**MSE**MASTER OF SCIENCE
IN ENGINEERING

MSE-Master-Thesis

Skillbasierter Robotereinsatz für Industrieaufgaben

Studiengang	Master of Science in Engineering
Autor	Spatz Yannick
Dozent	Prof. Borer Melchior
Experte	Stucki Simon

Version 1.0 vom 15. Januar 2025

- Technik und Informatik
- Automation und Mechatronik

Abstract

Roboter sind in der Industrie unverzichtbar, doch hohe Beschaffungs- und Inbetriebnahmekosten sowie komplexe Programmierung erschweren ihre Integration. Monotone Aufgaben werden oft weiterhin manuell ausgeführt, da die Umstellung auf einen automatisierten Prozess mit hohem Aufwand verbunden ist. Diese Thesis untersucht, wie Roboter durch eine standardisierte Softwarestruktur einfacher und schneller programmiert werden können. Der Fokus liegt auf einem skill-basierten Ansatz.

Um die Wettbewerbsfähigkeit der Schweiz auf dem internationalen Markt zu sichern, gewinnen automatisierte Prozesse immer mehr an Bedeutung. Ein zentraler Bestandteil dieser Entwicklung sind Themen wie Flexibilität, Modularität und die einfache Einrichtung sowie Bedienung von Industrieanlagen. Diese Anforderungen betreffen nicht nur die Hardware, sondern stellen auch spezifische Ansprüche an die Software.

Basierend auf TwinCAT wurde eine Software-Struktur konzipiert, entwickelt und getestet, die eine einfachere und schnellere Programmierung von Industrieanlagen ermöglicht. Im Mittelpunkt steht dabei ein skill-basierter Ansatz, bei dem die Fähigkeiten der Anlagenkomponenten in übersichtliche und klar definierte Skills unterteilt werden

Die Software-Struktur basiert auf drei Aspekten:

Aufteilung von Prozess und Anlage:

Die Software gliedert sich in ein Prozessmodell und ein Anlagenmodell. Das Prozessmodell steuert die Abläufe und definiert die Prozessparameter, während das Anlagenmodell die Funktionalität der Systemkomponenten abbildet. Beide Modelle sind unabhängig voneinander, arbeiten jedoch über definierte Schnittstellen zusammen.

Die SPS als zentrales Element:

Die Steuerung sämtlicher Anlagenkomponenten erfolgt über die SPS, die damit das zentrale Element der Anlage darstellt. Jede Komponente wird im Anlagenmodell durch eine spezifische Objektklasse repräsentiert. Für die Entwicklung dieser Objektklassen wurde eine standardisierte Struktur implementiert, welche Konsistenz und Effizienz gewährleistet.

Aufteilung der Funktionalitäten:

Die Funktionalitäten der Anlage werden in sogenannte Skills unterteilt. Diese Skills bilden die Grundlage des Prozessmodells und repräsentieren die einzelnen Arbeitsschritte der Anlage. Auch für die Erstellung von Skills wurde eine standardisierte Struktur entwickelt, die sowohl die Interaktion mit anderen Elementen innerhalb des Prozessmodells als auch die Interaktion mit dem Anlagenmodell umfasst.

Inhaltsverzeichnis

Abstract	iii
1 Rahmen der Arbeit	1
1.1 Einleitung in die Thesis	1
1.2 Auftragsinterpretation	2
1.3 Zeitplan der Thesis	3
2 Einarbeitung in Thematik	5
2.1 Vorwissen aus Referenzprojekt	5
2.2 Annahmen	6
2.3 Situationsanalyse	6
2.3.1 Fragen bezüglich Referenzprojekt	6
2.3.2 Grundlagenfragen zur Thematik	8
2.3.3 Technische Grundfragen	11
2.4 Marktanalyse	15
2.5 System- und Kontextgrenze	22
2.6 Projektanforderungen	23
3 Vorgehen	25
3.1 Arbeitspakete	25
3.2 Gate-Plan	26
4 Anwendung und Aufbau	27
4.1 Definition der Anwendung	27
4.2 Definition der Anlagenkomponenten	28
4.3 Mechanischer Aufbau	30
5 Struktur der Software	33
5.1 Schnittstellen innerhalb der Software	34
5.2 Interaktion innerhalb der Software	36
6 Entwicklung der Skills	39
6.1 Kompetenzen von Skills	39
6.2 Definition von Skills für Anwendung	40
6.3 Definierung der Skill-Struktur	41
6.4 Konfiguration eines Skills	45
6.5 Interaktion zwischen Skill und Objekt	45
6.6 Umsetzung in TwinCAT	47
6.6.1 Allgemeine Struktur	47
6.6.2 Aufbau des Grund-Skills	48
6.6.3 Erarbeitete Skills	50
6.6.4 Momentaner Stand	51
7 Entwicklung des Anlagenmodells	53
7.1 Definition der Objekt-Struktur	53

7.2	Umsetzung in TwinCAT	56
7.2.1	Allgemeine Struktur	56
7.2.2	Aufbau des Grundobjektes	58
7.2.3	Erarbeitete Objekte	60
7.2.4	Schnittstelle zwischen Anlagenmodell und realer Anlage	73
8	Entwicklung des Prozessmodells	75
8.1	Struktur des Prozessmodells	75
8.1.1	Struktur eines Arbeitsplans nach Ansatz 1	77
8.1.2	Struktur von Sequenzen	77
8.2	Umsetzung in TwinCAT	79
8.2.1	Allgemeine Struktur	79
8.2.2	Aufbau einer Sequenz	80
8.2.3	Aufbau eines Arbeitsplanes	83
9	Auswertung	85
9.1	Aktueller Stand und Erkenntnisse	85
9.2	Vergleich mit Anforderungen	86
9.3	Weiterführende Arbeiten	87
10	Schlussfolgerung / Fazit	89
11	Anhang	91
	Literaturverzeichnis	95
	Abbildungsverzeichnis	97
	Tabellenverzeichnis	99
	Glossar	101

1 Rahmen der Arbeit

1.1 Einleitung in die Thesis

Das Institut für Intelligente Industrielle Systeme (I3S) befasste sich im Rahmen von Forschungsprojekten bereits mit der Thematik von skill-basierten Softwareansätzen. ROS war dabei ein zentrales Element der Entwicklung. Jedoch gab es weiterhin viele offene Fragen in Bezug auf den Einsatz von Skills.

Die Thesis wird mit der Idee initialisiert, neue Ansätze und Herangehensweisen zu finden und zu erarbeiten. Der Fokus liegt auf einer industrienahen Umsetzung und den Einsatz einer SPS. Bereits gewonnene Erfahrungen aus Projekten im Bereich der Verfahrenstechnik sollen in die Entwicklung neuer Ansätze einfließen. Dabei wird die Frage erarbeitet, welche Parallelen es zur Verführungstechnik gibt und wie diese für Skill-Anwendungen übernommen werden könnten.

Die Thesis setzt sich mit grundlegenden Fragen auseinander, wie:

- ▶ Was ist ein Skill und für welche Aufgaben eignet sich der Einsatz dieser.
- ▶ Wie kann ein Skill definiert werden.
- ▶ In welche Software-Struktur muss ein Skill eingebettet werden.
- ▶ Welche Möglichkeiten und Grenzen hat der Einsatz von Skills.

Das allgemeine Vorgehen innerhalb des Projektes folgt dabei den klassischen Entwicklungsphasen (Analyse / Konzipieren / Ausarbeitung / Auswertung). Zu Beginn wird eine ausführliche Analyse der Thematik, durchgeführt um alle themenrelevanten Fragen zu beantworten. Anhand dieser Analyse werden die diversen Arbeitspakete des Projektes definiert, welche jeweils mit einer Konzept- und Ausarbeitungsphase umgesetzt werden. Die Dokumentation dieser Thesis konzentriert sich darauf, den Entwicklungsprozess sowie die zugrunde liegenden Überlegungen detailliert darzustellen. Der Fokus liegt dabei auf der Beschreibung der methodischen Herangehensweise und der Schritte, die zur Erarbeitung der finalen Lösung geführt haben. Ziel ist es, transparent nachzuvollziehen, wie die Lösung entwickelt wurde und welche Entscheidungen den Prozess geprägt haben.

Mit dem Start des Semesters am 16.09.2024 wurde auch mit der Arbeit an dieser Thesis begonnen. Die Thesis umfasst dabei ca. 900 Arbeitsstunden. Das Projekt wird von Prof. Melchior Borer betreut und in enger Zusammenarbeit mit dem I3S umgesetzt. Dabei wird ein stetiger Austausch mit Prof. Dr. Norman Urs Baier, dem Leiter des Instituts für Intelligente Industrielle Systeme, angestrebt.

1.2 Auftragsinterpretation

Problembeschreibung:

Roboter finden in der Industrie eine immer breiter werdende Anwendung und sind aus vielen Prozessen nicht mehr wegzudenken. Ein grosser Beschaffungs- und Inbetriebnahmeaufwand machen es jedoch für viele Unternehmen schwierig, einen Roboter in ihre Prozesse zu integrieren. Dadurch werden zeitintensive und monotone Arbeiten oftmals noch manuell von Mitarbeitern durchgeführt. Eine grosse Hürde des Robotersystems ist der konventionelle Robotersoftwareansatz, welcher ein Grundverständnis für die Programmierung des Roboters voraussetzt. Für Veränderungen im Prozessablauf wird dadurch zwingend ein Programmierer benötigt.

Beschreibung des Auftrages:

Die Thesis beschäftigt sich mit der Frage, wie ein Roboter einfacher und standardisiert programmiert werden kann. Hierfür wird der Ansatz einer Rezeptursteuerung (bekannt aus Pharmaprozessen) analysiert. Dabei wird untersucht, ob ein solcher Ansatz für Roboteranwendungen geeignet ist und wie dieser eingesetzt werden kann. Ein zentrales Element ist dabei der skill-basierte Aufbau eines Roboterprogramms.

Die konkreten Ziele dieser Thesis sind wie folgt definiert:

- ▶ Analyse des Ansatzes einer Rezeptursteuerung für Roboteranwendungen.
- ▶ Analyse und Definierung von geeigneten Skills.
- ▶ Softwaremässige Umsetzung des skill-basierten Ansatzes.
- ▶ Reaktion der Software auf Fehler des Roboters während der Prozessdurchführung.

Auftragskontext:

Die Thesis findet als BFH-internes Projekt statt, welches aus dem Forschungsprojekt «ACROBA» entstanden ist. Das Ziel des ACROBA-Forschungsprojektes ist die Entwicklung und Demonstration neuartiger Konzepte für Roboterplattformen bezüglich agiler Fertigung.

Abgrenzungen:

Die Thesis beschäftigt sich nicht mit der Umsetzung einer konkreten industriellen Anwendung. Es ist jedoch möglich, dass Erkenntnisse aus dieser Thesis für zukünftige Automatisierungsprojekte verwendet werden können.

Projektorganisation:

Rolle	Wer	Status	Kontakt
Advisor	Prof. Borer Melchior	Dozent BFH	melchior.borer@bfh.ch
Experte	Stucki Simon	Externer Experte	/
Student	Yannick Spatz	Master-Student	yannick.spatz@bfh.ch

Tabelle 1.1: Projektorganisation

1.3 Zeitplan der Thesis

Der erstellte Zeitplan dient als erste grobe Orientierung und stellt eine vorläufige Einschätzung des Projektverlaufs dar. Da der genaue Umfang der Aufgabenstellung für diese Thesis nur schwer im Voraus vollständig abzuschätzen ist, kann der Projektrahmen entsprechend variieren. Insbesondere die Entwicklung der Software ist ein dynamischer und iterativer Prozess.

Daher ist der Zeitplan als flexibles Arbeitstool zu verstehen, das zwar eine klare zeitliche Struktur vorgibt, sich jedoch im Laufe des Projekts an veränderte Anforderungen und Erkenntnisse anpassen kann. Er soll einerseits helfen, die einzelnen Meilensteine im Blick zu behalten und andererseits als Leitfaden dienen, um trotz aufkommender Änderungen einen festen Rahmen für den Projektfortschritt zu gewährleisten. Die kontinuierliche Überprüfung und gegebenenfalls Anpassung des Zeitplans ist ein integraler Bestandteil dieses Prozesses, um sicherzustellen, dass das Projekt im vorgesehenen zeitlichen Rahmen bleibt.

	PW1 KW38	PW2 KW39	PW3 KW40	PW4 KW41	PW5 KW42	PW6 KW43	PW7 KW44	PW8 KW45	PW9 KW46	PW10 KW47	PW11 KW48	PW12 KW49	PW13 KW50	PW14 KW51	PW15 KW52	PW16 KW1	PW17 KW2	PW18 KW3	PW19 KW4	PW20 KW5
Analyse-Phase																				
Planungs-Phase																				
Umsetzungs-Phase																				
Auswertungs-Phase																				

Abbildung 1.1: Grobzeitplan der Thesis



Eine detaillierte Version des endgültigen Zeitplans wird im Anhang aufgeführt.

2 Einarbeitung in Thematik

2.1 Vorwissen aus Referenzprojekt

Im Rahmen des Master-Projekt 2 wurde eine chemische Reaktoranlage auf Basis von Twin-Cat automatisiert. Innerhalb dieses Projekts wurden verschiedene Themen analysiert und erarbeitet, die für diese Thesis von Relevanz sein könnten. Innerhalb der weiteren Dokumentation dieser Thesis wird das Master-Projekt 2 als «Referenzprojekt» bezeichnet. Konkret wurde im Master-Projekt 2 eine Softwarestruktur für eine Reaktorsystem, bestehend aus 7 Reaktoren, entwickelt. Die Reaktoren wurden über eine Rezeptursteuerung von einer Gesamtsystemebene gesteuert, welche mit allen Reaktoren über eine OPC-UA-Schnittstelle kommuniziert hat [1].

Umsetzung einer Rezeptursteuerung:

Die Reaktoren werden mit einer Rezeptursteuerung betrieben. Während dem Projekt konnte viel Wissen über die Strukturierung von TwinCAT-Programmen für eine Rezeptursteuerung gesammelt werden. Die korrekte Strukturierung ermöglicht es, anlagenunspezifische Abläufe zu definieren, welche flexibel mit Prozessparametern betrieben werden können.

Objektorientierte Struktur:

Das objektorientierte Programmieren kann ich vielen Programmiersprachen angewendet werden. Es bildet einen essentiellen Aspekt einer Rezeptursteuerung und der benötigten Struktur.

Kommunikation via OPC UA:

Der Betrieb der Anlage basiert auf der Kommunikation zwischen zwei Ebenen. Die Gesamtsystemebene (Supervisory-Level) definiert über ein Rezept die Prozessparameter. Diese werden via OPC-UA-Kommunikationsschnittstelle an die Reaktoreinheitsebene (Control-Level) übergeben, welche die entsprechenden Abläufe startet und schlussendlich mit den Sensoren und Aktoren (Field-Level) interagiert.

Eine ausführlichere Beschreibung der verschiedenen Themen findet sich in der Dokumentation des Master-Projekt 2 (Automatisierung: Chemische Reaktoranlage). Die grundlegende Theorie zu den einzelnen Themen wird in der Dokumentation dieser Thesis nicht behandelt.



Die Dokumentation des Master-Projekt 2 wird im Anhang beigelegt.

2.2 Annahmen

Obwohl die Thesis keine konkrete praktische Anwendung zum Ziel hat, soll der Auftrag industrienah umgesetzt werden. Die gewonnenen Erkenntnisse sollen potenziell für zukünftige industrielle Projekte nutzbar sein. Dafür wird folgende Annahme für die Erarbeitung der Thesis gemacht.

Als Programmierumgebung wird TwinCAT von Beckhoff verwendet, da mit früheren Projekten umfassende Kenntnisse und Erfahrungen gesammelt wurden. Dank der zahlreichen Zusatzpakete von Beckhoff können viele Funktionalitäten in einer einzigen Software kombiniert werden, wodurch der Einsatz mehrerer verschiedener Software-Tools entfällt.

Die definierte Annahme sollen den Rahmen der Thesis klarer abstecken, um sicherzustellen, dass innerhalb der zur Verfügung stehenden Zeit ein verwertbares Ergebnis erzielt wird. Trotz dieser Fokussierung wird darauf geachtet, dass die erarbeiteten Strukturen und gewonnenen Erkenntnisse auch auf andere Systeme, abgesehen von Beckhoff, übertragbar sind. So bleibt die Arbeit nicht nur auf ein spezifisches System beschränkt, sondern bietet Potenzial für eine breitere Anwendung in verschiedenen industriellen Umgebungen.

2.3 Situationsanalyse

2.3.1 Fragen bezüglich Referenzprojekt

Frage 1.1: Was ist ein chargenorientierter Ansatz nach ANSI/ISA-88

Der internationale Standard ANSI/ISA-88 beschreibt Methoden und Strukturen zur Entwurfung von Chargensteuerungen in der Pharma- und Chemieindustrie. Eine Charge ist in diesem Kontext eine definierte Produktionsmenge, welche innerhalb eines Produktionsablaufes prozessiert wird. Das Rezept beschreibt dabei den Herstellungsweg einer Charge und definiert die notwendigen Prozessinformationen [2].

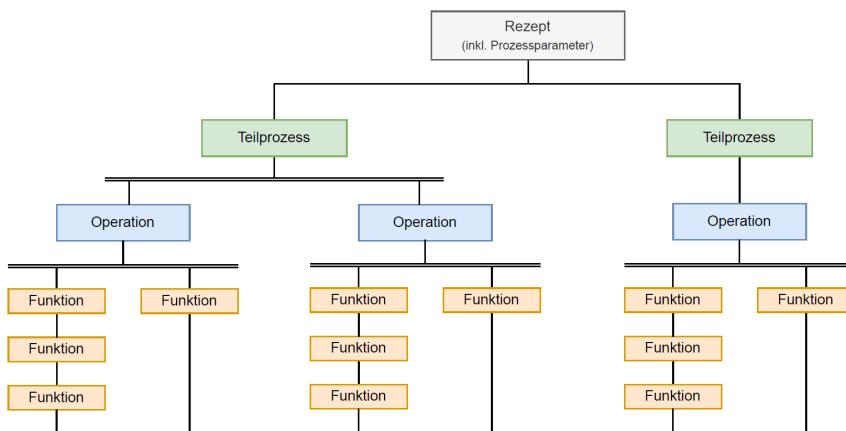


Abbildung 2.1: Prozessstruktur von ISA-88

Das Rezept besteht aus unterschiedlichen Teilprozessen, welche aus Operationen zusammengesetzt sind (Abb. 2.1). Auf der untersten Stufen befinden sich die Funktionen. Diese stellen die Grundfunktionalitäten der Anlagenkomponenten dar, z.B. das Öffnen und Schliessen eines Ventiles. Alle Prozesse können seriell oder parallel durchgeführt werden. Durch diese Struktur wird der Prozess nicht mit einem fest definierten Ablauf gesteuert, sondern durch die im Rezept definierten Schritte und Parameter. Dies ermöglicht ein flexibles Einsetzen der Anlage und deren Ressourcen. Sofern es die Infrastruktur der Anlage zulässt, kann jedes Rezept gefahren werden. Dies reduziert Stillstandszeiten der Anlage.

und macht diese effizienter. Dieses Prinzip kann so erweitert werden, dass das System auch selbstständig die Anlagenelemente definiert, welche für das Prozessieren der Charge verwendet werden. Hierbei spricht man dann von einer Rezeptur und nicht mehr von einem Rezept. Besteht das System z.B. aus mehreren chemischen Reaktoren, entscheidet die Rezeptur selbstständig, welcher der Reaktoren für den Prozess eingesetzt wird. Dies ermöglicht einen noch flexibleren und selbständigeren Prozess.

Frage 1.2: Wo und wie wird ein chargenorientierter Ansatz eingesetzt

Eine Chargensteuerung nach ANSI/ISA-88 wird hauptsächlich für die Verfahrenstechnik eingesetzt. In der Verfahrenstechnik werden verschiedene Produkte oft auf derselben Anlage hergestellt. Der Einsatz einer Rezeptursteuerung nach ANSI/ISA-88 vereinfacht und standardisiert das Prozessieren von unterschiedlichen Produkten somit erheblich. Für ein neues Produkt muss nur das Rezept angepasst werden, sofern alle Rezepte auf den definierten Funktionen und Operationen aufbauen.

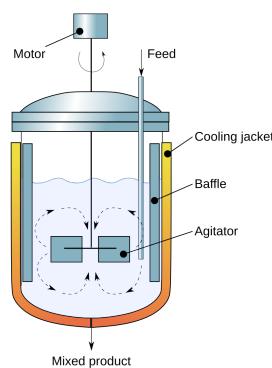


Abbildung 2.2: Reaktorbeispiel

Der dargestellte Reaktor (Abb. 2.2) hat z.B. folgende Funktionen:

- Befüllen / Kühlen / Mischen / Warten / Abfüllen

Aus diesen Funktionen lassen sich verschiedene Operationen definieren, welche wiederum vom Rezept verwendet werden können, um ein bestimmtes Produkt zu prozessieren. Ein zentraler Aspekt für die Flexibilität und Modularität von Chargensteuerungen nach ANSI/ISA-88 ist die Trennung von Prozessmodell und Anlagenmodell (Abb. 2.3) [3]. Das Prozessmodell beinhaltet das Rezept, die Operationen und Funktionen. Die Rezeptlogik ist unabhängig von den spezifischen Elementen, die in der Anlage verwendet werden. Es beschreibt lediglich, was getan werden soll, um das gewünschte Ergebnis zu erzielen, nicht wie oder mit welchen Elementen es durchgeführt wird. Das Anlagenmodell ist für das Ansteuern der Anlagenkomponenten zuständig. Die Trennung von Prozess und Anlage ermöglicht es, dass ein Rezept auf unterschiedlichen Anlagen ausgeführt werden kann.

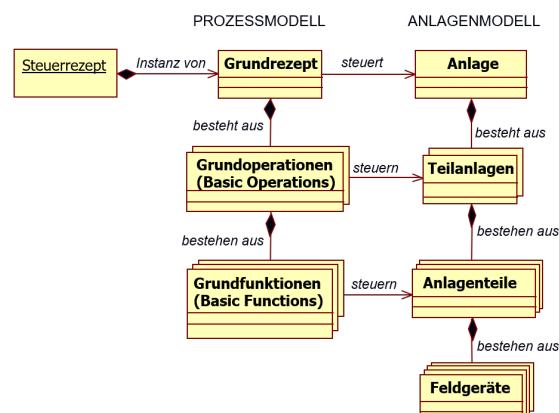


Abbildung 2.3: Prozess- und Anlagenmodell

Frage 1.3: Wie wird eine chargenorientierte Struktur in TwinCAT umgesetzt

Als Referenz für den Aufbau einer Rezeptursteuerung auf Basis von Codesys (TwinCAT basiert auf Codesys) dient das Buch «Speicherprogrammierbare Steuerungen in der Industrie 4.0» von Matthias Seitz.

Entsprechend wird auch in TwinCAT zwischen dem Prozessmodell und dem Anlagenmodell unterschieden. Das Prozessmodell wird mittels Ablaufsprache (SFC) umgesetzt. Innerhalb des Anlagenmodells werden die Objektklassen für die verschiedenen Feldgeräte der Anlage definiert. Diese müssen zwingend objektorientiert aufgebaut werden. Die Objektklassen werden in der Anlage instanziiert. Die instanzierten Objekte werden verwendet, um die Schnittstelle zu den Funktionen zu bilden.

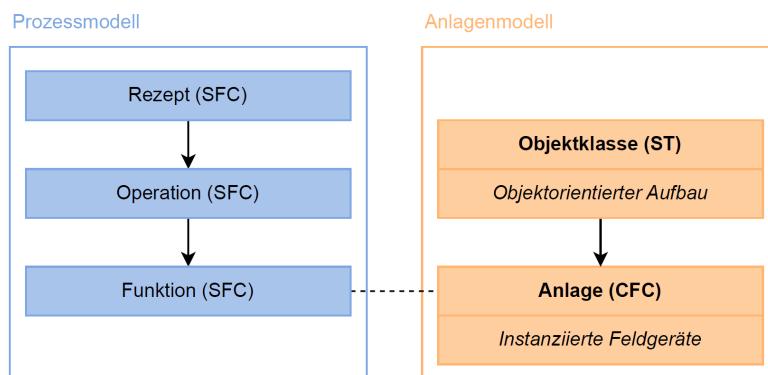


Abbildung 2.4: Softwarestruktur von ISA/ANSI-88

Die dargestellte Struktur (Abb. 2.4) stellt ein stark vereinfachtes Modell dar. Für eine detaillierte Beschreibung der definierten TwinCAT-Struktur, wird auf die Dokumentation des Master-Projekt 2 verwiesen [1].

2.3.2 Grundlagenfragen zur Thematik

Frage 2.1: Was versteht man unter einem skill-basierten Ansatz

Ein skill-basierter Ansatz basiert auf der Fähigkeit von Maschinen bestimmte Funktionen auszuführen. Der Skill beschreibt eine konkrete Grundfunktion des Systems und stellt die tiefste Abstraktionsebene der Systemfunktion dar. Die Idee hinter dem skill-basierten Ansatz ist, dass Abläufe auf einfache Skills heruntergebrochen werden können. Durch die Verkettung von Skills, mit der Zuweisung von Funktionsparametern, können komplexe Funktionalitäten erstellt werden. Das Ziel dieses Ansatzes ist die Vereinfachung der Programmierung eines Systems (z.B. Roboter). In einem traditionellen, nicht-skill-basierten Ansatz würde der gesamte Prozess in einem einzigen, umfassenden Programm geschrieben werden. Das Programm würde alle Schritte in fester Reihenfolge definieren. Änderungen an einem Prozess würden dann bedeuten, dass das gesamte Programm geändert werden muss. Dagegen erfordern Änderungen beim skill-basierten Ansatz nur Änderungen in der Reihenfolge oder Kombination der verwendeten Skills, nicht im gesamten Ablauf.

Frage 2.2: Vergleich zwischen einem chargenorientierten und skill-basierten Ansatz

Beide Ansätze besitzen eine vergleichbare Struktur. Die Funktionalitäten eines Systems werden auf der untersten Abstraktionsstufe auf Funktionen bzw. Skills heruntergebrochen. Diese können anschliessend verwendet werden, um komplexere Abläufe zu definieren. Beim skillbasierten Ansatz ist der Einsatz einer Rezeptur mit definierter Chargen-

grösse weniger sinnvoll. Die Verwendung eines Arbeitsplanes, welcher definierte Schritte vorgibt, eignet sich besser. Für den skill-basierten Ansatz werden für den Rahmen dieser Arbeit folgende Begriffe (Abb. 2.5) verwendet, um die verschiedenen Strukturelemente zu beschreiben.

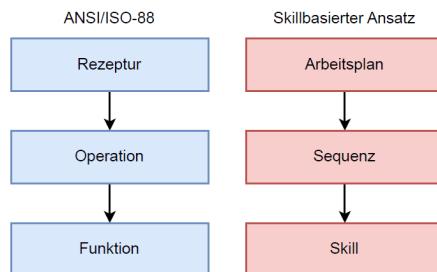


Abbildung 2.5: Softwarestrukturvergleich

Frage 2.3: Was sind die Vorteile eines skill-basierten Ansatzes

Komplizierte Abläufe werden in einfache Teilfunktionen aufgeteilt. Die gesamte Funktionalität des Systems wird entsprechend innerhalb der Skills umgesetzt. Das System kann dadurch einfacher und zugänglicher programmiert werden, da Skills modular und flexibel kombiniert und mit entsprechenden Parametern versehen werden können. Dies erhöht auch die Verständlichkeit des Programms für Außenstehende oder bei nachträglichen Anpassungen. Bei funktionalen Anpassungen werden nur die entsprechenden Skills bearbeitet und nicht das komplette Programm.

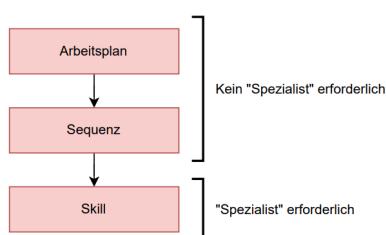


Abbildung 2.6: Kompetenzaufteilung

Da die Skills die Schnittstelle zu den Systemkomponenten darstellen, wird nur dort ein fundiertes Wissen über die Programmierung der Komponenten vorausgesetzt. Auf den oberen Stufen (Sequenz & Arbeitsplan) wird kein fundiertes Wissen über die Komponenten benötigt. Die Kompetenzen können bei einem skill-basierten Ansatz somit besser aufgeteilt werden (Abb. 2.6).

Ein System, welches skill-basiert aufgebaut wurde, ist einfach und schnell erweiterbar mit neuen Skills, wenn z.B. neue Produkte auf dem System prozessiert werden sollen. Es ist auch möglich einen Standard für die

Schnittstellen der Skills zu definieren. Dabei wird klar beschrieben, welche Inputs ein Skill benötigt und welche Outputs dieser liefern muss.

Sequenzen und Arbeitspläne können dadurch anlagenunabhängig umgesetzt werden, da Schnittstellen immer dieselben bleiben. Somit lassen sich Prozesse (Sequenzen & Arbeitspläne) bereits definieren, ohne dass die Komponenten im System bekannt sind. Dadurch können auch dynamische Anwendungen realisiert werden. Dies wird anhand einer Beispielsituation erklärt. Ein System besteht aus einem Arbeitsbereich, auf welchem Aufgabe A und B ausgeführt werden soll (Abb. 2.7). Zur Ausführung dieser Aufgaben besitzt das System zwei Roboter, welche über eine SPS gesteuert werden. Alle Arbeitsprozesse wurden in einzelne Skills aufgeteilt. Durch den anlagenunabhängigen Arbeitsplan kann das System selbst definieren, welcher Robo-

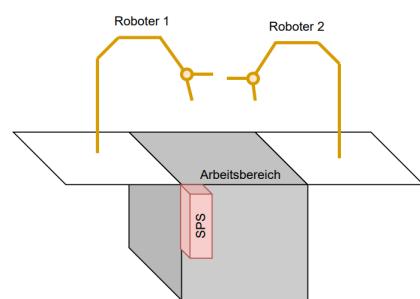


Abbildung 2.7: Beispieldiagramm

ter Aufgabe A oder B ausführt. Wenn nun Roboter 1 mit Aufgabe A beschäftigt ist, kann Roboter 2 Aufgabe B ausführen oder umgekehrt. Der dynamische Prozess führt zu einer hohen Flexibilität. Roboter 1 und 2 müssen auch nicht vom selben Hersteller kommen, sofern die Schnittstellen des Skills korrekt definiert wurden, spielt dies für den Prozess keine Rolle.

Frage 2.4: Was sind die Herausforderungen eines skill-basierten Ansatzes

Die grösste Herausforderung ist das Herunterbrechen der gesamten Funktionalität einer Roboteranwendung auf Skills. Eine Roboteranwendung kann sehr komplexe Prozesse realisieren, in welchen viele unterschiedliche Akteure und Sensoren miteinander interagieren. Die umfangreichen Möglichkeiten eines solchen Systems sollen nicht durch die Verwendung von Skills eingeschränkt werden. Bei einem definierten Standard, müssen die Systemkomponenten in der Lage sein, die definierten Schnittstellen zur Verfügung zu stellen. Dies wäre ein wichtiger Aspekt einer anlagenunabhängigen Prozessdefinierung auf Stufe der Sequenzen und Arbeitspläne. Zusätzlich soll das Arbeiten mit einem skill-basierten Ansatz nicht aufwändiger werden als das konventionelle Programmieren der Roboteranwendung.

Frage 2.5: Wie wird eine skill-basierte Struktur in TwinCAT umgesetzt

Grundsätzlich kann die Struktur aufgebaut werden wie bei einem chargenorientierten Ansatz. Man hat wieder zwei Modelle, das Prozessmodell und das Anlagemodell. Das Prozessmodell implementiert die Skills, aus welchen die Sequenzen und Arbeitspläne bestehen. Die wohl wichtigsten Elemente der Struktur sind die Objektklassen der Komponenten. Hier wird die Funktionalität der Komponenten, wie z.B. des Roboters, des Greifers aber auch des Vision-Systems, programmiert. Diese können anschliessend in der Anlage instanziert werden. Der Skill greift auf die instanzierten Objekte zu und führt entsprechend einen definierten Prozess aus.

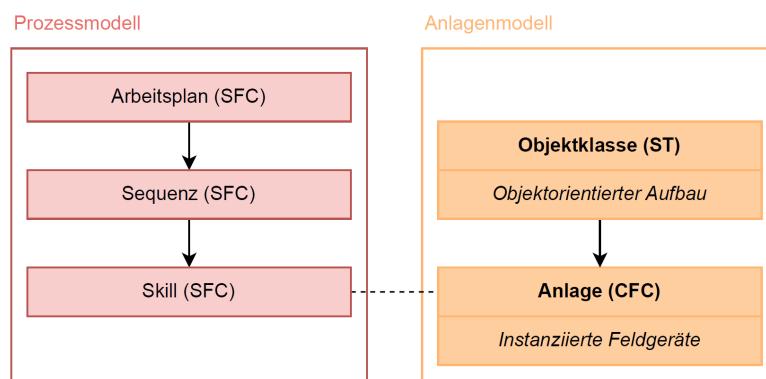


Abbildung 2.8: Beispiel für angepasste Struktur

Die angegebene Struktur (Abb. 2.8) stellt eine erste Einschätzung dar, welche anhand Erfahrungen des Master-Projekt 2 gemacht wurden. Während der Erarbeitung der Thesis kann sich diese noch verändern.

2.3.3 Technische Grundfragen

Frage 3.1: Welche Komponenten können im System zum Einsatz kommen

Ein komplettes System kann aus mehreren Aktoren und Sensoren bestehen, welche miteinander interagieren müssen, um einen bestimmten Prozess durchzuführen. Einige mögliche Komponenten werden in der folgenden Liste (Tab. 2.1) aufgezählt.

Aktoren	Sensoren
<ul style="list-style-type: none"> • Roboter • Greifer (am Roboter) • Greifer (nicht an Roboter) • Hydraulische Zylinder • Antriebsmotor 	<ul style="list-style-type: none"> • Endschalter • Distanzmessung • Vision-System • Bedienbuttons

Tabelle 2.1: Mögliche Komponenten für Robotersystem

Alle Komponenten eines Systems müssen mit der SPS interagieren können und haben einen Einfluss auf den Prozess. Die wohl aufwändigste Schnittstelle ist die SPS-Roboter-Schnittstelle. Fast alle anderen Komponenten können über entsprechende Beckhoff-Klemmen in die SPS integriert werden.

Frage 3.2: Welche Roboter stehen zur Verfügung

Um eine grosse Bandbreite an Möglichkeiten für das System zu haben, soll für das Projekt ein 6-Achs-Roboter eingesetzt werden. Damit der zeitliche Rahmen der Thesis voll genutzt werden kann, werden Roboter betrachtet, welche im Moment an der BFH, im Bereich der Maschinentechnik, zur Verfügung stehen (Tab. 2.2).

ABB	KUKA	Universal-Robots
IRB 1200	KR 6 R700-2	UR5
OmniCore C30	KR C5 Mirco	CB2
		

Tabelle 2.2: Gegenüberstellung der Roboter

Frage 3.3: Welche Schnittstellen bieten die Roboter zu einer SPS

Innerhalb dieser Analyse werden die Möglichkeiten aufgelistet, wie die SPS mit dem Roboter verbunden werden könnte. Details über die Kommunikationsschnittstelle oder die benötigten Hersteller-Tools werden bei der Erarbeitung weiter ausgeführt.

ABB (IRB 1200 / OmniCore C30):

Für die direkte Steuerung eines Roboters kann EGM von ABB verwendet werden. Mit EGM können Befehle via RAPID-Tasks an den Controller gesendet werden und es können Daten empfangen werden. Die Kommunikation wird über eine UdpUc-Schnittstelle realisiert (Abb. 2.9) [4].

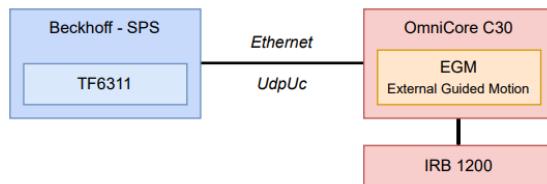


Abbildung 2.9: ABB-Schnittstelle über EGM

Es ist auch möglich den Roboter mit OPC-UA- oder TCP/IP-Schnittstellen anzusprechen. Hierbei kann jedoch nicht direkt mit dem Roboter kommuniziert werden. Die Verbindung zwischen SPS und Controller wird über RobotStudio von ABB hergestellt. In RobotStudio muss ein Programm laufen, welches die Schnittstellen-Variablen auswertet und interpretiert. Für die OPC-UA-Schnittstelle wird auf der SPS der OPC-UA-Server eingerichtet. RobotStudio dient als Client (Abb. 2.10) [5].

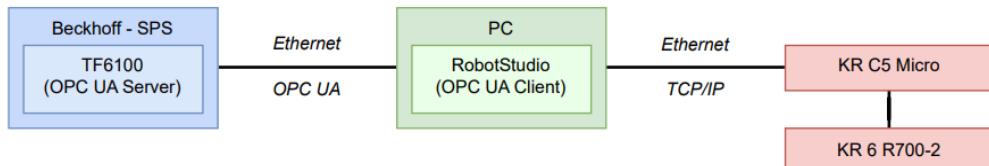


Abbildung 2.10: ABB-Schnittstelle über OPC

KUKA (KR 6 R700-2 / Kr C5 Mirco):

Mit «KUKA.PLC mx Automation» bietet KUKA eine standardisierte Schnittstelle zwischen der Robotersteuerung und der SPS [6]. Dies erlaubt es, den Roboter vollständig durch die SPS und in Echtzeit zu steuern. Dafür werden Funktionsbausteine zur Verfügung gestellt, welche in der SPS verwendet werden können. Um auf diese zugreifen zu können, muss das Beckhoff-Paket TF5120 (TwinCAT 3 Robotics mxAutomation) installiert werden (Abb. 2.11). Eine komplette Liste der Funktionen ist im entsprechenden Beckhoff-Handbuch aufgeführt.

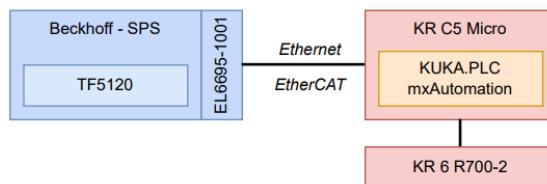


Abbildung 2.11: KUKA-Schnittstelle über mxAutomation

Damit der Controller und die SPS miteinander kommunizieren können, wird eine spezielle EtherCAT-Klemme (EL6695-1001) benötigt, welche von KUKA angeboten wird. Auch eine Schnittstelle über OPC-UA ist möglich [7]. Jedoch dient hierbei der Controller als OPC-UA-Server und definiert Kommunikationsvariablen. Die SPS wird als Client eingesetzt. Der Roboter könnte über diese Schnittstelle nur gesteuert werden, wenn auf dem Controller Programme ausgeführt werden, welche die Kommunikationsvariablen auswerten und entsprechende Aktionen auslösen.

Universal-Robots (UR5 / CB2):

Über eine TCP/IP-Schnittstelle kann die SPS mit dem Controller von Universal Robots kommunizieren [8]. Für eine solche Kommunikationsschnittstelle muss das Beckhoff-Paket TF6310 (TwinCAT 3 TCP/IP) installiert werden (Abb. 2.12) [9]. Über die von Universal Robots entwickelte Programmiersprache URScript kann der Roboter programmiert werden. URScript ermöglicht eine ausführliche Kontrolle über den Roboter.

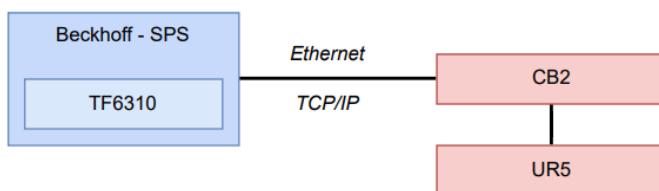


Abbildung 2.12: UR-Schnittstelle über TCP/IP

Frage 3.4: Was sind die Anwendungen eines Roboters in der Industrie

Das folgende Mindmap (Abb. 2.13) zeigt einen groben Umriss von Anwendungen für einen Roboter in der Industrie. Es wird nicht zwischen kollaborativen und industriellen Robotern unterschieden.

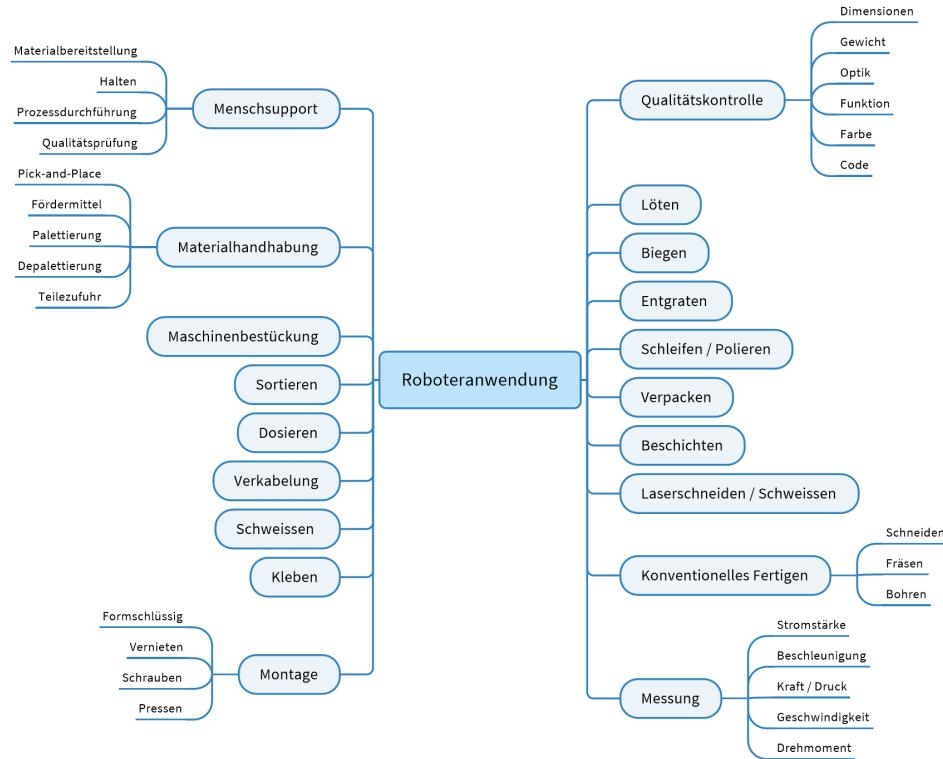


Abbildung 2.13: Roboteranwendung

Ein skill-basierter Ansatz eignet sich besonders für Aufgaben, bei denen Anpassungen am Ablauf erforderlich sind, wie beispielsweise bei der Positionierung oder Ausrichtung eines Objekts. Mit diesem Ansatz kann flexibel auf unterschiedliche Situationen reagiert werden. Das traditionelle Teach-In-Verfahren ist hingegen effizienter für Prozesse, die wiederholte und unveränderte Roboterbewegungen erfordern.

Während sich ein skill-basierter Ansatz grundsätzlich für viele Anwendungen eignet, entfaltet er seinen Vorteil besonders in Szenarien mit variierenden Bedingungen. Werden hingegen immer dieselben Bauteile verarbeitet, ist das klassische Verfahren oft die bessere Wahl.

2.4 Marktanalyse

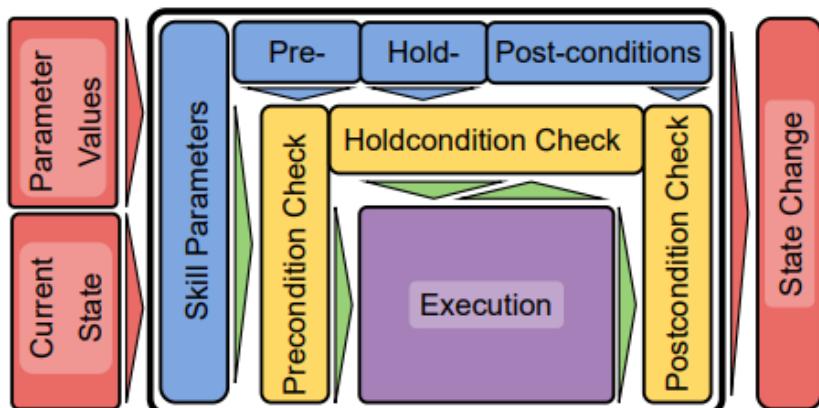
Frage 1: Wurden bereits vergleichbare Projekte und Ansätze durchgeführt

Vergleich 1:

Rahmen:	MSE-Master-Thesis an der BFH
Titel:	Skill-basierte Aufgabenplanung für Roboter in Handarbeitsplätzen von Herstellungsprozessen [10]
Beschreibung:	<p>Innerhalb der Master-Thesis wurde für die Firma Ypsomed AG eine spezifische Anwendung mit skill-basierter Aufgabenplanung umgesetzt. Das Ziel dieser Arbeit war eine Roboterlösung zu entwickeln, die einen raschen Einsatz eines Roboters in der Produktion zum Automatisieren verschiedener und wechselnder Aufgaben ermöglicht. Dafür wurden die Teilaufgaben durch parametrierbare Skills umgesetzt, wodurch komplexe Prozessabläufe abgebildet werden konnten. Das Projekt verwendete einen UR5e-Roboter in Kombination mit einem Greifer und einem Vision-System. Als wichtiges Softwareframework wurde ROS eingesetzt. Jedoch gab es noch weitere Software-Komponenten, welche für das Projekt eingesetzt wurden und miteinander interagiert haben. Das Projekt setzte sich mit folgenden Themen auseinander: Lokalisieren der Objekte, das Berechnen des Griffs und das Planen des Roboterablaufs.</p> <p>Die Grundfunktionalitäten, welche sich aus der vorgegebenen Anwendung der Firma Ypsomed AG ergaben, wurden als Skills umgesetzt. Die Logik des Skills wurde als Zustandsmaschine aufgebaut. Der umgesetzte Prototyp funktionierte für die definierte Anwendung. Einige Funktionalitäten wurden jedoch noch nicht ganz umgesetzt und die Ausführungszeiten des Prozesses sind höher als bei einer händischen Durchführung.</p>
Erkenntnisse:	<ul style="list-style-type: none"> ▶ Allgemeine Idee des skill-basierten Ansatzes hat funktioniert. ▶ Könnte als Referenz für mögliche Skill-Definition dienen (Parameter/Variablen). ▶ Die umgesetzte Lösung könnte auch mit einer SPS-Schnittstelle betrieben werden. ▶ Thematik der Trennung zwischen Prozess und Anlage wurde als zukünftiger Schritt definiert. Im Moment ist die Struktur anlagenspezifisch. ▶ Die umgesetzte Lösung setzt auf viele unterschiedliche Software-Tools mit entsprechenden Schnittstellen.

Vergleich 2:

Rahmen:	International Conference on Intelligent Robots and Systems (IROS)
Titel:	SkiROS2: A skill-based Robot Control Platform for ROS [11]
Beschreibung:	<p>Das Dokument beschreibt SkiROS2, welches eine skill-basierte Robotersteuerungsplattform auf Basis von ROS darstellt. Die Struktur wurde darauf ausgelegt, kleine Batchgrößen, welche komplexe Abläufe benötigen, umzusetzen. Die Architektur von SkiROS2 besteht aus einem Skill Manager und einem World Model, welche das Kernsystem bilden.</p> <p>Mit dieser Struktur ist es möglich, mehrere Roboter zu steuern und miteinander zu synchronisieren. Das World Model speichert Instanzen der aktuellen Situation. Es enthält semantische Informationen über Roboter, Objekte und Orte und bietet eine API für das Lesen und Schreiben von Daten. Es kann zur Planung und Parametrisierung von Skills genutzt werden. Jeder Roboter hat einen eigenen Skill Manager, der Skills aus Bibliotheken lädt, diese initialisiert und überwacht. Er übernimmt auch die Ausführung und teilt Aufgaben eindeutig zu.</p> <p>Alle Skills besitzen die gleiche Struktur. Die Skill-Schnittstelle definiert die benötigten Parameter, wie auch die Pre-, Hold- und Post-Bedingung. Diese werden als Transitionen verwendet, welche den Skill starten, beenden oder abbrechen. Es wird zwischen zwei Skill-Arten unterschieden. Die «primitive» Skills sind Aktionen, welche vom System in der realen Welt ausgeführt werden (z.B. Greifen). Diese haben drei Zustände: «running», «success» und «failure». Die «compound» Skills stellen einen Zusammenschluss von «primitive» oder «compound» Skills dar.</p>

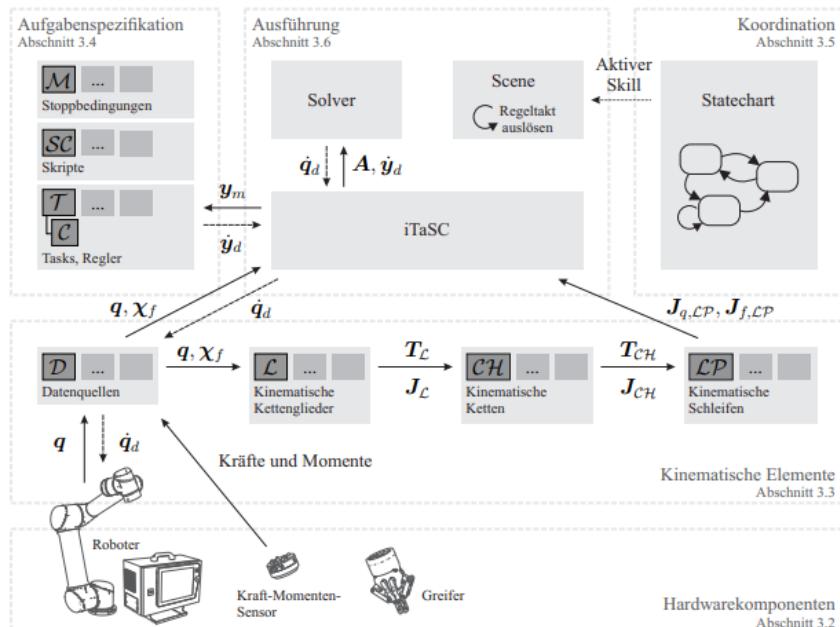


Erkenntnisse:

- ▶ SkiROS2 scheint eine bereits sehr ausgereifte Umsetzung eines skill-basierten Ansatzes zu sein.
- ▶ Ein Grundwissen über ROS, der allgemeinen Struktur von SkiROS2 und das Programmieren in Python ist erforderlich, um es einsetzen zu können.
- ▶ Ansätze der Strukturierung können als Referenz für diese These dienen.

Vergleich 3:

- Rahmen: Dissertation an der Universität Stuttgart
 Titel: Prototypbasiertes Skill-Modell zur Programmierung von Robotern für kraftgeregelte Montageprozesse [12]
 Beschreibung: Die Arbeit zielt darauf ab, den Einsatz von Industrierobotern in Montageanwendungen zu erleichtern, indem ein skill-basierter Ansatz zur Programmierung von kraftgeregelten Montageprozesse entwickelt wurde.
 Innerhalb der Arbeit wurde das komplette System entworfen. Die Skills werden in Bereich Koordination gehandhabt.



Für das Projekt wurde ein eigenes Skill-Modell entworfen, mit dem Namen «pitasc». Das entwickelte Skill-Modell basiert auf drei Grundpfeilern: Abstraktion, Komposition und Vererbung. Hierbei wird das Herunterbrechen der Funktionen des Systems auf anlagenunabhängige Parameter beschrieben.

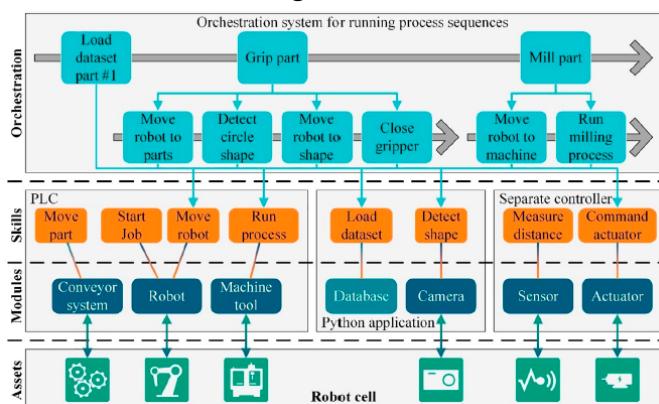
Die Arbeit wurde für das Fraunhofer-Institut für Produktionstechnik und Automatisierung erstellt. Das Institut bietet den pitasc-Systembaukasten mittlerweile zum Kauf an.

Erkenntnisse:

- ▶ Das pitasc-Skill-Modell könnte für das Projekt spannend sein.
- ▶ Die Thematik der Positions-, Geschwindigkeits- und Kraftregelung kann ein relevanter Punkt werden (z.B. Aufstecken von Klemme auf DIN-Schiene). Die iTaSC-Formulierung könnte dafür hilfreich sein.
- ▶ Das entworfene Skill-Modell ist anlagenunabhängig.
- ▶ Die Software macht einen sehr komplexen Eindruck, was sich wahrscheinlich in der Anwendung und Bedienung widerspiegelt.
- ▶ Das beschriebene System beinhaltet kein Vision-System.

Vergleich 4:

Rahmen:	Research-Paper des Fraunhofer-Institut für Werkzeugmaschinen und Umformtechnik
Titel:	Flexible skill-based control for robot cells in manufacturing [13]
Beschreibung:	Das Forschungsprojekt stellt die Methode zur Programmierung flexibler, auf Skills basierender Steuerungen für Roboteranwendung vor. Dabei wird stark auf die Vorteile einer solchen Methode eingegangen. Als Hauptvorteil wird die Fähigkeit angegeben, dass der Prozessablauf von Bedienern angepasst und erweitert werden kann, ohne den Steuerungscode zu ändern. Für die Entwicklung der Methode wurden vier Anforderungen gestellt: Erweiterbarkeit, Flexible Nutzbarkeit, Konfigurierbarkeit und Wiederverwendbarkeit. Um diese Anforderungen zu erfüllen, wurde die «skill-based control architecture (SBC)» angewendet, welche im Dokument «Evaluating Skill-Based Control Architecture for Flexible Automation Systems» von Kirill Dorojeff und Monika Wenger beschrieben wird.



Es wird auch darauf hingewiesen, dass es bereits Richtlinien für die Standardisierung von Skills (VDI/VDE/NAMUR 2658) des VDI gibt. Wichtige Begriffe dieser Richtlinie sind PEA (Schnittstellen modulärer Prozesseinheiten) und MTP (Module Type Package).

Innerhalb des Projektes wurde ein Prototyp auf Basis von TwinCAT umgesetzt (inkl. HMI). Ein Aspekt des Systems ist die TwinCAT-Hot-Connect-Funktionalität. Damit können Anlagenkomponenten während des Betriebs entfernt oder hinzugefügt werden. Dies kann für das Wechseln von Greifern während dem Betrieb relevant sein. Das Projekt hat gezeigt, dass ein Aufbau mit der SBC-Struktur aufgebaut werden kann und funktioniert. Als weiterführende Arbeiten wurde definiert, dass das benötigte Wissen zur Anwendung von skill-basierten Ansätzen weiter verringert werden muss und dass dabei das HMI eine wichtige Rolle spielt.

Erkenntnisse:

- ▶ Das Projekt zeigt, dass eine solche Anwendung auf Basis von TwinCAT umgesetzt werden kann.
- ▶ Das entwickelte HMI kann als Referenz für die Thesis dienen.
- ▶ Die TwinCAT-Hot-Connect-Funktionalität kann für das Projekt relevant sein.
- ▶ Die Richtlinie des VDI kann für das Projekt relevant sein.

Die Analyse verdeutlicht, dass bereits zahlreiche Projekte zu diesem Thema durchgeführt wurden, die unterschiedliche Ansätze verfolgen. Diese reichen von reinen Forschungsprojekten bis hin zu Lösungen, die direkt in der Industrie Anwendung finden. Ein skill-basierter Ansatz gewinnt aufgrund neuer und spezifischer Anforderungen der Industrie zunehmend an Bedeutung. Die Implementierung eines vollständig skill-basierten Systems umfasst viele unterschiedliche Aspekte, wobei die gewählte Struktur eine zentrale Rolle spielt. Es gibt jedoch zahlreiche Punkte, die für diese Arbeit als Referenz dienen und im Vorfeld geklärt werden müssen. Der Umfang des Themas verdeutlicht, wie wichtig es ist, die Anforderungen an das System präzise zu definieren. Keines der analysierten Projekte hat die ISA/ANSI-88-Norm als Grundlage für seine Struktur verwendet. Die Anwendung dieser Norm auf ein skill-basiertes System stellt daher eine innovative Herangehensweise dar.

Frage 2: Was beschreibt die Richtlinie VDI/VDE/NAMUR 2658

Die Richtlinie beschreibt das Engineering der Automatisierungstechnik modularer Anlagen, insbesondere in der Verfahrenstechnik 14. Sie behandelt sowohl das Modulengineering als auch das Anlagenengineering. Das Module Type Package (MTP) dient zur Definition und Beschreibung der Schnittstellen und Funktionen der Module, was die Integration in eine Prozessführungsebene ermöglicht. Die Schwerpunkte der Richtlinie umfassen:

- ▶ Allgemeines Konzept des modularen Anlagenengineering.
- ▶ Zustands- und Dienstmodelle modularer Anlagen.
- ▶ Aufbau und Struktur des MTP.
- ▶ Schnittstellen zwischen den Moduldiensten und der Prozessführungsebene (PFE).
- ▶ Definition des MTP-Manifests und der Kommunikationsschnittstellen (z.B. OPC-UA).
- ▶ Modellierungsvorgaben zur Erstellung des MTP-Manifests und der Kommunikationsbeschreibung.

Ziel der Richtlinie ist, die Integration von Feldgeräten zu vereinfachen, damit diese effizient mit anderen Systemen zusammenarbeiten. Die Richtlinie stellt sicher, dass die Geräte herstellerunabhängig funktionieren und Daten, wie Mess- oder Diagnosedaten, standardisiert ausgetauscht werden können. Dabei werden nicht nur die Integration in Planungs- und Engineering-Software, sondern auch der Betrieb in Prozessleitsystemen sowie der gesamte Lebenszyklus der Geräte, von der Planung über den Betrieb bis zur Wartung, berücksichtigt. Dadurch können Fehler reduziert und der Aufwand für Inbetriebnahme und Wartung minimiert werden.

Frage 3: Was ist MTP

MTP steht für Module Type Package und definiert standardisierte Schnittstellen zwischen Anlagenmodulen [15]. Innerhalb eines Systems gibt es verschiedene Module, welche für eine spezielle Prozessfunktion zuständig sind. Dieses Modul wird als Functional Equipment Assembly (FEA) bezeichnet und bildet ein eigenes, geschlossenes System mit Schnittstellen gegen aussen und innen.



Abbildung 2.14: MTP-Struktur von Beckhoff

Ein System besteht aus mindestens einem FEA und bildet das Process Equipment Assembly (PEA). Innerhalb des PEA können FEA entfernt oder hinzugefügt werden. Das MTP definiert eine einfache Konfiguration und Implementation der Anlagenmodule. MTPs definieren die Eigenschaften und Schnittstellen dieser Module. Das Prozesseitsystem liest die MTPs und nutzt diese, um einen Prozessablauf an die Anlagenmodule zu delegieren. Die gesamte Anlage ist dadurch flexibel einsetz- und erweiterbar.

Das MTP-Prinzip wird bis jetzt hauptsächlich in der Verfahrenstechnik eingesetzt. Unternehmen wie Beckhoff, ABB oder Siemens bieten MTP-Pakete für Ihre Systeme an (Abb. 2.14).

Die standardisierte Schnittstellendefinition könnte als Referenz für die Thesis verwendet werden. Jedoch muss die Richtlinie (VDI/VDE/NAMUR 2658) gekauft werden und steht im Moment nicht zur Verfügung.

Frage 4: Was ist der PLCopen-Standard

Der PLCopen-Standard ist eine internationale Initiative zur Standardisierung von Programmiersprachen und Funktionen einer SPS. Ziel von PLCopen ist es, die Programmierung, Entwicklung und Wartung von Steuerungssystemen zu vereinfachen und zu vereinheitlichen, um die Kompatibilität zwischen verschiedenen SPS-Systemen und Herstellern zu verbessern. Der PLCopen-Standard ergänzt die Norm IEC 61131-3, indem Bibliotheken, Modelle und Richtlinien zur Verfügung gestellt werden, die auf diesen Normen basieren und die Programmierung einer SPS noch effizienter gestalten [16].

Die Richtlinie gibt einen Ansatz vor, wie Funktionsbausteine aufgebaut werden können und funktionieren. Dabei wird zwischen zwei Arten von Funktionsbausteinen unterschieden. Der pegelgesteuerte Funktionsbaustein bleibt aktiv, so lange das Triggersignal aktiv ist. Der flankengesteuerte Funktionsbaustein wird bei einer Flanke aktiv und bleibt aktiv. Die Basisdefinition dieser Funktionsbausteine sieht wie folgt aus:

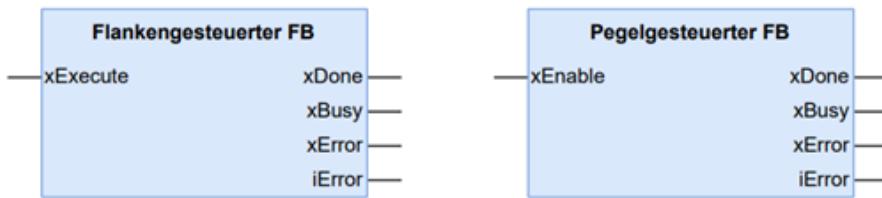


Abbildung 2.15: PLCopen-Basisfunktionsblock

Diese Basisdefinition kann beliebig erweitert werden, z.B. mit einer Abort-Funktionalität. Das Verhalten der Funktionsbausteine wird anhand des aufgeführten Diagramms (Abb. 2.16) erklärt. Die Variable «eState» gibt dabei den momentanen Zustand des Funktionsbausteins als Eigenschaft wieder. Der Ansatz der Funktionsbausteindefinition kann für gewisse Aspekte der Thesis relevant sein. Falls Skills als Funktionsbausteine definiert werden, könnte diese Definition helfen, diese übersichtlich und gleich zu strukturieren.

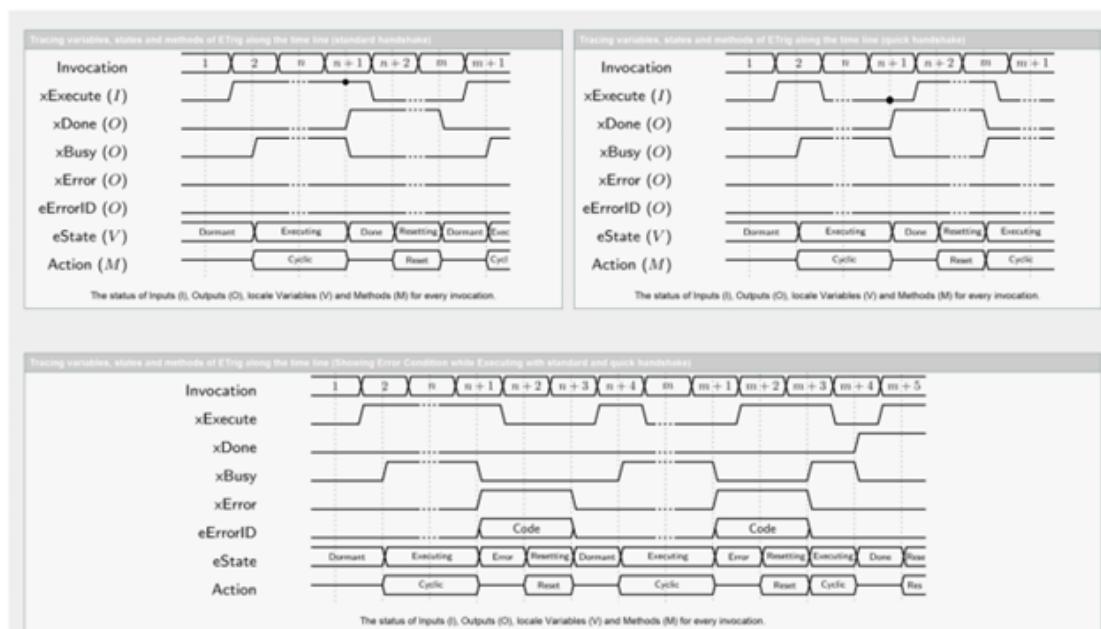


Abbildung 2.16: PLCopen-Basisablauf



Die relevanten Dokumentationen von PLCopen werden im Anhang beigelegt

2.5 System- und Kontextgrenze

Über die System- und Kontextgrenzen (Abb. 2.17) wird der Rahmen für das Projekt gesetzt. Man unterscheidet zwischen System (*kann beeinflusst werden*), Systemkontext (*ist relevant für das Projekt, jedoch können solche Aspekte nicht beeinflusst werden*) und irrelevanter Umgebung (*spielt zum jetzigen Zeitpunkt keine Rolle*).

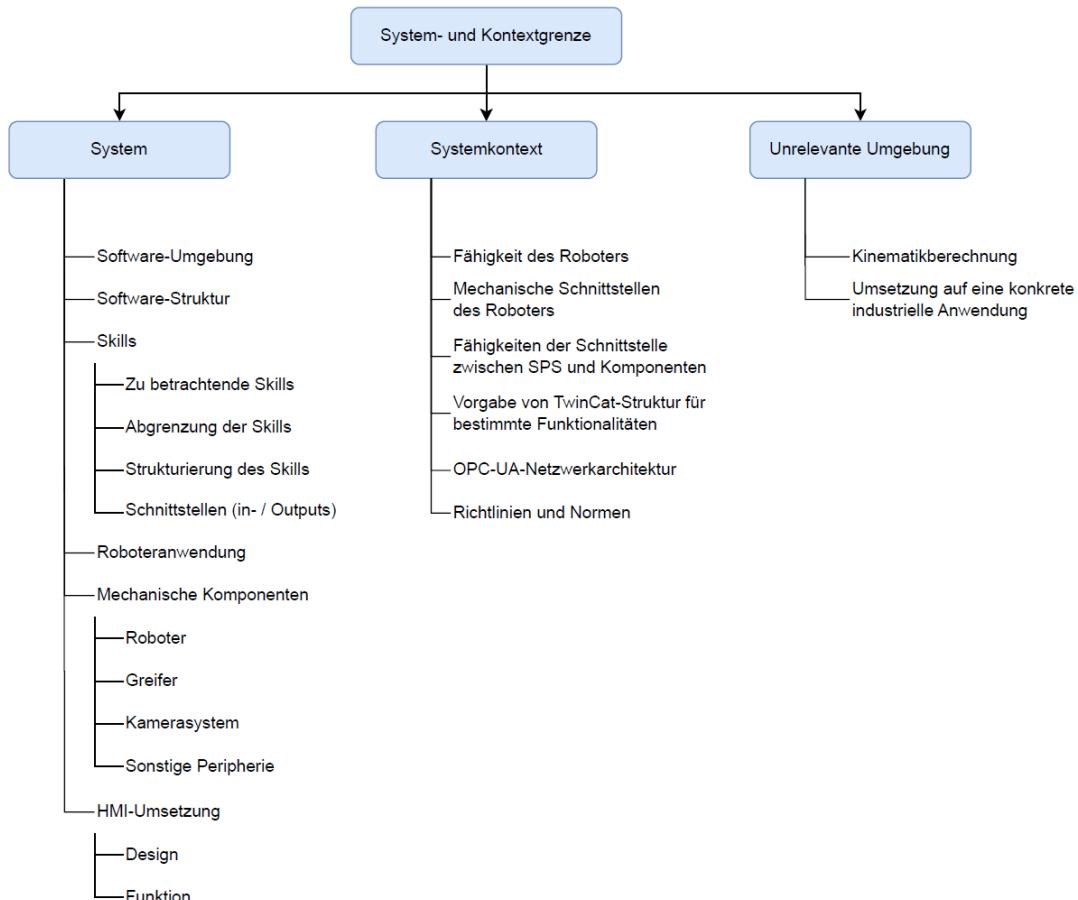


Abbildung 2.17: System- und Kontextgrenzen

2.6 Projektanforderungen

Die Thesis basiert auf einem offen definierten Auftrag ohne ein festgelegtes Lastenheft, wodurch die Projektanforderungen entsprechend flexibel gestaltet sind. Um jedoch einen effizienten und zielgerichteten Workflow zu gewährleisten – insbesondere bei einem Projekt dieser Größenordnung – ist es entscheidend, klare Projektziele zu definieren.

Für die Definition der Projektanforderungen wird eine bewährte Methode aus dem System- und Anforderungsmanagement genutzt, bei der «Needs», «Goals» und «Objectives» (NGOs) formuliert werden (Tab. 2.3). Ein «Need» beschreibt in einer einzigen Aussage, welches Problem das System lösen soll, ohne dabei konkrete Lösungen vorzugeben. Die «Goals» legen die Erwartungen an das System fest, die zur Erfüllung des «Needs» beitragen. Darauf aufbauend definieren die «Objectives» spezifische Anforderungen für jedes «Goal». Ein zentraler Aspekt dieser Anforderungen ist, dass sie messbar, quantifizierbar und überprüfbar sein müssen.

Folgende NGO's wurden für die Thesis definiert:

Need	Eine Roboteranwendung soll einfach und standardisiert programmierbar sein			
Goals	G1		G2	G3
	Personen ohne grossen Programmierhintergrund können einen Arbeitsablauf für das System definieren		Skills können verwendet werden, um komplexe Arbeitsabläufe abzubilden	Das System kann mit einem Versuchsaufbau getestet werden
Objectives	A1	Ein HMI soll die Möglichkeit bieten, Arbeitspläne zu definieren (Ablauf / Parameter).	A1	Ein Standard soll definiert werden für die Definition eines Skills
	A2	Ein HMI soll die Möglichkeit bieten, das System zu bedienen (Start, Stop, Reset)	A2	Ein Skill sollen die Funktionalität einer Komponente abbilden können
	A3	Die Software soll sich nach der ANSI/ISA-88-Norm richten und ein Prozess- und Anlagenmodell besitzen.	A3	Das System soll in der Lage sein, Prozessschritte zu überprüfen
	A4	Die Softwarestruktur soll eine anlagenunabhängige Definition des Prozessmodels ermöglichen	A4	Das System soll Korrekturen bei nicht erfüllten Prozessschritten vornehmen
	A5	Die Funktionalität der Systemkomponenten soll im Anlagenmodell abgebildet werden	A5	Der Roboter soll unterschiedliche Objekte greifen können
	A6	Die Objektklassen der Systemkomponenten sollen objektorientiert aufgebaut werden	A6	Die Software soll mit TwinCat umgesetzt werden
	A7	Alle Komponenten sollen über ein eigenes HMI im manuell-Modus betrieben werden können.	A7	Der Versuchsaufbau kann eine OPC-UA-Kommunikationsschnittstelle besitzen

Tabelle 2.3: Projektanforderungen

Mit der Definition der Projektanforderungen wird die Analyse-Phase abgeschlossen. Die gewonnenen Erkenntnisse werden in einem nächsten Schritt als Grundlage für die Planung der weiteren Phasen verwendet. Am Ende des Projektes werden die definierten Anforderungen mit der umgesetzten Lösung verglichen.

3 Vorgehen

Der skill-basierte Ansatz beinhaltet eine Vielzahl unterschiedlicher Aspekte. Für die Thesis ist es daher entscheidend, die Prioritäten klar zu setzen. Zur strukturierten Bearbeitung werden verschiedene Arbeitspakete festgelegt, die die zentralen Anforderungen des Projekts abdecken. Im Anschluss werden die Abhängigkeiten dieser Arbeitspakete bestimmt und in einem Gate-Plan visualisiert. Dieser gliedert die Bearbeitung in Phasen und zeigt die Reihenfolge auf, in der die Arbeitspakete abgearbeitet werden sollen.

3.1 Arbeitspakete

Grundsätzlich teilt sich das Gesamtsystem in die Hardware und Software ein (Abb. 3.1). Diese lassen sich anschliessend in verschiedene Arbeitspakete aufteilen.

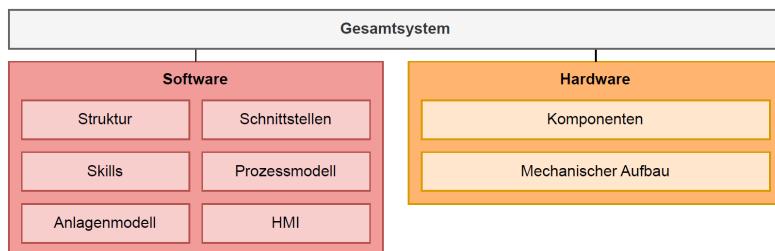


Abbildung 3.1: Definierte Arbeitspakete

Software:

- Struktur: Definieren der Software-Elemente und wie diese miteinander in Beziehung stehen.
- Schnittstellen: Definieren der Schnittstellen zwischen Hardware-Komponenten und Software.
- Skills: Planung und Entwicklung der Skills (Struktur, Umsetzung).
- Prozessmodell: Planung und Entwicklung des Prozessmodells (Struktur, Umsetzung).
- Anlagenmodell: Planung und Entwicklung des Anlagenmodell. (Struktur, Umsetzung).
- HMI: Planung und Entwicklung des HMI. (Design, Umsetzung, Anwendung).

Hardware:

- Komponenten: Definieren einer Anwendung und die dazu benötigten Komponenten.
- Mech. Aufbau: Konstruktion und Montage des mechanischen Aufbaus.

3.2 Gate-Plan

Der Gate-Plan besteht aus 4 Phasen (Abb. 3.2). Jede Phase wird durch ein Gate begonnen und abgeschlossen. In den Phasen werden die verschiedenen Arbeitspakete erarbeitet.

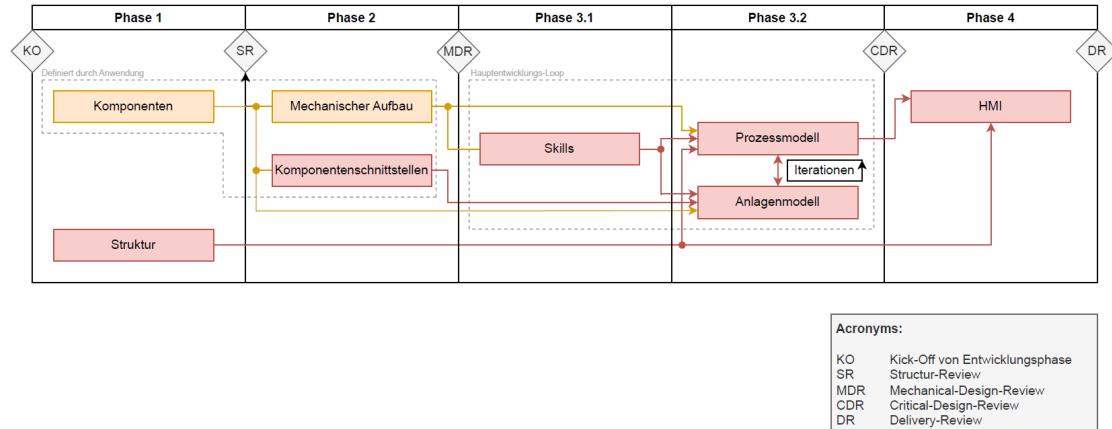


Abbildung 3.2: Definierter Gate-Plan

Phase 1:

Beschreibung: Phase 1 beschäftigt sich mit den Grundlagen der Hard- und Software. Die relevanten Komponenten müssen definiert und die notwendigen Fragen bezüglich der Software-Struktur geklärt werden.

Output:

- ▶ Alle Komponenten für den Versuchsaufbau sind bekannt.
- ▶ Die allgemeine Software-Struktur wurde definiert.

Phase 2:

Beschreibung: Innerhalb von Phase 2 wird der Hardware-Teil abgeschlossen. Dafür wird der Versuchsaufbau konstruiert und gebaut. Zusätzlich werden die Software-Schnittstellen der Komponenten analysiert und definiert.

Output:

- ▶ Gebauter Versuchsbau für das Testen der Software.
- ▶ Es ist bekannt, wie die Komponenten in die Software implementiert werden (in Theorie).

Phase 3:

Beschreibung: Phase 3 stellt die konkrete Umsetzung des skill-basierten Ansatzes in der Software dar. Hierbei wird iterativ die Software entwickelt. Diese umfasst die Umsetzung der Struktur, wie auch die Entwicklung der Skills, der Objektklassen und des kompletten Prozessmodells.

Output:

- ▶ Funktionaler Prototyp, welcher an der Versuchsanlage getestet werden kann.

Phase 4:

Beschreibung: Die letzte Phase beschäftigt sich mit der Entwicklung eines HMI.

Output:

- ▶ Über HMI bedienbarer Prototyp, welcher an der Versuchsanlage getestet werden kann.

4 Anwendung und Aufbau

4.1 Definition der Anwendung

Als Anwendung für den Versuchsaufbau wurde die Verbindung zweier Platten über einen Befestigungswinkel definiert. Die Platten bestehen aus Kunststoff und sind mit drei Befestigungslöchern, wie auch Fingerzinken versehen. Der Prozess wird durch 3 Teilprozesse gebildet.

Teilprozess 1: Positionieren der Platten

Dieser Teilprozess fügt die beiden Platten im 90° -Winkel zusammen (Abb. 4.1a). Die Seiten der Platten sind mit ineinandergreifenden Fingerzinken versehen, die ineinandergreifen. Der Roboter entnimmt zunächst die erste Platte aus dem Lager und positioniert sie auf einer Halterung. Anschliessend holt der Roboter die zweite Platte und stösst diese entsprechend der Fingerzinken in die erste Platte.

Teilprozess 2: Montieren von Befestigungswinkel

Der Roboter holt den Befestigungswinkel aus dem Lager und platziert diesen entsprechend der Löcher in den Platten (Abb. 4.1b). Die korrekte Position wird über ein Vision-System definiert. Das Befestigungsblech kann auch in die Halterung eingelegt werden.

Teilprozess 3: Verbinden von Platten mit Befestigungswinkel

Im letzten Schritt wird der Befestigungswinkel mit Stiften mit den Platten verbunden (Abb. 4.1c). Die Passung der Löcher ist dabei so ausgelegt, dass der Stift knapp mit Spiel montiert werden kann. Die Stifte müssen somit so mit der Platte verbunden werden, dass sich diese nicht verkanten.

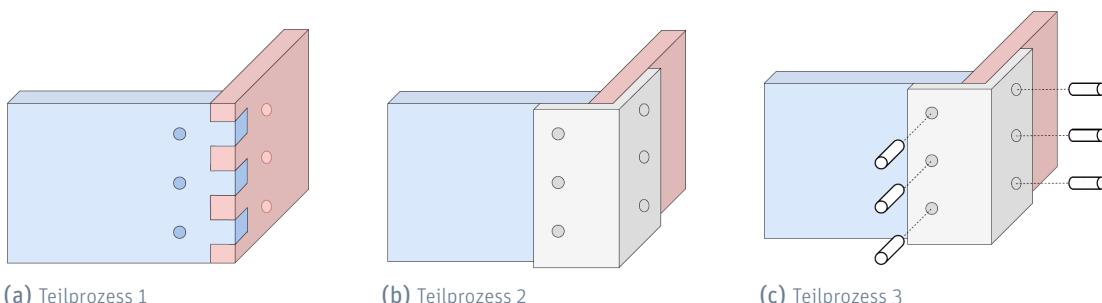


Abbildung 4.1: Anwendungsteilprozesse

4.2 Definition der Anlagenkomponenten

Die allgemeine Kommunikationstopologie (Abb. 4.2) der funktionsrelevanten Komponenten sieht wie folgt aus:

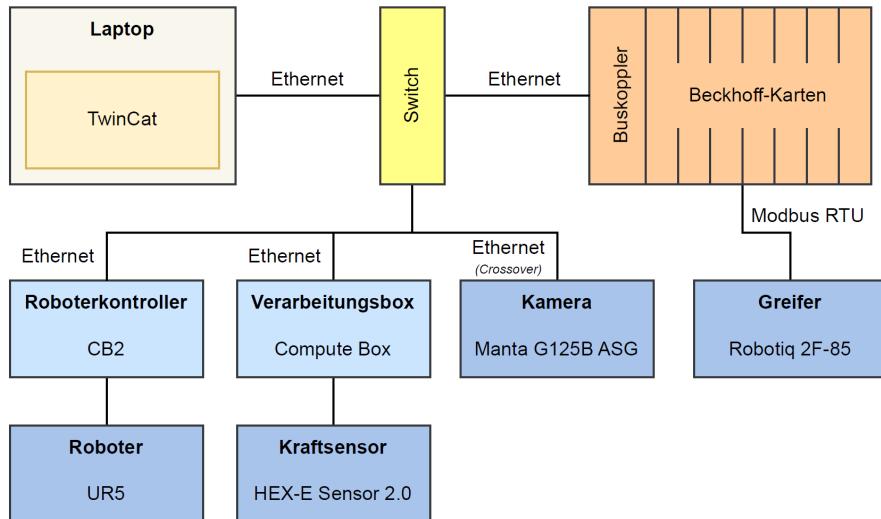


Abbildung 4.2: Kommunikationstopologie

Für die Umsetzung des Versuchsaufbaus wird ein Laptop mit TwinCAT als SPS eingesetzt. Der Grund dafür ist die Leistungsfähigkeit. Da alle Funktionalitäten innerhalb der SPS stattfinden sollen, muss diese auch die entsprechende Power mitbringen. Die Switch verbindet den Laptop mit dem Bus-Koppler und den entsprechenden Komponenten der Prozesszelle. Teil des Netzwerkes ist der Roboterkontroller, die Signalverarbeitungsbox des Kraftsensors und die Kamera.



Alle relevanten Datenblätter werden im Anhang beigelegt.

Roboter und Roboterkontroller:

Der UR5-Roboter von Universal Robots wurde als Robotersystem ausgewählt. Dieser kolaborative Roboter zeichnet sich durch eine benutzerfreundliche Kommunikationsschnittstelle aus und kann schnell und unkompliziert in Betrieb genommen werden. An der BFH besteht umfassende Erfahrung mit dem UR5 und es stehen verschiedene Peripherie-Tools zur Verfügung.

Mit einer Traglast von 5 kg und einer Reichweite von 850 mm ist der Roboter für vielfältige Anwendungen geeignet [8]. Als Kommunikationsschnittstelle zwischen TwinCAT und Kontroller wird eine TCP/IP-Schnittstelle verwendet. Dafür muss TwinCAT mit dem TF6310-Paket (TwinCAT 3 TCP/IP) versehen werden. Die TCP/IP-Schnittstellen ermöglichen das Programmieren des Roboters über URScript. Der Kontroller verfügt aber auch über digitale und analoge Ein- und Ausgänge.

Kamera:

Als Kamera kommt die „Manta G125B ASG“ zum Einsatz, die mit der weit verbreiteten GigE-Vision-Schnittstelle ausgestattet ist. Dadurch lässt sie sich direkt aus TwinCAT auslesen und konfigurieren. Um die Kamera in Betrieb zu nehmen, müssen die TF7XXX-Pakete (TwinCAT 3 Vision) installiert sein. Der Anschluss der Kamera erfolgt über eine Ethernet-Schnittstelle mithilfe eines Crossover-Kabels.

Kraftmessungssensor:

Im Gegensatz zum UR5e verfügt der UR5 nicht über integrierte Kraft-/Drehmomentsensoren in den Gelenken, was eine präzise Kraftregelung erschwert. Um dies zu kompensieren, kann der UR5 mit einem externen Kraftmesssensor ausgestattet werden. Hierfür eignet sich der „HEX-E Sensor 2.0“ von OnRobot (Abb. 4.3), der an der mechanischen Schnittstelle des UR5 montiert werden kann.

Der Sensor misst Kräfte und Drehmomente in und um drei Achsen [17]. Die erfassten Daten werden an eine Signalverarbeitungsbox weitergeleitet, die das Rohsignal verarbeitet und über eine Ethernet-Schnittstelle zur Verfügung stellt. Die Kommunikation kann über eine UDP- oder TCP-Schnittstelle erfolgen. Für die Integration in TwinCAT wird das TF6310-Paket (TwinCAT 3 TCP/IP) verwendet.

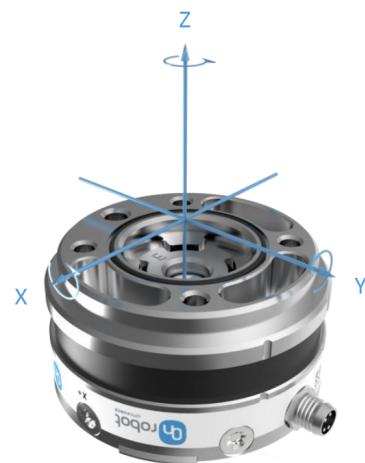


Abbildung 4.3: Kraftsensor

Greifer:

Für das System wird der 2F-85-Greifer von Robotiq verwendet (Abb. 4.4), der speziell für die Integration mit UR-Robotern entwickelt wurde. Die Montage erfolgt mittels einer Adapterplatte. Der Greifer kann nicht direkt über den Roboter-Kontroller betrieben werden, da die CB2-Version zu alt ist. Die Schnittstelle wird direkt zwischen Greifer und SPS aufgebaut. Die Schnittstelle wird in Kapitel 7.2.3 beschrieben.

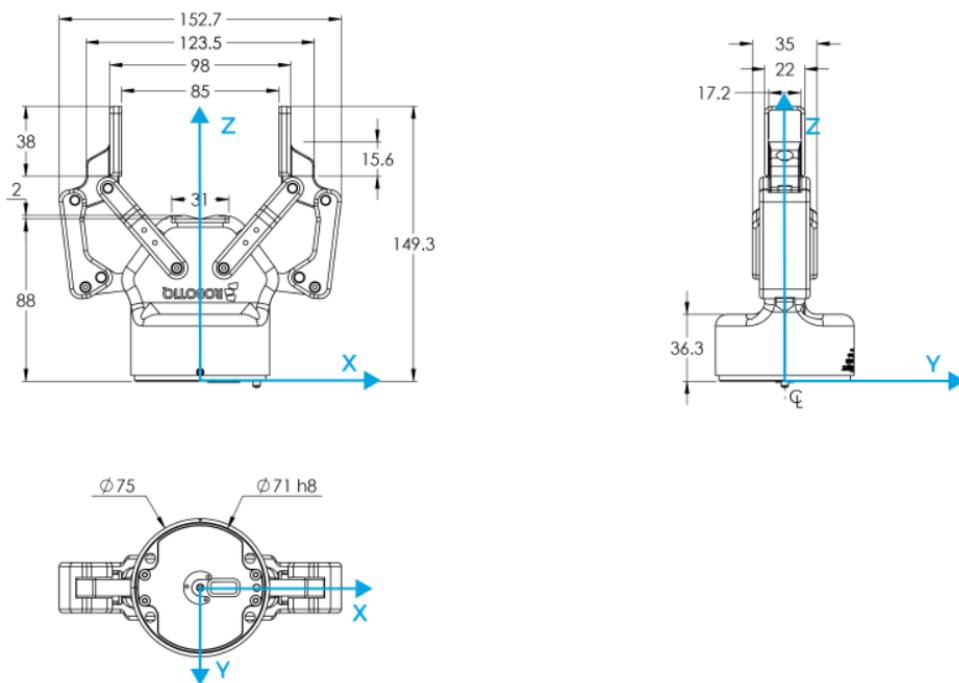


Abbildung 4.4: Kommunikationstopologie

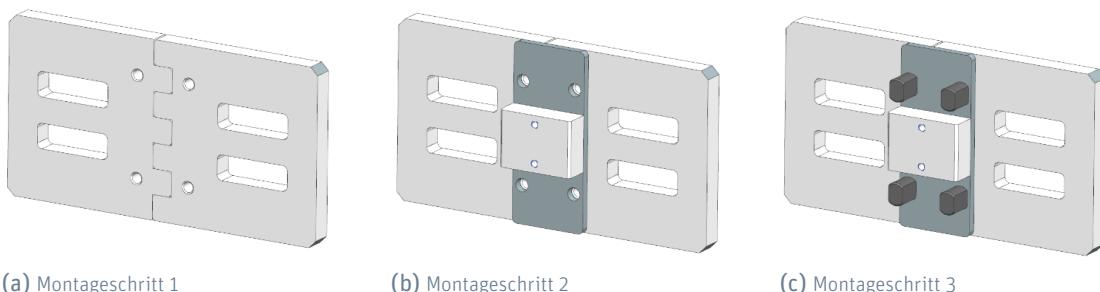
Der 2F-85-Greifer bietet eine maximale Greifweite von 85 mm und wird elektrisch betrieben. Dank integrierter Sensoren lässt sich die Greifkraft präzise überwachen und anpassen. Auch die Fingerposition wird überwacht, was dem Greifer eine hohe Flexibilität in verschiedenen Anwendungen ermöglicht [18].

4.3 Mechanischer Aufbau

Die mechanische Konstruktion wurde so einfach wie möglich gestaltet, um in erster Linie das schnelle Testen der Software zu ermöglichen. Der Aufbau besteht aus zwei Kunststoffplatten, einem Verbindungsblech und vier Stiften, welche miteinander verbunden werden sollen.

Der Montageprozess erfolgt in drei Schritten: Zunächst werden die beiden Platten miteinander verbunden (Abb. 4.5a). Dafür sind sie mit «Fingerzinkungen» ausgestattet, durch die sie ineinander gesteckt werden. Anders als im ursprünglichen Konzept vorgesehen, bei dem eine Eckverbindung angedacht war (Abb. 4.1), liegen die Platten in einer Ebene. Erste Simulationen mit der Software «RoboDK» ergaben, dass eine Umsetzung mit Eckverbindung die Erreichbarkeit der verschiedenen Positionen für den Roboter deutlich erschwert hätte. Durch den montierten Kraftsensor und Greifer wird der TCP versetzt, wodurch der Arbeitsbereich des Roboters zusätzlich eingeschränkt wird. Die flache Anordnung der Platten erleichtert die Zugänglichkeit erheblich, während der grundlegende Prozess unverändert bleibt.

Im zweiten Schritt wird das Verbindungsblech auf die zusammengefügten Platten aufgelegt (Abb. 4.5b). Es muss dabei exakt auf die Lochpositionen der Platten ausgerichtet werden. Im dritten Schritt werden die Platten und das Blech mithilfe von Stiften fixiert (Abb. 4.5c). Die Stifte werden in die vorgesehenen Löcher gedrückt, wobei eine enge, aber noch als Spielpassung definierte Toleranz vorliegt. Der Roboter muss dabei äußerst präzise arbeiten und in der Lage sein, eine Verkantung zu erkennen. Auf Verfahren wie Schrauben oder Nieten wurde bewusst verzichtet, um den Einsatz eines spezifischen Werkzeugs für den Roboter zu vermeiden.



(a) Montageschritt 1

(b) Montageschritt 2

(c) Montageschritt 3

Abbildung 4.5: Anwendungsteilprozesse



Alle Detailzeichnungen in Fertigungsdaten werden im Anhang beigelegt.

Die Montage wird auf einer Platte durchgeführt. Gleichzeitig dient die Platte auch als Halterung für die Komponenten (Abb. 4.6a). Im unteren linken Bereich werden die Platten miteinander montiert. Damit eine Referenzposition sichergestellt werden kann, dient ein L-Stück als Anschlag. Die erste Platte wird damit in X- und Y-Richtung positioniert. Das Ziel ist, dass der Roboter den Anschlag durch den Kraftsensor erkennt und entsprechend die Bewegung stoppt.

Der UR5 wird mit einem Kraftsensor und einem Greifer ausgestattet (Abb. 4.6b). Der Kraftsensor wird am UR5 montiert, während der Greifer am Kraftsensor befestigt wird. Um die Verbindung zwischen dem Kraftsensor und dem UR5 herzustellen, wurde eine Adapterplatte entwickelt. Der Greifer hingegen kann direkt am Sensor befestigt werden, ohne dass ein zusätzlicher Adapter erforderlich ist. Für ein präzises und zuverlässiges Greifen der Anwendungsteile wurden spezielle Backeneinsätze für den Greifer konstruiert.

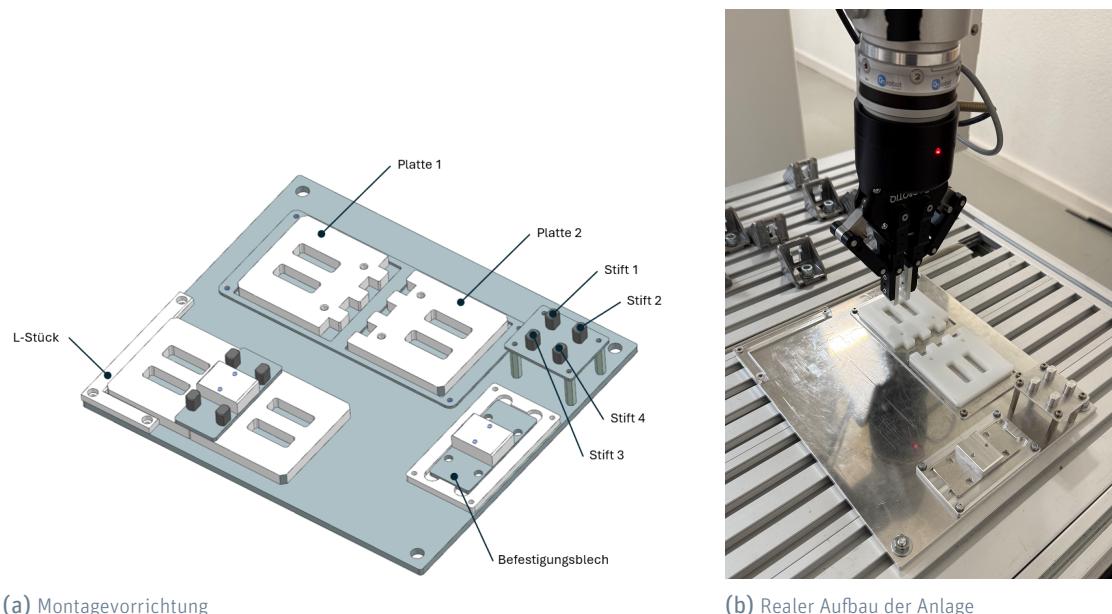


Abbildung 4.6: Umgesetzte Anwendung

Mit der Fertigung und Montage des mechanischen Aufbaus, kann in einem nächsten Schritt mit der Umsetzung der Software begonnen werden.

5 Struktur der Software

Die Struktur der Software muss auf einen skill-basierten Ansatz ausgelegt werden. Dafür müssen die Anforderungen an eine solche Struktur klar definiert und die Möglichkeiten, die TwinCAT bietet, analysiert werden. In einem ersten Schritt wird die allgemeine Grobstruktur des Systems dargestellt (Abb. 5.1). Dieses kann wie folgt definiert werden:

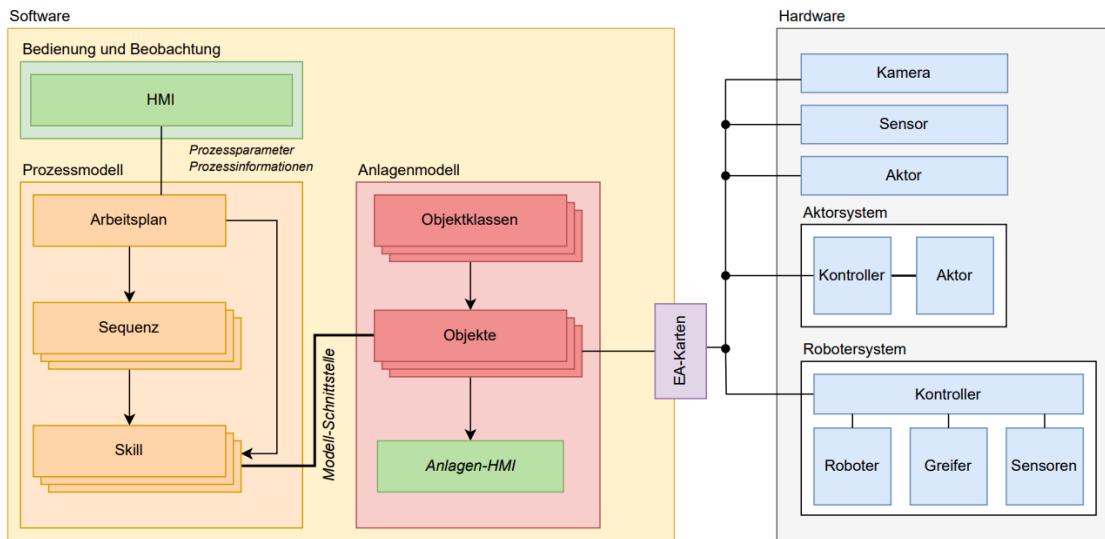


Abbildung 5.1: Strukturübersicht

Wie durch die ANSI/ISA-88-Norm vorgegeben (beschrieben in Kapitel 2.3.1), besteht die Software aus einem Prozess- und einem Anlagenmodell. Das Prozessmodell steuert den Ablauf, während das Anlagenmodell die Schnittstelle zu den einzelnen Anlagenkomponenten darstellt. Innerhalb des Prozessmodells werden die Skills definiert, die entweder von Sequenzen oder direkt aus dem Arbeitsplan zur Ablaufsteuerung genutzt werden können. Der Arbeitsplan beschreibt den gesamten Prozess.

Das Anlagenmodell implementiert die Objektklassen der verschiedenen Systemelemente und bildet deren Funktionalitäten ab. Die Struktur des Anlagenmodells ist klar und übersichtlich: Die Objektklassen werden instanziert und diese Objekte mit den entsprechenden Ein- und Ausgängen verknüpft, welche die Schnittstelle zum realen System darstellen. Der Grundgedanke dabei ist, dass alle Funktionalitäten zentral in der SPS gebündelt werden, um möglichst wenig Funktionalität auf den einzelnen Komponenten selbst zu beladen. Ziel ist es, dass sämtliche Elemente, von Robotern bis zu Kameras, über die SPS gesteuert werden können. Voraussetzung dafür ist, dass alle Komponenten über eine funktionale Schnittstelle zur SPS verfügen. Alle Komponenten werden im Anlagen-HMI visualisiert und können dort manuell gesteuert werden. Dies ermöglicht es beispielsweise, Roboterpositionen zu speichern, die später von einem Skill für Bewegungsabläufe genutzt werden können.

Abschliessend gibt es eine Bedienungs- und Beobachtungsebene, die als Benutzerschnittstelle dient, um Prozessparameter einzugeben und Informationen über den Prozess anzuzeigen. Diese Systemstruktur ermöglicht auch den modularen Aufbau von Systemen (Abb. 5.2). Das folgende Schema zeigt, wie ein solches System aussehen könnte:

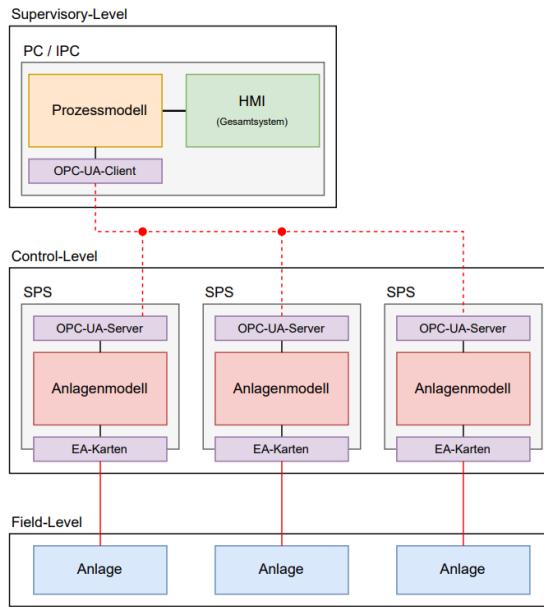


Abbildung 5.2: Gesamtsystemstruktur

Das Schema orientiert sich an der Automatisierungspyramide [19] und zeigt die ersten drei Ebenen (Supervisory-, Control- und Field-Level). Dabei handelt es sich um ein System, welches aus verschiedenen Teilanlagen bestehen, mit eigener SPS und Anlagenmodell. Auf dem Supervisory-Level befindet sich das Prozessmodell. Hier wird über die HMI ein Arbeitsplan ausgewählt / zusammengestellt und gestartet. Eine OPC-UA-Schnittstelle übermittelt die prozessrelevanten Daten an die entsprechende SPS im Control-Level. Die SPS steuert die Anlage im Field-Level basierend auf dem Anlagenmodell. Die verschiedenen Anlagen können flexibel für unterschiedliche Aufgaben genutzt werden oder, falls sie identische Fähigkeiten besitzen, nach Auslastung zugewiesen werden. Dadurch ist das System äusserst flexibel und kann ohne grossen Aufwand erweitert werden.

5.1 Schnittstellen innerhalb der Software

Die Software besteht aus den drei Hauptelementen: Bedienung und Beobachtung (HMI), Prozessmodell und Anlagenmodell (Abb. 5.1). Um die Schnittstellen zwischen diesen Elementen zu definieren, müssen die Kompetenzen klar definiert werden. Wer ist für was verantwortlich und welche Informationen werden dafür benötigt.

Bedienung und Beobachtung:

Über das HMI wird der Arbeitsplan erstellt und mit den erforderlichen Ablaufparameter versehen. Das System sowie der erstellte Arbeitsplan können über das HMI gestartet, gestoppt und gesteuert werden.

Prozessmodell:

Das Prozessmodell koordiniert die Ausführung des Arbeitsplans. Zusammen mit den Ablaufparametern werden die Prozessparameter festgelegt und weitergegeben. Der Arbeitsplan wird in einzelne Skills unterteilt, die wiederum die grundlegenden Funktionen der Anlagenkomponenten ausführen. Die effiziente und standardisierte Definition eines Skills, vereinfacht die Anwendung dieser in den Sequenzen und Arbeitsplänen.

Anlagenmodell:

Das Anlagenmodell bildet die Funktionalität der Systemkomponenten ab. Es stellt die Funktionen durch Methoden dar, während Zustände und Prozessinformationen über Eigenschaften wiedergegeben werden. Das Anlagenmodell verarbeitet die erhaltenen Prozessparameter und übersetzt diese in Anlagenparameter, mit denen die Komponenten der Anlage betrieben werden. Die vom Anlagenmodell zurückgemeldeten Daten (wie Zustände und Messwerte) werden als Zustandsparameter bezeichnet.

Die drei Hauptelemente müssen voneinander abgegrenzt werden (Abb. 5.3). Der modellbasierte Ansatz der Softwarestruktur bietet dabei mehrere Vorteile (siehe EVA-Referenz). Dank klarer Struktur und Übersichtlichkeit lassen sich die Prozesse und Abläufe leicht nachvollziehen. Dies erleichtert die Kommunikation, da durch die einheitliche Verwendung von Begriffen alle dieselbe «Sprache» sprechen. Darüber hinaus können Risiken früher und einfacher erkannt werden.

Die Abgrenzung der Hauptelemente geschieht über die Schnittstellen zwischen diesen. Die Schnittstellen werden durch die verschiedenen Parameter definiert. Der Begriff Parameter ist dabei ein Sammelbegriff für alle definierten Variablen, welche zwischen den Elementen ausgetauscht werden.

Prozessparameter:

Die Prozessparameter beschreiben den Prozess auf eine möglichst einfache Weise. Es werden nur Informationen weitergegeben, welche nötig sind um den Prozess eindeutig zu definieren. Die Prozessparameter hängen dabei von den Skills und deren Fähigkeiten ab. Die Parameter werden während der Erarbeitung des Prozessmodells definiert.

Ablaufparameter:

Ablaufparameter werden durch die Skills definiert und legen relevante Parameter für den Ablauf fest. Die Informationen sind jedoch nicht konkret auf die Komponenten im System ausgelegt. Die Ablaufparameter sind noch anlagenunabhängig und hängen z.B. nicht vom Typ des Roboters ab, welcher im System eingesetzt wird. Die genauen Parameter werden während der Erarbeitung des Prozessmodells definiert.

Anlagenparameter:

Die Parameter, welche von den instanzierten Objekten im Anlagenmodell vorbereitet werden, dienen als Schnittstelle zur realen Anlage und sind anlagenspezifisch. Die Art dieser Parameter hängt von den eingesetzten Komponenten im System ab. Die genauen Parameter werden während der Erarbeitung des Anlagenmodells definiert.

Zustandsparameter:

Die durch die instanzierten Objekte ausgewerteten und verarbeiteten Anlagenparameter werden als Zustandsparameter an das Prozessmodell zurückgegeben. Auf diese Parameter reagiert der Skill wie auch das System. Die genauen Parameter werden während der Erarbeitung des Anlagenmodells definiert.

Systemparameter:

Systemparameter sind systemübergreifende Parameter, welche zur Bedienung des gesamten Systems verwendet werden oder dessen Zustand darstellen.

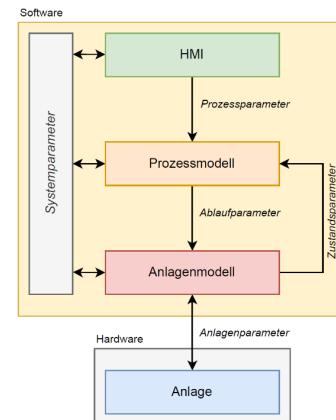


Abbildung 5.3: Systemstruktur

5.2 Interaktion innerhalb der Software

Um die Schnittstellen innerhalb der Software besser zu verstehen, muss die Interaktion zwischen Systemparameter, Prozessmodell (Skills) und Anlagenmodell (Objekten) abgegrenzt sein, da diese die wichtigsten Schnittstellen im System darstellen (Abb. 5.4). Die Interaktion findet dabei mit 3 Schnittstellen statt.

Objektschnittstelle:

Die Objektschnittstelle regelt die Interaktion zwischen den Systemparametern und den Objekten des Anlagenmodells. Die Systemparameter steuern dabei die grundlegenden Funktionen der Objekte, wie Ein- und Ausschalten, Zurücksetzen oder Stoppen. Diese Basisfunktionen werden nicht durch die Skills aktiviert, entsprechend bleibt deren Aufgabe auf die Verwaltung des Prozesses beschränkt. Dies ist besonders sinnvoll, da ein Objekt mehrere Skills besitzen kann und so Fragen zur Berechtigung der Skills vermieden werden. Im Gegenzug stellen die Objekte den Systemparametern Informationen über ihren Zustand und Fehler zur Verfügung.

Modellschnittstelle:

Die Modellschnittstelle ist für die Interaktion zwischen Prozess- und Anlagenmodell zuständig, genauer gesagt zwischen Skills und Objekten. Die Skills schicken Ablaufparameter an das Objekt, auf welche das Objekt reagiert. Das Objekt übergibt den aktuellen Zustand. Zusätzlich werden auch Messwerte vom Objekt an den Skill übergeben.

Koordinationsschnittstelle:

Die Koordinationsschnittstelle ist für die allgemeine Prozesskoordination verantwortlich. Es werden Information über den aktuellen Zustand und Fehler des Skills an die Systemparameter übergeben. Der Skill erhält den aktuellen Zustand des Systems. Der Skill kann somit auf systemübergreifende Situationen reagieren und das System kann auf Skill-Zustände reagieren.

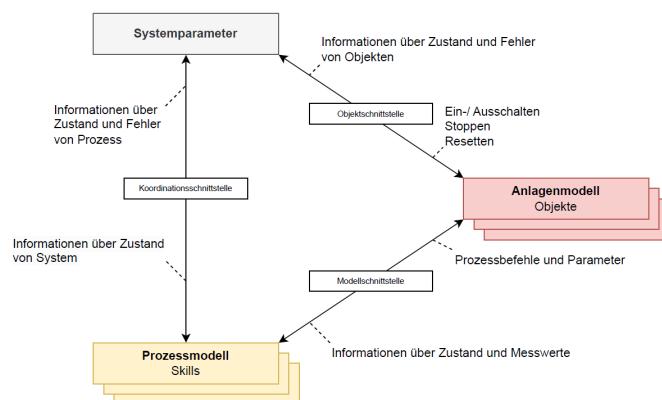


Abbildung 5.4: Schnittstellenübersicht

Die Schnittstellen und deren Abgrenzung dienen als Grundlage für die Bestimmung der Zustände. Dabei werden die Zustände für das System, die Skills und die Objekte bestimmt. Die System- und Objektzustände sind entscheidend für die grundlegende Struktur der Skills, da diese auf die jeweiligen Zustände reagieren müssen. Folglich stellen die definierten System- und Objektzustände lediglich die Mindestanforderungen dar, die notwendig sind, um eine Interaktion mit den Skills zu ermöglichen. Bei der Entwicklung der jeweiligen Systemelemente (Prozessmodell & Anlagenmodell) können noch weitere Zustände dazu kommen.

System:

Das System besitzt mindestens folgende 5 Zustände:

Zustand:	Beschreibung:
0 AUS	Das System ist ausgeschaltet (Startzustand)
1 BEREIT	Das System ist eingeschaltet und bereit einen Prozess durchzuführen
2 LAUFEND	Ein Prozess wird ausgeführt
3 GESTOPPT	Ein Prozess wurde gestoppt
4 FEHLER	Es gibt einen Fehler im System

Tabelle 5.1: Minimale Systemzustände

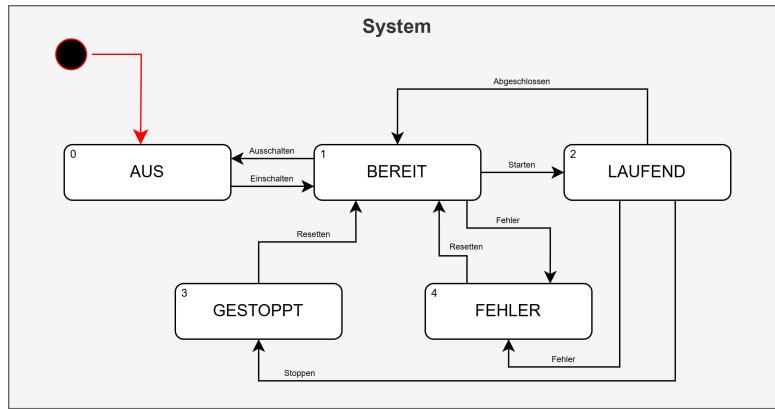


Abbildung 5.5: Minimale Systemzustände

Skill:

Ein Skill besitzt 6 Zustände:

Zustand:	Beschreibung:
0 BEREIT	Der Skill ist bereit einen Prozess auszuführen (Startzustand)
1 LAUFEND	Der Skill führt einen Prozess aus
2 ABGESCHLOSSEN	Der Prozess wurde abgeschlossen (Durch Objekt)
3 ERREICHT	Prozessziel wurde erreicht / Prozess beendet (Durch Skill)
4 LIMIT	Grenzwert wurde überschritten und Prozess wurde abgebrochen
5 FEHLER	Es gibt einen Fehler bezüglich des Prozesses

Tabelle 5.2: Minimale Skillzustände

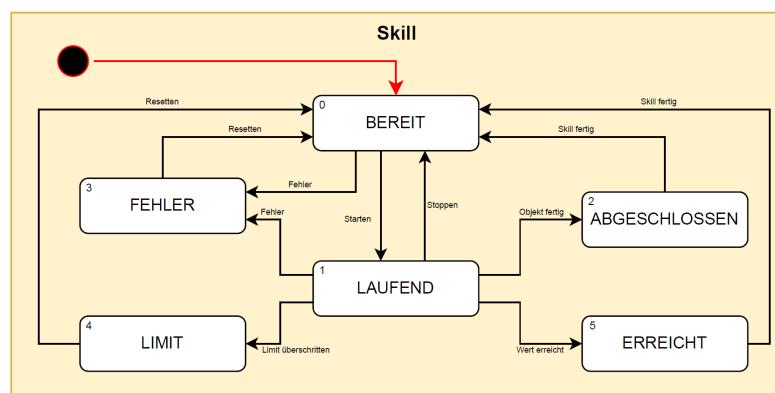


Abbildung 5.6: Minimale Systemzustände

Objekt:

Ein Objekt benötigt mindestens folgende 7 Zustände:

Zustand:	Beschreibung:
0 AUS	Das Objekt ist ausgeschaltet (Startzustand)
1 BEREIT	Das Objekt ist eingeschaltet und bereit
2 LAUFEND	Das Objekt ist aktiv
3 ABGESCHLOSSEN_INTERN	Prozess wurde abgeschlossen (Durch Objekt)
4 ABGESCHLOSSEN_EXTERN	Prozess wurde abgeschlossen (Durch Skill)
5 GESTOPPT	Prozess wurde gestoppt (Durch System)
6 FEHLER	Es gibt einen Fehler bezüglich des Objektes

Tabelle 5.3: Minimale Objektzustände

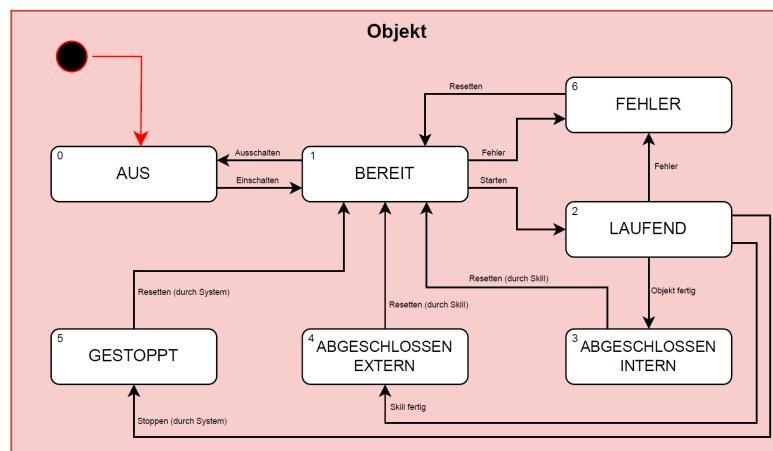


Abbildung 5.7: Minimale Objektzustände

Die Definition der allgemeinen Software-Struktur und ihrer Interaktionen bildet die Grundlage für die Ausarbeitung der detaillierten Struktur, Funktionsweise und Implementierung der Skills sowie des Anlagenmodells.

6 Entwicklung der Skills

6.1 Kompetenzen von Skills

Für die Definition eines Skills wird nicht die Funktion des Gesamtsystem so weit wie möglich in Teilfunktionalitäten heruntergebrochen, sondern die Funktionalitäten der Komponenten im System. Dabei wird jeder Komponenten-Typ soweit wie möglich einzeln betrachtet (Abb. 6.1). Der Vorteil dieser Definition ist, dass für neue oder angepasste Systeme dieselben Skills eingesetzt werden können. Ein Robotersystem hat in jeder Anlage die identischen Grundfunktionalitäten und somit Skills. Der Skill ist anwendungsunabhängig. Die aus den Skills erstellten Sequenzen bilden die Funktion der Anwendung ab. Damit unterscheidet sich die Definierung der Skills von vergleichbaren Projekten wie z.B. der bereits durchgeföhrten Master-Thesis auf Basis von ROS (beschrieben in Kapitel 2.4). Dort haben Skills Funktionalitäten von mehreren Komponenten ineinander kombiniert, wodurch der Skill stärker an das spezifische System gebunden war.

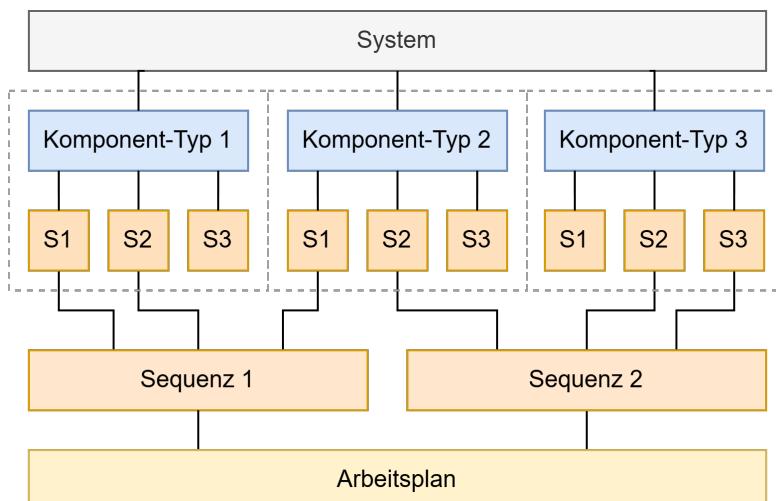


Abbildung 6.1: Skill-Integration in System

Die Kompetenzen eines Skills lassen sich in vier Bereiche aufteilen:

- | | |
|---------------|---|
| Zuweisung: | Der Skill ist für die Zuweisung einer Komponente zuständig. Es muss definiert werden können, welche Komponente den Skill ausführt. |
| Umsetzung: | Die definierte Grundfunktion muss innerhalb des Skills umgesetzt werden. Der Skill muss mittels Parameter-Inputs flexibel eingesetzt werden können. |
| Verarbeitung: | Die Informationen der Grundfunktion wird so ausgegeben, dass das Anlagenobjekt damit arbeiten und die reale Komponente ansteuern kann. |
| Auswertung: | Die momentane Situation der Komponente wird überwacht und ausgewertet. Der Skill kann auf bestimmte Situationen reagieren. |

6.2 Definition von Skills für Anwendung

Um die benötigten Skills für die Anwendung zu definieren, werden im ersten Schritt die allgemeinen Arbeitsschritte (Tab. 6.1) basierend auf dem mechanischen Aufbau (siehe Kapitel 4.3) festgelegt. Dabei wird von der Ausgangssituation ausgegangen, dass alle Teile in ihren Lagerpositionen abgelegt sind, der Roboter sich in der Home-Position befindet und alle Komponenten eingeschaltet sowie betriebsbereit sind.

Schritt:	Beschreibung:	Komponente:
1	Position von Platte 1 in Lagerung erkennen	K
2	Platte 1 mittels L-Stück ausrichten	R, G, K
3	Position von Platte 2 in Lagerung erkennen	K
4	Platte 2 und Platte 1 zusammenführen	R, G, K
5	Position der Befestigungslöcher in den Platten ermitteln	K
6	Position von Befestigungsblech in Lagerung erkennen	K
7	Befestigungsblech an Montageposition bringen	R, G, K
8	Position von Stift 1 in Lagerung erkennen	K
9	Stift 1 an korrekte Position bringen	R, G
10	Befestigungsblech mit Platte 1 verbinden (mit Stift)	R, G
11	Wiederholen von Schritt 8 – 10 für Stift 2 bis 3	

(K = Kamerasystem | R = Roboter | G = Greifer)

Tabelle 6.1: Arbeitsschritte

Eine detaillierte Auflistung der Arbeitsschritte wird im Anhang beigefügt. Aus diesem lässt sich erkennen, dass sich diverse Schritte mit kleinen Anpassungen wiederholen. Diese sich wiederholenden Arbeitsschritte definieren die Skills. Ein entscheidender Aspekt dabei ist, dass der Roboter und der Kraftsensor als separate Komponenten betrachtet werden. Der Kraftsensor erweitert die Fähigkeiten des Roboters zwar und damit dessen Skills, jedoch kann der Roboter auch ohne Kraftsensor betrieben werden. Der Kraftsensor wird als eigene Objektklasse abgebildet, jedoch besitzt dieser keinen eigenen Skill. Folgende Skills wurden definiert, welche den Prozess abdecken:

Komponenten:	Skill:	Bemerkung:
Kamerasystem	- Bild aufnehmen - Objekt erkennen - Greiferposition ermitteln	
Roboter	- Position anfahren - Kontrolliert bewegen	Bei Kraftüberschreitung wird gestoppt
Greifer (mit Sensor)	- Backenposition anfahren	

(Kamerasystem = Kamera + Vision)

Tabelle 6.2: Definierte Skills für Anwendung

6.3 Definierung der Skill-Struktur

Alle Skills sollten nach einer einheitlichen Struktur aufgebaut sein und lediglich um die prozessspezifischen Funktionen ergänzt werden. Ein zentraler Bestandteil dieser Grundstruktur sind die Schnittstellen, die ein Skill mindestens benötigt, um innerhalb der Systemstruktur funktionsfähig zu sein. Dazu zählen nicht nur Eingangs- und Ausgangsvariablen, sondern auch Eigenschaften und Methoden, da Informationen ebenfalls über diese übertragen werden können. Ein Skill muss über die definierten Schnittstellen (siehe Kapitel 5.2) sowohl mit dem System, mit dem Objekt und innerhalb des Prozessmodells interagieren können.

Innerhalb der Entwicklung wurden mehrere Iterationen von Skill-Strukturen geplant, umgesetzt und getestet, um die bestmögliche Struktur zu finden. Innerhalb dieser Dokumentation werden nicht alle Iterationsschritte erklärt, sondern nur die daraus folgenden Erkenntnisse.

Die Schnittstellen eines Skills wurden in folgende drei Kategorien aufgeteilt, welche das allgemeine Verständnis einfacher machen sollen.

Steuerungselemente:

Die Steuerungselemente sind für die Bedienung des Skills angedacht. Hier wird der Skill gestartet, gestoppt oder resettet. Zusätzlich werden auch Informationen über den Zustand des Skills angegeben.

Betriebselemente:

Die Betriebselemente sind für den allgemeinen Betrieb des Skills angedacht, welche nicht mit dem spezifischen Prozess zu tun haben. Dies ist z.B. die Schnittstelle zum System, welche eine Aussage über den Systemzustand macht.

Prozesselemente:

Die Prozesselemente sind spezifische Informationen, welche der Skill für die Ausführung benötigt und entsprechend an das Objekt weitergibt.

Zu Beginn wurden die Steuerungselemente auf Basis des PLC-Standards aufgebaut (beschrieben in Kapitel 2.4). Der Skill wurde dabei von einer BOOL-Variable (`bExecute`) gestartet und hat diverse Informationen als Ausgangsvariablen ausgegeben (`bDone`, `bBusy`, `bLimit`, `bError` und `iErrorID`). Die Ausgangsvariablen konnten als Transition-Bedingungen für Abläufe verwendet werden. Auf den ersten Blick ist dies eine einfache und sinnvolle Steuerung des Skills. Jedoch zeigte sich bei ersten Versuchen, dass diese Implementierung diverse Probleme mit sich bringen kann. Die Umsetzung eines `bExecute`-Triggers ist aufwändig und muss gut durchdacht werden, da der Skill nur einmal ausgeführt werden soll. Wenn der Skill beendet wurde und die `bExecute`-Variable immer noch betätigt ist, soll der Skill nicht ein zweites Mal gestartet werden. Der Trigger muss entsprechend so umgesetzt werden, dass dieser nur auf eine steigende Flanke reagiert. Das Management der Ausgangsvariablen ist geknüpft an die verschiedenen Zustände innerhalb des Skills. Es muss genau bestimmt werden, welcher Zustand einen Einfluss auf die Ausgänge hat. Durch die hohe Anzahl an Ausgangsvariablen kann man hierbei schnell den Überblick verlieren. Zusätzlich werden viele der Informationen, welche über die Ausgangsvariablen dargestellt werden, auch über den Zustand dargestellt. Die Konsequenz daraus ist, dass die Steuerungselemente mit Methoden und Eigenschaften umgesetzt werden (Tab. 6.3).

Art:	Bezeichnung:	Typ:	Beschreibung:
Methode	M_Start	BOOL	Methode zum Starten des Skills
Methode	M_Stop	BOOL	Methode zum Stoppen des Skills
Methode	M_Reset	BOOL	Methode zum Resetten des Skills
Eigenschaft	P_State	eSkillState	Eigenschaft des aktuellen Zustandes

Tabelle 6.3: Steuerungselemente eines Skills

Die Eigenschaft wird mit einem benutzerdefinierten ENUM-Datentyp umgesetzt (Tab. 6.4), welche die Zustände des Skills abbildet. Somit ist immer klar, in welchem Zustand sich der Skill im Moment befindet.

Listen-Nr:	eSkillState:
0	BEREIT
1	LAUFEND
2	ABGESCHLOSSEN
3	FEHLER
4	LIMIT
5	ERREICHT

Tabelle 6.4: Definition von eSkillState

Auch bei den Betriebselementen gab es im Verlauf der Entwicklung Veränderungen. Zu Beginn enthielten diese alle Variablen, die für den Betrieb des Skills erforderlich waren und dienten als Schnittstelle zwischen System und Objekt. Relevante Informationen für den Prozess wurden dabei über Ausgangsvariablen an das Objekt übergeben, welches diese wiederum als Eingangsvariablen entgegennahm.

Wenn jedoch zwei Skills auf dasselbe Objekt zugreifen sollen (nicht gleichzeitig) und beide mit dessen Eingangsvariablen verbunden sind, überschreibt stets einer der beiden Skills den Wert des anderen. Da es durchaus vorkommt, dass mehrere Skills mit einem Objekt interagieren (ebenfalls nicht gleichzeitig), muss die Interaktion zwischen Skill und Objekt so gestaltet sein, dass sie unabhängig von anderen Skills funktioniert.

Dies lässt sich durch die Instanzierung eines Interfaces erreichen. In der objektorientierten Programmierung wird ein Interface genutzt, um vorzugeben, welche Methoden und Eigenschaften ein Funktionsbaustein zwingend besitzen muss. Durch das Instanziieren eines Interfaces innerhalb eines Funktionsbausteins können dessen Methoden und Eigenschaften verwendet werden. Wenn eine Eingangsvariable mittels Interface instanziert wird, kann diese Funktionsbaustein-Eingangsvariable mit einem Objekt verbunden werden, insofern das selbe Interface beim Objekt implementiert wurde (Abb. 6.2). Beim Ausführen einer Methode innerhalb des Funktionsbausteins wird dadurch die entsprechende Methode im verknüpften Objekt aufgerufen. Gleiches gilt für die Eigenschaften. Dadurch kann z.B. der Zustand des Objektes über diese Schnittstelle abgefragt werden.

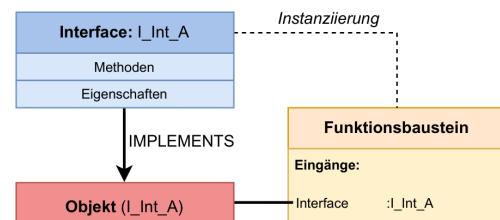


Abbildung 6.2: Schnittstellenstruktur

Somit wurden die Betriebselemente wie folgt definiert:

Art:	Bezeichnung:	Typ:	Beschreibung:
Eingangsvariabel	eSysCommand	eSystemCommand	Befehl von System zu Skill
Eingangsvariabel	eSysState	eSystemStatus	Informationen über System
Eingangsvariabel	fbObjekt	Objekt-Interface	Verknüpfung zu Objekt
Ausgangsvariabel	iErrorID	INT	Information über Fehler

Tabelle 6.5: Betriebselemente eines Skills

Da auch die Übergabe der Objektparameter an das Objekt über die Interface-Verknüpfung erfolgt (mittels einer Eigenschaft), wird für jede Art von Objekttyp ein eigenes Interface definiert, da Objekte unterschiedliche Parameter benötigen. Die Steuerungselemente für ein Objekt sind jedoch für alle Objekttypen einheitlich und ähnlich wie bei den Skills. Dazu gehören die Methoden «M_Start», «M_Stop», «M_Rest» sowie die Eigenschaft «P_State» (ObjectState).

Die folgenden benutzerdefinierten Datentypen sind für die Betriebselemente von Bedeutung:

Listen-Nr:	eSystemCommand:	ObjectState:	eSystemState:
0	KEINE	AUS	AUS
1	AUSSCHALTEN	BEREIT	BEREIT
2	EINSCHALTEN	MANUELL	LAUFEND
3	STOPPEN	LAUFEND	GESTOPPT
4	RESETTEN	ABGESCHLOSSEN_INTERN	FEHLER
5	/	ABGESCHLOSSEN_EXTERN	/
6	/	GESTOPPT	/
7	/	FEHLER	/

Tabelle 6.6: Benutzerdefinierte Datentypen

i Beim Datentyp «ObjectState» wird unterschieden zwischen einem Aktor oder einem Sensor. Die dargestellten Zustände beziehen sich auf einen Aktor. Innerhalb von Kaptiel 7 wird detaillierter auf den Unterschied zwischen Aktoren und Sensoren eingegangen.

Die letzte Kategorie umfasst die Prozesselemente. Die Prozesseigenschaften dienen dem bereitstellen von prozessrelevanten Informationen. Die vorhandenen Prozessmethoden werden vom Skill intern genutzt, beispielsweise zur Datenverarbeitung oder Auswertung.

Die Grundstruktur eines Skills lässt sich anhand des folgenden Schemas zusammenfassen (Abb. 6.3). Dieses Schema veranschaulicht auch die Interaktion des Skills über die verschiedenen Schnittstellen im System.

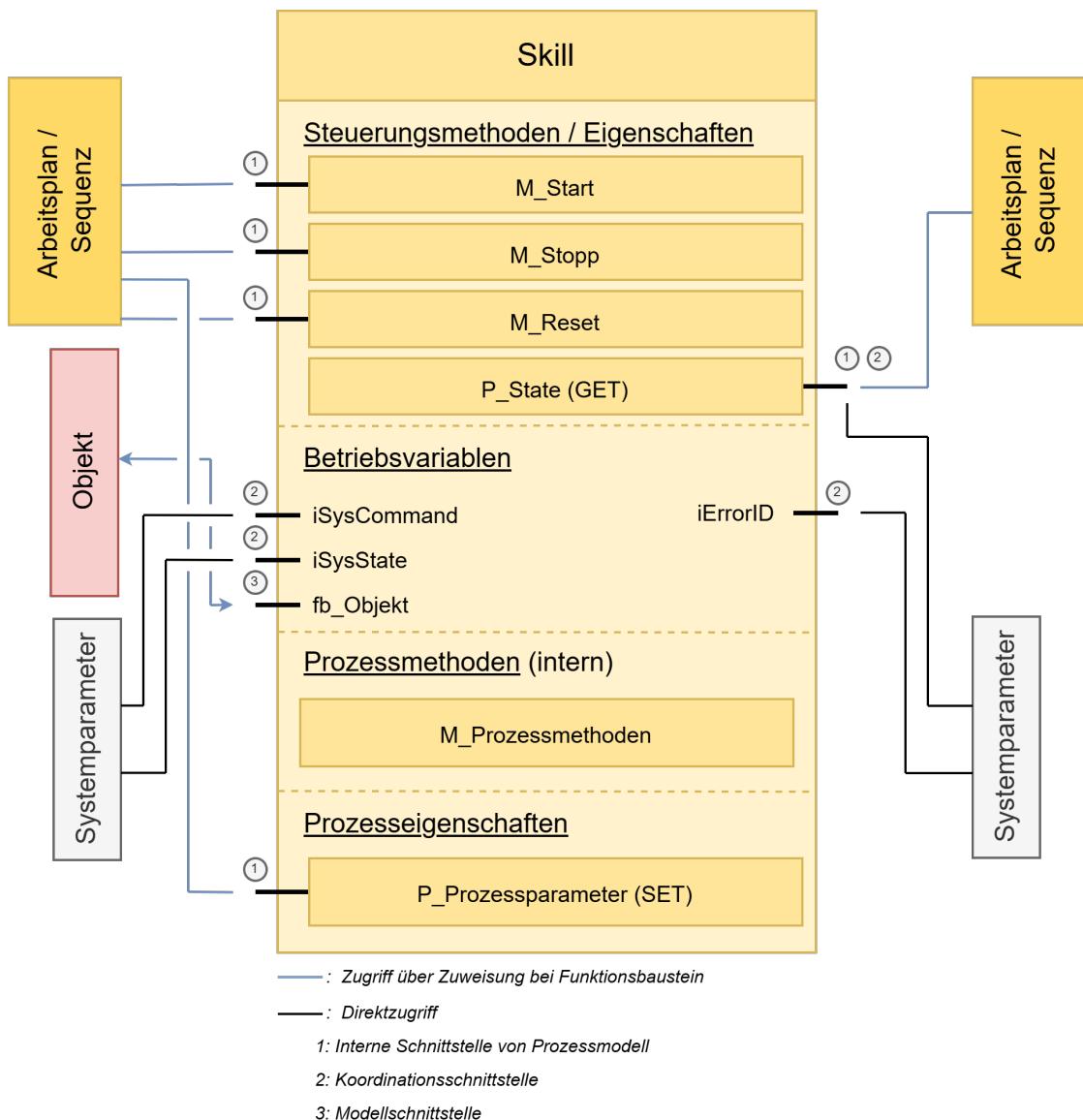


Abbildung 6.3: Skilldefinition und Schnittstellen

6.4 Konfiguration eines Skills

Nicht jeder Skill benötigt alle Zustände. Damit der Skill so übersichtlich wie möglich bleibt, sollen auch nur die Zustände verwendet werden, welche benötigt werden. Ein Skill kann drei Konfigurationen einnehmen (Abb. 6.4).

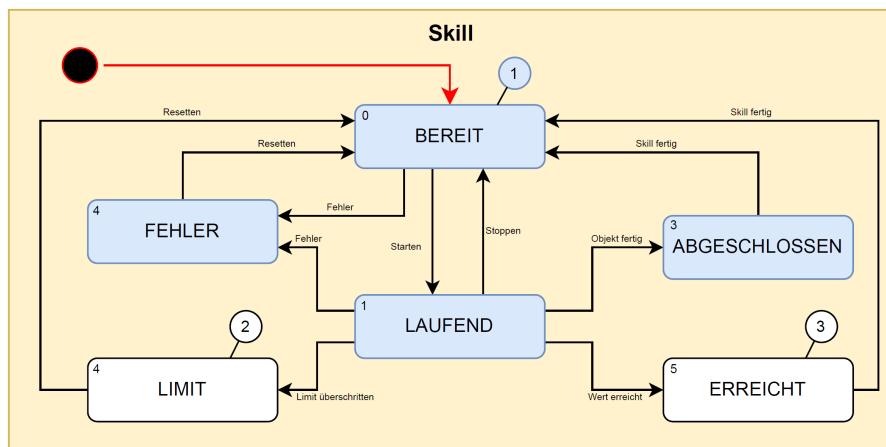


Abbildung 6.4: Skillkonfiguration

- Konfiguration 1: Stellt die Grundkonfiguration dar. Jeder Skill muss diese Zustände implementieren. Hierbei handelt es sich um Skills, welche nur durch das Objekt abgeschlossen werden, z.B. eine einfache Punkt-Zu-Punkt-Bewegung des Roboters.
- Konfiguration 2: Bei dieser Konfiguration kommt der LIMIT-Zustand dazu. Dieser wird benötigt, wenn es eine Limit-Bedingung gibt, z.B. einen Grenzwert für die Kraft oder eine maximale Zeitdauer.
- Konfiguration 3: Bei der letzten Konfiguration wird der ERREICHT-Zustand ergänzt. Dieser gibt an, ob ein definiertes Ziel erreicht wurde, dies kann z.B. eine Kraft sein.

6.5 Interaktion zwischen Skill und Objekt

Anhand eines Beispiels (6.5) soll die Interaktion zwischen einem Skill und den zugehörigen Objekten erläutert werden. Der Skill «Skill_P2P» ermöglicht es einem Roboter, zu einem definierten Punkt zu fahren. Falls der Roboter dabei mit einem Hindernis kollidiert, wird der Roboter durch den Skill gestoppt. Hindernisse werden dabei mithilfe eines Kraftsensors erkannt.

Der Skill wird über Eingangsvariablen mit den entsprechenden Objekten verbunden, was durch eine blaue Verbindungsleitung dargestellt wird. Über die Sequenz werden verschiedene Informationen an den Skill übergeben. Dazu zählen:

- Die anzufahrende Position des Roboters.
- Die Geschwindigkeit und Beschleunigung.
- Die Art der Bewegung bzw. Positionierung.
- Die Kraftgrenze, die ein Hindernis definiert.

Der Start des Skills erfolgt durch die Start-Methode, die von der Sequenz ausgelöst wird. Der Skill selbst initiiert über die Start-Methode des Aktor-Objekts den Bewegungsprozess und liest parallel dazu mithilfe der Messdaten-Eigenschaft die Daten des Sensor-Objekts aus. Die Objekte haben dabei ausschliesslich die Aufgabe, ihre spezifischen Funktionen auszuführen, die durch ihre jeweilige Komponente definiert sind. Sie enthalten keine Logik in Bezug auf das Gesamtsystem – diese liegt vollständig im Skill.

Der Skill analysiert die übermittelten Daten und reagiert entsprechend darauf. Im geschilderten Beispiel stoppt der Skill den Aktor, sobald die gemessenen Daten die vorgegebenen Schwellenwerte überschreiten.

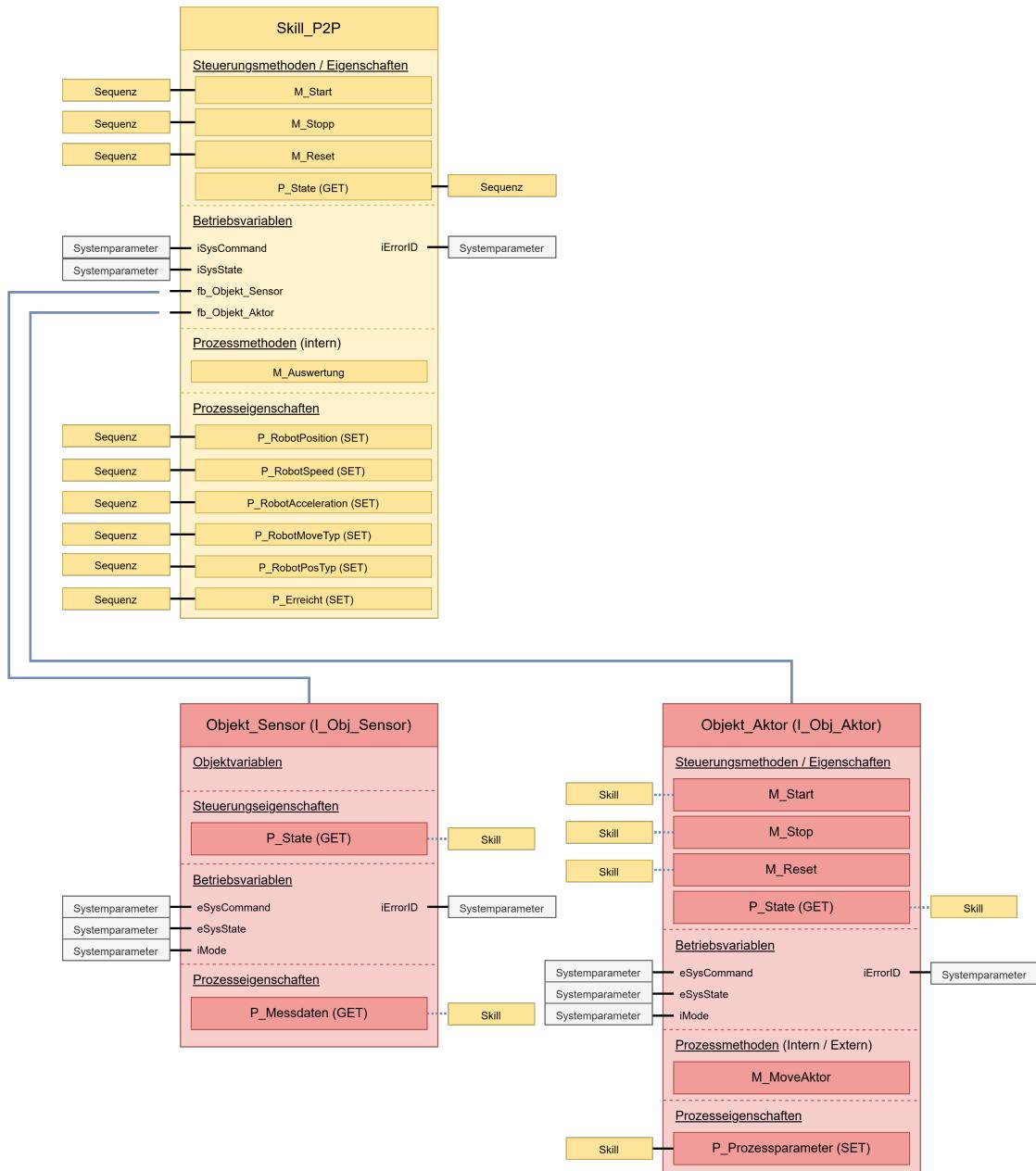


Abbildung 6.5: Skill-Interaktion

6.6 Umsetzung in TwinCAT

6.6.1 Allgemeine Struktur

Die Programmierung der Skills soll einfach und übersichtlich gehalten werden. Dafür wird mit Vererbungen von Funktionsbausteinen (Abb. 6.6) gearbeitet. Über eine Vererbung können Methoden und Eigenschaften (inkl. ihrer Funktionalität), sowie Variablen Deklaration von einem Funktionsbaustein übernommen werden. Dadurch kann ein Basis-Funktionsbaustein erstellt werden, welcher Methoden, Eigenschaften und Variablen zur Verfügung stellt, welche für alle Skills verwendet werden. Diese Elemente werden bei jeder Vererbung separat zur Verfügung gestellt. Mehrere Skills greifen nicht auf dieselben Elementen zu und beeinflussen sich somit nicht.

Da jedoch die Steuerungselemente auf das instanzierte Objekt zugreifen sollen (Abb. 6.5), können diese nicht in einem Basis-Funktionsblock abgebildet und vererbt werden. Für die Objekte gibt es verschiedene Interfaces, je nach benötigten Daten für den Prozess. Entsprechend werden in unterschiedlichen Skills auch unterschiedliche Objekt-Interfaces benötigt. Dadurch kann das Objekt nicht bereits im Basis-Funktionsbaustein instanziert werden und die Methoden und Eigenschaften können nicht darauf zugreifen. Aus diesem Grund, müssen «M_Start», «M_Stop», «M_Reset» und «P_State» im Skill selbst implementiert werden. Die Vererbung wird in diesem Fall nur für Variablen genutzt, welche für alle Skills gleich bleiben.

In TwinCAT wird eine Vererbung über den Zusatz «EXTENDS» durchgeführt. Für den Basis-Funktionsbaustein «FB_Basis_Skill» wurden die Variablen wie folgt definiert:

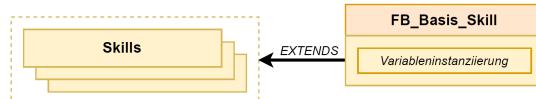


Abbildung 6.6: Umsetzungsstruktur

```

FUNCTION_BLOCK FB_Basis_Skill
VAR_INPUT
    // Betriebsvariablen
    eSysCommand      :eSystemCommand;           // Systembefehl an Skill
    eSysState         :eSystemState;             // Momentaner Zustand des Systems
END_VAR
VAR_OUTPUT
    iErrorID          :INT;                    // Fehlercode für System
END_VAR
VAR
    // Managementvariablen
    iState            :eSkillState;             // Information über Zustand von Skill
    bGestartet        :BOOL;                   // Skill wurde gestartet
    bGestoppt         :BOOL;                   // Skill wurde gestoppt
    bSkillAktiv       :BOOL;                   // Skill ist im Moment aktiv

    // Zustandsvariablen (Transitions)
    bStarten          :BOOL;
    bStoppen          :BOOL;
    bObjektFertig     :BOOL;
    bWertErreicht     :BOOL;
    blimitErreicht    :BOOL;
    bSkillFertig      :BOOL;
    bFehler           :BOOL;
    bResetten         :BOOL;
END_VAR

```

Abbildung 6.7: Definitionen für Skill-Basis-FB

6.6.2 Aufbau des Grund-Skills

Alle Skills sollten nach einem einheitlichen Grundmuster aufgebaut werden. Dabei wird ein Skill als Funktionsblock mit strukturiertem Text realisiert. Die Verwendung von strukturiertem Text bietet hohe Flexibilität bei der Umsetzung.

Jeder Skill hat folgende drei Steuerungsmethoden:

```
// Prüfen von Bedingungen
// Ist das System im korrekten Status
IF eSysState = 1 OR eSysState = 2 THEN
    // Ist der Skill im korrekten Status
    IF iState = 0 THEN
        // Prüfen auf Fehler
        IF ParameterError OR PositionError THEN
            bFehler := TRUE;
        ELSE
            Objekt.P_Prozessparameter := eRoboterPos;
            Objekt.M_Start();
            bSkillActiv := TRUE;
        END_IF
    END_IF
END_IF
```

(a) Methode: Start

```
Objekt.M_Stop();
bSkillActiv := FALSE;
```

(b) Methode: Stop

```
Objekt.M_Reset();
bSkillActiv := FALSE;
```

(c) Methode: Reset

Abbildung 6.8: Steuerungsmethoden eines Skills

Innerhalb der Methode «M_Start» (Abb. 6.8a) wird in einem ersten Schritt geprüft, ob sich das System im korrekten Zustand befindet. Danach wird der Zustand des Skills geprüft. Da die Methode von aussen (z.B. durch die Sequenz) ausgelöst wird, darf der Skill nur im BEREIT-Zustand gestartet werden. Zuletzt werden die Parameter überprüft, falls diese Auflagen erfüllen müssen. Falls alle Bedingungen erfüllt werden, werden die Prozessparameter (in diesem Fall eine Roboterposition) an das Objekt übergeben und die Start-Methode des Objektes wird ausgelöst. Die Stop- und Reset-Methode lösen die entsprechenden Methoden beim Objekt aus (Abb. 6.8b / Abb. 6.8c).

Der Aufbau eines Skills gliedert sich in drei Hauptbereiche: Fehlerüberwachung, Aufruf interner Steuerungsmethoden und Zustandsmanagement. Im Bereich des Aufrufs interner Steuerungsmethoden wird überprüft, ob das System vorgibt, den Skill zu stoppen oder zurückzusetzen (Abb. 6.9). In diesen Fällen wird entweder die Skill-Methode «M_Stop» oder «M_Reset» ausgelöst. Zudem wird geprüft, ob der Skill ein definiertes Limit oder einen Zielwert erreicht. Auch hier führt dies zur Ausführung der Methode «M_Stop».

```
// Interner Steuerungsmethodenaufruf:
// Stoppen
IF eSysCommand = 3 THEN
    bStoppen := M_Stop();
ELSE
    bStoppen := FALSE;
END_IF

// Resetten
IF eSysCommand = 4 THEN
    bResetten := M_Reset();
ELSE
    bResetten := FALSE;
END_IF

// Limit
IF bLimit THEN
    bLimitErreicht := M_Stop();
ELSE
    bLimitErreicht := FALSE;
END_IF

// Erreicht
IF bErreicht THEN
    bWertErreicht := M_Stop();
ELSE
    bWertErreicht := FALSE;
END_IF
```

Abbildung 6.9: Interner Steuerungsmethodenaufruf

Unter Zustandsmanagement werden die, in Kapitel 5.2 definierten Zustände des Skills definiert und umgesetzt. Für die Umsetzung wurde mit einer CASE-Anwendung gearbeitet. Die meisten Zustände sind für alle Skills identisch, jedoch beim Zustand «LAUFEND» gibt es kleinere Unterschiede.

Zustand 0 - BEREIT:

Innerhalb des Zustandes prüft der Skill den Zustand des Objektes. Falls das Objekt durch «M_Start» gestartet wurde und sich nun im Zustand «LAUFEND» befindet, so wechselt auch der Skill auf den Zustand «LAUFEND». Der Skill darf jedoch nur dann auf den Objektzustand reagieren, falls der Skill den Start des Objektes auch ausgelöst hat, da unter Umständen mehrere Skills auf ein Objekt zugreifen.

Zustand 1 - LAUFEND:

Der Skill kann auf verschiedene Arten gestoppt werden, welche innerhalb des Zustandes überprüft werden. Das System kann den Skill stoppen, welcher anschliessend direkt in den Zustand «BEREIT» wechselt. Falls der Objektzustand meldet, dass das Objekt den Prozess abgeschlossen hat (z.B. bei einer Punkt-zu-Punkt-Bewegung eines Roboters), wechselt der Skill in den Zustand «ABGESCHLOSSEN». Falls eine Limit- oder Erreicht-Bedingung erfüllt wurde, wird das Objekt durch den Skill gestoppt und wechselt in den entsprechenden Zustand. Während des Zustands «LAUFEND» können aber auch interne Methoden aufgerufen werden (z.B. für das Auswerten von Messwerten).

Zustand 2 - ABGESCHLOSSEN:

Der Zustand gibt den Befehl zum Resetten an das Objekt weiter. Sobald der Objektzustand meldet, dass sich das Objekt im Zustand «BEREIT» befindet, kehrt auch der Skill in den Zustand «BEREIT» zurück.

Zustand 3 - FEHLER:

Im Fehlerzustand wartet der Skill, dass dieser durch das System (eSystemCommand) resettet wird.

Zustand 4 - LIMIT:

Der Zustand gibt den Befehl zum Resetten an das Objekt weiter. Sobald der Objektzustand meldet, dass sich das Objekt im Zustand «BEREIT» befindet, kehrt auch der Skill in den Zustand «BEREIT» zurück.

Zustand 5 - ERREICHT:

Der Zustand gibt den Befehl zum Resetten an das Objekt weiter. Sobald der Objektzustand meldet, dass sich das Objekt im Zustand «BEREIT» befindet, kehrt auch der Skill in den Zustand «BEREIT» zurück.

6.6.3 Erarbeitete Skills

Zum Stand der Dokumentation wurden drei Skills umgesetzt, welche sich auf den Roboter (2 Skills) und den Greifer (1 Skill) beziehen.



Aus Zeitgründen wurde während der Erarbeitung entschieden, dass das Kamerasystem noch nicht in den Prozess integriert wird. Der Schwerpunkt dieser Arbeit liegt auf der Software-Struktur und wie die verschiedenen Elemente miteinander interagieren. Auch ohne Kamerasystem konnte die Struktur definiert, aufgebaut und getestet werden. Die Einarbeitung in das Vision-Tool von TwinCAT ist ein zeitintensiver Prozess, mit wenig Mehrwert für das Ziel dieser Arbeit. Im Rahmen einer weiterführenden Arbeit wäre die Integration des Kamerasystems aber ein relevanter Punkt.

Skill 1 (Roboter): Skill_Position_Anfahren

Mit dem Skill kann eine Position vom Roboter angefahren werden. Die Bewegung wird selbstständig vom Objekt abgeschlossen. Es ist nicht vorgesehen, dass der Skill die Bewegung stoppt. Damit der Skill ausgeführt werden kann, benötigt er folgende Prozessparameter, welche als Eigenschaften (SET) dem Skill übergeben werden:

Bezeichnung:	Typ:	Beschreibung:
P_RobotPosition	Array[0..5] of LREAL	Gibt die kartesischen Koordinaten oder Gelenkwinkel an, je nachdem welcher Bewegungstyp ausgewählt wurde.
P_RobotSpeed	LREAL	Gibt an, wie schnell sich der Roboter bewegen soll.
P_RobotAcceleration	LREAL	Gibt an, mit welcher Beschleunigung der Roboter arbeiten soll.
P_RobotMoveTyp	eRobPosTyp	Gibt den Bewegungstyp des Roboters an.
P_RobotPosTyp	eRobMoveTyp	Gibt den Positionstyp der angegebenen Position an.

Tabelle 6.7: Skill 1: Eigenschaften

Für den Bewegungstyp und Positionstyp wurden benutzerdefinierte ENUM-Datentypen angelegt. Eine detaillierte Erklärung ist bei der Beschreibung des Roboter-Objektes (Kapitel 7.2.3) vorhanden. Diese sind wie folgt definiert:

Listen-Nr:	eRobPosTyp:	eRobMoveTyp:
0	Absolut	LinearToolSpace
1	Relativ	LinearJointSpace
2	/	CircularToolSpace
3	/	BlendCircularMoveLinear

Tabelle 6.8: Benutzerdefinierte Datentypen für Roboter

Skill 2 (Roboter): Skill_Position_Anfahren_Erweitert

Ist ähnlich wie Skill 1, jedoch kann der Skill zusätzlich auf eine Kraft reagieren. Es ist möglich, einen Grenzwert für die Kraft und deren Richtung anzugeben. Der Skill stoppt das Objekt und somit die Bewegung, falls dieser Grenzwert überschritten wurde. Damit ist es möglich, Hindernisse zu erkennen, um z.B. einen Anschlag zu definieren. Damit der Skill ausgeführt werden kann, werden die Eigenschaften von Skill 1 mit folgender Eigenschaft ergänzt:

Bezeichnung:	Typ:	Beschreibung:
P_Erreicht	sKraftvariablen	Gibt den Grenzwert für die entsprechenden Richtungen an.

Tabelle 6.9: Skill 2: Eigenschaften

Der Datentyp «sKraftvariablen» ist eine Struktur, welche drei LREAL-Elemente besitzt. Diese stellen die Kraft in X-, Y- und Z-Richtung dar.

Skill 3 (Greifer): Skill_GreiferPos_Anfahren

Der letzte Skill bezieht sich auf den Greifer. Der Skill kann in Verbindung mit einem Greifer verwendet werden, welcher eine definierte Position anfahren kann. Die Bewegung wird selbstständig vom Objekt abgeschlossen. Damit der Skill ausgeführt werden kann, benötigt er folgende Prozessparameter, welche als Eigenschaften (SET) dem Skill übergeben werden:

Bezeichnung:	Typ:	Beschreibung:
P_GreifBreite	LREAL	Gibt an, welche Breite die Greifbacken anfahren sollen.
P_GreifGeschwindigkeit	LREAL	Gibt an, wie schnell sich die Greifbacken bewegen sollen.
P_GreifKraft	LREAL	Gibt an, mit welcher Kraft die Greifbacken zudrücken sollen.

Tabelle 6.10: Skill 3: Eigenschaften

6.6.4 Momentaner Stand

Die Skills wurden derzeit so entwickelt, dass die definierte Anwendung erfolgreich umgesetzt werden kann. Dabei lag der Fokus nicht darauf, bereits jede mögliche Situation abzudecken. Viel Zeit und Aufwand wurden in die Konzeption und das Testen der Skill-Struktur investiert. Diese Struktur bildet das stabile und zuverlässige Fundament, auf dem die Skills aufbauen. Sobald dieses Fundament steht, können die Funktionen des Skills schrittweise erweitert und optimiert werden.

Als potenzielle Erweiterung könnten zusätzliche Einstellungsmöglichkeiten integriert werden. Beim Greifer könnte beispielsweise ein Offset angegeben werden, der die Dicke der Bauchenaufsätze berücksichtigt. Dadurch liesse sich die angefahrene Breite der Greifbacken an unterschiedliche Aufsätze anpassen.

7 Entwicklung des Anlagenmodells

Die Objekte stellen die Komponenten in der Anlage dar und sollen die Funktionalitäten dieser abbilden. Ein Objekt wird immer nur für sich selbst betrachtet und besitzt möglichst keine Abhängigkeiten von anderen Objekten (Kann in gewissen Situation durchaus Sinn machen). Man baut alle Komponenten der Anlage aus und betrachte die Fähigkeiten dieser. Eine erstelle Objektklasse kann anschliessend für die Instanziierung eines Objektes verwendet werden. Dies ermöglicht das Erstellen von mehreren Objekten auf Basis einer Objektklasse.

7.1 Definition der Objekt-Struktur

Wie bereits für die Skills wird auch für die Objekte eine Grundstruktur festgelegt, die als Basis für den Aufbau aller Objekte dient. Diese einheitliche Struktur erleichtert die Standardisierung der Interaktionen innerhalb der Software. Dabei sollen die Objektklassen möglichst objektorientiert gestaltet werden, sodass die Funktionalität der Objekte in klar abgegrenzten Methoden abgebildet wird. Zum Ausführen einer Funktionalität reicht es daher, die entsprechende Methode aufzurufen.

Die Objekte wurden zusammen mit den Skills iterativ erarbeitet. Änderungen bei der Skill-Struktur hatten auch einen Einfluss auf die Objektstruktur.

Die Schnittstellen eines Objektes wurden in 4 Kategorien aufgeteilt, welche ähnlich wie beim Skill definiert wurden (siehe Kapitel 6.3) und entsprechend das allgemeine Verständnis einfacher machen sollen.

Objektvariablen:

Die Objektvariablen umfassen die Eingangsvariablen (Objektparameter) und die Ausgangsvariablen (Anlagenparameter), welche für den Betrieb des Objektes benötigt werden. Dabei kann es sich z.B. um eine IP-Adresse handeln, wenn das Objekt über eine TCP/IP-Schnittstelle angesprochen wird. Falls das Objekt direkt mit E/A-Klemmen interagieren muss, kann dies über die Anlagenparameter gemacht werden.

Steuerungselemente:

Die Steuerungselemente sind für die Bedienung des Objektes angedacht. Hier wird das Objekt gestartet, gestoppt oder resettet. Zusätzlich werden auch Informationen über den Zustand des Objektes angegeben.

Betriebselemente:

Die Betriebselemente sind für den allgemeinen Betrieb des Objekts angedacht, welche nicht mit dem spezifischen Prozess zu tun haben. Dies ist z.B. die Schnittstelle zum System, welche eine Aussage über Systemzustand macht.

Prozesselemente:

Die Prozesselemente sind spezifische Informationen, welche das Objekt für die Ausführung benötigt.

Die Steuerungs-, Betriebs- und Prozesselemente erfüllen somit die gleichen Aufgaben wie beim Skill. Zwischen der Struktur von Objekten und Skills gibt es bewusste Parallelen, welche die Verständlichkeit und Übersicht der Gesamtsoftware verbessern sollen. Die Steuerungselemente umfassen die gleichen Methoden und Eigenschaften wie die Skills (Tab. 7.1) (Umsetzung dieser unterscheidet sich jedoch).

Art:	Bezeichnung:	Typ:	Beschreibung:
Methode	M_Start	BOOL	Methode zum Starten des Objektes
Methode	M_Stop	BOOL	Methode zum Stoppen des Objektes
Methode	M_Reset	BOOL	Methode zum Resetten des Objektes
Eigenschaft	P_State	eObjectAktorState / eObjectSensorState	Eigenschaft des aktuellen Zustandes

Tabelle 7.1: Steuerungselemente eines Objektes

Die Eigenschaft wird mit einem benutzerdefinierten ENUM-Datentyp (Tab. 7.2) umgesetzt, welcher die Zustände des Objektes abbildet. Somit ist immer klar, in welchem Zustand sich das Objekt im Moment befindet. Dabei unterscheidet man zwischen Aktor und Sensor.

Listen-Nr:	eObjectAktorState:	eObjectSensorState:
0	AUS	AUS
1	BEREIT	LAUFEND
2	MANUELL	FEHLER
3	LAUFEND	/
4	ABGESCHLOSSEN_INTERN	/
5	ABGESCHLOSSEN_EXTERN	/
6	GESTOPPT	/
7	FEHLER	/

Tabelle 7.2: Benutzerdefinierte Objekt-Datentypen

In Kapitel 5.2 (Struktur) wurden die Zustände eines Objekts definiert. Dies stellte einen wichtigen Schritt dar, um die allgemeine Interaktion zwischen System, Skill und Objekt festzulegen. Basierend auf diesen Zuständen wurden die finalen Zustände eines Objekts spezifiziert (Abb. 7.1).

Ein Aktor-Objekt erweitert diese Zustände um einen zusätzlichen Zustand für den manuellen Betrieb. Ein Sensor-Objekt hingegen kann einfacher gestaltet werden: Sobald das System eingeschaltet wird, ist der Sensor aktiv. Dieser Zustand wird nur durch das Ausschalten des Systems oder das Auftreten eines Fehlers verlassen.

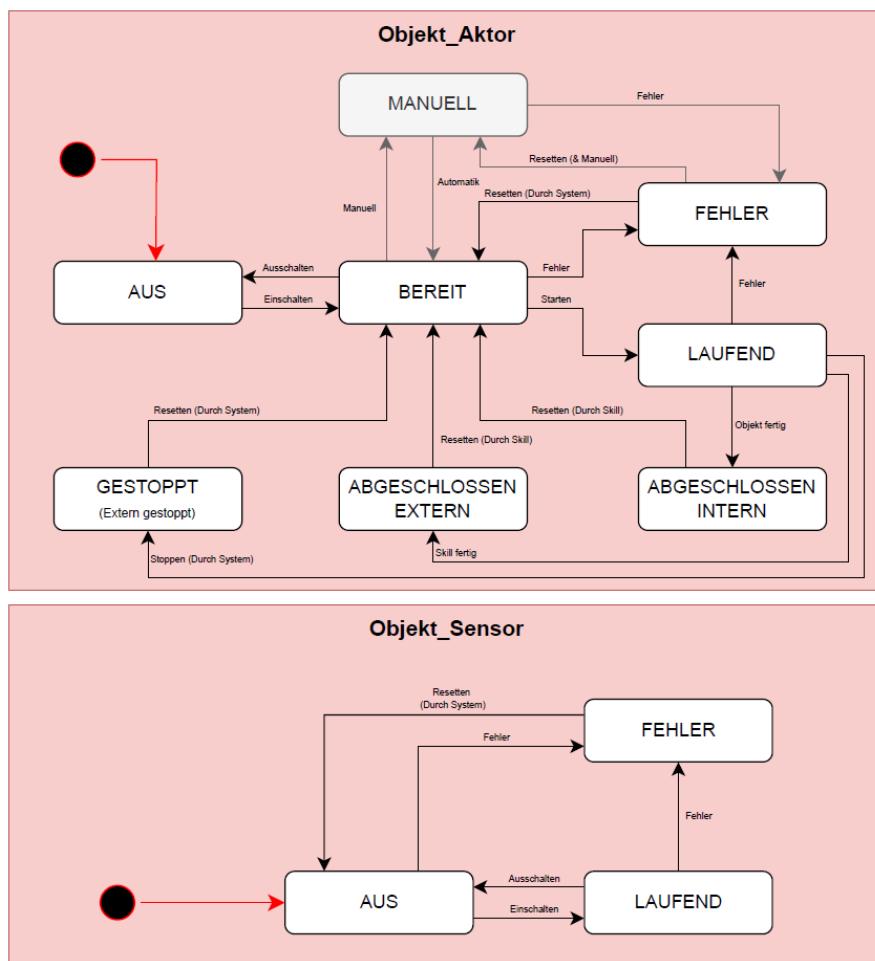


Abbildung 7.1: Objektzustände

Auch in der Struktur unterscheiden sich Aktor- und Sensor-Objekte geringfügig. Sensor-Objekte verfügen über keine Steuerungsmethoden wie Start, Stopp oder Reset. Stattdessen werden diese Funktionen vom System über die Betriebsvariablen gesteuert. Beide Objekt-Typen verwenden jedoch dieselben Betriebsvariablen:

Art:	Bezeichnung:	Typ:	Beschreibung:
Eingangsvariabel	eSysCommand	eSystemCommand	Befehl von System zu Skill
Eingangsvariabel	eSysState	eSystemStatus	Informationen über System
Eingangsvariabel	iMode	INT	Objektmodus (Manuell / Automatik)
Ausgangsvariabel	iErrorID	INT	Information über Fehler

Tabelle 7.3: Betriebselemente eines Objektes

Wie in Kapitel 6.3 beschrieben, verfügt ein Objekt über ein Interface, das die Steuerungselemente und Prozesseigenschaften definiert. Dieses Interface hängt von den erforderlichen Prozessparametern bzw. Prozessinformationen ab, sowie ob es sich um einen Aktor oder einen Sensor handelt. Die Prozessinformationen umfassen beispielsweise Messwerte eines Sensors oder die aktuelle Position eines Roboters.

Die Grundstruktur eines Objekts wird im folgenden Schema (Abb. 7.2) dargestellt. Es zeigt übersichtlich, wie das Objekt über die verschiedenen Schnittstellen im System interagiert.

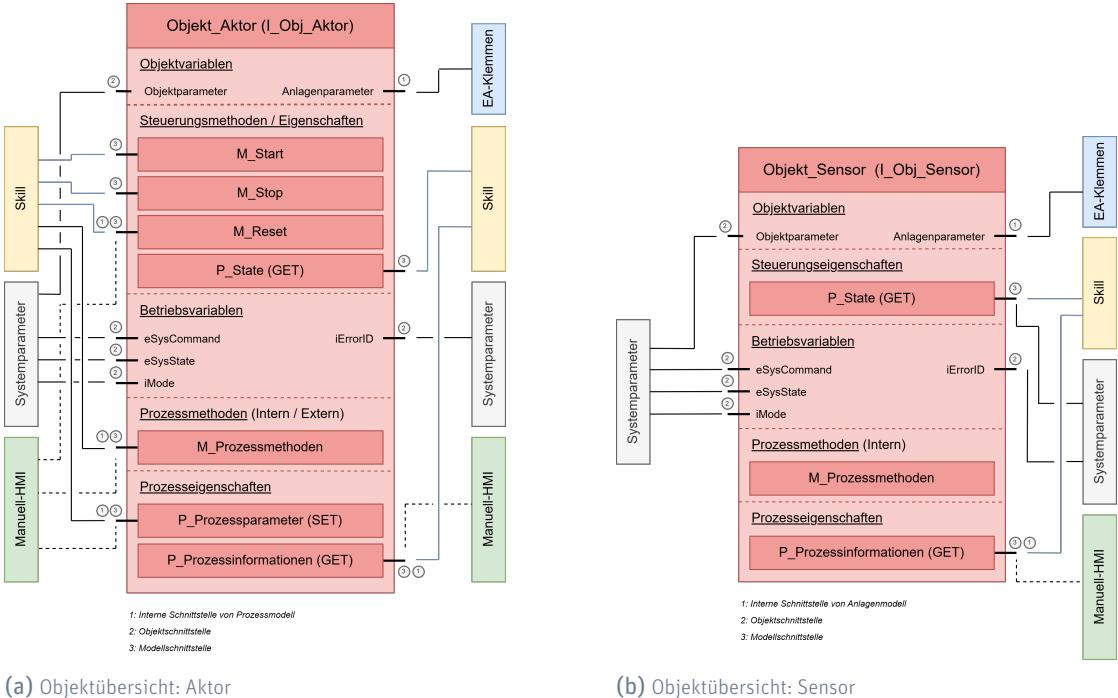


Abbildung 7.2: Objektübersicht

7.2 Umsetzung in TwinCAT

7.2.1 Allgemeine Struktur

Die Programmierung der Objekte soll, ähnlich wie bei den Skills, möglichst einfach und übersichtlich gestaltet sein. Die Grundstruktur ist dabei identisch zu der der Skills. Ein Interface legt fest, welche Methoden und Eigenschaften ein Objekt zwingend besitzen muss. Mit dem Zusatz «**IMPLEMENTES**» kann ein Interface zugewiesen werden.

Die gemeinsamen Steuerungselemente und Betriebsvariablen werden durch Vererbung implementiert, sodass sie für alle Objekte einheitlich verfügbar sind (Abb. 7.3). Unterschiede ergeben sich zwischen Aktoren und Sensoren: Während ein Aktor in der Regel umfangreicher definiert werden muss, kann ein Sensor deutlich einfacher und schlanker gestaltet werden.

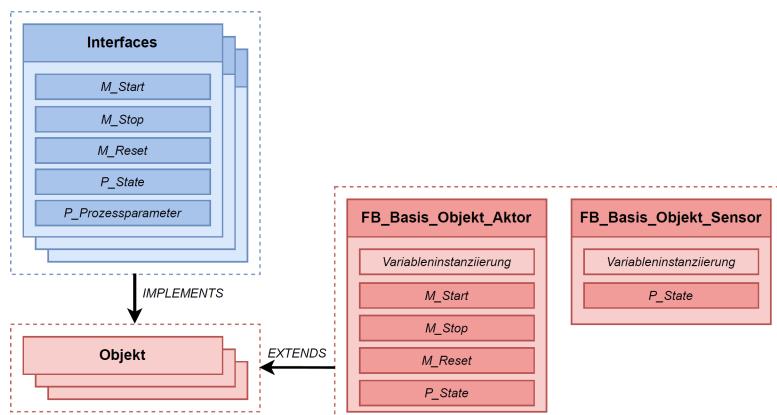


Abbildung 7.3: Umsetzungsstruktur eines Objektes

Im Basis-Funktionsbaustein «FB_Basis_Objekt_Aktor» (Abb. 7.4) wurden die Betriebs-, Management- und Zustandsvariablen festgelegt. Auch hier zeigt sich die Parallele zum Aufbau des Skill-Basis-Funktionsbausteins, wodurch eine einheitliche Struktur gewährleistet wird.

```

FUNCTION_BLOCK FB_Basis_Objekt_Aktor
VAR_INPUT
    // Betriebsvariablen
    eSysState      :eSystemState;           // Information über Stand von System
    eSysCommand    :eSystemCommand;         // Steuerungsvariabel von System
    iMode          :INT;                   // Information über aktuellen Betriebsmodi
END_VAR
VAR_OUTPUT
    iErrorID       :INT;                   // Information um welchen Fehler es sich handelt
END_VAR
VAR
    // Managementvariablen
    iState         :eObjectAktorState;     // Information über Zustand von Objekt
    // Zustandsvariablen
    bEinschalten   :BOOL;                 // Objekt einschalten
    bAusschalten   :BOOL;                 // Objekt ausschalten
    bStarten        :BOOL;                 // Objekt starten
    bObjektFertig   :BOOL;                 // Objekt hat Prozess abgeschlossen
    bSkillFertig    :BOOL;                 // Skill hat Prozess abgeschlossen
    bStoppen        :BOOL;                 // Objekt wurde gestoppt
    bResettenSkill  :BOOL;                 // Das Objekt wird durch den Skill resettet
    bResettenSystem :BOOL;                 // Das Objekt wird durch das System resettet
    bFehler         :BOOL;                 // Das Objekt hat einen Fehler
END_VAR

```

Abbildung 7.4: Definitionen für Objekt-Basis-FB (Aktor)

Der Basis-Funktionsbaustein «FB_Basis_Objekt_Sensor» verwendet dieselben Variablen wie der Aktor-Baustein. Allerdings wird für die Variable «*iState*» der benutzerdefinierte ENUM-Datentyp «*eObjectSensorState*» verwendet.

Da ein Sensor im Vergleich zu einem Aktor weniger Zustände implementiert, sind bei den Zustandsvariablen deutlich weniger Transitionen abzubilden.

Der Basis-Funktionsbaustein des Aktor-Objekts, wie auch des Sensor-Objekts, implementiert die Steuerungsmethoden der Objekte. Die Methoden wurden wie folgt umgesetzt:

```

// Prüfen von Bedingungen
// Ist das System im korrekten Status
IF eSysState = 1 THEN
    // Ist das Objekt im korrekten Status
    IF iState = 0 THEN
        bStarten := TRUE;
    END_IF
END_IF

IF bStarten THEN
    M_Start := TRUE;
ELSE
    M_Start := FALSE;
END_IF

```

(a) Methode: Start

```

// Ist das Objekt im korrekten Status
IF iState = 3 THEN
    bStoppen := TRUE;
END_IF

IF bStoppen THEN
    M_Stop := TRUE;
ELSE
    M_Stop := FALSE;
END_IF

```

(b) Methode: Stop

```

bResettenSkill := TRUE;

IF bResettenSkill THEN
    M_Reset := TRUE;
ELSE
    M_Reset := FALSE;
END_IF

```

(c) Methode: Reset

Abbildung 7.5: Steuerungsmethoden eines Objektes

Die Methode «*M_Start*» (Abb. 7.5a) überprüft zunächst den Status des Systems und anschliessend den Status des Objekts. Beide müssen sich im BEREIT-Zustand befinden, damit die Start-Variablen aktiviert werden kann. Die Aktivierung der Variablen erfolgt nur, wenn sowohl das System als auch das Objekt die erforderlichen Voraussetzungen erfüllen.

Die Stopp-Methode (Abb. 7.5b) kann ausschliesslich im LAUFEND-Zustand des Objekts ausgeführt werden. Für die Reset-Methode (Abb. 7.5c) gibt es hingegen keine Bedingungen; sie kann unabhängig vom aktuellen Zustand des Systems oder des Objekts verwendet werden.

7.2.2 Aufbau des Grundobjektes

Das Grundobjekt definiert den Aufbau, nachdem sich alle Objekte richten. Der Aufbau gliedert sich in fünf Hauptbereiche: Instanziierungen, Input-Command-Verwaltung, Interner Methodenaufruf, Datenerfassung und Zustandsmanagement.

Instanziierungen:

Der Bereich wird genutzt, um notwendige im Objekt-Funktionsbaustein instanzierte TwinCAT-Funktionsbausteine aufzurufen. Diese werden nicht in den Methoden aufgerufen, da es dabei zu Problemen kommen kann. Während der Entwicklung wurde die Erfahrung gemacht, dass beim Anwenden von Kommunikationsbausteinen für TCP/IP-Schnittstellen innerhalb einer Methode Fehler auftreten können. Dies ist zurückzuführen auf die Laufzeit der Methode. Wenn der Kommunikationsbaustein aber im Objektbaustein selbst aufgerufen wird und die Methode nur auf das Eingangssignal einwirkt, ist die Funktionalität stabiler und reproduzierbarer. Bei der detaillierten Beschreibung der umgesetzten Objekte wird gezeigt, wie Funktionsbausteine aufgerufen werden.

Input-Command-Verwaltung:

Die Input-Command-Verwaltung analysiert über die Variabel «eSysCommand», ob das System einen Befehl vorgibt. Das System kann das Objekt einschalten, ausschalten, stoppen und resetten. Gewisse Befehle können nur ausgeführt werden, wenn sich das Objekt in einem bestimmten Zustand befindet.

Interner Methodenaufruf:

Wie der Name beschreibt, werden Methoden aufgerufen, welche nicht von aussen ausgeführt werden. Dabei kann es sich um Methoden handeln, welche z.B. ausgeführt werden müssen, wenn das Objekt eingeschalten wird und von «AUS» zu «BEREIT» wechseln soll.

Datenerfassung:

Im Bereich Datenerfassung wird der Prozess durchgeführt, welcher für die Erfassung von notwenigen Daten zuständig ist. Dazu gehört der Aufbau der Verbindung, das Auslesen von Daten und das Verarbeiten der Daten, sodass das Objekt damit arbeiten kann. Dies ist erforderlich, da manche Objekte bestimmte Informationen für Funktionen benötigen. Damit z.B. ein Roboter-Objekt beurteilen kann, ob die Bewegung abgeschlossen wurde und dadurch in den Zustand «ABGESCHLOSSEN_INTERN» wechseln kann, muss die aktuelle Position mit der gewünschten Position verglichen werden. Die aktuelle Position wird hierbei unter Datenerfassung erfasst. Es geht grundsätzlich um Daten, die während dem Betrieb des Objektes ausgewertet werden müssen.

Zustandsmanagement:

Wie bei den Skills werden unter Zustandsmanagement die Zustände des Objektes umgesetzt. Für die Umsetzung wurde auch hier mit einer CASE-Anwendung gearbeitet.

Zustand 0 - AUS:

Dieser Zustand prüft ausschliesslich, ob das Objekt durch das System eingeschaltet wird. Sobald die entsprechende Variabel aktiviert ist, wechselt das Objekt in den Zustand «BEREIT».

Zustand 1 - BEREIT:

Das Objekt kann entweder gestartet, wieder ausgeschaltet oder in den Manuell-Modus geschalten werden. Beim Start des Prozesses durch einen Skill, wird die entsprechende interne Methode ausgeführt und der Zustand wechselt zu «LAUFEND». Das Ausschalten kann nur durch das System ausgelöst werden.

Zustand 2 - MANUELL:

Der Zustand implementiert alle Funktionalitäten, welche das Objekt hat, sodass diese manuell verwendet werden können. Dies kann z.B. das manuelle Anfahren von Punkten mit dem Roboter sein.

Zustand 3 - LAUFEND:

Ein Objekt kann auf 2 Arten gestoppt werden, durch externen Einfluss oder das Objekt beendet den Prozess von selbst. Beides muss im Zustand überwacht werden. Beim Stoppen durch den Skill, wechselt das Objekt in den Zustand «ABGESCHLOSSEN_EXTERN». Falls das Objekt selbst den Prozess stoppt, wechselt dieses in den Zustand «ABGESCHLOSSEN_INTERNAL». Auch das System kann das Objekt stoppen, in diesem Fall wechselt das Objekt in den Zustand «GESTOPPT», da dies ein Stoppen ausserhalb des ordentlichen Prozesses darstellt. Wie der Stop-Prozess innerhalb des Zustandes umgesetzt wird, kann von Objekt zu Objekt unterschiedlich sein.

Zustand 4 - ABGESCHLOSSEN_INTERNAL:

Im Zustand wird auf den Reset-Befehl des Skills gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT».

Zustand 5 - ABGESCHLOSSEN_EXTERN:

Im Zustand wird auf den Reset-Befehl des Skills gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT».

Zustand 6 - «GESTOPPT»:

Im Zustand wird auf den Reset-Befehl des Skills gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT».

Zustand 7 - FEHLER:

Im Zustand wird auf den Reset-Befehl des Skills gewartet. Sobald dieser erkannt wird, wechselt das Objekt auf «BEREIT».

7.2.3 Erarbeitete Objekte

Objektklasse für UR5: FB_Roboter

Die Objektklasse (Abb. 7.6) implementiert das Interface «I_Objekt_Roboter», welches die Steuerungselemente der Objektklasse vorgibt. Als Objekt-Eingangsvariablen benötigt die Objektklasse eine IP-Adresse und zwei Port-Angaben. Die genaue Definition dieser Angaben wird in den folgenden Seiten erklärt. Es gibt keine Objekt-Ausgangsvariablen, welche mit EA-Klemmen verbunden werden müssten. Die Verbindung zum Roboter wird über TwinCAT-Funktionsbausteine durchgeführt, welche gleichzeitig auch die direkte Schnittstelle darstellen.

Die Steuerungs- und Betriebsvariablen bleiben unverändert zum Basis-Funktionsbaustein.

Als Prozessmethoden wurden «M_Verbinden», «M_Trennen», «M_BewegungZuPosition» und «M_BewegungStoppen» definiert. Diese Methoden decken die Grundfunktionen der Schnittstelle zum Roboter und dessen Funktionalität ab, um die definierte Anwendung umsetzen zu können. Damit weitere Funktionalitäten abgebildet werden könnten, müssten weitere Methoden hinzugefügt werden. Die Prozesseigenschaften bestehen aus den Prozessparametern, die für eine Punkt-zu-Punkt-Bewegung des Roboters benötigt werden.

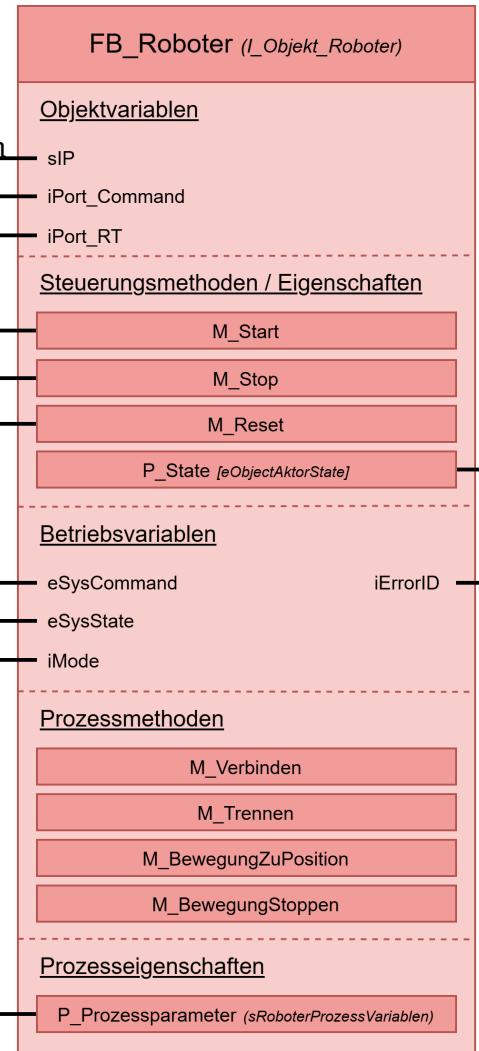


Abbildung 7.6: Umsetzungsstruktur von Roboter-Objekt

Wie bereits in Kapitel 4.2 angesprochen, besitzt der UR5 eine TCP/IP-Schnittstelle. Unterschiedliche Ports haben dabei unterschiedliche Funktionen (Abb. 7.7).

CB-Series							
	Primary		Secondary		Real-time		Real-time Data Exchange (RTDE)
Port no.	30001	30011	30002	30012	30003	30013	30004
Frequency [Hz]	10	10	10	10	125	125	125
Receive	URScript commands	-	URScript commands	-	URScript commands	-	Various data
Transmit	See attachment from the bottom		See attachment from the bottom		See attachment from the bottom		See RTDE Guide

Abbildung 7.7: UR5-Verbindungsports

i Beim eingesetzten Kontroller handelt es sich um einen älteren Kontroller vom Typ CB2. Diverse Funktionalitäten der Schnittstellen oder zur Verfügung gestellten Dokumentationen richten sich an Kontroller ab Version CB3. Dies hat während der Entwicklung zu grossen Problemen geführt und viel Zeit in Anspruch genommen, vor allem bezüglich des Real-Time-Interfaces. Dieses wird in dieser Form nicht mehr unterstützt und wurde durch RTDE (Real-time Data Exchange) ersetzt, welche deutlich umfangreichere Funktionen bietet. Mittels Referenzprogrammen und alten Dokumentation konnte aber auch mit dem Real-Time-Interface gearbeitet werden. Die entsprechenden Dokumentationen werden im Anhang beigelegt.

Über die Secondary-Schnittstelle (Port 30002) können dem Roboter Befehle in der UR-eigenen UR-Script-Programmiersprache übergeben werden. Alle möglichen Befehle lassen sich aus der entsprechenden Dokumentation herauslesen [20].

Über die Real-Time-Schnittstelle (Port 30003) können Informationen über den Roboter in Echtzeit erhalten werden [21]. Dabei erhält man ein Datenpaket von 812 Byte, bei welchem jede Position für eine bestimmte Information steht (Tab. 7.4).

Bezeichnung:	UR-Dt:	TwinCAT-Dt:	Byte:	Beschreibung:
messageSize	INT	LREAL	4	Gesamtlänge der Nachricht in Bytes
timestamp	DOUBLE	LREAL	8	Zeitdauer seit Controller gestartet wurde
qTarget	DOUBLE	LREAL	48	Zielposition der Gelenke
qdTarget	DOUBLE	LREAL	48	Zielgeschwindigkeit der Gelenke
qddTarget	DOUBLE	LREAL	48	Zielbeschleunigung der Gelenke
iTarget	DOUBLE	LREAL	48	Zielstrom der Gelenke
mTarget	DOUBLE	LREAL	48	Zielmoment der Gelenke
qActual	DOUBLE	LREAL	48	Aktuelle Position der Gelenke
qdActual	DOUBLE	LREAL	48	Aktuelle Geschwindigkeit der Gelenke
iActual	DOUBLE	LREAL	48	Aktueller Strom der Gelenke
toolAccelerometerValues	DOUBLE	LREAL	24	Tool-Beschleunigungswerte (x, y und z)
unused_1	/	/	120	/
tcpForce	DOUBLE	LREAL	48	Kräfte beim TCP
toolVector	DOUBLE	LREAL	48	Zielkoordinaten des Tools
tcpSpeed	DOUBLE	LREAL	48	Zielgeschwindigkeit des Tools
digitalInputBits	DOUBLE	LREAL	8	Aktueller Zustand der digitalen Eingänge
unused_2	/	/	120	/

Dt = Datentyp

Tabelle 7.4: UR-Datenpaket

Das korrekte Auslesen des Datenpaketes ist für den fehlerfreien Betrieb des Roboters essenziell. Eine falsche Interpretation der Daten kann zu unerwarteten Verhalten des Roboters führen. Der Schnittstelle muss auch genug Zeit gelassen werden, dass das komplette Datenpaket von 812 Byte ausgelesen werden kann.

i Die Dokumentation zu URScript und der Real-Time-Schnittstelle wird in Anhang beigelegt.

In TwinCAT wurde für die Kommunikation mit dem Roboter das Paket TF6310 (TwinCAT 3 | TCP/IP) verwendet. Das Paket stellt verschiedene Funktionsbausteine zur Verfügung, welche für die Kommunikation über TCP/IP benötigt werden (Abb. 7.8). Der UR5 ist dabei der Server und das TwinCAT der Client. Folgende vier Bausteine sind relevant für die Kommunikation mit dem Roboter:

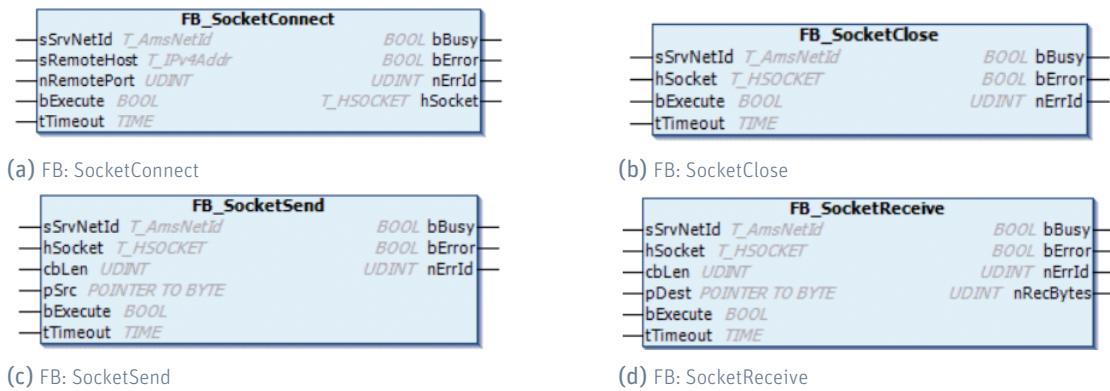


Abbildung 7.8: TwinCAT-Funktionsbausteine für TCP/IP



Dokumentationen der relevanten Beckhoff-Pakete werden im Anhang beigelegt.

Für eine detaillierte Beschreibung des TF6310-Pakets kann die entsprechende Dokumentation angeschaut werden [9]. «FB_SocketConnect» ist für die Verbindung mit dem Server zuständig. Folgende zwei Informationen werden für die Verbindung zum UR5 benötigt:

<code>sRemoteHost:</code>	192.168.1.100	IP-Adresse des Roboters
<code>sRemotePort:</code>	30002	Verbindung mit Secondary-Schnittstelle
	30003	Verbindung mir Real-Time-Schnittstelle

Die Variable «`sSrvNetId`» kann mit einem Leerstring versehen werden, da der TwinCAT-TCP/IP-Connection-Server auf dem lokalen Rechner läuft. Sobald der Funktionsbaustein über «`bExecute`» gestartet wird und eine Verbindung zum TCP/IP-Server erfolgreich aufgebaut werden konnte, wird ein TCP/IP-Verbindungshandle erstellt. Dieser wird über «`hSocket`» zur Verfügung gestellt. Über dieses Handle können Daten an einen Socket gesendet oder empfangen werden. Mit «`FB_SocketClose`» kann die Verbindung zum Kommunikationssocket geschlossen werden.

Zum Senden und Empfangen von Daten werden «`FB_SocketSend`» und «`FB_SocketReceive`» verwendet. Beide Funktionsbausteine benötigen den definierten TCP/IP-Verbindungshandle, die Anzahl der zu sendenden Daten in Bytes («`cbLen`») und die Pointer-Adresse des Sendepuffers («`pSrc`» / «`pDest`»).

Die UR5-Objektklasse hat grundsätzlich folgenden Aufgaben:

- ▶ Aufbau und Verwaltung der Verbindung zum UR5-Roboter über TCP/IP.
- ▶ Entgegennahme und Vorbereitung der Prozessdaten.
- ▶ Senden des entsprechenden Befehls an den Roboter, damit Bewegung gestartet wird.
- ▶ Senden des entsprechenden Befehls an den Roboter, dass Bewegung gestoppt wird.
- ▶ Entgegennahme, Verarbeitung und Auswertung der vom Roboter gesendeten Daten.

Für den Aufbau und die Verwaltung der Kommunikation werden alle relevanten Funktionsbausteine im Bereich «Instanziierungen» der Objektklasse aufgerufen und mit den allgemeinen Variablen verknüpft (Abb. 7.9). Die Zuweisung der Sende- oder Empfangsvariablen wird erst innerhalb der entsprechenden Methode vorgenommen.

```
Send_START(sSrvNetId := sSrvNetId,
            hSocket := hSocket,
            tTimeout := tTimeout,
            bError => bFehler,
            nErrId => iErrorID);
```

Abbildung 7.9: Aufrufen eines Funktionsbausteines

Über die Methoden «M_Verbinden» und «M_Trennen» werden die entsprechenden Trigger der Funktionsbausteine aktiviert um diese auszulösen. Die Prozessdaten, welche durch den Skill zur Verfügung gestellt wurden, werden über die Eigenschaft «P_Prozessparameter» an das Objekt übergeben. Die Daten werden mittels einer benutzerdefinierten Struktur mit folgender Definition übergeben:

```
TYPE sRoboterProzessVariablen :
STRUCT
    q1           :LREAL;          // X-Koordinate oder Gelenkwinkel 1
    q2           :LREAL;          // Y-Koordinate oder Gelenkwinkel 2
    q3           :LREAL;          // Z-Koordinate oder Gelenkwinkel 3
    q4           :LREAL;          // Orientierung um X-Achse oder Gelenkwinkel 4
    q5           :LREAL;          // Orientierung um Y-Achse oder Gelenkwinkel 5
    q6           :LREAL;          // Orientierung um Z-Achse oder Gelenkwinkel 6
    v            :LREAL;          // Geschwindigkeit
    a            :LREAL;          // Beschleunigung
    PostTyp      :eRobPostType;   // Position-Typ definieren (Absolut / Relativ)
    MoveTyp     :eRobMoveType;   // Bewegungsart definieren (Linear / Circular etc.)
END_STRUCT
END_TYPE
```

Abbildung 7.10: Struktur: sRoboterProzessVariablen

Die Prozessdaten werden innerhalb der Methode «M_BewegungZuPosition» analysiert, in die korrekte Form gebracht und versendet. Die Methode prüft in einem ersten Schritt, um welche Positionsart es sich handelt. Es kann zwischen absoluten und relativen Positionen unterschieden werden. Je nach Positionsart werden die Koordinaten anders ausgelegt. Der nächste Schritt ist die Bestimmung der Bewegung. Jede Bewegungsart hat einen definierten Befehl in der URScript-Programmiersprache. Am Ende kann der Befehl-String zusammengesetzt werden. Ein Befehl-String besteht in der Regel aus dem entsprechenden Befehls-Wort, der Position, der Geschwindigkeit und Beschleunigung.

Beispiel für einen solchen Befehl-String:

moveL(p[x,y,z,rx,ry,rz],a,v)	moveL:	Lineare Bewegung im kartesischen Koordinatensystem
	P [x,y,z,rx,ry,rz] :	Informationen über anzufahrenden Punkt [m & rad]
	a	Beschleunigung [m/s^2]
	v	Geschwindigkeit [m/s]



Jeder URScript-Befehl muss mit «\n» beendet werden. Der String kann aber nicht einfach mit diesen Angaben ergänzt werden. Bei der Umwandlung von einem String in einen Byte-Array wird diese Endung «falsch» übersetzt (mit dem Wert 42). Für die korrekte Interpretation des Befehls durch den Kontroller muss am Ende aber eine 10 stehen. Diese muss dem Byte-Array angehängt werden. Wahrscheinlich ist dies auf eine unterschiedliche Interpretation von Low- und High-Byte zurückzuführen (die Reihenfolge der Bytes wird dabei vertauscht und führt zu falschen Resultaten). Dies hat zu diversen Problemen während der Entwicklung geführt in Zusammenhang mit der TCP/IP-Schnittstelle. Ein Beispiel, wie diese Problematik gelöst wurde, wird in Abbildung 7.14 gezeigt.

In folgendem Code-Snippet (Abb. 7.11) wird das Byte-Array mit dem Wert 10 ergänzt und anschliessend über den TCP/IP-Funktionsbaustein «FB_SocketSend» an den Server versendet.

```
// Vorbereiten von Array
CommandArray.sValue := Command;
CommandArray.arrByte[LEN(CommandArray.sValue)] := 10;

// Senden von Array
Send_START(cbLen := LEN(CommandArray.sValue),
            pSrc := ADR(CommandArray.arrByte),
            bExecute := TRUE);
```

Abbildung 7.11: Ergänzung von Array

Mit der Methode «M_BewegungStoppen» wird grundsätzlich dasselbe gemacht. Nur der Befehl-STRING ändert sich auf den entsprechenden Befehl.

Im Bereich «Datenerfassung» wird kontinuierlich der aktuelle Wert der Gelenkposition und Geschwindigkeit erfasst, wie auch die TCP-Position. Die Erfassung arbeitet dabei fortlaufend vier Schritte ab:

- | | |
|-----------|--|
| Schritt 1 | Verbindungsaufbau zur Real-Time-Schnittstelle des Roboters |
| Schritt 2 | Empfangen des Datenpakets (812 Bytes) |
| Schritt 3 | Verarbeitung der Rohdaten |
| Schritt 4 | Trennung der Verbindung zur Real-Time-Schnittstelle |

Erfahrungen haben gezeigt, dass die Schnittstelle immer wieder geschlossen und neu geöffnet werden muss, damit Daten verlässlich ausgelesen werden konnten. Wird dies nicht gemacht, so werden die Daten nur das erste Mal empfangen und behalten anschliessend diese Werte.

Ein Positions- oder Geschwindigkeitswert setzt sich jeweils aus 8 Bytes zusammen. Diese können über eine UNION-Datenstruktur zu einem LREAL-Wert zusammengefügt werden. Da TwinCAT die Low- und High-Bytes anders interpretiert als der Roboter-Kontroller, muss die Reihenfolge des Arrays in einem ersten Schritt umgedreht werden. Anschliessend können diese mittels einer UNION-Datenstruktur umgewandelt werden.

Diese Werte werden im Zustand «LAUFEND» verwendet, um zu prüfen, ob der Roboter die gewünschte Position bereits erreicht hat. Falls die Differenz zwischen der aktuellen und gewünschten Position einen definierten Grenzwert unterschreitet, gilt die Position als erreicht (Abb. 7.12).

```

// Vergleich zwischen IST und SOLL
// Kartesisch
IF Prozessvariablen.MoveTyp = 0 OR Prozessvariablen.MoveTyp = 3 THEN
    IF ABS(ABS(TargetPosition.q1) - ABS(ToolPosition.q1)) < 0.001 THEN
        bQ1 := TRUE;
    END_IF

    IF ABS(ABS(TargetPosition.q2) - ABS(ToolPosition.q2)) < 0.001 THEN
        bQ2 := TRUE;
    END_IF

    IF ABS(ABS(TargetPosition.q3) - ABS(ToolPosition.q3)) < 0.001 THEN
        bQ3 := TRUE;
    END_IF

    IF ABS(ABS(TargetPosition.q4) - ABS(ToolPosition.q4)) < 2 THEN
        bQ4 := TRUE;
    END_IF

    IF ABS(ABS(TargetPosition.q5) - ABS(ToolPosition.q5)) < 1 THEN
        bQ5 := TRUE;
    END_IF

    IF ABS(ABS(TargetPosition.q6) - ABS(ToolPosition.q6)) < 1 THEN
        bQ6 := TRUE;
    END_IF
END_IF

```

Abbildung 7.12: Vergleich von IST- und SOLL-Positionen

Beim Stoppen der Bewegung kann über die aktuelle Geschwindigkeit ermittelt werden, ob der Roboter angehalten hat und somit die Bewegung abgeschlossen wurde.

Objektklasse für Kraftsensor: FB_Kraftsensor

Die Objektklasse (Abb. 7.13) implementiert das Interface «I_Objekt_Kraftsensor» und gehört zur Kategorie der Sensor-Objekte. Daher enthält sie keine Steuerungsmethoden, sondern lediglich eine Steuerungseigenschaft. Für den Betrieb der Objektklasse sind eine IP-Adresse und eine Port-Nummer als Eingabeveriablen erforderlich. Analog zur Roboterobjektklasse werden keine Ausgangsvariablen definiert. Die Kommunikation mit dem Sensor erfolgt ebenfalls über TwinCAT-Funktionsbausteine.

Die Steuerungs- und Betriebsvariablen bleiben unverändert gegenüber dem Basis-Funktionsbaustein. Die Objektklasse benötigt keine Prozessmethoden. Die Prozesseigenschaft stellt die Messwerte des Sensors zur Verfügung.

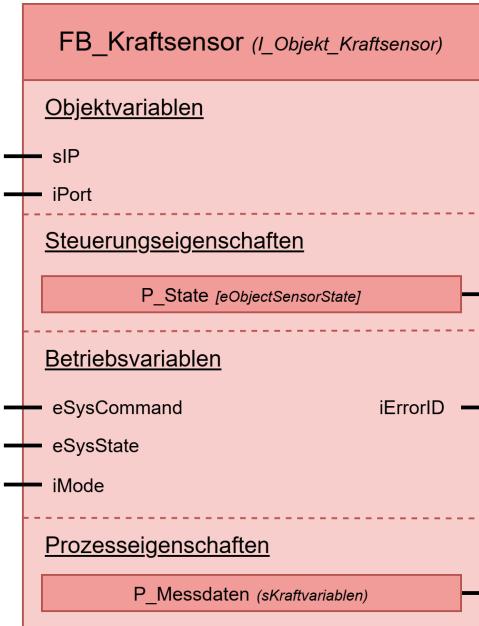


Abbildung 7.13: Umsetzungsstruktur von Kraftsensor-Objekt

Wie bereits in Kapitel 4.2 angesprochen, besitzt der definierte Kraftsensor eine TCP/IP-Schnittstelle. Die Schnittstelle kann verwendet werden, um die Sensor-Rohdaten auszulesen. Um sich mit dem Sensor zu verbinden, muss als Port-Nummer 49151 verwendet werden [22].

Der Sensor kann Anfragebefehle (20 Bytes) entgegennehmen und Datenpakete (16 Bytes) senden. Dabei ist der Aufbau des Befehls und des Datenpaketes genau vorgegeben. Damit

die Daten des Sensors ausgelesen werden können, muss in einem ersten Schritt ein Startbefehl (Tab. 7.5) gesendet werden.

Bezeichnung:	Sensor-Dt:	TwinCAT-Dt:	Byte:	Wert:
Command	UINT8	UINT	1	Der Wert muss 0 sein
Reserved	UINT8	UNIT	19	Alle 19 Werte müssen 0 sein

(Dt = Datentyp)

Tabelle 7.5: Startbefehl für Kraftsensor

Der Sensor stellt anschliessend das Datenpaket mit folgender Struktur zur Verfügung:

Bezeichnung:	Sensor-Dt:	TwinCAT-Dt:	Byte:	Beschreibung:
Header	UINT16	UINT	2	Ein fixer Wert «0x1234»
Stats	UINT16	UINT	2	Statuswort des Sensors
Fx	INT16	INT	2	Kraftwert in X-Richtung
Fy	INT16	INT	2	Kraftwert in Y-Richtung
Fz	INT16	INT	2	Kraftwert in Z-Richtung
Tx	INT16	INT	2	Momentwert um X-Achse
Ty	INT16	INT	2	Momentwert um Y-Achse
Tz	INT16	INT	2	Momentwert um Z-Achse

(Dt = Datentyp)

Tabelle 7.6: Datenpaket von Sensor



Die Dokumentation der OnRobot-Compute-Box wird in Anhang beigelegt.

Die Rohmessdaten müssen nun in Newton und Newton/Meter umgewandelt werden. Dies kann mit folgenden Formeln gemacht werden:

Umrechnung der Kraft in N:	Umrechnung des Moments in N/m:
$F(\text{in N}) = F * \text{ScaleFactor} / \text{CPF}$	$T(\text{in N/m}) = T * \text{ScaleFactor} / \text{CPT}$

Tabelle 7.7: Umrechnungsformeln

Alle Faktoren werden durch den Sensor zur Verfügung gestellt, müssen jedoch über eine Anfrage beantragt werden. Mit folgendem Befehl werden die Faktoren gesendet:

Bezeichnung:	Sensor-Dt:	TwinCAT-Dt:	Byte:	Wert:
Command	UINT8	UINT	1	Der Wert muss 1 sein
Reserved	UINT8	UNIT	19	Alle 19 Werte müssen 0 sein

(Dt = Datentyp)

Tabelle 7.8: Befehl für Umrechnungsfaktoren

Der Sensor schickt darauf folgend ein Datenpaket mit 24 Byte mit folgender Struktur:

Bezeichnung:	Sensor-Dt:	TwinCAT-Dt:	Byte:	Beschreibung:
Header	UINT16	UINT	2	Ein fixer Wert «0x1234»
Unit_Force	UINT8	USINT	1	Gibt an, welche Einheit mit Faktoren ermittelt wird
Unit_Torque	UINT8	USINT	1	Gibt an, welche Einheit mit Faktoren ermittelt wird
CPF	INT32	UDINT	4	Zählwerte pro Kraftwert (Counts per Force value)
CPT	INT32	UDINT	4	Zählwerte pro Momentwert (Counts per Torque value)
ScaleFactorFx	INT16	UINT	2	Skalier-Faktor für Kraft in X-Richtung
ScaleFactorFy	INT16	UINT	2	Skalier-Faktor für Kraft in Y-Richtung
ScaleFactorFz	INT16	UINT	2	Skalier-Faktor für Kraft in Z-Richtung
ScaleFactorTx	INT16	UINT	2	Skalier-Faktor für Moment um X-Achse
ScaleFactorTy	INT16	UINT	2	Skalier-Faktor für Moment um Y-Achse
ScaleFactorTz	INT16	UINT	2	Skalier-Faktor für Moment um Z-Achse

(Dt = Datentyp)

Tabelle 7.9: Datenpaket von Sensor für Umrechnungsfaktoren

Folgende Faktoren werden für die Umrechnung in Newton und Newton/Meter verwendet:

$$\begin{array}{ll}
 \text{CPF} = 10000 & \text{ScaleFactorFx} = 200 \\
 \text{CPT} = 10000 & \text{ScaleFactorFy} = 200 \\
 & \text{ScaleFactorFz} = 200 \\
 & \text{ScaleFactorTx} = 100 \\
 & \text{ScaleFactorTy} = 100 \\
 & \text{ScaleFactorTz} = 65
 \end{array}$$

In TwinCAT wird wieder mit dem Paket TF6310 (TwinCAT 3 | TCP/IP) gearbeitet, um die Kommunikation zum Sensor aufzubauen [9]. Der Sensor stellt hierbei wieder den Server dar. Entsprechend können die gleichen vier Funktionsbausteine wie beim Roboter verwendet werden.

Für die Verbindung zum Server mittels «FB_SocketConnect» werden folgende zwei Informationen benötigt:

sRemoteHost:	192.168.1.10	IP-Adresse des Sensors
sRemotePort:	49151	Port für TCP-Schnittstelle

Die allgemeine Verwendung der Kommunikationsbausteine bleibt identisch zum Roboter, wie auch der Aufbau der Objektklasse. Die Kraftsensor-Objektklasse hat grundsätzlich folgende Aufgaben:

- ▶ Aufbau und Verwaltung der Verbindung zum Sensor über TCP/IP.
- ▶ Entgegennahme, Verarbeitung und Auswertung der vom Sensor gesendeten Daten.

Die Objektklasse wertet die Sensordaten aus, wandelt die Rohdaten um und stellt diese über die Prozesseigenschaft zur Verfügung. Der Prozess der Datenauslesung ist dabei wieder in 4 Schritte aufgeteilt:

Schritt 1	Verbindungsaufbau zu Sensor über TCP/IP
Schritt 2	Empfangen des Datenpakets (16 Bytes)
Schritt 3	Verarbeitung der Rohdaten
Schritt 4	Trennung der Verbindung von TCP/IP-Schnittstelle

Auch hier muss berücksichtigt werden, dass die Low- und High-Bytes von TwinCAT anders interpretiert werden, als vom Server vorgesehen. Da der Messwert nur aus 2 Byte besteht, kann dies relativ simpel gemacht werden. Das Prinzip ist wie folgt:

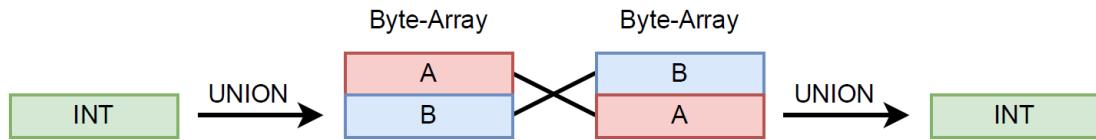


Abbildung 7.14: Byte-Switch von Low- und High-Byte

Der über die Kommunikationschnittstelle erhaltene INT-Wert wird mittels eines UNION-Datentyps in eine Array mit 2 Byte umgewandelt. Die Reihenfolge der Bytes wird umgedreht, damit das Array anschliessend mit einem weiteren UNION-Datentyp in eine INT-Variablen zurückgewandelt werden kann. Der korrekte Rohwert kann jetzt für die Umwandlung in den entsprechenden Einheit verwendet werden. In TwinCAT sieht dies folgendermassen aus:

```

RohArray.sValue := ReceivedData.Fx;
SwitchArray[0] := RohArray.arrByte[1];
SwitchArray[1] := RohArray.arrByte[0];
SwichValue.arrByte := SwitchArray;
Rohdaten.Fx := SwichValue.sValue;
Messung.Fx := Rohdaten.Fx * ScalefactorFx / (CPF);
  
```

Abbildung 7.15: Code für Byte-Switch

Die umgewandelten Daten werden über die Prozesseigenschaft «P_Messdaten» zur Verfügung gestellt. Für den Datentyp der Eigenschaft wurde die Struktur «sKraftvariablen» angegeben. Diese benutzerdefinierte Struktur ist wie folgt definiert:

```

TYPE sKraftvariablen :
STRUCT
    Fx          :LREAL;           // Kraft in X-Richtung
    Fy          :LREAL;           // Kraft in Y-Richtung
    Fz          :LREAL;           // Kraft in Z-Richtung
    Tx          :LREAL;           // Moment auf X-Achse
    Ty          :LREAL;           // Moment auf Y-Achse
    Tz          :LREAL;           // Moment auf Z-Achse
END_STRUCT
END_TYPE
  
```

Abbildung 7.16: Struktur: sKraftvariablen

Objektklasse für Greifer (mit Sensoren): FB_SmartGreifer

Die letzte Objektklasse (Abb. 7.17) wird mit dem Interface «I_Objekt_SmartGreifer» implementiert. Da es sich beim Objekt wieder um einen Aktor handelt, werden durch das Interface auch Steuerungsmethoden vorgegeben. Die Objektklasse besitzt keine Objektvariablen. Die Kommunikation zum Greifer wird über TwinCAT-Funktionsbausteine aufgebaut, welche direkt auf EA-Klemmen zugreifen.

Die Steuerungs- und Betriebsvariablen bleiben unverändert zum Basis-Funktionsbaustein. Als Prozessmethoden des Funktionsbausteins wurden «M_Aktivieren», «M_Resetten», «M_StatusLesen» und «M_Position_Anfahren» definiert. Diese Funktionen decken die Grundfunktionalitäten des Greifers ab. Durch die Methoden kann eine bestimmte Backenposition angefahren und überprüft werden, ob diese Position erreicht wurde oder wann der Greifer in ein Hindernis gefahren ist.

Mit den Prozesseigenschaften werden die prozessrelevanten Parameter für den Greifer definiert. Über «P_AnschlagPos» wird die Backenposition zur Verfügung gestellt, bei welcher der Greifer in eine Hindernis gefahren ist.

Die Kommunikation zum Greifer wird über ein Modbus RTU Protokoll realisiert. Innerhalb der Dokumentation wird nicht weiter auf die Spezifikation dieses Protokolls eingegangen. Bei der Kommunikation handelt es sich um eine Master-Slave-Kommunikation [18]. Der Greifer stellt dabei den Slave dar. Die wichtigsten Eigenschaften, welche die Verbindung zum Greifer einhalten muss, sind folgende:

Anschlusschnittstelle:	RS-485
Baud-Rate:	115200 bps (Bits per Second)
Data-Bits:	8 Bits
Stop-Bits:	1 Bit
Parität:	Keine
Slave-ID:	9
Packetgrösse:	2 Byte (16 bit)

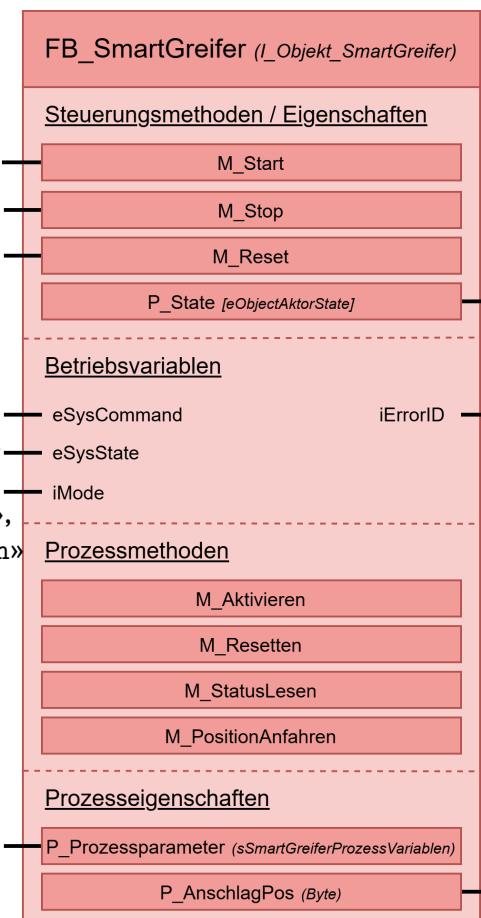


Abbildung 7.17: Umsetzungsstruktur von Greifer-Objekt

Die Modbus RTU Schnittstelle des Greifers unterstützt verschiedene Funktionalitäten, welche unterschiedliche Register darstellen, auf welche man zugreifen kann. Die Register haben verschiedene Anfragebefehle und Antworten. Um die Kommunikationsschnittstelle korrekt anwenden zu können, ist es wichtig diese Funktionalitäten und ihre Befehle und Antworten zu verstehen. Der genaue Aufbau der Nachricht und Antwort kann aus der Dokumentation des Greifers entnommen werden.

Als Beispiel wird der Befehl für das komplette Schliessen des Greifers bei 100% Geschwindigkeit und Kraft gezeigt:

Array-Nr.	Bits (Hex):	Beschreibung:
0	09	Slave-ID
1	10	Funktionscode 16 (Preset Multiple Registers)
2 + 3	03 E8	Adresse des ersten Registers
4 + 5	00 03	Anzahl der zu beschreibenden Register
6	06	Anzahl der Daten-Bytes (3 Register x 2 Bytes = 6 Bytes)
7 + 8	09 00	Wert auf Register schreiben (Aktivieren von Greifer)
9 + 10	00 FF	Wert auf Register schreiben (Position für komplett geschlossenen Greifer angeben 255/255)
11 + 12	FF FF	Wert auf Register Schreiben (100% Geschwindigkeit und Kraft)
13 + 14	42 29	Zyklische Redundanzprüfung (CRC)

Tabelle 7.10: Beispiel für das Schliessen des Greifers

Die Interaktion mit dem Greifer über die herstellereigene Software funktionierte einwandfrei. Der Greifer wurde über einen RS-485-USB-Adapter am PC angeschlossen. Der Greifer-Port wurde sofort erkannt und konnte verwendet werden. Auch über Programme wie „Modbus-tester“ konnte der Greifer verbunden und getestet werden. Die Integration des Greifers in TwinCAT via Modbus RTU war jedoch eine aufwändige und zeitintensive Arbeit. TwinCAT stellt grundsätzlich ein Paket für die Kommunikation mit Modbus RTU zur Verfügung, TF6255 (TwinCAT 3 | Modbus RTU). Das Paket stellt verschiedene Funktionsbausteine für die Kommunikation zur Verfügung [23]. Einer dieser Funktionsbausteine stellt die Verbindung über eine COM-Port-Schnittstelle her. Nach unzähligen Versuchen war es aber nicht möglich eine Verbindung mit dem Greifer aufzubauen. Mit Hilfe einer Software zur Überwachung von Serial-Port-Schnittstellen wurde festgestellt, dass kein Signal TwinCAT verlässt. Es wurde kein Weg gefunden eine Verbindung über eine COM-Port-Schnittstelle mittels TwinCAT zum Greifer herzustellen.

Als Alternative wurde versucht sich über eine RS-485-Interface-Karte von Beckhoff mit dem Greifer zu verbinden. Dafür wurde die KL6041 (1-Kanal-Kommunikations-interface) ausgewählt [24]. Wichtig beim Anschluss der Karte ist, dass der Greifer als «Half-Duplex» angeschlossen werden muss (Abb. 7.18). Dabei wird die Datenleitung für das Senden und Empfangen verwendet. Im aufgeführten Schema wird dargestellt, wie die Klemme mit den entsprechenden Anschlüssen des Kabels verbunden werden muss. Über einen Busskoppler (BK1120) kann die Karte mit einer EtherCAT-Schnittstelle ausgewertet werden. Es war schlussendlich möglich, den Greifer über die KL6041-Karte anzusteuern und die Daten auszulesen.

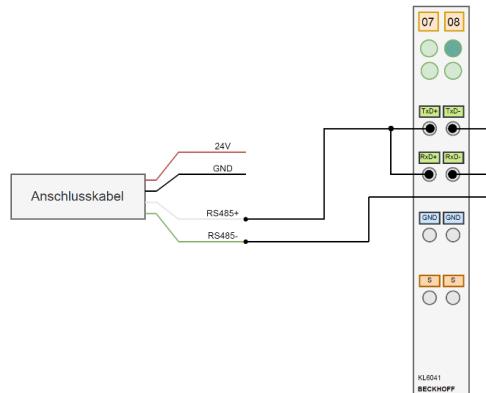


Abbildung 7.18: Anschluss der Beckhoff-Karte



Die Dokumentation der KL6041-Klemme wird im Anhang beigelegt.

Die Karte muss mit dem Konfigurationstool KS2000 von Beckhoff (Abb. 7.19) korrekt konfiguriert werden. Die entsprechenden Einstellungen können aus der Dokumentation der KL6041 entnommen werden. Anschliessend ist diese verknüpfungs- und einsatzbereit.

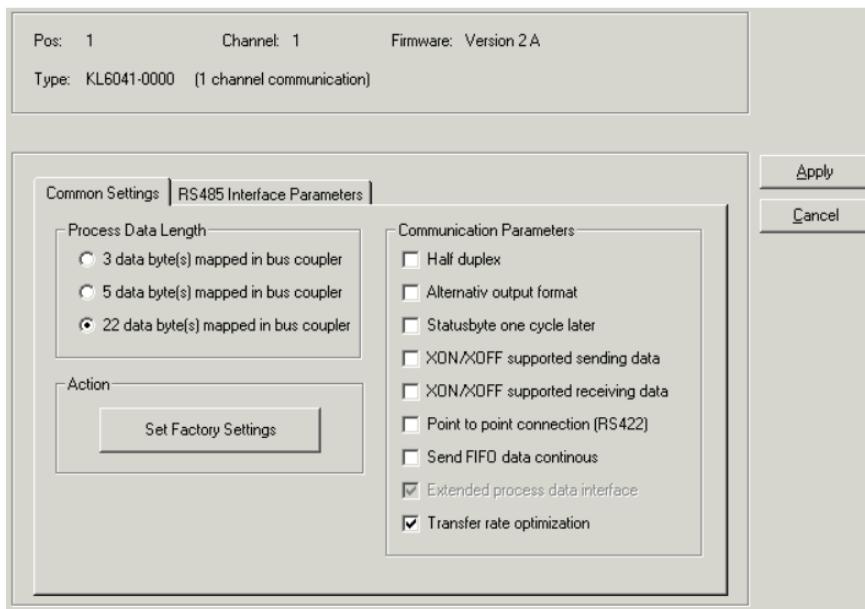


Abbildung 7.19: KS2000-Konfigurationstool

Das Paket TF6255 stellt für den Einsatz einer KL6041-Karte einen separaten Funktionsbaustein zur Verfügung (Abb. 7.20).

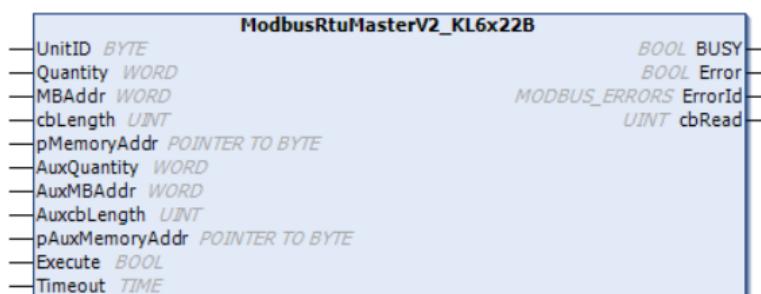


Abbildung 7.20: Kommunikationsbaustein für Modbus RTU

Der Funktionsbaustein erstellt anhand der Eingangsvariablen den zu sendenden Befehlsframe für die Modbus RTU Kommunikation. Folgende Eingangsvariablen sind relevant für die korrekte Erstellung des Frames:

Bezeichnung	Beschreibung	Beispielwert (Tabelle 7.10)
UnitID	Slave-ID des Greifers	09 (Hex) 9 (Dez)
Quantity	Anzahl der zu beschreibenden Register	03 (Hex) 3 (Dez)
MBAddr	Adresse des ersten Registers	03E8 (Hex) 1000 (Dez)
cbLength	Anzahl der zu sendenden Daten in Bytes	06 (Hex) 6 (Dez)

Tabelle 7.11: Eingabeparameter für Funktionsbaustein

Der folgende Bildausschnitt aus der Methode «M_PositionAnfahren» veranschaulicht, wie eine entsprechende Implementierung in TwinCAT aussehen kann. Die Slave-ID wurde dabei bereits übergeordnet definiert, da sie für alle Befehle unverändert bleibt.



Auch ein wichtiger Aspekt bei der Zusammenstellung der Daten für die Register ist die Problematik von High- und Low-Bit. Die Registerdaten sind im Vergleich zur Dokumentation des Greifers vertauscht. Konkret bedeutet dies, dass die Array-Nummern aus dem Beispiel (Tab. 7.10) [7, 8, 9, 10, 11, 12] in TwinCAT zu [8, 7, 10, 9, 12, 11] angepasst werden müssen. Das gleiche Prinzip gilt auch für das Auslesen der Register.

Der Funktionscode wird durch die ausgewählte Aktion des Funktionsblocks definiert. Für dieses Beispiel (Abb. 7.21) wurde der Funktionscode 16 «Communication.WriteRegs» ausgewählt. Der Basis-Funktionsbaustein, in diesem Fall «Communication» kann nicht in seiner Grundform aufgerufen werden, sondern nur zusammen mit einer Aktion.

```
// Definieren der Anzahl von Daten-Bytes
Quantity := 3;

// Definieren von Register
MBAddr := 1000;

// Definieren der Daten
DataSend[0] := 0;
DataSend[1] := 9;
DataSend[2] := iPos;
DataSend[3] := 0;
DataSend[4] := iKraft;
DataSend[5] := iGeschw;

// Senden von Daten an entsprechendes Register
Communication.WriteRegs(
    Quantity      := Quantity,
    MBAddr        := MBAddr,
    cbLength     := SIZEOF(DataSend),
    pMemoryAddr  := ADR(DataSend),
    Execute       := TRUE);
```

Abbildung 7.21: Anwendung des Kommunikationsbausteines

7.2.4 Schnittstelle zwischen Anlagenmodell und realer Anlage

Für die Interaktion zwischen Anlagenmodell und der realen Anlagen werden komponentenspezifische Schnittstellen benötigt.

- UR5-Roboter: Der Kontroller und somit der Roboter wird über einer Ethernet-Schnittstelle mit der SPS (Laptop) verbunden.
- HEX-E-Sensor: Der Sensor wird an eine Signalverarbeitungsbox von OnRobot angeschlossen. Die Box kann mit einer Ethernet-Schnittstelle mit der SPS (Laptop) verbunden werden.
- 2F-85-Greifer: Wie in Kapitel 7.2.3 beschrieben, wird für die Interaktion mit dem Greifer eine KL6041-Karte von Beckhoff eingesetzt. Diese wird wiederum an einen Beckhoff-Buskoppler (BK1120) angeschlossen. Dieser kann über einer Ethernet-Schnittstelle mit der SPS (Laptop) verbunden werden.

Um die Schnittstelle übersichtlicher zu gestalten, wurde eine einfache Halterung umgesetzt, auf welcher alle Schnittstellenkomponenten montiert werden können. Die Vorrichtung enthält die Signalverarbeitungsbox, die Beckhoff-Komponenten, Anschlussklemmen das Greiferkabel, ein 24V-Netzteil und eine Ethernet-Switch.

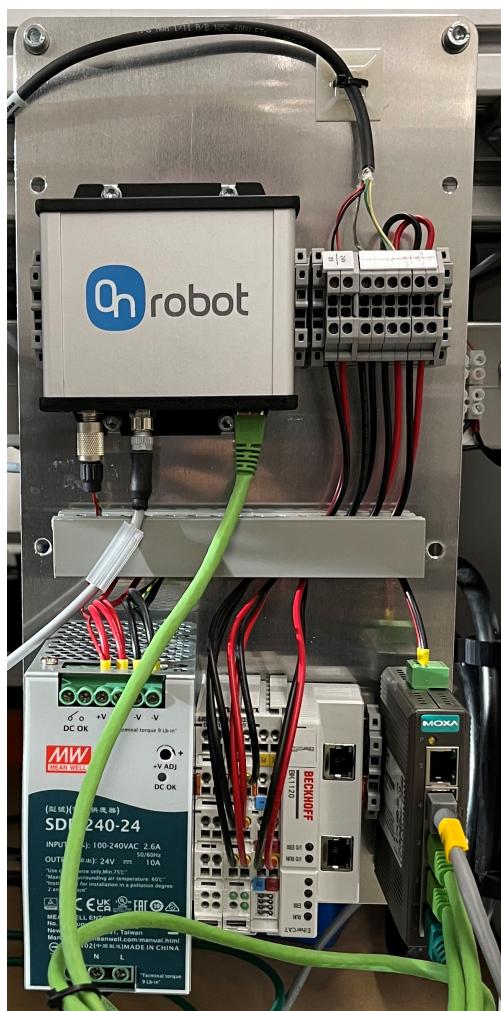


Abbildung 7.22: Schnittstelle zwischen Anlagenmodell und realer Anlage

8 Entwicklung des Prozessmodells

Das Prozessmodell ist für den korrekten Ablauf des Prozesses zuständig, Arbeitsplan, Sequenzen und Skills sind Teil des Prozessmodells. Wie in Kapitel 2.3.2 beschrieben, wird das Prozessmodell so definiert, dass dieses anlagenunabhängig aufgebaut werden kann. Das Modell muss nicht wissen, welche Komponenten im System vorhanden sind, sondern nur die Fähigkeiten dieser. Dadurch können Prozess- und Anlagenmodell parallel entwickelt werden, was die Kosten und Entwicklungszeit verringern kann. Zudem erlaubt die klare Trennung eine flexible Anpassung oder den Austausch von Komponenten im Anlagenmodell, ohne das Prozessmodell zu beeinflussen. Umgekehrt können neue Prozesse schnell und unkompliziert implementiert werden, ohne dass detaillierte Kenntnisse der Anlagenkomponenten erforderlich sind.

Diese Flexibilität führt insgesamt zu einer effizienteren und schnelleren Entwicklung sowie Inbetriebnahme der Software.

8.1 Struktur des Prozessmodells

Die grundlegende Struktur des Prozessmodells lässt sich klar und einfach definieren: Der gesamte Prozess wird durch einen Arbeitsplan abgebildet. Dieser besteht aus verschiedenen Sequenzen und Skills, wobei Sequenzen aus mehreren Skills zusammengesetzt sind (Abb. 8.1). Skills repräsentieren die grundlegenden Funktionalitäten einzelner Komponenten, während Sequenzen die Funktionen des Gesamtsystems oder von Teilsystemen darstellen.

Das Prozessmodell verfügt über zwei zentrale Schnittstellen: zum Anlagenmodell und zum HMI. Die Interaktion mit dem Anlagenmodell erfolgt dabei ausschließlich über die Skills. Über das HMI wird der Arbeitsplan gesteuert, wodurch eine intuitive Bedienung und Anpassung des Prozesses ermöglicht werden soll. Die Struktur des Prozessmodells beeinflusst massgeblich die Flexibilität der Erstellung und Nutzung eines Arbeitsplans. Es stellt sich die Frage, ob das Ziel darin besteht, über ein HMI flexibel Abläufe zusammenstellen zu können, oder ob lediglich ein vordefinierter Ablauf gestartet werden soll. Beide Ansätze erfordern unterschiedliche Programmstrukturen:

Ansatz 1: Vordefinierte Abläufe (Abb. 8.2a)

Hier werden feste Abläufe vorab definiert und können mit minimalem Aufwand gestartet werden. Dies erfordert eine weniger dynamische, dafür jedoch robuste Struktur, die auf vorgefertigten Arbeitsplänen basiert.

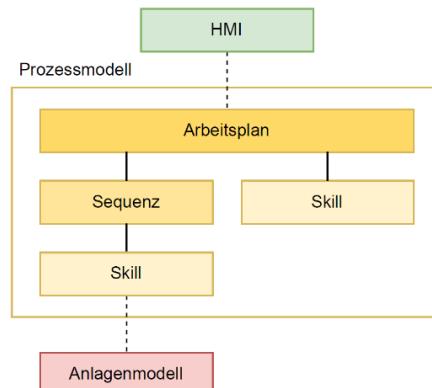


Abbildung 8.1: Grundstruktur des Prozessmodells

Ansatz 2: Flexible Erstellung über das HMI (Abb. 8.2b)

Diese Variante ermöglicht eine dynamische Konfiguration von Abläufen direkt über die Benutzeroberfläche. Hierfür ist eine modulare und erweiterbare Programmstruktur notwendig, die es erlaubt, Sequenzen und Skills zur Laufzeit zu definieren und zu kombinieren. Folgende Schemas zeigen grob die Struktur dieser Ansätze:

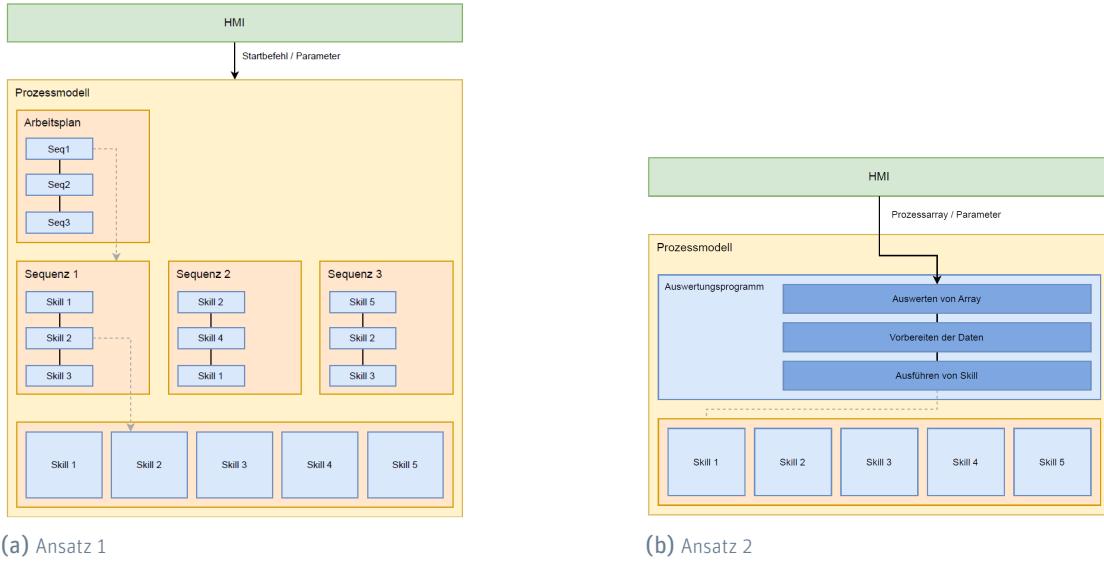


Abbildung 8.2: Ansatzvergleich

Ansatz 1 folgt einer bewährten Methode, wie sie bereits in TwinCAT für Rezeptursteuerungen eingesetzt wird. Hier werden die Prozessabläufe durch vordefinierte Schrittabfolgen realisiert, die je nach Bedarf ausgelöst werden. Über das HMI erfolgt lediglich der Start des Prozesses sowie die Eingabe der erforderlichen Parameter.

- | | |
|------------|---|
| Vorteile: | <ul style="list-style-type: none"> ▶ Klare und übersichtliche Struktur. ▶ Einfache Implementierung dank zahlreicher Referenzen und standardisierter Konzepte. ▶ Schnelle Umsetzbarkeit. |
| Nachteile: | <ul style="list-style-type: none"> ▶ Eingeschränkte Flexibilität: Abläufe können nicht dynamisch angepasst oder neu erstellt werden. ▶ Begrenzte Übertragbarkeit auf vollständig flexible Prozesse. |

Ansatz 2 bietet eine wesentlich flexiblere Gestaltung der Prozessabläufe. Über das HMI können Abläufe frei durch die Kombination von Skills und Sequenzen definiert werden. Jeder Skill wird durch eine eindeutige Nummer repräsentiert, während Sequenzen als Arrays von Nummern aufgebaut sind, die den Prozessablauf abbilden. Ein Programm im Prozessmodell analysiert dieses Array und führt die Skills in der definierten Reihenfolge aus.

- | | |
|------------|---|
| Vorteile: | <ul style="list-style-type: none"> ▶ Hohe Flexibilität: Prozesse können individuell und dynamisch über die HMI erstellt werden. ▶ Anpassbar an variable Anforderungen oder neue Arbeitsabläufe. |
| Nachteile: | <ul style="list-style-type: none"> ▶ Höhere Komplexität bei der Implementierung. ▶ Mögliche Herausforderungen bei parallelen Prozessen, insbesondere in Bezug auf Synchronisation und Timing. |

Zunächst wird die Struktur gemäss Ansatz 1 umgesetzt, da sie bewährt und schnell realisierbar ist. Je nach dem welche Erfahrungen mit dieser Struktur gemacht werden, kann in einer zweiten Iteration auf Ansatz 2 umgestellt werden, um mehr Flexibilität zu ermöglichen. Die Skills bleiben in beiden Ansätzen unverändert und können somit nahtlos übernommen werden.

8.1.1 Struktur eines Arbeitsplans nach Ansatz 1

Die Struktur eines Arbeitsplans ist klar und übersichtlich. Ein Arbeitsplan wird als eigenständiges Programm (PRG) definiert, das den Prozess in Form eines Schrittablaufs abbildet. Jeder Schritt führt entweder Skills oder Sequenzen aus. Innerhalb des Programms werden verschiedene Variablen und Funktionsbausteine instanziert, um den Ablauf zu steuern.

Die Bedienvariablen dienen zum Starten, Stoppen und Zurücksetzen des Schrittablaufs. Prozessvariablen enthalten alle prozessrelevanten Informationen, die für die Ausführung des Arbeitsplans erforderlich sind. Zusätzlich müssen im Arbeitsplan alle benötigten Skills und Sequenzen instanziert werden. Dabei werden ausschliesslich die Skills instanziert, die im Schrittablauf des Arbeitsplans tatsächlich verwendet werden. Skills, die innerhalb von Sequenzen genutzt werden, werden direkt in den Sequenzen selbst instanziert.

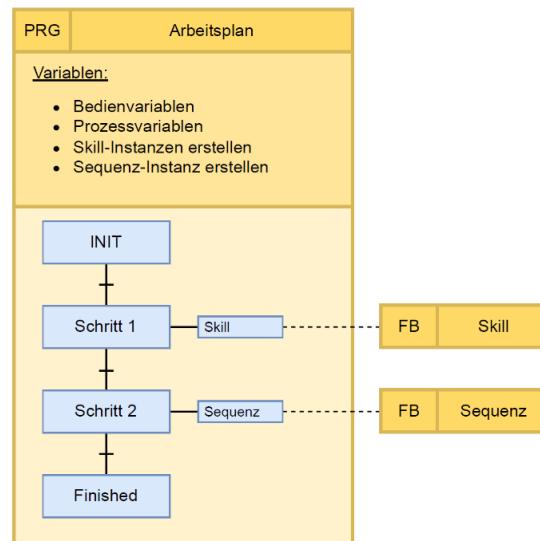


Abbildung 8.3: Struktur des Arbeitsplans

8.1.2 Struktur von Sequenzen

Sequenzen werden analog zu Skills als Funktionsbausteine definiert, wobei auch hier das Ziel darin besteht, eine einheitliche Struktur bereitzustellen. Ein zentraler Bestandteil dieser Struktur ist die Definition der Schnittstellen einer Sequenz, die in zwei Kategorien unterteilt wurden. Die Steuerungselemente sind für die Bedienung der Sequenz vorgesehen. Die Sequenz wird gestartet, gestoppt oder resettet. Zusätzlich werden auch Informationen über den Zustand der Sequenz angegeben. Der Arbeitsplan startet hierbei die Sequenz oder resettet diese. Über den Zustand weiss der Arbeitsplan, wann eine Sequenz abgeschlossen wurde.

Die Betriebsvariablen sind für den Betrieb der Sequenz angedacht. Im Gegensatz zum Skill wurden die Prozessparameter hier direkt in die Betriebsvariablen integriert.

Die Struktur der Sequenz hat während der Entwicklung mehrere Iterationen durchlaufen. In der Dokumentation wird jedoch ausschliesslich der aktuelle Stand beschrieben, da nicht alle Entwicklungsschritte detailliert erläutert werden. Während der Entwicklungs- und Testphase wurden zahlreiche Erkenntnisse und Erfahrungen gesammelt, die die Struktur massgeblich beeinflusst haben. Viele Anpassungen waren notwendig, um TwinCAT-spezifische Probleme zu bewältigen. Dabei konnte wertvolles Know-how über TwinCAT und dessen Arbeitsweise gewonnen werden.

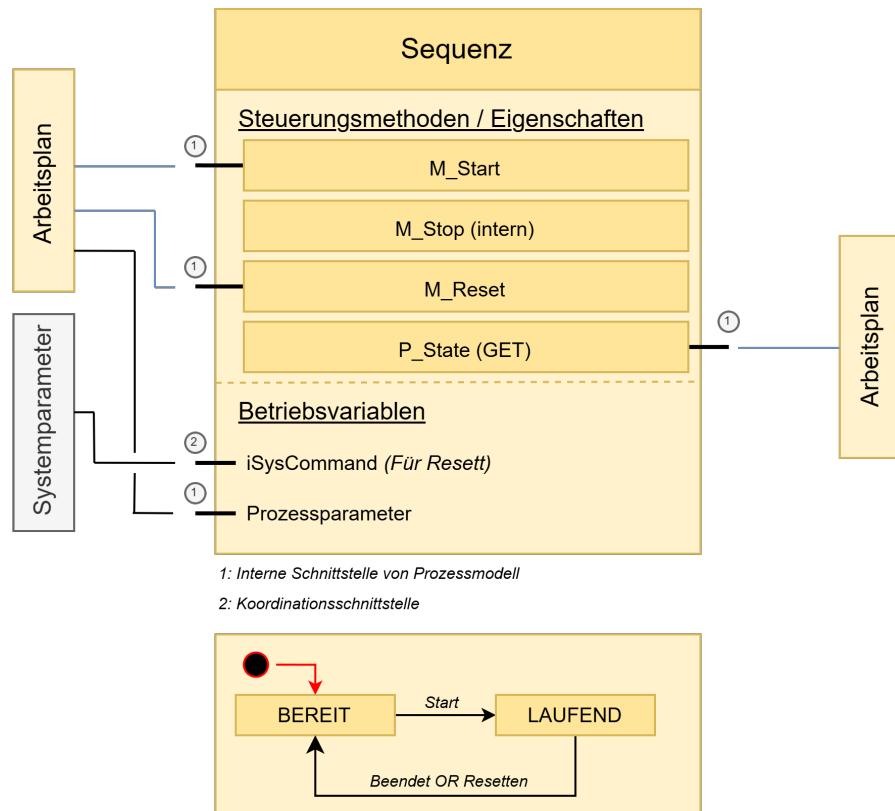


Abbildung 8.4: Struktur einer Sequenz

Die Steuerungselemente umfassen die gleichen Methoden und Eigenschaften wie die Skills, die Umsetzung dieser unterscheidet sich jedoch.

Art:	Bezeichnung:	Typ:	Beschreibung:
Methode	M_Start	BOOL	Methode zum Starten der Sequenz und des Schrittablaufs
Methode	M_Stop	BOOL	Methode zum Beenden des Schrittablaufs (Intern)
Methode	M_Reset	BOOL	Methode zum Resetten der Sequenz und des Schrittablaufs
Eigenschaft	P_State	INT	Eigenschaft des aktuellen Zustandes

Tabelle 8.1: Steuerungselemente einer Sequenz

8.2 Umsetzung in TwinCAT

8.2.1 Allgemeine Struktur

Innerhalb der Umsetzung wurden zwei verschiedenen Strukturen erarbeitet und getestet.

Struktur 1: Skills instanziert in separatem Programm

In der ersten Struktur wurden die Skills in einem Programm namens «SkillSet» instanziert, vergleichbar mit den Objekten innerhalb der Anlage. Diese instanziierten Skills können anschliessend vom Arbeitsplan oder über Sequenzen verwendet werden. Der Arbeitsplan greift dabei direkt auf die im SkillSet-Programm instanzierten Skills zu. Sequenzen hingegen erhalten die Skills über Eingangsvariablen, die mit dem entsprechenden Interface definiert wurden.

Der Vorteil dieser Struktur liegt darin, dass die Skills kontinuierlich ausgeführt werden, wodurch jederzeit der aktuelle Zustand eines Skills festgestellt werden kann. Die Interaktion mit den Skills wird durch diese Struktur einfacher und transparenter (Probleme bei der Interaktion werden im Zusammenhang mit der zweiten Struktur genauer beschrieben).

Ein Nachteil dieser Struktur besteht darin, dass alle benötigten Skills vorab im SkillSet-Programm instanziert werden müssen. Beispielsweise: Wenn ein Roboter 10 Skills besitzt und das System um einen zweiten Roboter erweitert wird, müssen auch für diesen zweiten Roboter die 10 entsprechenden Skills im SkillSet-Programm instanziert werden. Dadurch entsteht eine gewisse Abhängigkeit des Prozessmodells vom Anlagenmodell.

Ein weiteres Risiko ist die mögliche Verwechslung von Skills, insbesondere bei zwei gleichartigen Skills für unterschiedliche Komponenten. Wenn beispielsweise Roboter A angeprochen werden soll, könnte versehentlich der Skill von Roboter B ausgewählt werden.

Struktur 2: Instanziierung von Skills innerhalb von Arbeitsplänen und Sequenzen

Bei dieser Struktur werden ausschliesslich die Skills innerhalb des Arbeitsplans oder der Sequenz instanziert, die tatsächlich benötigt werden. Auch die Zuweisung der Objekte erfolgt direkt innerhalb des Arbeitsplans oder der Sequenz. Eine Interface-Implementierung der Skills ist hierbei nicht erforderlich. Der entsprechende Skill wird innerhalb eines Schritts aufgerufen, alle notwendigen Parameter (z.B. das Objekt) werden zugewiesen, und der Skill wird ausgeführt.

Ein wesentlicher Vorteil der Struktur besteht darin, dass die Skills lokal in dem jeweiligen Arbeitsplan oder der Sequenz instanziert werden. Ebenso erfolgt die Zuweisung der Objekte lokal, wodurch das Prozessmodell keine Informationen über das Anlagenmodell benötigt, wie etwa die Anzahl der im System vorhandenen Roboter. Stattdessen wird der be-

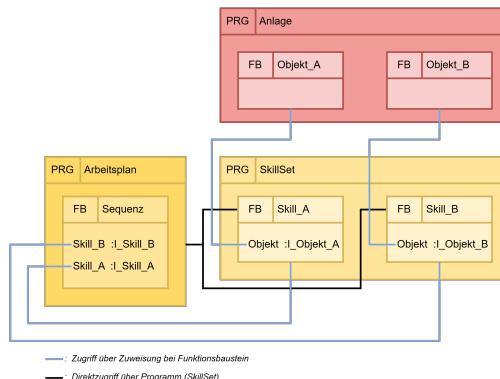


Abbildung 8.5: Prozessmodell-Struktur 1

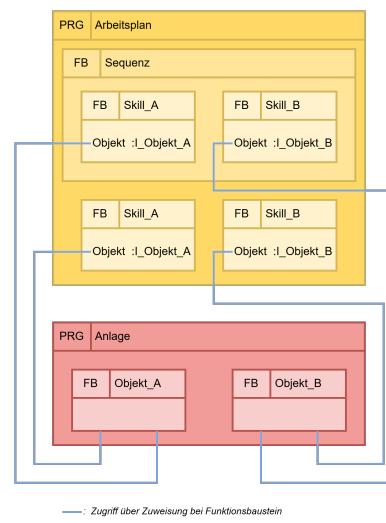


Abbildung 8.6: Prozessmodell-Struktur 2

nötigte Roboter einfach dem entsprechenden Skill innerhalb des Arbeitsplans oder der Sequenz zugewiesen. Diese Herangehensweise verbessert zudem die Übersichtlichkeit und Verständlichkeit der Software: Durch die instantiierten Skills lässt sich schnell erkennen, welche Aufgaben eine bestimmte Sequenz oder ein Arbeitsplan ausführt.

Ein Nachteil dieser Struktur ist jedoch, dass beim Aufruf, der Zuweisung und der Ausführung der Skills potenziell mehr Probleme auftreten können. Es ist essenziell, ein Verständnis darüber zu haben, wie lange Informationen für die Verarbeitung benötigen und wie auf Methoden zugegriffen werden kann. Diese Struktur erfordert daher ein fundiertes Wissen über TwinCAT und dessen Funktionsweise.

8.2.2 Aufbau einer Sequenz

Der Schwerpunkt liegt dabei, den allgemeinen Aufbau einer Sequenz zu erklären. Es werden nicht alle definierten Sequenzen aufgezeigt und beschrieben. Die Sequenzen unterscheiden sich grundsätzlich nur in den definierten Ablaufschritten und den verwendeten Skills. Zusätzlich konnten zum Zeitpunkt dieser Dokumentation noch nicht alle Sequenzen komplett abgeschlossen werden.

Eine Sequenz basiert auf einem Funktionsbaustein, der in strukturiertem Text (ST) erstellt ist. Der Funktionsbaustein umfasst verschiedene Aktionen. Eine Aktion ist für den Ablauf verantwortlich, der als SFC umgesetzt wird. Die übrigen Aktionen bestehen aus Zuweisungen, die ebenfalls in strukturiertem Text geschrieben sind. Diese dienen innerhalb des Ablaufs dazu, Skills aufzurufen und Parameter an diese zu übergeben. Über den Funktionsbausteine «FB_Basis» werden der Sequenz die Steuerungselemente vererbt.

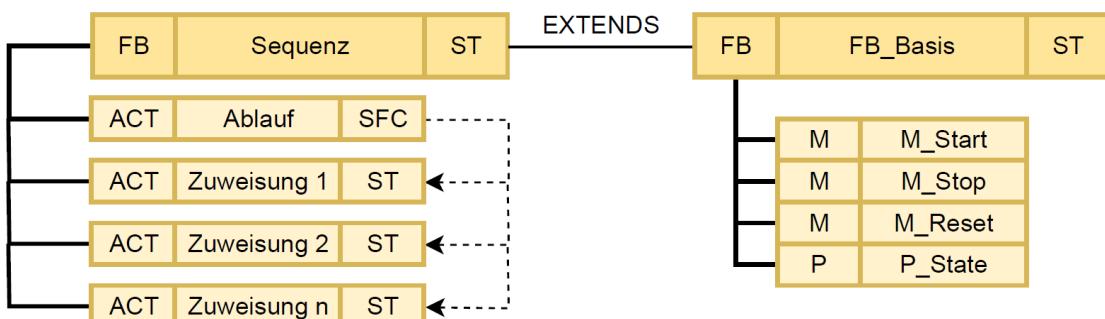


Abbildung 8.7: Aufbau einer Sequenz

Als Beispiel wird die Sequenz «Seq_Teile_Hohlen» betrachtet. Die Aufgabe dieser Frequenz ist es, mit dem Roboter an eine definierte Position zu fahren und ein Teil mit dem Greifer zu packen.

Der Funktionsbaustein wurde dabei wie folgt definiert:

<pre> FUNCTION_BLOCK Seq_Teile_Hohlen EXTENDS FB_Basis VAR_INPUT // Prozessparameter Param_StartPos :sRoboterProzessVariablen; Param_Backenbreite :USINT; // Objekt-Schnittstellen I_Roboter :I_Objekt_Roboter; I_Greifer :I_Objekt_SmartGreifer; END_VAR VAR_OUTPUT END_VAR VAR // Skill-Instanzierung Skill_R_P2P :Skill_Position_anfahren; Skill_G_P2P :Skill_Greifer_Pos_Anfahren; // Interne Prozessvariablen LRobotVar :sRoboterProzessVariablen; LGreiferVar :sSmartGreiferProzessVariablen; LOffsetVar :LREAL; END_VAR </pre>	<pre> // Ablauf ACT_Ablauf(); // Zustandshandling CASE iState OF 0: // BEREIT IF bStart THEN iState := 1; END_IF 1: // LAUFEND IF eSysCommand = 4 THEN bReset := TRUE; SFCReset := TRUE; END_IF IF bBeendet OR bReset THEN bReset := FALSE; SFCReset := FALSE; iState := 0; END_IF 2: // ABGESCHLOSSEN IF SFCCurrentStep = 'Init' THEN iState := 0; END_IF END_CASE </pre>
---	--

(a) Variablen-deklaration

(b) Funktionsbausteinprogrammierung

Abbildung 8.8: Sequenz-Funktionsbaustein

Für die Funktionalität des Bausteins müssen zwei Prozessparameter sowie Objekte als Eingabeveriablen definiert werden. Die Prozessparameter legen die Startposition und die Greifer-Backenbreite im geöffneten Zustand fest. Die erforderlichen Objekte umfassen den Roboter und den Greifer, die jeweils über das entsprechende Interface instanziert werden. Innerhalb der internen Variablen werden sowohl die benötigten Skills als auch Variablen zur Zuweisung der Prozessparameter initialisiert.

Im Baustein selbst wird zunächst der Ablauf «ACT_Ablauf» ausgeführt. Diese Aktion läuft kontinuierlich. Das Zustandshandling erfolgt durch eine CASE-Struktur, die den aktuellen Zustand verarbeitet und die entsprechenden Aktionen steuert.

Die Aktion «ACT_Ablauf» ist wie folgt definiert:

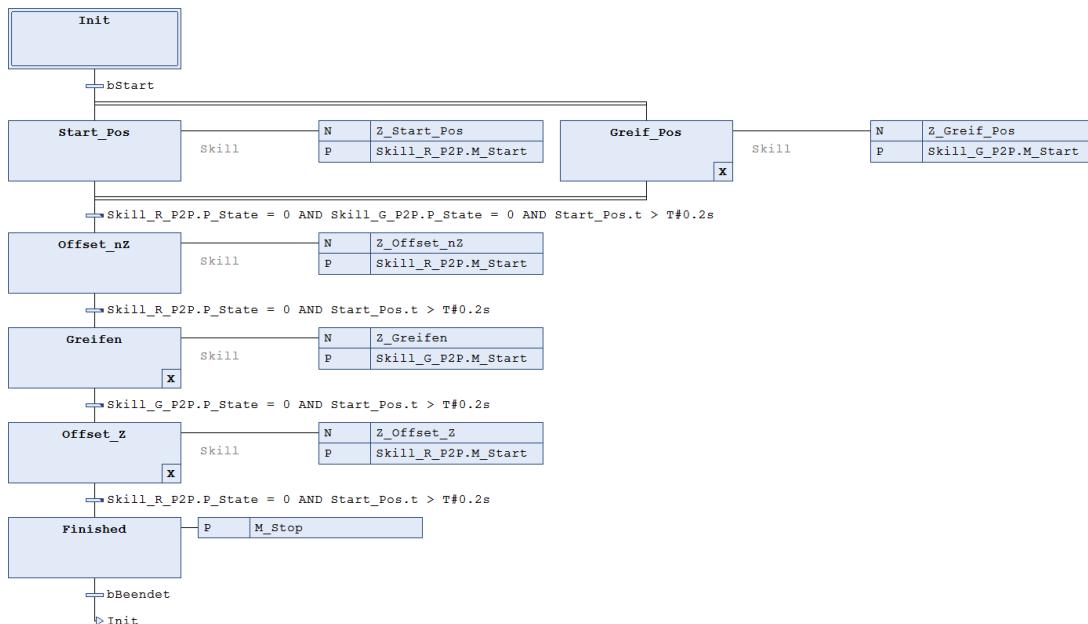


Abbildung 8.9: Beispiel für Sequenzablauf

Zu Beginn wird der Roboter und Greifer in eine definierte Startposition gebracht. Von dieser Position fährt der Roboter nach unten und greift nach dem Teil. Als letzter Schritt bewegt sich der Roboter mit dem Teil nach oben. Ein grundsätzlich sehr einfacher und überschaubarer Ablauf. Mit der Variablen Startposition für Roboter und Greifer, kann diese Sequenz für alle Elemente der Anwendung so übernommen werden.

Jeder Schritt ruft zwei Aktionen auf. Innerhalb der ersten Aktion wird der Skill aufgerufen und die Parameter werden zugewiesen. Diese Aktion wird kontinuierlich durchgeführt, so lange der Schritt aktiv ist.

```
// Skillaufruf
Skill_R_P2P(eSysCommand := GVL.SystemCommand,
             eSysState := GVL.SystemState,
             Objekt := I_Roboter,
             iErrorID => GVL.ErrorID);

// Parameterbestimmung
LRobotVar := Param_StartPos;

// Parameterzuweisung
Skill_R_P2P.P_RobotParam := LRobotVar;
```

Abbildung 8.10: Zuweisungsaktion der Sequenz

Die zweite Aktion führt die die Methoden zum Starten des Skills aus. Im Moment werden die Start-Methoden mit dem SFC-Qualifizierer «P» umgesetzt. Das «P» steht dabei für «Pulse». In TwinCAT bedeutet dies, dass die Methode zwei Mal durchgeführt wird: Einmal, wenn der Schritt aktiv wird und ein zweites Mal im darauffolgenden Zyklus. Dieses Verhalten hat zu Problemen geführt. Zusätzlich kann mit dem Qualifizierer «P» auch nicht innerhalb eines SFC-Ablaufs zwei Mal, in separaten Schritten, auf eine Methode zugegriffen werden. Aus im Moment unbekannten Gründen, ist die Methode nach dem ersten Schritt gesperrt.

Es stehen zwei mögliche Herangehensweisen zur Verfügung:

Möglichkeit 1: Tiefergehende Analyse des TwinCAT-Verhaltens

Ziel wäre es, eine Lösung für das Problem zu finden, sodass weiterhin mit dem Qualifizierer «P» gearbeitet werden kann.

Möglichkeit 2: Umstellung auf den Qualifizierer «N»

Mit «N» wird die Methode so lange ausgeführt, wie der Schritt aktiv ist. Dabei wird die Methode in jedem Zyklus erneut aufgerufen. Um Fehler zu vermeiden müsste die Methode so umgeschrieben werden, dass diese trotzdem nur einmal ausgelöst wird.

Diese Problematik wurde zum Zeitpunkt dieser Dokumentation noch nicht gelöst. Die Transition zum nächsten Schritt wird basierend auf dem Zustand des Skills ausgelöst. Der Skill muss sich dafür wieder im Zustand «BEREIT» befinden. Zusätzlich ist sichergestellt, dass der aktuelle Schritt mindestens 0,2 Sekunden lang ausgeführt wird. Diese zeitliche Verzögerung gibt dem Skill ausreichend Zeit, um ordnungsgemäß zu starten. Ohne diese Bedingung würde die Transition sofort TRUE werden, sobald der Schritt aktiv wird, da sich der Skill bereits vor dem Start im Zustand «BEREIT» befindet.

8.2.3 Aufbau eines Arbeitsplanes

Der Arbeitsplan wird als Programm umgesetzt, das in Form eines Schrittablaufs (SFC) strukturiert ist. Der grundlegende Aufbau entspricht dem des SFC einer Sequenz. Jeder Schritt enthält zwei Aktionen: eine Zuweisung, die als CFC realisiert ist, sowie eine Aktion zum Starten der Sequenz oder des Skills.



Abbildung 8.11: Aufbau eines Arbeitsplans

Bei der Variablen Deklaration des Arbeitsplanes werden folgende Elemente instanziert:

- ▶ Bedienparameter
- ▶ Prozessparameter
- ▶ Skill-Instanzen
- ▶ Sequenz-Instanzen

Die Bedienparameter dienen zum Start des Ablaufs. Unter Prozessparameter werden alle Parameter zusammengefasst, die für den Betrieb des Arbeitsplans erforderlich sind. Alle Informationen, die als Eingabevariablen für Skills und Sequenzen verwendet werden, müssen in diesem Bereich definiert sein. Die Instanzen von Skills und Sequenzen repräsentieren die im Arbeitsplan verwendeten Skills und Sequenzen.

Die Prozessparameter könnten über eine «Rezeptverwaltung» verwaltet werden. Damit bietet TwinCAT die Möglichkeit benutzerdefinierte Variabellenlisten zu erstellen und zu verwalten. Für einen Arbeitsplan könnte dadurch einfach ein bestimmtes Rezept geladen werden, welches sämtliche Prozessvariablen enthält. Dieses Feature wurde innerhalb dieser Arbeit noch nicht implementiert.

9 Auswertung

9.1 Aktueller Stand und Erkenntnisse

Der mechanische Aufbau für die Umsetzung der definierten Anwendung konnte umgesetzt in Betrieb genommen werden. Der einfach gehaltene Aufbau erfüllt dabei seinen Zweck und konnte für das Testen der Software eingesetzt werden. Die Software erfüllt dabei die grundlegend gestellten Anforderungen, befindet sich jedoch noch in der Entwicklung und somit im Prototypenstatus.

Mit dem Prototypen kann bereits gezeigt werden, dass die definierte Struktur funktioniert. Abläufe können erstellt und durchgeführt werden. Dabei greifen diese auf die definierten Skills zu, welche wiederum mit den umgesetzten Objekten interagieren. Der allgemeine Workflow der Software konnte dadurch getestet und in ersten Schritten optimiert werden. Die umgesetzten Objekte erfüllen dabei ihre spezifischen Aufgaben. Dies sind einige der Vorteile der umgesetzten Software-Struktur:

- ▶ Einfache Umsetzung von Abläufen.
- ▶ Einfache Anpassung von bestehenden Abläufen.
- ▶ Übersichtlich- und Verständlichkeit der Software.
- ▶ Gesamte Funktionalität der Anlage wird in der SPS zusammengefasst.
- ▶ Anlagen- und Prozessmodell können separat voneinander aufgebaut werden.
- ▶ Komponenten können einfach ergänzt oder ausgetauscht werden, ohne grosse Veränderungen am Prozessmodell.

Einige vorgesehene Elemente der Software konnten jedoch noch nicht umgesetzt werden. Die Einbindung des Vision-Systems in den Prozess wurde aus zeitgründen weggelassen. Auch die Umsetzung des HMI für die Anlagenkomponenten und des Gesamtsystem-HMI wurde noch nicht gemacht. Das momentane HMI ist nur für das schnelle Testen vorgesehen. Das System kann dabei eingeschaltet und der Prozess gestartet werden.

Auch der komplette Ablauf konnte zum Stand der Dokumentation noch nicht umgesetzt werden. Der Fokus während der Entwicklung wurde auf eine stabile Struktur gelegt, da dies das Fundament der Software darstellt. Viel Zeit wurde dadurch in das Testen und optimieren der Struktur investiert. Sobald dieser robust und sauber umgesetzt ist, ist die Umsetzung des kompletten Ablaufs kein grosser Aufwand mehr. Der Ablauf wurde entsprechend soweit aufgebaut, dass alle Funktionalitäten und Interaktionen getestet werden konnten. Ein aktueller Fehler, welcher noch behoben werden muss, tritt beim Starten von Skills auf (in Kapitel 8.2.2 beschrieben). Dies und weitere kleinere Optimierungen sollten in den Wochen, nach Abgabe dieser Arbeit, noch umgesetzt werden können.

9.2 Vergleich mit Anforderungen

Alle Anforderungen werden mit der erarbeiteten Lösung verglichen und auf die in Kapitel 2.6 definierten Anforderungen bezogen. Die Anforderungen werden wie folgt bewertet:

Goals	Objectives	Beschreibung	Erfüllt
G1	A1	Ein HMI soll die Möglichkeit bieten, Arbeitspläne zu definieren (Ablauf / Parameter).	Nein
	A2	Ein HMI soll die Möglichkeit bieten, das System zu bedienen (Start, Stop, Reset)	(Nein)
	A3	Die Software soll sich nach der ANSI/ISA-88-Norm richten und ein Prozess- und Anlagemodell besitzen.	Ja
	A4	Die Softwarestruktur soll eine anlagenunabhängige Definition des Prozessmodells ermöglichen	Ja
	A5	Die Funktionalität der Systemkomponenten soll im Anlagenmodell abgebildet werden	Ja
	A6	Die Objektklassen der Systemkomponenten sollen objektorientiert aufgebaut werden	Ja
	A7	Alle Komponenten sollen über ein eigenes HMI im manuell-Modus betrieben werden können.	Nein
G2	A1	Ein Standard soll definiert werden für die Definition eines Skills	Ja
	A2	Ein Skill sollten die Funktionalität einer Komponente abbilden können	Ja
	A3	Das System soll in der Lage sein, Prozessschritte zu überprüfen	Nein
	A4	Das System soll Korrekturen bei nicht erfüllten Prozessschritten vornehmen	(Ja)
	A5	Die Skills sollen zu Sequenzen zusammengefügt werden können	Ja
	A6	Die Skills und Sequenzen sollen zu Arbeitsplänen zusammengefügt werden können	Ja
G3	A1	Der Versuchsaufbau soll eine Vision-System besitzen	Nein
	A2	Der Versuchsaufbau soll ein Robotersystem mit Greifer besitzen	Ja
	A3	Der Versuchsaufbau soll in einer Laborumgebung betrieben werden	Ja
	A4	Der Roboter soll unterschiedliche Objekte greifen können	Ja
	A5	Die Software soll mit TwinCat umgesetzt werden	Ja
	A6	Der Versuchsaufbau kann eine OPC-UA-Kommunikationsschnittstelle besitzen	Nein

Tabelle 9.1: Vergleich mit Anforderungen

Viele der Anforderungen wurden mit der erarbeiteten Lösung erfüllt. Dennoch gibt es einige Anforderungen, die nicht oder nur teilweise erfüllt wurden:

- G1 / A1: Im Moment wurde nur ein sehr einfaches und provisorisches HMI realisiert, damit der allgemeine Ablauf getestet werden kann. Das HMI bietet nicht die Möglichkeit verschiedene Abläufe oder Parameter vorzugeben.
- G1 / A2: Das HMI kann grundsätzlich das System starten, stoppen und resetten. Die Umsetzung ist jedoch noch so rudimentär, dass die Anforderung als nicht erfüllt betrachtet wird. Das HMI dient nur dem Testen der Anlage und stellt nicht eine, vom Endbenutzer, anwendbare Lösung dar.
- G1 / A7: Für die verschiedenen Objektklassen wurden noch keine HMI erstellt.
- G2 / A3: Die ursprünglich angedachte Vision, dass sich das System selbstständig korrigieren kann, wurde nicht implementiert. Im Verlauf des Projektes wurde festgestellt, dass dieses Feature nur wenig Überschneidungen mit dem definierten Thema hat und grundsätzlich in einem eigenen Projekt umgesetzt werden müsste. Der Fokus dieser Arbeit wurde auf die Software-Struktur gelegt.
- G2 / A4: Das System reagiert zwar nicht selbstständig auf nicht erfüllte Prozessschritte, über die Zustände der Skills kann jedoch im Ablauf auf gewisse Situationen reagiert werden (z.B. das Erreichen eines Schwellenwertes).
- G3 / A1: Aus Zeitgründen wurde die Integration eines Vision-System nicht umgesetzt.
- G3 / A6: Für den Aufbau wurde keine OPC-UA-Schnittstelle benötigt.

9.3 Weiterführende Arbeiten

Kurzfristig:

Die unmittelbar weiterführenden Arbeiten beziehen sich auf das Abschliessen der Software für die definierte Anwendung. Dazu gehören folgende Punkte:

- ▶ Allgemeine Optimierung der Software (Stabilität, Bug-Behebung).
- ▶ Umsetzung von HMI und sammeln von Erfahrungen mit TE2000.
- ▶ Implementierung von Kamerasystem in Prozess.

Mit dem gesammelten Know-How und den gemachten Erfahrungen sollte die gesamte Struktur in einer weiteren Iteration nochmals optimiert und angepasst werden. Viele Elemente können einfacher, effizienter oder eleganter gelöst werden.

Mittelfristig:

In Kontakt treten mit Unternehmen, für welche eine solche Software-Struktur interessant sein könnte, um folgende Fragen zu klären:

- ▶ Was ist die allgemeine Meinung zu dieser Struktur und ihrer Vorteile.
- ▶ Für welche Unternehmen könnte eine solche Struktur interessant sein.
- ▶ Wie könnte diese Struktur bei Unternehmen implementiert werden.
- ▶ Welche Anforderungen stellen Unternehmen an die Struktur.

Das Ziel sollte sein, einen Überblick über den Markt und seiner Interessen und Anforderungen zu haben. Daraus kann eingeschätzt werden, welches Potential eine solche Software-Struktur im Markt haben könnte.

Langfristig:

Langfristig sollen die Erkenntnisse aus der Erarbeitung der Software-Struktur mit der Analyse des Marktes verbunden werden. Die Struktur soll anhand der Bedürfnisse des Marktes ausgebaut und optimiert werden. Mit dem Ziel, möglichst alle Anforderungen der Industrie (innerhalb eines sinnvollen Kontextes) mit der Software abdecken zu können. Dafür muss das Know-How in Bezug auf Beckhoff und TwinCAT weiter ausgebaut und vertieft werden. Dazu gehört die Entwicklung eines separaten Software-Tools (basierend auf z.B. C#), mit welchen ein TwinCAT-Programm auf Basis der definierten Struktur konfiguriert werden kann. Mit dem «TwinCAT Automation Interface» können TwinCAT-Programme automatisiert erzeugt und konfiguriert werden. Somit könnte es möglich sein, über ein solches Programm Skills und Objekte zu konfigurieren welches anschliessend aufgrund dieser Konfiguration ein komplettes TwinCAT-Projekt erstellt, basierend auf der entwickelten Struktur. Die Möglichkeiten dieser Schnittstelle müssten jedoch in einem ersten Schritt getestet und verstanden werden. Jedoch könnte ein solches Tool interessant für Unternehmen sein, da die Software-Entwicklungszeit massiv reduziert und vereinfacht werden könnte.

10 Schlussfolgerung / Fazit

Das Hauptziel des Projekts, eine Roboteranwendung einfach und standardisiert programmieren zu können, wurde aus meiner Sicht erfolgreich erreicht. In TwinCAT wurde eine Struktur entwickelt, die eine übersichtliche und unkomplizierte Erstellung von Prozessen ermöglicht. Sobald Objektklassen und Skills definiert sind, lässt sich der Prozess lediglich durch das Erstellen von Schrittabläufen realisieren – hierfür sind keine tiefgehenden Programmierkenntnisse erforderlich. Das Entwickeln der Objektklassen setzt jedoch ein gutes Verständnis der zu implementierenden Komponenten voraus.

Das Projekt hat auch gezeigt, dass solche Anwendungen komplett innerhalb einer SPS umgesetzt werden können. Die SPS und vor allem TwinCAT, bieten viele Möglichkeiten eine solche Software zu realisieren und unterschiedlichste Komponenten in das System zu integrieren.

Die Software wurde erfolgreich mit einem Versuchsaufbau getestet, wobei noch Optimierungspotenziale identifiziert werden konnten. Das Testen stellte einen essentiellen Bestandteil der iterativen Entwicklung der Software-Struktur dar und trug massgeblich zur Verfeinerung des Systems bei.

Für mich war das Projekt eine äusserst spannende und lehrreiche Erfahrung, geprägt von vielen Höhen, aber auch einigen Herausforderungen. Zu Beginn war ich aufgrund des offenen Projektauftrags skeptisch und unsicher, in welche Richtung sich das Vorhaben entwickeln würde. Mit der Zeit wurden jedoch der Rahmen und die Zielrichtung immer klarer und somit auch interessanter.

Im Verlauf des Projekts verlagerte sich der Fokus von der reinen Skill-Entwicklung hin zur Gestaltung einer allgemeinen Software-Struktur, die für die Implementierung und Nutzung dieser Skills erforderlich ist. Die Entwicklung dieser Struktur war für mich zweifellos ein Höhepunkt der Arbeit und bereitete mir grosse Freude. Besonders motivierend war das positive Feedback aus der Industrie zu meinem Ansatz, das meinen Antrieb zur Umsetzung nochmals deutlich gesteigert hat.

Ein weiterer spannender Aspekt war die intensive Arbeit mit TwinCAT. Für die Umsetzung der Software musste und konnte viel Neues gelernt werden, was jedoch auch diverse Herausforderungen mit sich brachte. Besonders zeitaufwendig war die Entwicklung der Objektklassen, die eine gründliche Einarbeitung erforderte. Dies hatte einen erheblichen Einfluss auf den Zeitplan. So wurden beispielsweise allein für die Integration einer Modbus-RTU-Verbindung in TwinCAT drei Wochen benötigt, um diese erfolgreich umzusetzen.

Dieser Prozess umfasste mehrere Schritte: Zunächst musste die Arbeitsweise und Struktur des Kommunikationsprotokolls verstanden werden. Anschliessend galt es, dessen Einsatz in TwinCAT zu analysieren und herauszufinden, warum es anfänglich nicht funktionierte. Es wurden mögliche Lösungen erarbeitet, und letztlich konnte eine alternative Schnittstelle erfolgreich in TwinCAT integriert werden. Dieser langwierige Prozess war zwar herausfordernd, doch die Freude war umso grösser, als die Kommunikation schliesslich einwandfrei funktionierte. Dies ist nur eines von vielen Beispielen für die Herausforderungen, die bei der Entwicklung der Objektklassen gemeistert werden mussten.

Abschliessend kann ich sagen, dass ich mit meiner Arbeit sehr zufrieden und stolz bin. Es ist für mich klar, dass ich diese Thematik auch nach Abschluss meines Masters weiterverfolgen möchte. Ich bin gespannt, wie sich das Projekt in Zukunft weiterentwickeln wird.

11 Anhang

1. Auftrag
2. Zeitplan
3. Referenzprojekt (Master-Thesis 2)
4. Normen
5. Datenblätter
6. Fertigungsdaten
7. Arbeitsschritte
8. Software

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Sämtliche Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, habe ich als solche kenntlich gemacht.

Hiermit stimme ich zu, dass die vorliegende Arbeit in elektronischer Form mit entsprechender Software überprüft wird.

15. Januar 2025



Yannick Spatz

Literaturverzeichnis

- [1] Yannick Spatz. *Automatisierung: Chemische Reaktoranlage*. BFH, 2024.
- [2] ISA (Instrument Society of America). Isa88: Batch control.
- [3] Matthias Seitz. *Speicherprogrammierbare Steuerungen in der Industrie 4.0 - Kapitel 6*. Hanser Fachbuch, 2021.
- [4] ABB. *Anwendungshandbuch Externally Guided Motion*. ABB, 2021.
- [5] ABB. Connect your robots using iot gateway - opc ua or mqtt.
- [6] KUKA. Kuka.plc mxautomation.
- [7] KUKA. *KUKA.OPC UA 2.0*. KUKA, 2019.
- [8] Universal Robots. *User Manual Version 1.6 (UR5 with CB2)*. Universal Robots, 2012.
- [9] Beckhoff. Tf6310 | twincat 3 tcp/ip.
- [10] Simon Kaderli. *Skill-basierte Aufgabenplanung für Roboter in Handarbeitsplätzen von Herstellungsprozessen*. BFH, 2020.
- [11] Francesco Rovida und Volker Krueger Matthias Mayr. *SkiROS2: A skill-based Robot Control Platform for ROS*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, USA, 2023.
- [12] Frank Nägele. *Prototypbasiertes Skill-Modell zur Programmierung von Robotern für kraftgeregelte Montageprozesse*. Frauenhofer Verlag, 2021.
- [13] Arvid Hellmich und Steffen Ihlenfeldt Christian Friedrich. Flexible skill-based control for robot cells in manufacturing.
- [14] Anwendung der Automatisierungstechnik. Automatisierungstechnisches engineering modularer anlagen in der prozessindustrie.
- [15] Beckhoff. Twincat mtp: modulare automation mit module type package.
- [16] PLCopen. Higher efficiency in your application software development.
- [17] OnRobot. *HEX-E SENSOR 2.0 DATASHEET*. OnRobot, 2018.
- [18] ROBOTIQ. *2F-85 Instruction Manual*. ROBOTIQ, 2018.
- [19] IPH-Hannover. Automatisierungspyramide: Aufbau und bedeutung.
- [20] Universal Robots. *The URScript Programming Language for SW5*. Universal Robots, 2024 - 5.16.
- [21] Universal Robots. *Realtime Client Interface*. Universal Robots, 2021.
- [22] OnRobot. *DESCRIPTION Compute Box Robot Controller Interface*. OnRobot, 2018.
- [23] Beckhoff. Tf6255 | twincat 3 modbus rtu.

- [24] Beckhoff. Kl6041 | busklemme, 1-kanal-kommunikations-interface, seriell, rs422/rs485.

Abbildungsverzeichnis

1.1	Grobzeitplan der Thesis	3
2.1	Prozessstruktur von ISA-88	6
2.2	Reaktorbeispiel	7
2.3	Prozess- und Anlagenmodell	7
2.4	Softwarestruktur von ISA/ANSI-88	8
2.5	Softwarestrukturvergleich	9
2.6	Kompetenzaufteilung	9
2.7	Beispielszenario	9
2.8	Beispiel für angepasste Struktur	10
2.9	ABB-Schnittstelle über EGM	12
2.10	ABB-Schnittstelle über OPC	12
2.11	KUKA-Schnittstelle über mxAutomation	12
2.12	UR-Schnittstelle über TCP/IP	13
2.13	Roboteranwendung	14
2.14	MTP-Struktur von Beckhoff	20
2.15	PLCopen-Basisfunktionsblock	21
2.16	PLCopen-Basisablauf	21
2.17	System- und Kontextgrenzen	22
3.1	Definierte Arbeitspakete	25
3.2	Definierter Gate-Plan	26
4.1	Anwendungsteilprozesse	27
4.2	Kommunikationstopologie	28
4.3	Kraftsensor	29
4.4	Kommunikationstopologie	29
4.5	Anwendungsteilprozesse	30
4.6	Umgesetzte Anwendung	31
5.1	Strukturübersicht	33
5.2	Gesamtsystemstruktur	34
5.3	Systemstruktur	35
5.4	Schnittstellenübersicht	36
5.5	Minimale Systemzustände	37
5.6	Minimale Systemzustände	37
5.7	Minimale Objektzustände	38
6.1	Skill-Integration in System	39
6.2	Schnittstellenstruktur	42
6.3	Skilldefinition und Schnittstellen	44
6.4	Skillkonfiguration	45
6.5	Skill-Interaktion	46
6.6	Umsetzungsstruktur	47
6.7	Definitionen für Skill-Basis-FB	47

6.8	Steuerungsmethoden eines Skills	48
6.9	Interner Steuerungsmethodenaufruf	48
7.1	Objektzustände	55
7.2	Objektübersicht	56
7.3	Umsetzungsstruktur eines Objektes	56
7.4	Definitionen für Objekt-Basis-FB (Aktor)	57
7.5	Steuerungsmethoden eines Objektes	57
7.6	Umsetzungsstruktur von Roboter-Objekt	60
7.7	UR5-Verbindungsports	60
7.8	TwinCAT-Funktionsbausteine für TCP/IP	62
7.9	Aufrufen eines Funktionsbausteines	63
7.10	Struktur: sRoboterProzessVariablen	63
7.11	Ergänzung von Array	64
7.12	Vergleich von IST- und SOLL-Positionen	65
7.13	Umsetzungsstruktur von Kraftsensor-Objekt	65
7.14	Byte-Switch von Low- und High-Byte	68
7.15	Code für Byte-Switch	68
7.16	Struktur: sKraftvariablen	68
7.17	Umsetzungsstruktur von Greifer-Objekt	69
7.18	Anschluss der Beckhoff-Karte	70
7.19	KS2000-Konfigurationstool	71
7.20	Kommunikationsbaustein für Modbus RTU	71
7.21	Anwendung des Kommunikationsbausteines	72
7.22	Schnittstelle zwischen Anlagenmodell und realer Anlage	73
8.1	Grundstruktur des Prozessmodells	75
8.2	Ansatzvergleich	76
8.3	Struktur des Arbeitsplans	77
8.4	Struktur einer Sequenz	78
8.5	Prozessmodell-Struktur 1	79
8.6	Prozessmodell-Struktur 2	79
8.7	Aufbau einer Sequenz	80
8.8	Sequenz-Funktionsbaustein	81
8.9	Beispiel für Sequenzablauf	81
8.10	Zuweisungsaktion der Sequenz	82
8.11	Aufbau eines Arbeitsplans	83

Tabellenverzeichnis

1.1	Projektorganisation	2
2.1	Mögliche Komponenten für Robotersystem	11
2.2	Gegenüberstellung der Roboter	11
2.3	Projektanforderungen	23
5.1	Minimale Systemzustände	37
5.2	Minimale Skillzustände	37
5.3	Minimale Objektzustände	38
6.1	Arbeitsschritte	40
6.2	Definierte Skills für Anwendung	40
6.3	Steuerungselemente eines Skills	42
6.4	Definition von eSkillState	42
6.5	Betriebselemente eines Skills	43
6.6	Benutzerdefinierte Datentypen	43
6.7	Skill 1: Eigenschaften	50
6.8	Benutzerdefinierte Datentypen für Roboter	50
6.9	Skill 2: Eigenschaften	51
6.10	Skill 3: Eigenschaften	51
7.1	Steuerungselemente eines Objektes	54
7.2	Benutzerdefinierte Objekt-Datentypen	54
7.3	Betriebselemente eines Objektes	55
7.4	UR-Datenpaket	61
7.5	Startbefehl für Kraftsensor	66
7.6	Datenpaket von Sensor	66
7.7	Umrechnungsformeln	66
7.8	Befehl für Umrechnungsfaktoren	66
7.9	Datenpaket von Sensor für Umrechnungsfaktoren	67
7.10	Beispiel für das Schliessen des Greifers	70
7.11	Eingabeparameter für Funktionsbaustein	71
8.1	Steuerungselemente einer Sequenz	78
9.1	Vergleich mit Anforderungen	86

Glossar

- CFC Continuous Function Chart:** Grafische Programmiersprache nach IEC 61131-3 in TwinCAT, die eine flexible, frei platzierbare Darstellung von Steuerungslogik ermöglicht. Ideal für komplexe Signalflüsse.
- EGM Externally Guided Motion:** Ein Steuerungsmodus von ABB-Robotern, der eine externe Echtzeitführung der Roboterbewegungen ermöglicht. Dabei werden Positions- und Geschwindigkeitsbefehle direkt von einem externen System an den Robotercontroller übermittelt, was hochpräzise und dynamische Bewegungen erlaubt.
- HMI Human Machine Interface:** Eine Benutzerschnittstelle, die die Interaktion zwischen Mensch und Maschine ermöglicht. HMIs visualisieren Daten, steuern Prozesse und erleichtern die Überwachung und Bedienung automatisierter Systeme.
- MTP Module Type Package:** Ein Standard zur modularen Prozessautomatisierung, der eine herstellerunabhängige Integration von Automatisierungskomponenten ermöglicht. MTP definiert eine standardisierte Schnittstelle für die Kommunikation und Interaktion zwischen Modulen und einem Prozessleitsystem, was Flexibilität und Skalierbarkeit fördert.
- ROS Robot Operating System:** Ein flexibles Framework zur Entwicklung von Robotersoftware. Es bietet Tools und Bibliotheken für Funktionen wie Sensorintegration, Steuerung und Bewegungsplanung. ROS ermöglicht modulare, plattformübergreifende Robotikanwendungen und wird in Forschung und Industrie weltweit eingesetzt.
- SFC Sequential Function Chart:** Eine grafische Programmiersprache nach IEC 61131-3, die in TwinCAT zur Modellierung und Steuerung sequenzieller Abläufe genutzt wird. Sie besteht aus Schritten, Übergängen und Aktionen.
- SPS Speicherprogrammierbare Steuerung:** Ein elektronisches Steuerungssystem zur Automatisierung von Maschinen und Prozessen. Es basiert auf einem Mikroprozessor und wird mit Programmiersprachen nach IEC 61131-3, wie in TwinCAT, programmiert.
- ST Structured Text:** Eine textbasierte Programmiersprache nach IEC 61131-3 in TwinCAT, die eine strukturierte Syntax für komplexe Steuerungslogik bietet. Ideal für mathematische und datenbasierte Operationen.
- TCP Tool Center Point:** Der Tool Center Point ist der Referenzpunkt am Roboterwerkzeug, an dem alle Bewegungen und Positionierungen des Roboters ausgerichtet sind.
- VDI Verein Deutscher Ingenieure:** Ein deutscher Ingenieurverein, der technische Normen, Richtlinien und Empfehlungen entwickelt. VDI ist auch bekannt für seine Veröffentlichungen, Weiterbildungen und Fachveranstaltungen in verschiedenen Ingenieurdisziplinen.