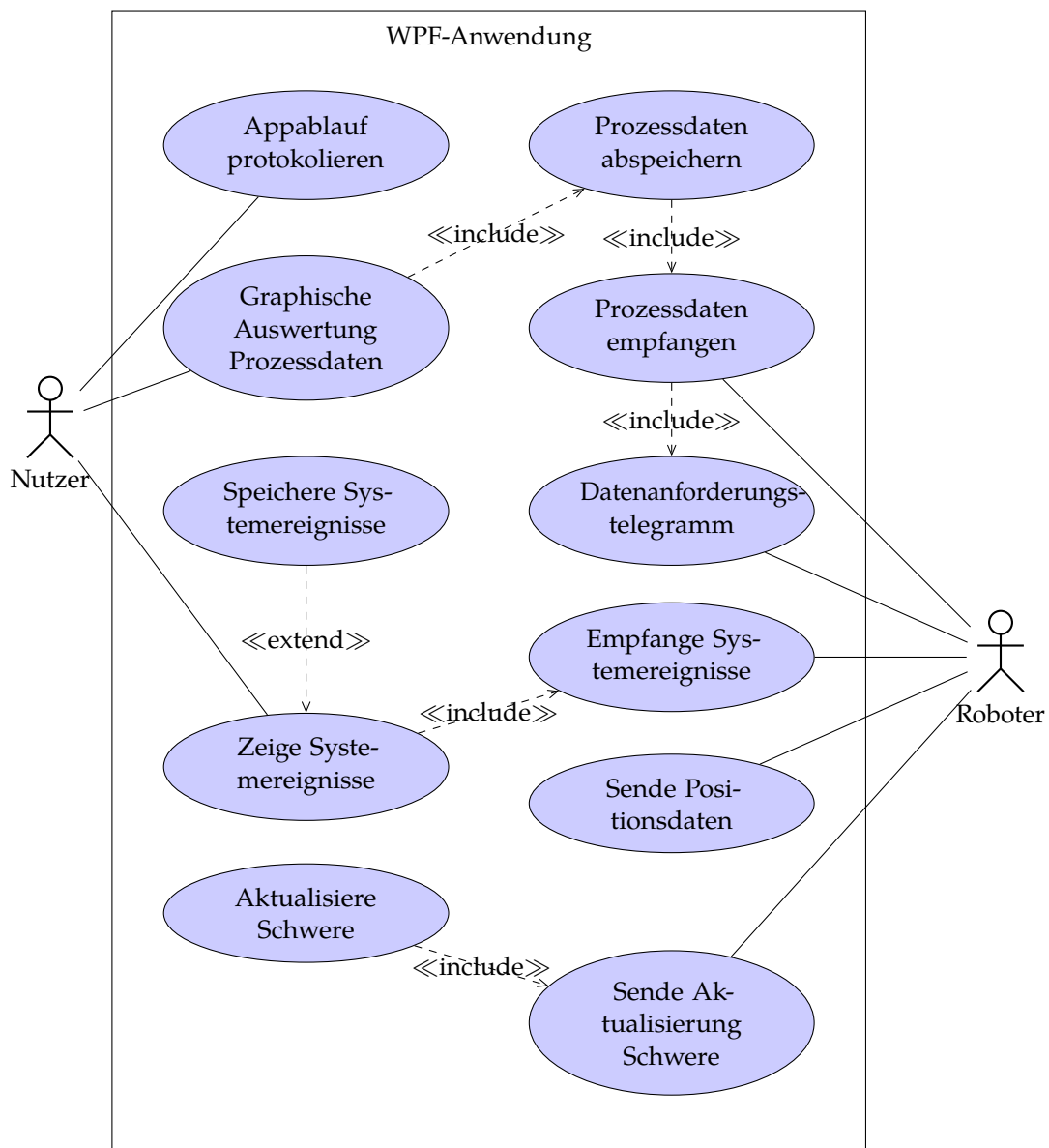


Abbildung 4.3: UML-Anwendungsdiagramm WPF-Anwendung

[25].

Nach [25] lassen sich Muster im Entwurfsprozess in drei Gruppen einteilen:

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

#### 4 Konzeptionierung

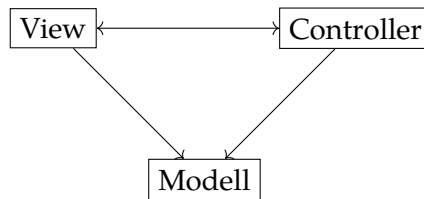


Abbildung 4.4: MVC-Muster nach [25]

- **Idiome:** Ein Idiom ist die Umsetzung eines abstrakten Musters in Form eines aus programmierten Entwurfsmusters. 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], [26], [25], [24]) 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 4.4).

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. [26] [25] 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. MVVM (Model-View-ViewModel) ist dabei eine Variation von MVC, kommt dessen Prinzip jedoch sehr nahe. [27] [28] [29] Dass MVVM bereits in der Anwendung roboTools zum Einsatz kommt unterstreicht, dass MVVM das geeignetste Architekturmuster für die zu entwerfende Applikation ist. MVVM setzt sich dabei aus den Komponenten View, ViewModel und Model zusammen (vgl. Abbildung 4.5)

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

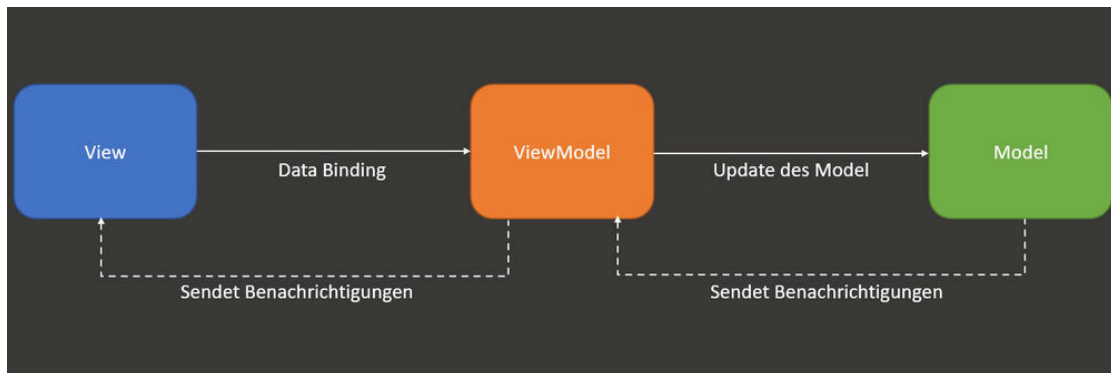


Abbildung 4.5: MVVM [27]

- **View:** In der View wird die Graphische Oberfläche definiert. Zur Darstellung von Daten nutzt Sie ViewModel, welches Informationen für die View zur Vergügung 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 passieren soll, geschehen soll implementiert. ViewModel holt sich von Model Daten, welche die View für Darstellungszwecke benötigt. [27] [29]

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

##### Entwurfsmuster

Nach [25] 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. [25]. 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. [25] 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 4.3) definiert wurden auf das

#### 4 Konzeptionierung

Entwurfsmuster aufgeteilt werden. Ebenso müssen die Klassen für die zu entwerfende WPF-Anwendung in C# definiert werden. Gemäß WVVM 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) verknüpft 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 4.6). Diese Klassen sind in der Umsetzung mit C# zu implementieren.

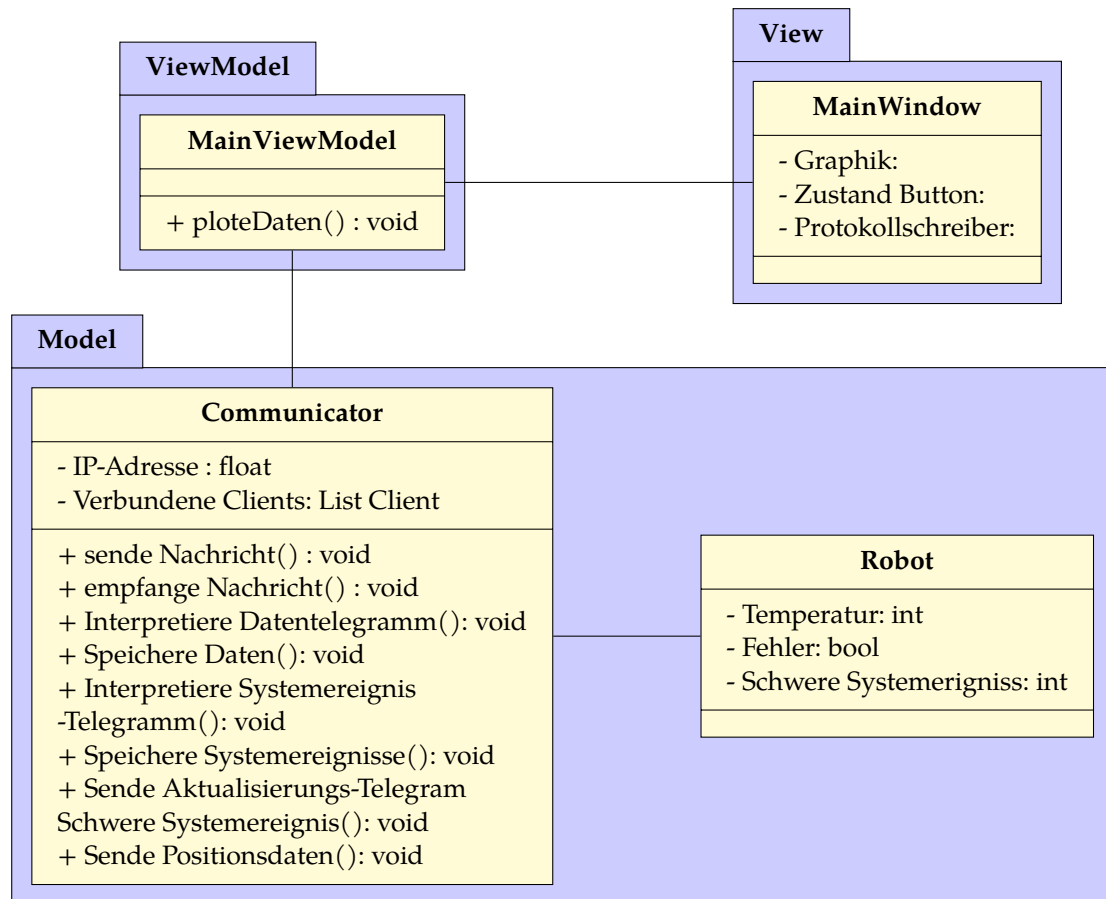


Abbildung 4.6: UML-Klassendiagramm WPF-Anwendung



## 5 Design und Implementierung

Design: Vergleich von Lösungsbausteinen (z.B. Vergleich Softwarebibliotheken) und Auswahl) auf Basis von Anforderungen und Recherche. Wissenschaftliche Bewertungskriterien nötig. Harvey-Diagramm.

Neben grundlegenden Entwurfsentscheidungen wird die Beschreibung einzelner Komponenten detailliert. Für komplexe Komponenten UML-Klassendiagramm Definition Schnittstelle zwischen den Komponenten und Kommunikation in Form von Sequenzdiagramm

Implementierung - Programmierung erfolgt gemäß den in Softwarespezifikation und entwurf festgelegten Randbedingungen, Kommentierung von Source Code ist essentiell. Gemäß bestehenden Standards (z.B. xdoc) - Generierung einer HTML-Hilfe aus den Kommentaren des Source-Codes z.B. durch Doxygen - Entwicklung erfolgt unter Git - Einheitliche Konventionen für das Schreiben des Quellcodes: Coding Conventions!

### 5.1 Stäubli-Roboter in VAL3

#### 5.1.1 EtherCAT

#### 5.1.2 TCP/IP

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. Bei 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 im Rahmen des Programmablaufs z.B. durch die Befehle `sioSet` und `sioGet` benötigt wird. Der Client versucht dann eine Verbindung zum Server aufzubauen. `usepackageffcode`

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

Mit dieser Funktion kann in VAL3 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.

Das Versenden von Nachrichten erfolgt über einen Byte-Array, das heißt durch die Aneinanderreihung mehrerer Bytes. Folglich muss die zu versendete Nachricht in einen Byte-Array umgewandelt werden und beim Empfangen muss der Byte-Array interpretiert 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 `nDataSize` kann die Anzahl der zu kodierenden Zeichen beschränkt werden.

`num fromBinary(num nDataByte[], num nDataSize, string sDataFormat, num& nValue[])`

Umgekehrt ermöglicht diese Funktion, einen empfangenen Byte-Array in numerische Werte zu konvertieren. Das Ergebnis im Datenformat `nDataFormat` wird in `nValue` gespeichert. Die Anzahl der zu decodierenden Bytes wird festgelegt durch `nDataSize`, wenn nicht alle Bytes des Eingangs-Array `nDataByte` decodiert werden sollen.

## 5.2 WPF-Anwendung

Wie in Stand der Technik. WPF, C, VisualStudio and DevOps.

### 5.2.1 Architektur

### 5.2.2 TCP/IP-Server

### 5.2.3 Daten-Logger

### 5.2.4 Graphen

## **6 Validierung**

## **7 Ausblick und Fazit**

# 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	Telegrammbearbeitung [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 [18] . . . . .	15
2.10	Stäubli CS9 Controller [19] . . . . .	16
2.11	CS9 Controller Computer-Einschub [19] . . . . .	16
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	UML-Anwendungsdiagramm Roboterprogramm . . . . .	38
4.3	UML-Anwendungsdiagramm WPF-Anwendung . . . . .	41
4.4	MVC-Muster nach [25] . . . . .	42
4.5	MVVM [27] . . . . .	43
4.6	UML-Klassendiagramm WPF-Anwendung . . . . .	45

# Tabellenverzeichnis

1.1	Terminplanung mit Gantt-Diagramm . . . . .	4
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	Verfügbare Daten . . . . .	33
4.4	Verfügbare Daten . . . . .	34
4.4	Verfügbare Daten . . . . .	35

# 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] 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.
- [23] Beckhoff, „EtherCAT Bridge Terminal Documentation - Version 3.4,“ *Beckhoff Automation GmbH & Co. KG*, 2002.
- [24] K.-H. Rau und T. Schuster, *Agile objektorientierte Software-Entwicklung*. Springer, 2015.
- [25] J. Goll, M. Koller und M. Watzko, *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer, 2023.
- [26] E. Gamma, *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2004.
- [27] Kraus, *Was bedeutet MVVM?* [Online, aufgerufen am 13. Januar 2024], 2022.
- [28] J. Kotz und C. Wenz, *C# und .NET 6–Grundlagen, Profiwissen und Rezepte*. Carl Hanser Verlag GmbH Co KG, 2022.
- [29] A. Troelsen und P. Japikse, *Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming*. Springer, 2022.