Masterarbeit

BETRACHTUNG DES STEUERUNGSENTWURFS VON SPEICHERPROGRAMMIERBAREN STEUEREINRICHTUNGEN – DIE FUNKTION IM ZENTRUM DER PROGRAMMIERUNG

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
BSc. Daniel Weger
Personenkennzeichen: 2010322022

betreut und begutachtet von
FH-Prof. DI Dr.techn. Udo Traussnigg

Graz, im Juli 2022 .....Daniel Weger.....

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Daniel Weger

.....

Unterschrift

DANKSAGUNG

An dieser Stelle möchte ich mich bei all meinen Vortragenden seitens der Fachhochschule CAMPUS 02 bedanken, die mich in den letzten Jahren bestens belehrt und betreut haben. Ein großes Danke möchte ich an Herrn FH-Prof. DI Dr.techn. Udo Traussnigg für seine hervorragende Betreuung und investierte Zeit aussprechen. Insbesondere sein kritischer Blick auf die Auswahl geeigneter Quellen und seine motivierenden Worte haben dazu beigetragen, diese Arbeit voranzubringen.

Ich danke den Unternehmen Selmo Technology GmbH und Selmo Automation GmbH für die beständige Unterstützung und das entgegengebrachte Vertrauen. Ich danke allen Kollegen, insbesondere DI DI (FH) Markus Gruber, für das entgegengebrachte Verständnis, für die Hilfe in der praktischen Umsetzung und der beruflichen Entlastung in den letzten Monaten der Ausarbeitung dieser Arbeit.

Ich danke der gesamten Familie Fasching und besonders meiner Freundin Lena für den emotionalen Beistand und den gezeigten Spürsinn, versteckte Fehler und Ungenauigkeiten in der Ausdrucksweise, aufzudecken.

Ich danke meiner ganzen Familie für die entgegengebrachte Geduld und das Verständnis für meine Arbeit, die neben meiner beruflichen Vollauslastung zahlreiche Wochenenden und Nachstunden in Anspruch genommen hat. Ich danke ganz besonders meinen Eltern Ingrid und Florian, sie haben mich während meines Studiums mit Rat und Tat unterstützt und in schweren Zeiten immer zu mir gehalten.

KURZFASSUNG

Traditionell wird das Erfahrungswissen in der Automatisierungstechnik aus Versuch und Irrtum generiert, so wird auch das Vorgehen im Entwurf von Steuerungssystemen geprägt. Die kosten- und zeitintensive Methodik eine Steuerung zu entwerfen, macht das Bedürfnis nach einer alternativen Methodik des Steuerungsentwurfs groß. Die Konzeption einer neuartigen Entwurfsmethodik, die Modellierung des Zustand-Zonen Modells (ZZM), brachte vielversprechende Behauptungen hervor. Ein ZZM sei eine vollständige und zuverlässige formale Spezifikation einer informell spezifizierten Steuervorgabe. Es gilt diese Aussage zu widerlegen oder Argumente für deren Richtigkeit zu beweisen. Die Arbeit zielt darauf ab, dem ZZM ein Modell einer vielversprechenden und bereits etablierten Modellierungs-Methodik gegenüberzustellen. Eine umfassende Analyse des Steuerungsentwurfsprozesses soll zeigen, in welcher Phase Qualität entsteht und wie diese übertragen und gesichert werden kann. Eine neu gedachte Anordnung der Qualitätsmerkmale von Software zeigt eindrucksvoll, welche Prämissen seit langem fehlinterpretiert wurden und wie sich die Realität abbildet. Die Definition zielgerichteter Fragen sichert eine korrekte Bewertung und die anschließende Gegenüberstellung der Modelle. In der Schlussbetrachtung fließen weitreichende Definitionen und Erkenntnisse aus der gesamten Arbeit mit ein.

ABSTRACT

Traditionally, experiential knowledge in automation engineering is generated from trial and error, and this is also how the approach to control system design is shaped. The costly and time-consuming methodology to design a control system makes the need for an alternative control system design methodology great. The conception of a novel design methodology, state-zone modeling (ZZM), produced promising claims. A ZZM, it is claimed, is a complete and reliable formal specification of an informally specified control specification. The task is to refute this statement or to prove arguments for its correctness. The work aims at contrasting the ZZM with a model of a promising and already established modeling methodology. A comprehensive analysis of the control design process shall show in which phase quality emerges and how it can be transferred and secured. A newly conceived arrangement of the quality characteristics of software impressively shows which premises have been misinterpreted for a long time and how reality is reflected. The definition of purposeful questions secures a correct evaluation and the following comparison of the models. In the conclusion, wide-ranging definitions and findings are incorporated throughout the paper.

INHALTSVERZEICHNIS

1 EINLEITUNG

Industrie 4.0 oder auch als vierte industrielle Revolution bezeichnet, repräsentiert die technologischen Ansätze der intelligenten Fabrik sowie der computerintegrierten Produktion. Kurz gesagt ein Technologieversprechen, das jedem*jeder ambitionierten Techniker*in dazu verpflichtet den technologischen Fortschritt voranzubringen und Wissen weiterzugeben. Traditionell wird Erfahrungswissen aus Versuch und Irrtum generiert, so auch in der Steuerungstechnik. Seit Anbeginn werden immer effizientere Werkzeuge zur Problemlösung vorgestellt, die im Kern dies widerspiegeln, was seit eh und je praktiziert wird. Die informell spezifizierte Steuervorgabe wird von dem*der Entwickler*in erfasst und je nach Qualifikation oder Erfahrung umgesetzt. Inbetriebnehmen setzt damit am altbewährten Prinzip von Versuch und Irrtum an. Um diese alten Angewohnheiten aufzubrechen und durch neue, womöglich bessere Vorgehensweisen zu ersetzen, wird in dieser Arbeit der Steuerungsentwurfsprozess näher betrachtet.

1.1 Definition des Problems

Es gilt ein Programm für speicherprogrammierbare Steuerungen (**SPS**) zu entwerfen, die zur Lösung einer Steuervorgabe eingesetzt werden kann. Dabei soll die Steuereinrichtung dafür sorgen, dass ein Prozessverhalten in der vorgedachten Weise abläuft. Dieses Prozessverhalten wird in einer informellen Spezifikation mit allen Anforderungen an den gesteuerten Prozess definiert. Der Steuerungsentwurfsprozess ist in der Regel ein Prozess, bei dem der*die Entwickler*in direkt aus der informellen Spezifikation einer Steuervorgabe ein ausführbares Programm erstellt. Ohne umfangreiche Verifizierung oder Validierung wird das Programm erst während der Inbetriebnahme getestet, mit meist schwerwiegenden Folgen. Dieser Prozess ist dafür verantwortlich, dass viele Steuerungssysteme mit Ausfällen und Störungen behaftet sind. Um dies in Zukunft zu vermeiden, soll der Steuerungsentwurfsprozess untersucht werden. Von Interesse ist dabei die Frage, wie Softwarequalität entsteht und wie diese erhöht werden kann. Dafür wird der Prozess der Softwareentwicklung analysiert, in drei Phasen eingeteilt und dessen Einfluss auf die Qualitätsmerkmale überprüft. Besonders die formale Beschreibung einer informellen Spezifikation ist von Interesse, weshalb Modellierungs-Methoden untersucht werden. Die etablierten Methoden der Modellierung werden um eine in dieser Arbeit neu vorgestellten Methodik erweitert.

1.2 Aufbau der Arbeit und Untersuchungsziel

Die Grundlage bildet die systemtheoretische Betrachtung verschiedener Steuerungssysteme. Um einen Überblick über den Entwicklungsprozess von Steuerungssystemen zu erlangen, wird dieser näher vorgestellt. In den folgenden Kapiteln werden verschiedene Modellierungsverfahren zur formalen Beschreibung einer informell spezifizierten Steuervorgabe beschrieben. Die Methode des Zustand-Zonen Modells (**ZZM**) wird vorgestellt und soll einer vielversprechenden bereits etablierten Modellierungs-Methodik gegenübergestellt werden. Zur Bewertung wird eine Steuervorgabe anhand einer informellen Spezifikation eines einfachen Beispiels vorgestellt und mit den jeweiligen Methoden je ein Modell modelliert. Ziel ist es, die beiden Methoden anhand der Qualitätsanforderungen zu bewerten und

gegenüberzustellen. Dafür werden Qualitätsanforderungen definiert und anschließend die entworfenen Modelle nach definierten Bewertungskriterien bewertet.

2 STEUERUNG ALS GRUNDFUNKTION DER AUTOMATISIERUNG

Mit diesem einführenden Kapitel ist beabsichtigt, die Begrifflichkeiten der Steuerung als Grundfunktion der Automatisierung zu erörtern und in eine gewisse Ordnung zu bringen. Dabei wird auf die Ziele der Automatisierung sowie auf Teilgebiete der Systemtheorie näher eingegangen, da diese eine wesentliche Rolle im Entwurf von Steuerungssystemen spielen. Um Verwechslungen auszuschließen, sowie Gemeinsamkeiten und Unterschiede zu verwandten Gebieten der Automatisierung zu erkennen, werden Begrifflichkeiten und Definitionen näher erklärt, visuell hervorgehoben und mit einem Index versehen, siehe **Definition 2-1**.[1]

2.1 Grundprinzip der Automatisierung

Die Automatisierung befasst sich mit der Steuerung und Überwachung von technischen Systemen. Maschinen, Geräte oder Anlagen werden durch Automatisierungseinrichtungen gesteuert und überwacht, damit diese ihre Funktion weitgehend selbständig erfüllen.

Definition 2-1: Automatisierung ist ein Vorgang zur Erfassung und Beeinflussung dynamischer Prozesse, um eine vorgegebene Aufgabe oder Funktion autark zu erfüllen.[2]

Die Entwicklung der Automatisierung ist historisch durch vier große Revolutionen unterteilt und erstreckt sich über drei Jahrzehnte. Jede für sich ist durch verschiedene Gesichtspunkte und Merkmale gekennzeichnet, wobei die folgenden Aspekte seit jeher als Antrieb für die Weiterentwicklung herangezogen werden können:

- Die Beherrschung schneller, komplizierter oder unnahbarer Prozesse.
- Steigerung der Qualität und Quantität einer technischen Einrichtung, um dadurch bessere wirtschaftliche Ergebnisse zu erzielen.
- Gewährleistung der Zuverlässigkeit, Sicherheit und Lebensdauer einer technischen Einrichtung.
- Verbesserung der Lebens- und Arbeitsbedingungen.

Diese Aspekte zur Weiterentwicklung der Automatisierung zeigen, wie vielfältig die zu erfüllenden Ziele sein können. Im Wesentlichen soll der Mensch durch eine Automatisierungseinrichtung ersetzt werden. [3]

[1] Vgl. Lunze (2020), S. 3 f.

[2] Vgl. Litz (2013), S. 181

[3] Zander (2015), S. 1

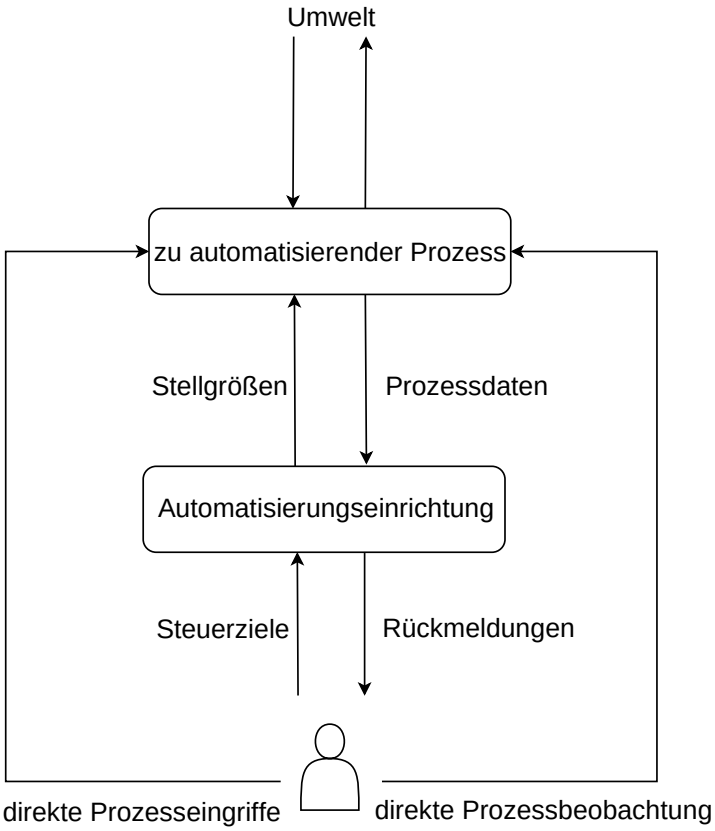


Abbildung 1: Grundstruktur automatisierter Systeme, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2020), S. 20

Die Lösung einer Automatisierungsaufgabe erfolgt stehts nach derselben Grundstruktur. Diese setzt sich aus zwei Komponenten zusammen, dem zu automatisierenden Prozess und der Automatisierungseinrichtung siehe Abbildung 1. Unter dem zu automatisierenden Prozess versteht man, dass eine Aufgabe oder ein Ablauf nicht mehr manuell, sondern automatisiert abläuft, sprich durch Software unterstützt wird. Handelt es sich um eine Steueraufgabe, so kann man den zu automatisierenden Prozess auch als Steuerstrecke bezeichnen. Die Steuerstrecke ist der Teil des Systems, in dem Prozessgrößen beeinflusst werden sollen. Die Automatisierungseinrichtung ist ein System, bestehend aus einem lesenden, einem verarbeitenden sowie einem beeinflussenden Teil. Es können Messgrößen und Steuerziele erfasst und daraus Steuerbefehle erzeugt werden. Aus diesem Grund kann eine Automatisierungseinrichtung auch als Steuerungssystem bezeichnet werden. Betrachtet man den zu automatisierenden Prozess, so sind für Automatisierungseinrichtungen nur Informationen von Bedeutung, die diesen Prozess beschreiben. Prozessdaten wie z.B. wichtige Messdaten werden an die Automatisierungseinrichtung übertragen, verarbeitet und können in anderer Richtung über Stellgrößen auf den Prozess einwirken. Mit Automatisierungseinrichtungen ist es möglich, eine neue Komponente zur Überwachung und Steuerung dieser Prozesse zu ergänzen.[4]

[4] Vgl. Lunze (2020), S. 19 f.

2.2 Ziele der Automatisierung

Ziel der Automatisierung ist es, das Verhalten von Prozessen in technischen Einrichtungen (Maschinen, Anlagen, Geräten usw.) gezielt zu beeinflussen, um diese in bestimmter und vorgedachter Weise ablaufen zu lassen. Um dieses Ziel zu erreichen, muss die Automatisierung spezielle Funktionen ausführen.

Folgende vier Automatisierungsfunktionen sind von Bedeutung:

- **Steuern:** Prozesse werden zielgerichtet beeinflusst, um einen bestimmten Ablauf einer technischen Einrichtung trotz des Einwirkens von Störungen zu realisieren. Um Störungen auszugleichen, müssen diese auch gemessen werden. Werden Störungen nicht oder fehlerhaft gemessen, kann der gewünschte Ablauf von Prozessen einer technischen Einrichtung nicht garantiert werden.
- **Regeln:** Ein Vorgang, bei dem fortlaufend eine Regelgröße erfasst wird, mit einer Führungsgröße verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird.
- **Überwachen/Melden:** Fortlaufend werden physikalische Größen oder Prozesszustände auf Abweichungen von Sollgrößen oder Sollzuständen überwacht. Wichtige Prozessgrößen werden gemessen, ausgewertet und in entsprechender Form an den*die Bediener*in übermittelt.
- **Anzeigen/Bedienen:** Informationen werden verarbeitet und über eine Schnittstelle an den*die Bediener*in übermittelt. Fehler oder unerwünschte Prozesszustände müssen erkannt und entsprechend angezeigt werden, um einen korrigierenden Eingriff zu ermöglichen z.B. durch eine Justierung von Sollwerten.[5]

Definition 2-2: Automatisieren ist ein Vorgang des Steuerns und Überwachen.

Die Methoden der Automatisierungstechnik basieren auf Rückkoppelungsstrukturen, die in vielfältiger Form auftreten können. Man bezeichnet die positive Rückkopplung als Mitkopplung, wenn eine Größe verstärkend auf sich selbst wirkt. Die negative Rückkopplung hingegen wird als Gegenkopplung bezeichnet, wenn die Ausgangsgrößen entgegen den Eingangsgrößen rückgeführt werden. Prinzipiell handelt es sich bei diesen Strukturen um jene, die Informationen in einem Kreislauf zwischen dem zu automatisierenden Prozess und der Automatisierungseinrichtung übermitteln. Die Informationen die zwischen der Automatisierungseinrichtung und dem*der Benutzer*in übermittelt werden, werden als Rückmeldungen bezeichnet.

[5] Vgl. Litz (2013), S. 7

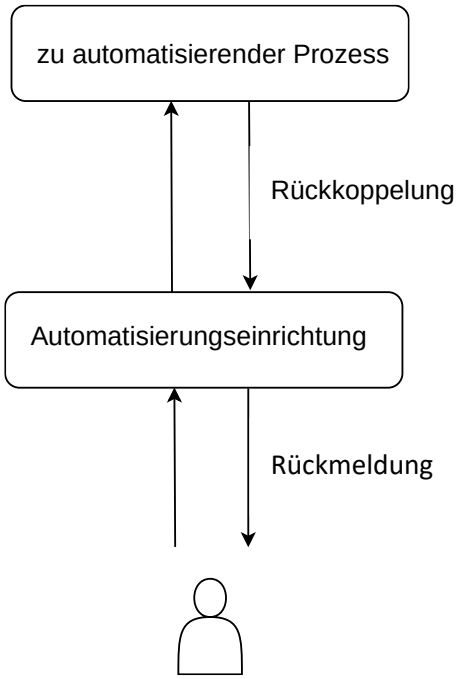


Abbildung 2: Rückmeldungs- Rückkoppelungsprinzip, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2020), S. 21

Abbildung 2 zeigt wie Informationen an den*die Benutzer*in automatisierten Systemen übermittelt werden. Im oberen Kreislauf werden Informationen im Prozess über Sensoren erfasst und über die Automatisierungseinrichtung an den zu automatisierenden Prozess rückgekoppelt. Mithilfe dieser Informationen ist es möglich, das Prozessverhalten automatisiert zu beeinflussen. Im zweiten Kreislauf werden Informationen über das Prozessverhalten an den Menschen rückgemeldet. Der*Die Benutzer*in kann aus diesem Prozessverhalten ableiten, ob und welche Steuerentscheidungen einzuleiten sind. Der Zeitpunkt, in dem der*die Benutzer*in auf das Prozessverhalten einwirkt, ist unregelmäßig und von der Zeitdauer um einiges langsamer als die erste beschriebene Rückkoppelung. Diese Rückmeldungen an den*die Benutzer*in sind in den meisten Fällen unverzichtbar, da in der Praxis kaum eine Automatisierungseinrichtung die Steuerung einer Anlage autonom betreiben kann.

Definition 2-3: Eine Automatisierungseinrichtung ist nur in seltenen Fällen in der Lage, eine Steuerung einer Anlage autonom zu betreiben. Um den zu automatisierenden Prozess zu realisieren, sind häufig leitende Eingriffe in das Prozessverhalten durch den Menschen erforderlich.[6]

Viele Methoden der Automatisierungstechnik basieren auf dem Verständnis von Rückkoppelungsstrukturen. Die Grundlage derartiger Strukturen werden oftmals systemtheoretisch betrachtet. Diese Theorie wird in den Grundzügen im folgenden Kapitel 2.3 erklärt.

[6] Vgl. Lunze (2020), S. 19

2.3 Grundbegriffe der Systemtheorie

Die Systemtheorie beschäftigt sich mit den Beziehungen, die das System zwischen den anliegenden Signalen, meist in Abhängigkeit der Zeit, herstellt. Dies ist ein Werkzeug zur Untersuchung und Beschreibung von Systemen, das einen formalen Überblick über die Zusammenhänge herstellt.[7]

Definition 2-4 Ein System ist eine Menge geordneter Elemente, dessen Eigenschaften mit Relationen zueinander verknüpft sind. In einem bestimmten Zusammenhang werden Elemente eines Systems als Ganzes betrachtet und können sich durch definierte Systemgrenzen gegenüber der Umwelt abgrenzen.

Die Automatisierungstechnik befasst sich mit dynamischen Abläufen in technischen Systemen. Für die Beschreibung dieser Abläufe, werden die Begrifflichkeiten Signal, Prozess und System sowie deren Beziehung zueinander betrachtet. Unter einem dynamischen System versteht man eine Einrichtung, welche unter dem Einfluss von einem oder mehreren Eingangssignalen steht. Das Verhalten des dynamischen Systems wird von einem oder mehreren Ausgangssignalen ausgedrückt. Um dieses Konzept zu verstehen, muss zunächst erklärt werden, was Signale sind und wo diese auftreten können.[8]

Definition 2-5: Ein Signal s beschreibt sich zeitlich ändernde physikalische Größen, durch die Informationen zwischen Teilsystemen mit jeweils einem Teilsystem als Sender und Empfänger ausgetauscht werden.

Ein Signal s wird als eine Funktion der Zeit in vektorieller Schreibweise dargestellt, siehe Formel (2-1). Zum Beispiel kann die Temperatur eines Materials oder der Füllstand eines Behälters dadurch beschrieben werden.

$$s: T \rightarrow S \qquad (2\text{-}1)$$

s	Signal
T	Menge der Zeitpunkte
S	Menge der Signalwerte

Die sich zeitlich ändernden Größen werden durch Signale beschrieben. Dabei spielt die Zeit in diesem Vorgang eine ordnende Rolle.

[7] Vgl. Vgl. Döring (2011), S. 6

[8] Vgl. Lunze (2020), S. 45 f.

Definition 2-6: Die Zeit ist eine ordnende Größe in einem Vorgang, bei dem ein Wert eines Signals zeitlich verändert werden kann. Dabei wirkt die Zeit wie ein Werkzeug zur Einordnung von vergangenen, aktuellen und zukünftigen Werten eines Signals. Die Summe dieser Zeitpunkte nennt man Zeitachse.[9]

Die in einer zu automatisierenden Anlage ablaufenden Prozesse sind durch Veränderung wie beispielsweise die Speicherung oder den Transport von Energie, Stoffen oder Informationen gekennzeichnet. Der Prozessbegriff ist jedoch nicht an bestimmte physikalische Vorgänge gebunden. So ist das Befüllen eines Behälters oder das Erwärmen eines Werkstücks jeweils ein Prozess mit unterschiedlichem physikalischem Hintergrund. Für jeden dieser Vorgänge können ein oder mehrere Parameter definiert werden, die sich mit der Zeit ändern. Diese Parameter werden in Signalen ausgedrückt und zur Darstellung des Prozesses verwendet.[10]

Definition 2-7: Ein Prozess bezeichnet die Gesamtheit der zeitlich parallelen oder nacheinander ablaufenden Vorgänge in einem Automatisierungssystem.[11]

Da es in Automatisierungsaufgaben in der Regel keine Unterscheidung zwischen der technischen Einrichtung und dem Prozessablauf gibt, werden beide Begriffe als Synonyme verwendet. Es gilt in dieser Arbeit diese Begrifflichkeiten klar zu definieren. Das Verhalten von Systemen wird durch ein oder mehrere Signale beschrieben. Ein System bringt diese Signale zueinander in Beziehung. Somit beschreibt ein System die abstrakte technische Vorrichtung, in der Vorgänge ablaufen können. Im Gegensatz dazu, beschreibt ein Prozess den ablaufenden physikalischen Vorgang in einem System an sich.[12]

2.3.1 Systembegriff

Ein System ist eine Menge geordneter Funktionseinheiten in einer technischen Einrichtung, die durch Wirkzusammenhänge und Ordnungsbeziehungen verknüpft sind. Verknüpfungen zwischen Elementen eines Systems werden Relationen genannt. Die kleinste, nicht weiter teilbare Einheit eines Systems wird als Element bezeichnet. Die Menge aller Elemente und Relationen eines Systems bilden dessen Systemstruktur, welche durch Systemgrenzen von der Umwelt abgegrenzt ist. Die Systemtheorie beschreibt die Wechselwirkung zwischen der Struktur und Ordnung der Elemente eines Systems.[13]

In einem technischen System kann die Struktur über das Verhalten des Systems bestimmen. Wird z.B. eine Schaltung über Widerstände, Spulen und Kondensatoren realisiert, so kann eine unterschiedliche

[9] Vgl. Zander, S. 4 f.

[10] Vgl. Lunze (2020), S. 46

[11] Vgl. Zander (2015), S. 8

[12] Vgl. Lunze (2020), S. 47

[13] Vgl. Feess (2018), S. 1

Anordnung dieser Elemente das Verhalten der Schaltung beeinflussen. Ein System kann nach der Signalart und der Prozessart unterschieden werden.[14]

- **Signalart**
  - Analoge Signale
    Verarbeitung von analogen Eingangssignalen zu analogen Ausgangssignalen
  - Digitale Signale
    Digitale Eingangssignale werden zu digitalen Ausgangssignalen verarbeitet
- **Prozessart**
  - Kontinuierliche Systeme
    Im System laufen kontinuierliche Prozesse ab
  - Ereignisdiskrete Systeme
    Im System werden ereignisdiskrete Prozesse ausgeführt

In der Systemtheorie ist die Betrachtung aller Signale, die auf ein System wirken können von Bedeutung. In Abbildung 3 werden Einflussgrößen beschrieben, die auf das Verhalten des Systems wirken.

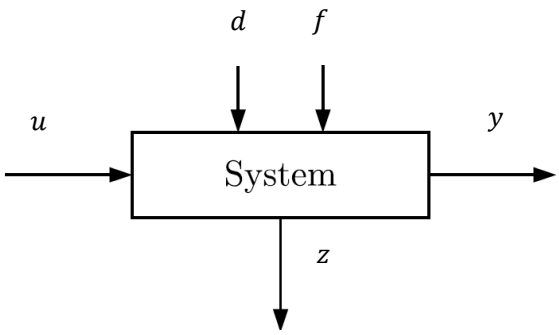


Abbildung 3: Wirkende Signale an einem System, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2020), S. 26

Welche Signaltypen unterschieden werden, wird wie folgt definiert:

- **Eingangsgrößen** $u(t)$ werden auch als Stellgrößen bezeichnet, die durch einen Operator (Mensch, Fremdsystem) festgelegt und verändert werden können, um zielgerichtet das Systemverhalten zu beeinflussen.
- **Störgrößen** $d(t)$ sind Eingangssignale, über die die Umwelt ein System beeinflussen kann. Diese sind möglicherweise messbar, können aber nie durch das System selbst beeinflusst werden.
- **Fehler** $f(t)$ werden als Eingangssignale interpretiert und sind innere Veränderungen, die unzulässige Abweichungen vom gewünschten Prozessverhalten des Systems hervorrufen.
- **Zustandsgrößen** $z(t)$ kennzeichnen den Zustand des Systems. Der Systemzustand ist nur bedingt messbar, obwohl alle Zustandsgrößen auf die messbaren Ausgangsgrößen $y(t)$ wirken.
- **Ausgangsgrößen** $y(t)$ werden durch Signale gebildet, die ihre Umgebung beeinflussen können. Über diese häufig messbaren Signale kann ein ablaufender Prozess beobachtet werden.[15]

[14] Vgl. Zander (2015), S. 13 ff.

[15] Vgl. Lunze (2020), S. 25

2.3.2 Verhalten eines Systems

Die wichtigste Aufgabe eines Systems besteht darin, eine Beziehung zwischen den unterschiedlichen zeitlichen Veränderungen von Signalen herzustellen. Die Reaktion auf eine Änderung der Eingangsgrößen u als Zeitfunktion erzeugt im System die Zeitfunktion der Ausgangsgrößen y. Diese Zeitfunktionen stellen nicht nur die Einzelwerte der Ein- bzw. Ausgangsgrößen dar, sondern Zeitfunktionen mit derselben Zeitachse.

$$u: T \rightarrow U \qquad (2\text{-}2)$$

u	Zeitfunktion der Eingangsgrößen
T	Zeitachse
U	Eingangsfolge

$$y: T \rightarrow Y \qquad (2\text{-}3)$$

y	Zeitfunktion der Ausgangsgrößen
T	Zeitachse
Y	Ausgangsfolge

Ein Ereignisdiskretes Modell ist eine formale Beschreibung eines Systems. Die Werte der Eingangsgrößen u zu den Zeitpunkten $k = 0{,}1, \dots k_e$ wird als Eingangsfolge U dargestellt. Dabei ist es ausreichend eine diskrete Zeitachse mit dem Zähler k_e zu verwenden, auf der nacheinander Ereignisse angeordnet werden können.

$$U = (u(0), u(1), \dots, u(k_e)) \qquad (2\text{-}4)$$

U	Eingabefolge
u	Zeitfunktion der Eingangsgrößen
k_e	Zähler der diskreten Zeitachse

Die Werte der Ausgangsgrößen y wird zu den Zeitpunkten $k = 0{,}1, \dots k_e$ als Ausgangsfolge Y dargestellt.

$$Y = (y(0), y(1), \dots, y(k_e)) \qquad (2\text{-}5)$$

Y	Ausgangsfolge
y	Zeitfunktion der Ausgangsgrößen
k_e	Zähler der diskreten Zeitachse

2.3.3 Zustand eines Systems

Informationen eines Systems, die benötigt werden, um vom gegenwärtigen Zeitpunkt auf den zukünftigen schließen zu können, können unter anderem als Zustand bezeichnet werden. Der Zustand z zum Zeitpunkt k eines ereignisdiskreten Systems wird mit $z(k)$ beschreiben. Für die Ermittlung des zukünftigen Zustandes $z(k+1)$ eines Systems ist die Funktion G erforderlich, sowie die Eingangsfunktion $u(k)$ und der Zustand $z(k)$ gegenwärtigen Zeitpunkt.

$$z(k+1) = G(z(k), u(k)) \quad (2\text{-}6)$$

z	Zustand eines Systems
k	Zeitpunkt
G	Zustandsübergangsfunktion
u	Zeitfunktion der Eingangsgrößen

Der Zustand $z(0)$ wird als Anfangszustand bezeichnet und ist der Zustand, der das System zum Zeitpunkt $k = 0$ beschreibt.

$$z(0) = z_0 \quad (2\text{-}7)$$

z	Zustand eines Systems
z_0	Anfangszustand

Ereignisdiskrete Systeme besitzen eine wichtige Eigenschaft, die Kausalität.

Definition 2-8 Ein System kann als kausal bezeichnet werden, wenn die Wirkung der Ursache zeitlich folgt.

Daraus folgt, dass ein Wert $y(k)$ des Ausgangssignals nicht von zukünftigen Werten des Eingangssignals zu einem späteren Zeitpunkt $k + 1$ usw. beeinflusst werden kann.

$$y(k) = H(z(k), u(k)) \quad (2\text{-}8)$$

y	Zeitfunktion der Ausgangsgrößen
k	Zeitpunkt
H	Ausgangsfunktion
u	Zeitfunktion der Eingangsgrößen

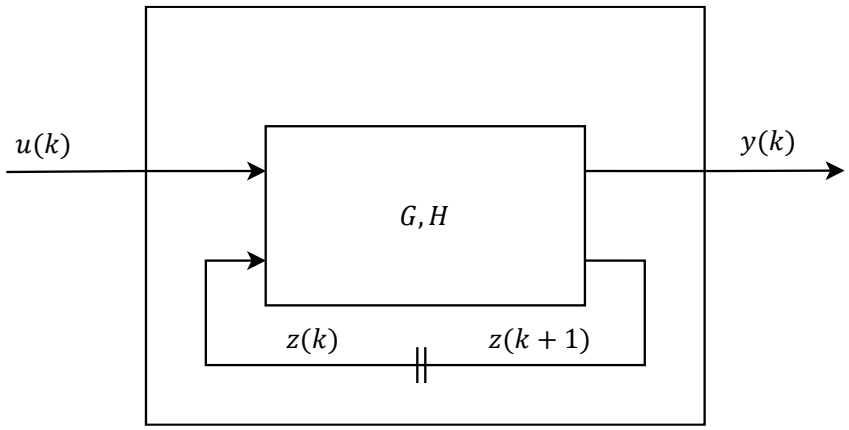


Abbildung 4: Zustandsraumbeschreibung, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2012), S. 43

Die Zustandsraumdarstellung eines ereignisdiskreten System ist in Abbildung 4 dargestellt. Dieses System kann über zwei Funktionen, die Zustandsübergangsfunktion und die Ausgangsfunktion gebildet werden,

siehe Formel (2-6) und (2-8). Da technische Systeme mit zahlreichen Aktoren und Sensoren ausgestattet sind, kann zur Darstellung dieser Ein- bzw. Ausgangsgrößen die vektorielle Darstellung hilfreich sein. Diese Darstellungsform ist zwar aufwendig, stellt aber einen sofortigen Bezug zu den Werten der jeweiligen Größen her. Ein Vektor der Eingangsgrößen $u(k)$ mit drei binären Eingangssignalen kann acht Werte annehmen:

$$u \in U = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right) \quad (2\text{-}9)$$

u Zeitfunktion der Eingangsgrößen

U Eingangsfolge

In dieser Arbeit werden ereignisdiskrete Systeme untersucht, die von mindestens einer Eingangsgröße $u(k)$, Zustandsgröße $z(k)$ und Ausgangsgröße $y(k)$ gebildet werden. Wird ein Wert einer Komponente für den Signalvektor U bedeutungslos, kann dieser durch einen Stern in der vektoriellen Darstellung vermerkt werden.

$$U = \left(\begin{pmatrix} 1 \\ * \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ * \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ * \\ 1 \end{pmatrix} \right) \quad (2\text{-}10)$$

U Eingangsfolge

2.3.4 Prozessbegriff

Ein Prozess ist als die Gesamtheit von ablaufenden Vorgängen in einem System definiert, durch die Materie, Energie oder Information umgeformt, transportiert oder gespeichert wird. Dieser Prozessbegriff ist nur teilweise bei der Prozessbeschreibung von Automatisierungslösungen von Bedeutung, da eine grundsätzliche Unterscheidung nach technologischen Eigenschaften und Vorgängen keine Relevanz aufweist. Allgemein kommt es nicht darauf an, ob Materie, Energie oder Information umgeformt, transportiert oder gespeichert werden. Das Entscheidende ist der ablaufende Vorgang, dass eine Steuerung einen Prozess gezielt durch die Änderung von Prozessgrößen beeinflusst.[16]

Um das Verhalten eines Prozesses gezielt zu beeinflussen, müssen zuallererst Prozessgrößen (z.B. Wege, Winkeln, Drucke usw.) erfasst werden. Prozessgrößen sind physikalische Größen, die Eigenschaften eines Prozesses kennzeichnen. Informationen über Prozessgrößen werden in einem System über physikalische Größen wie Strom, elektrische Spannung oder Wege übertragen. Eine Darstellung dieser Größen kann als Funktion der Zeit erfolgen. Die Informationen lassen sich durch einen veränderlichen Parameter unterscheiden, z.B. die Amplitude der Spannung eines Sensors. Ganz allgemein lässt sich ein Prozess in zwei Arten unterteilen:

- **Kontinuierlicher Prozess:** Ist ein technischer Prozess der Verfahrenstechnik, der ohne Unterbrechung abläuft. Dies kann z.B. ein Rührkessel sein, oder allgemein betrachtet, jeder Prozess ohne Produktwechsel während der Produktionszeit.

[16] Vgl. Zander (2015), S. 7 ff.

- **Diskontinuierlicher Prozess:** Ein diskontinuierlicher Prozess verhält sind nur abschnittsweise kontinuierlich, da sich die Prozessgrößen nur zu diskreten Zeitpunkten ändern. Eine durchgängige Kontinuität kann hierdurch nicht gegeben sein. Meist werden solche Änderung von Prozessgrößen durch Stellsignale im zeitlichen Ablauf eines diskontinuierlichen Prozesses ausgelöst.

Diskontinuierliche Prozesse können in Chargenprozesse oder auch Stückgutprozesse unterteilt werden. In beiden Prozessen können die Vorgänge jeweils parallel oder nacheinander ablaufen. Ein Chargenprozess wird als solcher erkannt, wenn Prozessaktivitäten nacheinander in streng definierter Reihenfolge ablaufen. Die Abarbeitung der Prozessschritte erfolgt stapelweise, weshalb sich die Namensgebung Batchprozess etablierte. Der Stückgutprozess wird als solcher bezeichnet, da die Vorgänge immer als zusammenhängendes Ganzes zu betrachten sind.[17]

In einem Prozessablauf werden Vorgänge eines diskontinuierlichen Prozesses beschrieben. Um das Prozessverhalten gänzlich darzustellen, werden einzelne Vorgänge aneinandergereiht und verkettet. Dazu ist es wichtig die Bedingung für einen Wechsel der Vorgänge zu definieren. Über sogenannte Ereignisse kann dies realisiert werden.

Definition 2-9: Ereignisse sind in einem Steuerungssystem Vorgänge, die durch Erfüllung bestimmter Bedingungen eintreten können. Bedingungen können vielfältig definiert werden, z.B. eine Prozessgröße wird durch einen Vorgang verändert und erreicht einen festgelegten Wert, z. B. eine definierte Zeitdauer oder ein bestimmter Wert wird erreicht.

Wird in einem technischen System eine Bedingung erfüllt, z.B. ein Schwellwert einer Sollgröße wird erreicht, so kann dies einen Zustands- oder Signalwechsel auslösen. Dazu wird die Bezeichnung ereignisdiskreter Prozess als Erweiterung zur Prozessbeschreibung eingeführt. Tritt ein Ereignis in einem in technischen Prozess auf und wird dabei die Prozessgröße verändert, so spricht man von einem ereignisdiskreten Prozess. Dafür muss eine neue Begrifflichkeit in der vorliegenden Arbeit definiert werden, siehe Definition 2-10.

Definition 2-10: Ein ereignisdiskreter Prozess wird durch Vorgänge bestimmt, die ausgelöst durch Ereignisse zeitlich parallel oder nacheinander in einer Steuerstrecke ablaufen können. Das Auftreten eines Ereignisses kann entweder die Ausführung eines Vorgangs stoppen oder dadurch weitere Vorgänge starten.

Erfüllt die Prozessgröße eine laut Definition 2-9 bestimmte Anforderung, beispielsweise durch Erreichen eines Schwellwertes, entsteht ein Ereignis an einem Ort, den man die Steuerstrecke nennt. Eine Rückmeldung dieser Ereignisse an die Steuereinrichtung erfolgt über Messsignale. Hervorgerufen durch Bedienhandlungen entstehen Bediensignale in Bedieneinrichtungen. Informative Aufgaben werden über den Teil einer Steuerungseinrichtung abgehandelt, in der keine Ereignisse stattfinden können.

[17] Vgl. Zander (2015), S. 8 ff.

2.4 Komplexität

In der Steuerungstechnik hängt die Komplexität einer Steuervorgabe vom Verhalten, dem Umfang und der Vorhersehbarkeit des zu beschreibenden Systems ab. Die Komplexität ist somit eine messbare Größe eines Systems, die bestimmt unter welchen Umständen und zu welchem Zeitpunkt die Vorhersehbarkeit verloren geht. Bei Beschreibungsmethoden für diskrete Systeme, wird die praktische Anwendbarkeit besonders durch die Komplexität eines Systems bestimmt. Das Zusammenwirken vieler Elemente eines Systems steigert den Aufwand, ein solches vollständig zu beschreiben. Ein Maß für die Komplexität K in diskreten Systemen ist die Anzahl an möglichen Systemzuständen. Ein Systemzustand ist die Betrachtung der Zustände aller Signale eines Systems zu einem bestimmten Zeitpunkt, siehe Kapitel 2.3.3. In einem System können Signale auf andere Signale wirken und sind dadurch miteinander verbunden. Dies bedeutet, dass mit steigender Anzahl an Signalen die Komplexität eines Systems erheblich ansteigen kann. In industriellen Anwendungen ist ein System mit mehreren hundert Stell- und Bediensignalen keine Ausnahme.[18]

Definition 2-11: Komplexität wird in der Systemtheorie mit der Vielschichtigkeit und Größe von Systemen in Verbindung gebracht. Die Komplexität eines Systems wird durch die Anzahl an Elementen und ihre Verknüpfungen zueinander bestimmt.

Soll ein System entworfen werden, um eine Steuervorgabe korrekt umzusetzen, ist es generell wünschenswert, dass die Zielsetzung der Steuervorgabe korrekt abgebildet wird. Dabei bezieht sich die Korrektheit auf die gestellten Anforderungen an das System. Werden die in den Anforderungen definierten Ziele fehlerfrei umgesetzt, kann man von einem korrekten System sprechen. Um ein System auf dessen Korrektheit zu überprüfen, werden seit langem verschiedene Methoden wie z. B. Model-Checking oder die Simulation eingesetzt, genaueres siehe Kapitel 6.4.2. Diese Methoden werden unter anderem eingesetzt, da es ab einem bestimmten Umfang oder Maß an Komplexität der Steuervorgabe nur schwer möglich ist, die Korrektheit eines Programms vollständig zu überprüfen. Die Vollständigkeit bezieht sich dabei auf den Umfang des Systems, welches durch die maximal möglichen Systemzustände definiert ist. Durch Entscheidungsprobleme kann es sogar vorkommen, dass Systeme nicht vollständig auf deren Korrektheit überprüft werden können.[19]

Eine Methode zur Überprüfung der Korrektheit eines Systems, ist die Simulation aller möglichen Systemzustände. Dabei wird das reaktive Verhalten eines Systems beobachtet, während alle möglichen Systemzustände simuliert werden. Reagiert das System zu jedem Zeitpunkt der Simulation gemäß den Anforderungen, kann von der korrekten Umsetzung der Steuervorgabe gesprochen werden. Um darzustellen, welche erheblichen Auswirkungen der Anstieg der Komplexität auf ein System hat, wird folgendes Beispiel vorgestellt:

[18] Vgl. Litz (2013), S. 351

[19] Vgl. Kleuker (2009), S. 13

- **Ausgangslage**: Zur Lösung einer Steuervorgabe werden fünf unterschiedliche Systeme entworfen. Diese unterscheiden sich lediglich durch die Anzahl ihrer Eingangssignale. Das erste System wird mit zwei Signalen realisiert und jedes weitere System verdoppelt die Anzahl an Eingangssignalen im Vergleich zum vorherigen System. Die Anzahl an Eingangssignalen ist wie folgt definiert:
  - **2** für **System 1**,
  - **4** für **System 2**,
  - **8** für **System 3**,
  - **16** für **System 4** und
  - **32** für **System 5**.
- **Vorgang:** Informationen der Eingangssignale sind binäre Werte mit dem Wertebereich null oder eins. Dadurch kann jedes Eingangssignal zwei Zustände annehmen. Die Anzahl möglicher Zustände der Eingangssignale werden als Z bezeichnet. Aufgrund der erhaltenen Informationen über die Eingangssignale, kann die Anzahl der Systemzustände Z_{sm} berechnet werden. Der Index n steht für die Anzahl der Eingangssignale und der Index m steht für das jeweilige System. Dargestellt ist dieser Zusammenhang in Formel (2-11).

$$Z_{sm} = Z^n \qquad (2\text{-}11)$$

Z_{sm}	Anzahl der möglichen Systemzustände
Z	Anzahl möglicher Zustände je Eingangssignal
n	Anzahl der Eingangssignale
m	System

	Anzahl der Eingangssignale	**Anzahl möglicher Zustände je Eingangssignal**	**Anzahl der möglichen Systemzustände**
System m	n	Z	Z_{sm}
System 1	2	2	2
System 2	4	2	16
System 3	8	2	256
System 4	16	2	65536
System 5	32	2	4294967296

Tabelle 1: Rechenbeispiel Komplexität, Quelle: Eigene Darstellung

- **Ergebnis:** Dargestellt werden die Ergebnisse in Tabelle 1. Die Anzahl an möglichen Systemzuständen steigt mit jedem zusätzlichen Signal exponentiell an. Soll nun jeder mögliche Systemzustand eines Systems überprüft werden, steigt auch der Aufwand für diesen Vorgang

exponentiell an. Dieses Systemverhalten ist als das State-Explosion Problem bekannt und bildet die Schranke für die Beherrschbarkeit eines technischen Systems.[20]

- **Relevanz:** Um sichere Aussagen zur Korrektheit eines Steuersystems zu tätigen, müssen alle möglichen Systemzustände überprüft werden. Wird dieses Steuersystem durch eine speicherprogrammierbare Steuerung (SPS) realisiert, so kann diese Überprüfung mithilfe eines Simulationsverfahrens durchgeführt werden. Eine Simulation ist jedoch nur möglich und sinnvoll, wenn die Berechnungszeit nicht unverhältnismäßig hoch ist. Wird pro Rechenzyklus ein Systemzustand überprüft, benötigt die Simulation für das System 5 aus Tabelle 1 insgesamt **1193,05 h** (mit 1 ms Zykluszeit). Damit ist **keine vollständige Simulation** realistisch umsetzbar. Deshalb gilt es, die Anzahl der Eingangssignale und somit die Anzahl an möglichen Systemzustände gering zu halten.

2.5 Struktur eines Automatisierungssystems

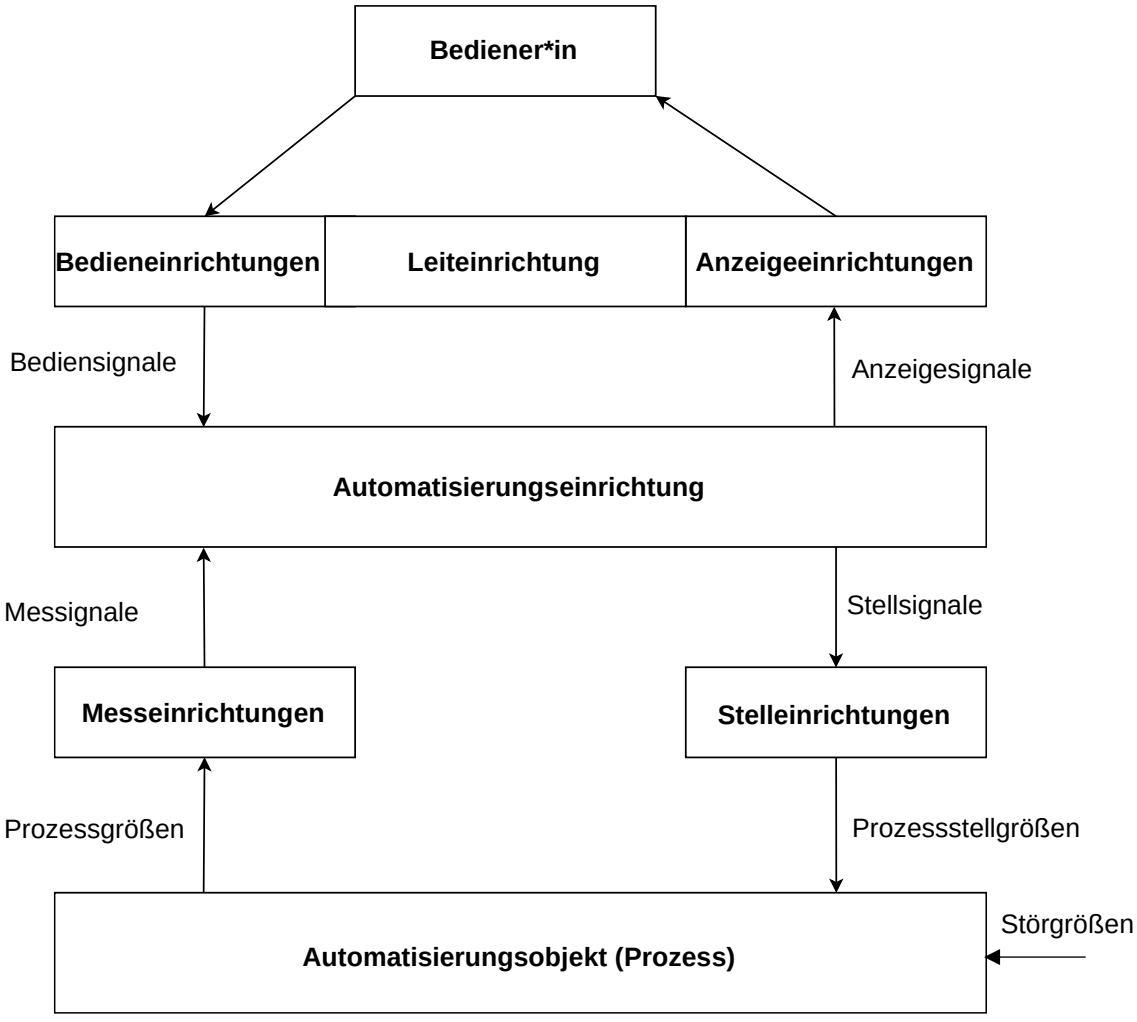


Abbildung 5: Allgemeine Struktur von Automatisierungssystemen, Quelle: Eigene Darstellung, in Anlehnung an Zander (2015), S. 3

[20] Vgl. Zander (2015), S. 317

In Abbildung 5 wird die Allgemeine Struktur eines Automatisierungssystems in Form von zwei geschlossenen Informationskreisläufen dargestellt. Der erste Informationskreislauf wird durch folgende Schritte zur Beeinflussung von Prozessgrößen eines Automatisierungsobjektes definiert:

1. **Informationsgewinnung:** Damit eine Steuerung einen Prozess gezielt beeinflussen kann, müssen dazu die Prozessgrößen erst durch geeignete Messeinrichtungen gemessen werden.
2. **Informationsübertragung und -speicherung:** Über Messignale werden die ermittelten Informationen über den Zustand der Prozessgrößen an die Automatisierungseinrichtung geleitet. Mess- und Bediensignale werden erfasst und zu Stellsignalen verarbeitet.
3. **Informationsnutzung:** Die gebildeten Stellsignale werden an die Stelleinrichtungen übertragen, um dort Prozessstellgrößen zu bilden, um gezielt eine zeitliche Veränderung der Prozessgrößen zu bewirken. Das Ergebnis dieser Prozessbeeinflussung wird über neue Messsignale an die Automatisiereinrichtung rückgemeldet, um erneut Stellsignale über den entsprechenden Steueralgorithmus zu bilden. Dieser Informationskreislauf wird als Rückkoppelung bezeichnet.

Oftmals existiert ein zweiter Informationskreislauf, nämlich jener zwischen Automatisierungseinrichtung und Bediener*in. Der*Die Bediener*in hat über eine Leiteinrichtung die Möglichkeit, den Prozess zu beobachten und gezielt Bedienhandlungen durchzuführen. Änderungen am Prozess selbst werden jedoch nicht während des Betriebs durch den*die Bediener*in durchgeführt.

Ein Automatisierungssystem besteht aus drei Teilsystemen:

- Das Automatisierungsobjekt ist der zu beeinflussende Prozess, der als Steuerstrecke bezeichnet wird und in einer technischen Einrichtung abläuft.
- Die Automatisierungseinrichtung die für die Prozessbeeinflussung des zu implementierenden Steueralgorithmus verantwortlich ist.
- Die Leiteinrichtung als überlagertes System und Schnittstelle zum*zur Bediener*in.

Im Sprachgebrauch ist leider der Begriff Steuerung mehrfach verwendet:

- **Steuerung**: Analoge und/oder binäre Signale beeinflussen einen kontinuierlichen Prozess
- **Steuerung**: Der Oberbegriff für binäre oder digitale Steuereinrichtungen, z.B. SPS

Folglich lässt sich erkennen, dass das technische System selbst bereits als Steuerung bezeichnet wird. In dieser Arbeit wird aus diesem Grunde vorwiegend die Abkürzung für speicherprogrammierbare Steuerungen (SPS) oder verbindungsprogrammierbare Steuerungen (VPS) verwendet.[21]

2.6 Die Steuerung

Der Begriff Steuerung bezieht sich auf die Grundfunktion eines automatisierten Systems zur Manipulation von Prozessen. Das Ziel der Steuerung ist es, technische Einrichtungen mithilfe der Funktionen der Automatisierungstechnik selbständig, in einer bestimmten vorgedachten Weise ablaufen zu lassen. Dabei ergeben sich folgende Einsatzzwecke:

[21] Vgl. Zander (2015), S.15

- Beherrschung komplizierter, schneller oder unzugänglicher Prozesse.
- Steigerung der Qualität und Quantität.
- Zuverlässige und sichere Systeme mit hoher Lebensdauer zu erschaffen.

Definition 2-12: Eine Steuerung ist nach DIN IEC 60050 - 351 ein Vorgang in einem System, bei dem Prozessgrößen aufgrund eigentümlicher Gesetzmäßigkeiten beeinflusst werden. Eine Steuerung wird durch einen offenen Wirkungsweg gekennzeichnet. Es ist möglich das mehrere Eingangsgrößen auf andere Größen, wie z.B. Ausgangsgrößen, wirken.

2.6.1 Steuerungseinrichtung

In diesem Abschnitt ist beschrieben, welche Aufgaben und Eigenschaften eine Steuerungseinrichtung in einem technischen System erfüllt. Verfügt eine Steuerungseinrichtung über Ein- und/oder Ausgangssignale, so spricht man von einem Steuerungssystem. Dieses System ist so aufgebaut, dass es Signale zeitlich verzögern, logisch verknüpfen oder auch speichern kann. Man unterscheidet zwischen kombinatorischen und sequentiellen Steuerungseinrichtungen, je nachdem ob Signale verknüpft werden oder ob zusätzlich binäre Speicherelemente zum Einsatz kommen.

Beschrieben wird das Verhalten einer Steuerungseinrichtung durch die Modelle der Systemtheorie. Das Verhalten von kombinatorischen Steuerungseinrichtungen kann durch Logik-Pläne oder Schaltausdrücke beschrieben werden. Die sequentiellen Steuerungseinrichtungen werden hingegen durch Mittel der Netz- bzw. Automatentheorie beschrieben. Dabei spielen besonders die steuerungstechnisch interpretierte Petrinetze (SIPN) eine führende Rolle. SIPN stellt eine Methode dar, welche es ermöglicht neben sequentiellen-, auch parallele Abläufe darzustellen. Besonders von Bedeutung sind diese Mittel im Bereich des Steuerungsentwurfs.[22]

2.6.2 Steuerkreis

In Abbildung 6 ist die allgemeine Struktur des Steuerkreises dargestellt. Teil dieses Systems sind z.B. Aktuatoren, Sensoren, Rückkoppelungen und Störungen des technischen Prozesses. Diese Struktur soll ermöglichen, eine Steuervorgabe möglichst effizient und genau durch einen Steueralgorithmus zu erfüllen. Das bedeutet, es ist die Aufgabe einer Steuerung, die Verarbeitung einer Eingangsgröße über den Steueralgorithmus so auszuführen, dass anschließend eine entsprechende Handlung durch eine Stellgröße ausgeführt werden kann, damit der Sollwert der Ausgangsgröße erreicht wird.

[22] Vgl. Zander (2015), S. 51 ff.

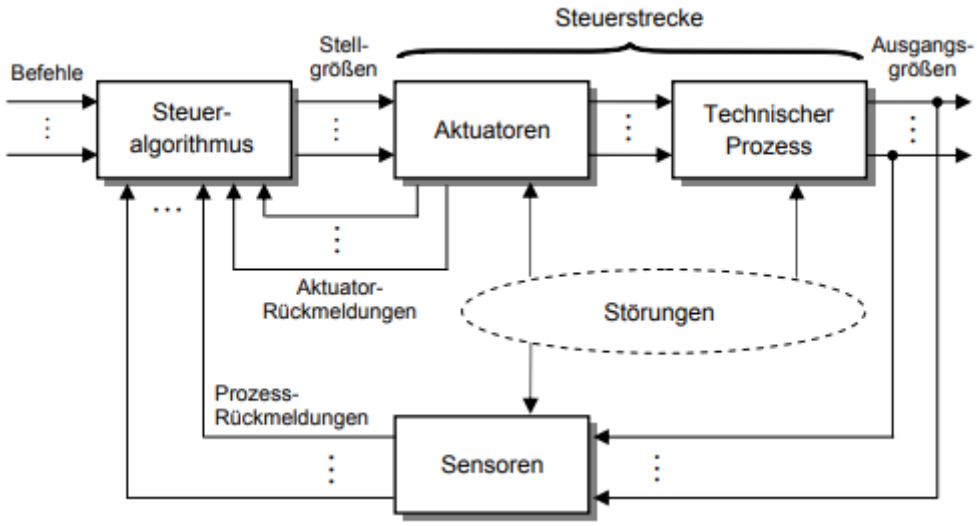


Abbildung 6: Strukturbild des Steuerkreises, Quelle: Litz (2013), S. 180

Der Steueralgorithmus verarbeitet Eingangsinformationen die aus Steuerbefehlen, Prozessrückmeldungen und/oder Aktuator-Rückführungen bestehen können. Dadurch entstehen nicht selten bis zu mehreren hundert Eingangsinformationen, die die Komplexität des Steueralgorithmus in die Höhe treiben, siehe Kapitel 2.4. Die Signale in diesem System können als Stell-, Bedien- oder Störgrößen auftreten. Eine Rückmeldung beschreibt ein Ereignis, dass durch Erreichen einer Endlage, durchbrechen einer Lichtschranke oder Störungsmeldung eines Aktuators etc. auftreten kann. Weiters können auch im Steuerkreis Störungen auftreten, die auf den Prozess sowie auch auf Aktuatoren oder Sensoren wirken können.[23]

Definition 2-13: Es existiert kein **Soll-Ist-Vergleich** im Steuerkreis. Das bedeutet, dass bei Abweichungen oder Fehler in der Steuerstrecke zusätzliche Rückmeldungen nötig werden, um diese zu beherrschen. Daraus resultieren Aktuator-Rückmeldungen, deren Aufgabe es ist, die richtige Wirkung der Stellgrößen zu überwachen.[24]

Es handelt sich bei den Befehlen eines Steuerkreises oft um Start- oder Stoppbefehle. Dadurch können Vorgänge eingeleitet werden, oder ein bestimmter Ablauf gesteuert werden. Die Steuerstrecke wird durch die Kombination des technischen Prozesses, der Aktuatorik und Sensorik gebildet. Diese Darstellung der Steuerstrecke wird als vereinfachtes Strukturbild der Steuerung betrachtet. Weiters wird angenommen, dass es sich im System bei Stellgrößen, Befehlen sowie Prozess- und Aktuator-Rückmeldungen um binäre Größen handelt.

[23] Vgl. Litz (2013), S. 179 ff.

[24] Vgl. Litz (2013), S. 9

3 SPEICHERPROGRAMMIERBARE STEUERUNG

In diesem Abschnitt der Arbeit werden Funktionen, Besonderheiten und Eigenschaften speicherprogrammierbarer Steuerungen (SPS) vorgestellt, die wichtig für den Steuerungsentwurf sind. Der allgemeine innere Aufbau wird betrachtet, um die Abläufe sowie die Arbeitsweise einer SPS zu verstehen. Weit entfernt vom ursprünglichen verwendeten Standard des Steuerungsentwurfs der ersten SPS, unterliegen heutige Programmiersprachen einem international verbindenden Standard EN 61131-3:2014-06.[25]

Dieser Standard umfasst fünf spezifische Sprachen, die entweder textuell oder grafisch beschrieben werden. Jede Sprache stellt dabei verschiedene Werkzeuge wie Funktionen oder Funktionsblöcke zur Verfügung, die vom jeweiligen SPS-Hersteller als Software-Bibliotheken zur Verfügung gestellt werden. Angewandt werden diese Programmiersprachen, um syntaktisch und semantisch Modelle einer Steuervorgabe nach definierten Regeln zu beschreiben.[26]

Definition 3-1: Eine SPS stellt ein digital arbeitendes System mit internem, programmierbarem Speicher der anwenderorientierten Steueranweisung dar. Dieses Steuerungssystem ist für die Implementierung spezifischer Funktionen wie z.B. Ablaufsteuerung, Verknüpfungssteuerung, Zeitfunktionen, Zählfunktionen und arithmetische Funktionen eingesetzt. Ausgelegt ist dieses Steuersystem in Hard- und Software nach standardisierten Vorgaben, die eine einfache und ausfallsichere Implementierung für Anwendungen in einer industriellen Umgebung gewährleisten.[27]

Als erstes ist die Abgrenzung zu verbindungsprogrammierbarer Steuerungen (VPS) zu definieren, dessen Steuerungsprogramm durch die Verdrahtung entsteht. Diese Art von Steuerung wurde früher über Schütze und Relais programmiert, dabei wurde die Logik durch eine je nach Anwendung variierende Anordnung erzeugt. Diese namensgebende Art der Steuerung hat jedoch den großen Nachteil, dass jede Steuerung immer wieder neu aufzubauen ist, selbst bei ähnlichen Anwendungen. Der Aufwand zur Realisierung einer Steuerung ist im Gegensatz zur SPS immens und schließt die Betrachtung dieser Art der Steuerung in dieser Arbeit aus. Die Entwicklung der letzten Jahrzehnte hat gezeigt, dass SPS, für die Lösung immer komplexer werdender Steuervorgaben, eingesetzt werden. Um diese Herausforderung zu meistern, ist es notwendig der Qualitätsanforderung an die Programmierung einer Steuereinrichtung mehr Bedeutung zu schenken. Dazu folgt die Betrachtung des allgemeinen Aufbaus und der Struktur in 3.1, sowie ein Überblick über die Grundfunktionen einer SPS in Kapitel 3.2. [28]

[25] Vgl. IEC61131-3 (2014)

[26] Vgl. Marwedel (2021), S. 34 ff.

[27] Vgl. Linke (2021), S. 1497 ff.

[28] Vgl. Zander (2015), S. 51 ff.

3.1 Aufbau und Struktur einer SPS

Diskrete Steuerungen werden durch Programme beschrieben, deren Ausgangsgrößen sowie die dabei erhaltenen Stellgrößen an die Steuerstrecke übertragen werden. Diese Art der Steuerung kann aus komplizierten und vielfältigen Steueralgorithmen bestehen und wird mit speicherprogrammierbarer Steuerung (SPS) umgesetzt. Es besteht die Möglichkeit, den Steueralgorithmus durch Programmänderungen abzuändern, ohne dabei die Hardware anzupassen.

Abbildung 7 zeigt den schematischen Aufbau einer SPS. Das Prozessabbild des Eingangs ist ein Speicherbereich der SPS, in den die Ausgangsgrößen der Steuerstrecke geschrieben werden. Die Abarbeitung des Programms erfolgt anhand des Zugriffes auf die Werte des Prozessabbilds des Eingangs. Dieses Programm realisiert den Steueralgorithmus. Das Ergebnis wird in das Prozessabbild des Ausgangs geschrieben, einen weiteren Speicherbereich der SPS. Stellgrößen werden aus diesem Speicherbereich ausgelesen und an den zu steuernden Prozess übermittelt. Die SPS arbeitet zyklisch in folgendem Schrittmuster:

- Einlesen der Ausgangsgrößen
- Abarbeiten des Programms
- Ausgang der Steuereingriffe

Verändern sich Ausgangswerte am Prozess, werden diese vom Steueralgorithmus nur am Anfang des Zyklus wahrgenommen. Die Steuerung reagiert entsprechend auf diese Änderung nach der Abarbeitung des Programms und greift in die Steuerstrecke ein. Dadurch verhält sich eine SPS anders als häufig vermutet und wartet also nicht darauf, dass das nächste Ereignis bei den Ausgangsgrößen der Steuerstrecke sichtbar wird. Die Zykluszeit einer SPS liegt im Bereich von Millisekunden und sollte im Vergleich zum zeitlichen Verhalten der Steuerstrecke klein sein. Dieser Unterschied wirkt sich nicht auf die Qualität der Steuerung aus und folglich kann die Arbeitsweise der SPS als quasi-kontinuierlich angesehen werden.

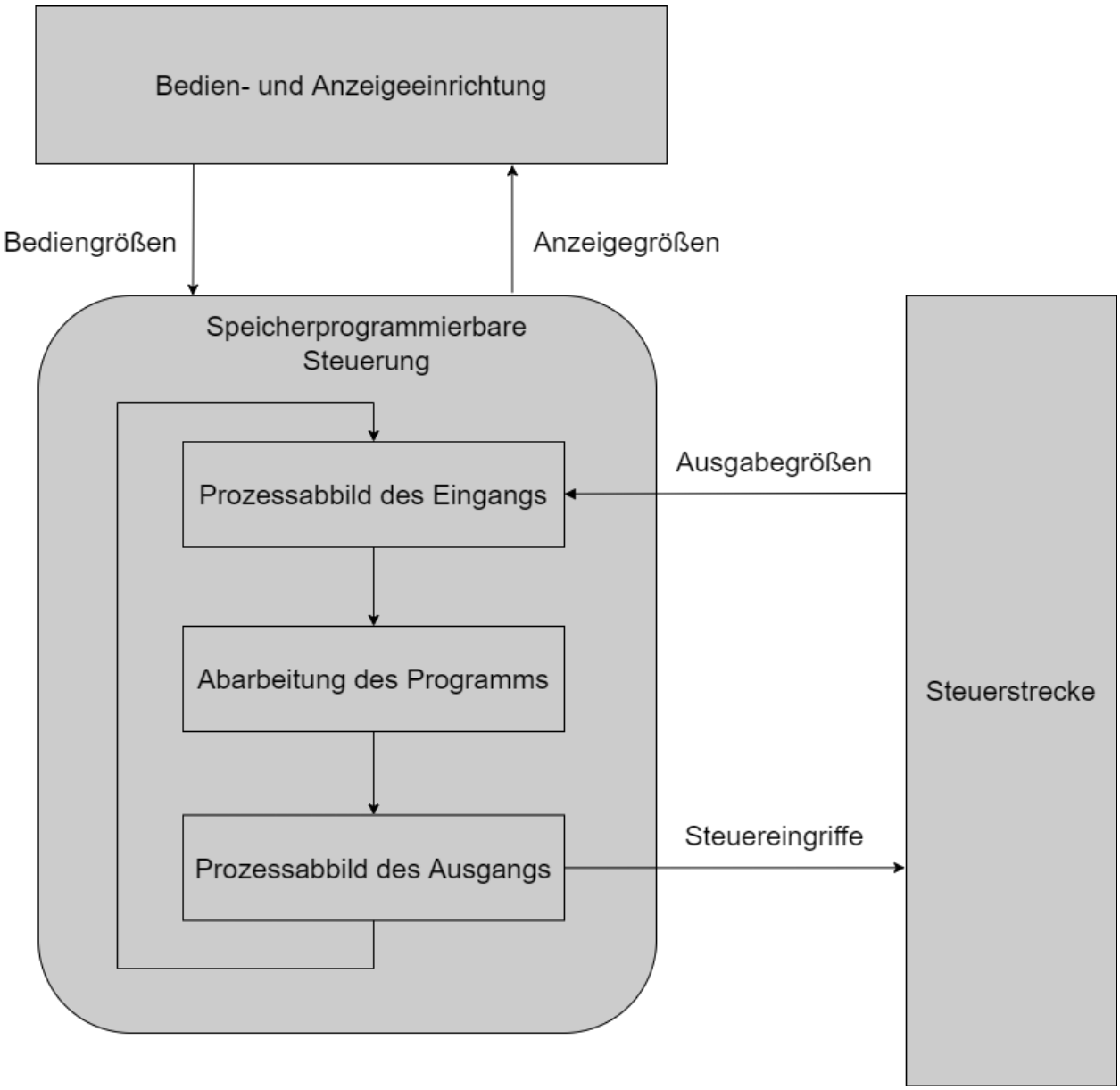


Abbildung 7: Strukturabbild einer speicherprogrammierbarer Steuerung, Quelle: In Anlehnung an Lunze (2020), S. 473

Speicherprogrammierbare Steuerungen verfügen über weitreichende Hilfen für die Programmierung (Entwicklungsumgebungen). Programme können durch grafisches Programmieren erstellt werden, wobei einzelne Modelle bzw. Prozessschritte der Steuerung als Blöcke dargestellt sind. Steueralgorithmen werden am häufigsten durch Anweisungslisten, Funktionspläne oder Kontaktpläne dargestellt. Dabei enthalten diese Pläne jeweils gemessene Signale, die zu den Stellgrößen des Systems verknüpft werden. Erfolgt die Umwandlung der Messgrößen binär, so können die weiteren Verarbeitungsschritte auf Operationen der booleschen Algebra zurückgeführt werden.[29]

[29] Vgl. Lunze (2020), S. 473-474

3.2 Grundfunktionen der SPS

Eine SPS ist eine Rechenmaschine die speziell für die Lösung von Steuervorgaben Ende der 1960er Jahre entwickelt wurde. Bei der Konzeption dieser Steuerung wurde auf den geringen Programmieraufwand Wert gelegt. Die Abbildung 7 ist eine Übersicht der grundlegenden Gemeinsamkeiten aller Varianten der SPS, dabei gilt das EVA-Prinzip:

- **E:** Eingangssignale
  Werden auch Eingänge genannt, werden zu einem fixen Zeitpunkt eingelesen und gespeichert. Es bildet sich über eine definierte Zeit (Zykluszeit) lang das sogenannte Eingangsbild.
- **V**: Verarbeiten des Programmcodes
  Das Eingangsbild bleibt innerhalb der Zykluszeit unverändert, während der Wert des Ausgangs verändert wird.
- **A**: Ausgangssignale
  Die entsprechenden Ausgangswerte werden am Ende der Zykluszeit in das Ausgangsbild übertragen.

Die Abarbeitung der Anweisungen einer SPS erfolgt seriell, das bedeutet, dass Schritt für Schritt eine Befehlskette abgearbeitet wird. Die Reihenfolge wird durch das Steuerungsprogramm bestimmt und bindet Systemparameter mit ein. Das Steuerungsprogramm bestimmt folglich, welche Eingangssignale verwendet werden, welche miteinander verknüpft werden und wo das Ergebnis als Ausgangssignal zur Verfügung steht. Eine weitere Besonderheit der SPS ist der zyklische Programmablauf. Die Arbeitsweise einer SPS kann je nach Auslegung mit variablen oder unveränderlichen Zykluszeiten erfolgen. Dabei gilt es, einen Echtzeitbetrieb herzustellen. Darunter wird die rechtzeitige Reaktion an den Ausgängen, auf ein eintreffendes Signal am Eingang verstanden. [30]

Definition 3-2: Echtzeitfähig wird ein System bezeichnet, dass innerhalb einer definierten Zeitspanne auf Ereignisse im Ablauf eines technischen Prozesses reagiert.[31]

In einer SPS können neben binären Signalen auch digitale Signale verarbeitet werden. Ein binäres Signal ist dadurch gekennzeichnet, dass es nur zwei Werte im Wertebereich annehmen kann. Durch aneinander Reihung mehrerer digitaler binärer Signale, entstehen Bitketten die erheblich größere Werte im Wertebereich annehmen können. Eine Codierung des Wertebereichs verleiht diesem eine definierte Bedeutung, wie z.B. einen Zahlenwert. Um diese Werte verarbeiten zu können werden digitale Funktionen wie Schiebungen, Umwandlungen, Vergleiche oder arithmetische und mathematische Funktionen etc. eingesetzt. Regelmäßig wird der digitalen Steuerung auch eine Verarbeitung von analogen Signalen abverlangt. Ein Sensor misst eine physikalische Größe und stellt diese dann kontinuierlich in Form einer Spannung oder eines Stromes dar. Entsprechende Analog-Digital-Umsetzer sind elektrische Bauteile, die

[30] Vgl. BOCK (2018), S. 9 ff.

[31] Vgl. Linke (2021), S. 1497

diese analogen Signale in digitale Signale umwandeln, um dadurch eine weitere Verarbeitung durch die digitale Steuerung zu ermöglichen.[32]

3.3 Programmierung

Das Steuerprogramm einer SPS wird mithilfe einer speziellen Programmiersprache beschrieben. Es stehen mehrere dieser Sprachen zur Verfügung, die nach ihrer Form der Beschreibung unterschieden werden. Weiters gibt es noch Sonderformen wie Zustandsgraph oder die Hochsprache C oder C++ als Programmiersprache. Um in dieser Industrie einen gewissen Qualitätsstandard zu gewährleisten, ist es nicht zwingend erforderlich eine Standardsprache für die Programmierung zu verwenden, wird jedoch empfohlen. Demnach wird zwischen grafischen sowie textuellen Programmiersprachen unterschieden. Die Norm DIN EN 61131-3:2014-06 definiert dafür die folgenden Programmiersprachen:

- Textbasierend
    - Anweisungsliste (AWL)
    - Strukturierter Text (ST)
    - Ablaufsprache, textbasiert (AS oder Sequential Function Chart (SFC))
- Grafisch
    - Funktionsbaustein-Sprache (FBS)
    - Kontaktplan (KOP)
    - Ablaufsprache, grafisch (AS)

Die Programmierung mittels Anweisungsliste (AWL), Kontaktplan (KOP) und Funktionsbaustein-Sprache (FBS) erfolgt vorrangig für Steuerungseinrichtungen, die als Verknüpfungssteuerung ausgeführt werden. Die Ablaufsprache (AS) ist eine speziell für Ablaufsteuerungen konzipierte Programmiersprache und wird häufig in Kombination mithilfe eines Zustandsgraphs eingesetzt.[33]

3.4 Konzeptionierung von Steuereinrichtungen

Ein wesentlicher Bestandteil für Automatisierungsprojekte ist die Schaffung einer Basis durch die Steuereinrichtung. Bei der Konzeptionierung der Steuereinrichtung muss neben der Forderung der korrekten Umsetzung der Steuervorgabe, auch die Forderung nach einem geringen Kostenaufwand eingehalten werden. Die Entwicklung der letzten Jahre zeigt, dass Steuerungen mit möglichst wenig Steuer- (Bauteile wie z. B. Relais), Logik- bzw. Speicherelementen an Bedeutung verlieren. Wichtiger jedoch ist, dass die Funktionalität, Integrität und Konnektivität der Steuerungseinrichtung an erster Stelle stehen. In den vergangenen Jahren lag der Fokus im Entwurf von Steueralgorithmen für SPS darin, mithilfe der Anwendung von SPS-Fachsprachen diesen effizienter zu gestalten. Diese Hauptforderung hat nach wie vor Bestand, nur dessen Priorität hat sich aktuell verändert. Wichtiger ist die Frage wie sich die Effizienz des Entwurfs von Steuerungen verbessern lässt, da sich die Kosten für die benötigte Hardware stark

[32] Vgl. Linke (2021), S. 1503

[33] Vgl. Linke (2021), S. 1501

reduzieren. Neben den bereits erwähnten Forderungen zur Reduktion der Hardwarekosten, sowie zur effizienten Vorgehensweise beim Entwurf, ist die übersichtliche sowie leicht verständliche Umsetzung der Steueralgorithmen eine zentrale Forderung. Ausgehend davon, werden für den Menschen leicht verständliche und übersichtliche Modelle zur Hilfestellung herangezogen. Der aus diesen Modellen abgeleitete Steueralgorithmus soll eine leichte Verifizierbarkeit sowie Testbarkeit vorweisen. Diese Vorgehensweise soll dazu führen, eine zuverlässige und kostengünstige Lösung der Steuervorgabe umsetzen zu können.

Definition 3-3: Die Zuverlässigkeit, Verifizierbarkeit sowie Testbarkeit eines Steuerungssystems werden durch den systematischen Entwurf einer Steuerung erheblich gesteigert.

4 STEUERUNGSENTWURF

Das folgende Kapitel des Steuerungsentwurfs handelt von der Vorgangsweise die nötig ist, um aus einer informell gegebenen Steuervorgabe eine formale Darstellung der geforderten Steuerung zu erarbeiten. Dieser Formalismus ermöglicht eine Übersetzung in einen Maschinencode, der dazu verwendet wird, eine entsprechende Steuereinrichtung zu realisieren.

Definition 4-1: Unter dem Steuerungsentwurf versteht man die Konzeption einer Steuerung eines technischen Systems mithilfe von definierten Entwurfswerkzeugen.

Der Steuerungsentwurf ist, wie dessen Begrifflichkeit andeutet, ein in höchstem Maße kreativer Vorgang. Dieser Entwurfsprozess bietet unter Berücksichtigung der Grenzwerte und Randbedingungen verschiedene Freiheitsgrade eine Steuerung zu entwerfen. Ziel ist es, den gewünschten Prozessablauf in der Steuerstrecke zu realisieren. In der Vergangenheit wurde der systematische Entwurf einer Steuerung wenig priorisiert und die direkte Umsetzung aus der informellen-, in eine formale Spezifikation bevorzugt. Gegenwärtig wird der Wunsch nach mehr Flexibilität, Funktionalität und Wiederverwendbarkeit der Steuerungsalgorithmen größer und somit das Verlangen nach mehr Kontrolle über die immer komplexer werdenden Prozessabläufe. Dafür wird ein Entwurfswerkzeug benötigt, das ausgehend von einer informellen Steuervorgabe eine formale Darstellung entwickelt.[34]

4.1 Entwurfsprinzipien

Die Aufgabe des Steuerungsentwurfs ist es, mithilfe von Entwurfswerkzeugen den Prozessablauf einer informell vorgegebenen Steuerungsvorgabe formal darzustellen. Aus dieser formalen Darstellung soll eine Steuereinrichtung realisierbar sein, die anhand ihrer ausgehenden Stellsignale und empfangenen Messsignale den gewünschten ereignisdiskreten Prozess in der Steuerstrecke ablaufen lassen kann. Es gilt, die in Kapitel 2.4 beschriebene Komplexität von Systemen und vor allem dessen Anstieg zu vermeiden. Je umfangreicher die zu beschreibende Steuervorgabe wird, desto schwieriger ist es für den Menschen Zusammenhänge und Funktionen ganzheitlich zu begreifen. Folglich werden vier wichtige Entwurfsprinzipien vorgestellt, die beim Entwurf von Steuerungen für umfangreiche Steuervorgaben behilflich sind:

- **Strukturierung**
  - Kriterien vorgeben und anhand derer Beziehungen innerhalb eines Systems festlegen.
  - Zerlegung eines Systems anhand bestimmter Kriterien, um deren innere Beziehungen sichtbar zu machen.
- **Dekomposition**
  - Ein System in dessen Grundbausteine zerlegen.
  - Dekomposition des Systems in kleinere Subsysteme.

[34] Vgl. Janschek (2010), S. 9 ff.

  - Anhand definierter Kriterien Struktur in das System bringen.
  - Verhalten des Systems in einem Modell abbilden.
- **Aggregation**
  - Subsysteme zu einem System aggregieren.
  - Einzelelemente eines Systems zusammenfügen.
- **Hierarchie**
  - Aufbau einer Rangordnung
  - Schichtenmodell: Systemdefinitionen werden hierarchisch angeordnet und können als Subsystem auch einzeln betrachtet werden.[35]

Um eine Steuerung zu entwerfen, muss eine Verbindung zwischen der technischen Funktion des betrachteten Systems und den Methoden des Automatisierungssystems hergestellt werden. Dazu gibt es ein iteratives Vorgehen, aufgeteilt in fünf Schritten, die zur Lösung einer Steuervorgabe wegweisend sein können:

- **Prozessanalyse:** Wichtige Informationen über das Verhalten des Prozesses, der in einem System ablaufen soll, werden durch ein Modell dargestellt.
- **Analyse des Systemverhaltens:** Analyse der wichtigsten Eigenschaften des zu steuernden Systems mithilfe des Modells.
- **Steuerungsentwurf:** Entwicklung eines Steueralgorithmus der die Funktionen des gesteuerten Systems beschreibt.
- **Modellerprobung:** Der Steueralgorithmus wird durch Simulationen untersucht und getestet, ob sich das Gesamtsystem in entscheidenden Situationen entsprechend der Anforderungen verhält.
- **Implementierung und Inbetriebnahme:** Realisierung der Steuerungseinrichtung-Hardware, sowie die Inbetriebnahme über die Vereinigung mit der Software.[36]

4.2 Die Steuervorgabe und die Entwurfsaufgaben

Als Steuervorgabe wird die grafisch oder textuell beschriebene Aufgabenstellung bezeichnet. Sie dient als Vorgabe für den Steueralgorithmus und ist der wichtigste Informationslieferant. Meist ist die textuelle Beschreibung in der Sprache des*der Auftraggebers*Auftraggeberin verfasst. Oft ist der Autor dieser Notizen ein*e Maschinenbauer- oder Anlagenbetreiber*in. In den meisten Fällen wird diese Aufgabenstellung von keinem*keiner Regelungs- oder Steuerungstechniker*in formuliert und beschreibt das Verhalten der Steuervorgabe nicht vollständig. Eine typische Aufgabenstellung zur Lösung einer Steuervorgabe kann wie folgt definiert sein: Gegeben ist ein System mit dem Eingangssignal $u(k)$ und dem Ausgangssignal $y(k)$ sowie die Eigenschaften in Form einer informellen Spezifikation. Gesucht ist die Steuerungseinrichtung, die in Anhängigkeit der Eingangsfolge U entsprechend auf die Ausgangsfolge Y reagiert. Diesem Prinzip nach können entsprechende Entwurfsaufgaben gestellt werden. Das Ziel kann

[35] Vgl. Janschek (2010), S. 59 ff.

[36] Vgl. Lunze (2020), S. 41 f.

unterschiedlich formuliert sein, je nachdem welche Eigenschaften und welchen Charakter die Aufgabenstellung aufweist. Folgende Arten von Systemzuständen können formuliert werden:

- **Vorgegebener Endzustand:** Das zu steuernde System soll in den Endzustand $z(k_n)$ überführt werden.
- **Vorgegebener Ablauf:** Das zu steuernde System soll eine vorgegebene Zustandsfolge $Z(k)$ durchlaufen.
- **Verbotene Zustände:** Das zu steuernde System darf eine Menge vorgegebener Zustände $\tilde{Z} \subset Z$ nicht annehmen.
- **Verbotene Zustandsübergänge:** Das zu steuernde System darf die Zustandsübergänge einer vorgegebenen Menge nicht ausführen.

Bringt man diese in eine bestimmte Form, so wird das Verhalten der Steuerstrecke unter der Wirkung der Steuerung beschrieben und man spricht von einer formalen Spezifikation.[37]

4.3 Spezifikation des Steuerungssystems

Die Spezifikation gilt als Hilfselement und Vorgabe für den Steuerungsentwurf und beschreibt alle funktionalen und nicht funktionalen Anforderungen an das Steuerungssystem sowie die zu bedienenden Schnittstellen.

Definition 4-2: Es existiert keine allgemein gültige Spezifikation zur Beschreibung des funktionellen Verhaltens von Steuerungssystemen.[38]

In der Spezifikation werden Anforderungen an die jeweilige Steuerung, die es in dieser Arbeit zu entwickeln gilt, individuell beschrieben. Am Anfang stehen immer die funktionalen Vorgaben der zu erfüllenden Aufgabe. Diese Vorgaben können sich teilweise ähneln, jedoch sind diese durch das Fehlen einer allgemein gültigen Spezifikation nicht als sich wiederholende Standardvorgaben anzusehen. Vielmehr ist die Beschreibung einer Steuervorgabe ein individueller Vorgang, der je nach Umfang erst nach und nach zu Stande kommt. Jedoch lässt sich der Prozess des Steuerungsentwurfs in einzelne Schritte zerlegen.[39]

[37] Vgl. Lunze (2020), S. 457 f.

[38] Vgl. Litz (2013), S. 189

[39] Vgl. Litz (2013), S. 188

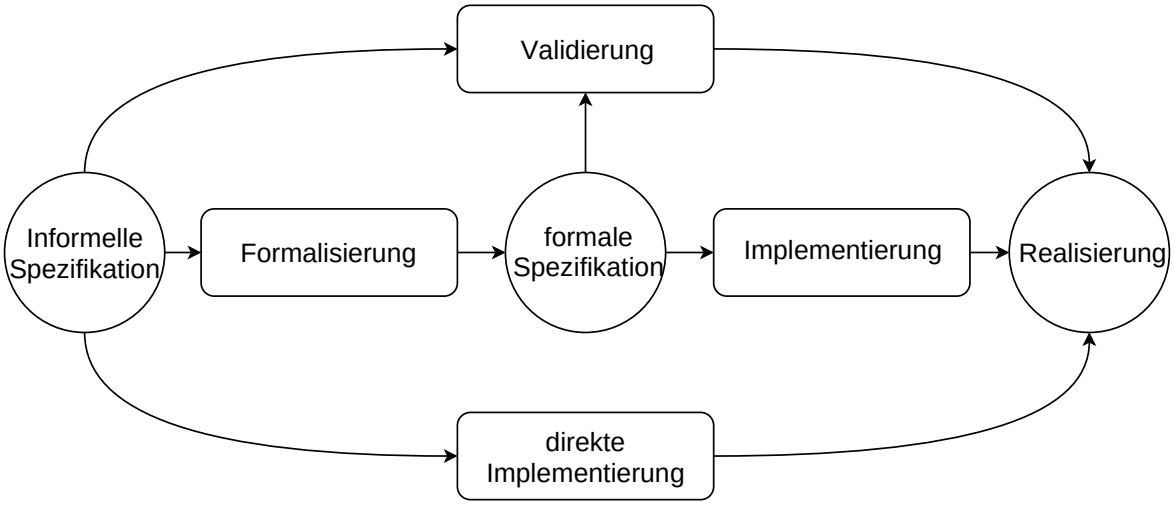


Abbildung 8: Prozess des Steuerungsentwurfs, Quelle: Georg Frey (1999), S. 146

4.4 Informelle Spezifikation

In Abbildung 8 werden zwei Wege des Steuerungsentwurfs dargestellt. Der Startpunkt ist dabei jeweils die informelle Spezifikation der Steuervorgabe. Dies bedeutet, dass diese Spezifikation an keine Syntax gebunden ist. Folglich kann eine informelle Beschreibung unvollständig, mehrdeutig oder inkonsistent sein.

Definition 4-3: Die Syntax einer Spezifikation oder Modellierungssprache definiert, wie die Symbole dieser Sprache verwendet werden und miteinander korrelieren können.

Die Semantik der informellen Spezifikationen ist umgangssprachlich oder indirekt durch die etablierte Anwendungspraxis definiert.

Definition 4-4: Die Semantik einer Spezifikation oder Modellierungssprache definiert die Bedeutung des verwendeten Konzepts und ermöglicht damit die Interpretation des resultierenden Modells.

Das Ziel der informellen Spezifikation ist eine verbale Beschreibung des ungesteuerten Prozesses und die Definition von Anforderungen an den gesteuerten Prozess. Beschrieben wird somit die Aufgabenstellung des Steueralgorithmus, der mithilfe der technischen Gegebenheiten den Prozess erfüllen soll. Die Bestandteile einer informellen Spezifikation setzen sich aus den folgenden Komponenten zusammen:

- Konstruktion- oder Aufbaupläne der technischen Anlage
- Textuelle oder grafische Beschreibung der Anforderungen an das Verhalten des Steuerkreises
- Quantitative Übersicht über die Art, Anzahl und Wirkungsweise der verwendeten Aktuatoren und Sensoren
- Beschreibung der hard- und softwaremäßigen Bedingungen für die Realisierung des Steueralgorithmus

Die Art und Gestalt dieser Konstruktions- und Aufbaupläne hängen vom Umfeld der technischen Anlage ab. Das Beispiel einer einfachen Windkessel-Steuerung aus Kapitel 7.1 wird in einer Form dargestellt, die in Fließgutprozessen häufig verwendet wird. Dem sogenannten R&I Fließbild. Diese Darstellung dient zur visuellen Beschreibung von Aktuatoren und Sensoren sowie der Aufgabenstellung selbst. Darin enthalten sind mittels Pfeile gekennzeichnete Zusammenhänge, die symbolisieren welche Elemente in einem Informationskreislauf miteinander verbunden sind. Es ist allerdings nicht möglich, durch eine solche grafische Darstellung die Steuervorgabe vollständig zu beschreiben, da dessen Komplexität mit der Anzahl an mitwirkenden Aktuatoren und Sensoren exponentiell ansteigt. Deshalb werden Dokumente angefertigt, die keiner streng definierten Syntax unterliegen, wie z. B. eine einfache Ablaufbeschreibung. Diese Dokumente dienen dazu, Festlegungen bezüglich des Steuervorgangs sowie des Steuerziels zu treffen. Diese Festlegungen sind meist unvollständig und mehrdeutig beschrieben, da der Überblick mit steigender Komplexität der Steuervorgabe leicht verloren gehen kann. Mit der Wirkung, dass sich eine solche informelle Spezifikation nicht ohne Zweifel auf Richtigkeit, Vollständigkeit und Widerspruchfreiheit überprüfen lässt. Diese Eigenschaften stellen die Herausforderung an den Steuerungsentwurfs dar. Um sich diese Herausforderung zu stellen, gilt es die Schwächen der informellen Spezifikation aufzudecken und beheben zu können.[40]

4.5 Formalismus

Die Herausforderung des Steuerungsentwurfs besteht darin, Modelle ausgehend von der informellen Spezifikation in höchster Qualität zu erschaffen und diese in eine formale Darstellung umzuwandeln. Dieser Vorgang nennt sich Formalisierung und ist eine nicht vollständig automatisierbare Leistung, die vom Menschen mithilfe technischer Einrichtungen umgesetzt wird. Die Qualität dieser Arbeit ist stark von der Kompetenz der ausführenden Fachkraft abhängig und gilt als eine herausfordernde Ingenieursleistung.[41]

Definition 4-5: Die Formalisierung ist der Vorgang zur Umsetzung einer informellen Spezifikation in eine formale Spezifikation und wird als der Schlüssel zur systematischen Lösung einer Steuervorgabe angesehen.

[40] Vgl. Litz (2013), S. 189

[41] Vgl. Litz (2013), S. 194

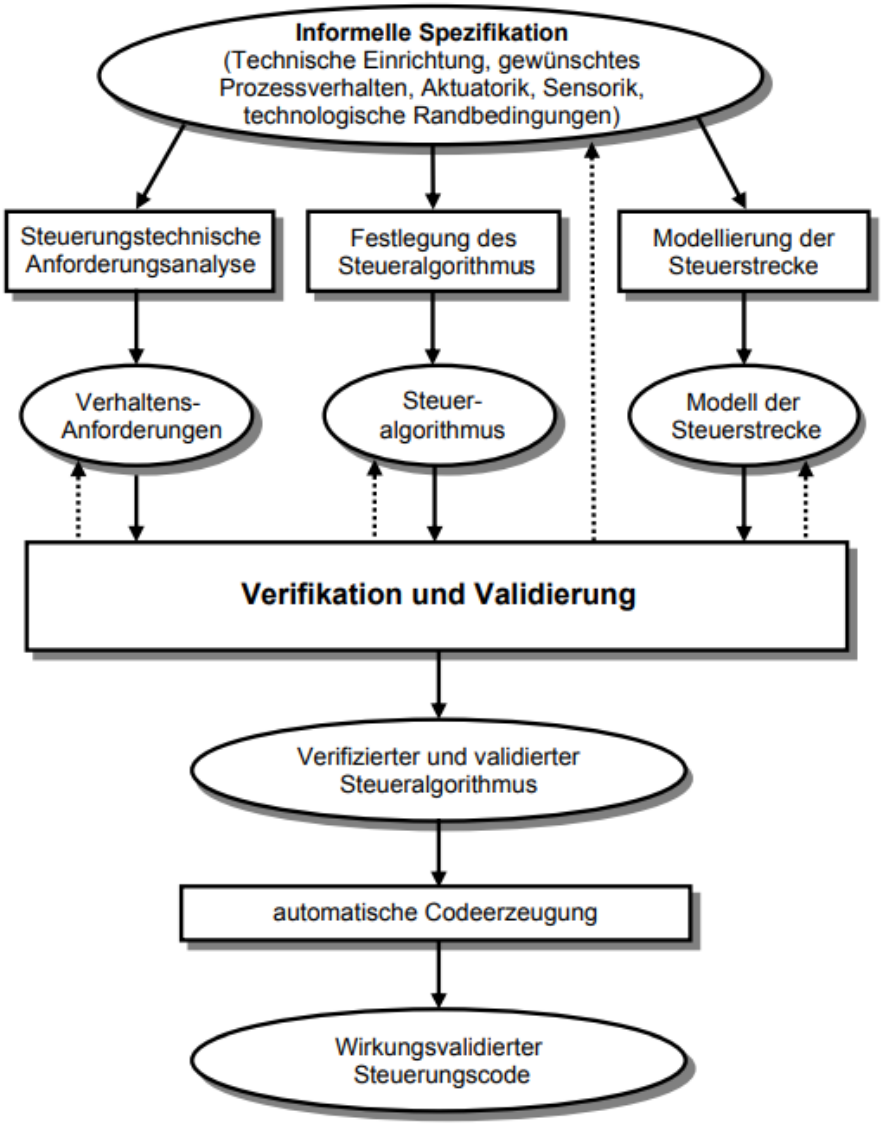


Abbildung 9: Steuerungsentwurfsprozess, Quelle: Litz (2013), S. 193

In Abbildung 9 wird der Steuerungsentwurfsprozess ausgehend von der informellen Spezifikation bis hin zum Steuercode dargestellt. Am Anfang steht im Steuerungsentwurfsprozess die informelle Spezifikation der Steuervorgabe. Beschrieben wird das geforderte Prozessverhalten der zu steuernden technischen Einrichtung sowie die Sensorik und Aktuatorik, inklusive bestimmter technologischer Randbedingungen. Ausgehend von dieser Gesamtheit werden drei formale Spezifikationen erstellt, das Modell der Verhaltensanforderungen, des Steueralgorithmus sowie der Steuerstrecke. Diese drei Modelle können durch verschiedene Modellierungsverfahren ermittelt werden, siehe Kapitel 5. Der Steueralgorithmus soll nach seiner Fertigstellung verifiziert und validiert werden, bevor dieser durch einen Automatismus in einen Maschinencode konvertiert werden kann. Je nach Zielsystem ist eine automatische Code-Erzeugung nach festen Regeln möglich. Dieser Vorgang kann als fehlerfrei und ausfallsicher bezeichnet werden und ist somit aus dem Untersuchungsinteresse dieser Arbeit ausgeschlossen.

Die Werkzeuge der Verifikation und Validierung werden angewandt, um verschiedene Arten von Fehlern in den drei verschiedenen formalen Darstellungen, sowie auch insbesondere im gewünschten Prozessverhalten zu identifizieren. Zu den formalen Darstellungen gehören die Verhaltensanforderungen, der Steueralgorithmus und das Streckenmodell. Werden Fehler erkannt, so werden entsprechende Korrekturen durchgeführt. In Abbildung 9 wird dies durch gestrichelte Pfeile dargestellt.[42]

Jedoch muss erwähnt werden, dass der systematische Steuerungsentwurf nur sinnvoll ist, wenn der Prozess zur Generierung des Steueralgorithmus automatisiert abläuft. Somit werden nur Methoden des Steuerungsentwurfs in Betracht gezogen, die einen Automatismus zur Realisierung des Steueralgorithmus ermöglichen.[43]

Liegt der Steueralgorithmus vor, ist dieser zu verifizieren und validieren. In einem Steuerkreis betrachtet man bei der Wirkvalidierung die entstehende Wirkung, die auf ein auftretendes Ereignis folgt. Diese Methodik ist keine vollständige Validierung und wird in der Entwicklungsphase des Steuerungsentwurfsprozess eingesetzt, um etwaige Fehler oder unerwünschte Wirkungen aufzudecken. Ergänzend zur Wirkvalidierung, erhofft man sich durch die praktische Validierung, die meist während der Inbetriebnahme der Steuerung abläuft, kleine Anpassungen oder die letzten kleinen Fehler im Prozess auszubessern.

Definition 4-6: Die Validierung ist ein Vorgang, der nach Ablauf eines Entwicklungsprozesses ermittelt, inwiefern das realisierte Steuerungssystem die spezifizierten Anforderungen erfüllt.

In einem Steuerungssystem wirkt sich eine Änderung, wenn auch noch so klein, immer auf das gesamte System aus. Betrachtet man einen Zustand eines binären Signales, gibt es kein etwas mehr oder etwas weniger, sondern nur konkrete Werte wie 0 oder 1. Auch die Betrachtung der möglichen Systemzustände zeigt, dass jede Änderung der Systemstruktur einen Einfluss auf alle möglichen Systemzustände hat, siehe Kapitel 2.3.3. Für ein Steuerungssystem gilt folglich die in der Definition 4-7 beschriebene Eigenschaft.

Definition 4-7: In einem Steuerungssystem hat jede Änderung eines einzelnen Bits Einfluss auf das Systemverhalten. Die praktische Validierung sollte deshalb nicht während der Inbetriebnahme, sondern durch ein modellbasierendes Verfahren (z.B. Wirkvalidierung) durchgeführt werden, um eventuelle Fehlerquellen auszuschließen. [44]

Der in dieser Arbeit bisher beschriebene indirekte Weg des Steuerungsentwurfs über einen Formalismus, ist in der Praxis nur selten anzutreffen. Vielmehr wird die direkte Umsetzung der informellen Steuervorgabe gewählt. In diesem Vorgang geht der Programmcode direkt aus der informellen Spezifikation hervor und wird so ohne formale Zwischenschritte erstellt. Obwohl man durch die direkte Umsetzung recht schnell zum

[42] Vgl. Litz (2013), S. 372 ff.

[43] Vgl. Litz (2013), S. 193 f.

[44] Vgl. Litz (2013), S. 173

gewünschten Programmcode gelangt, sind damit viele Nachteile verbunden. Der entstandene Programmcode wird meistens nicht verifiziert. Dadurch können Fehler bestenfalls erst während der Inbetriebnahme als solche erkannt. Schlimmer jedoch ist es, wenn Fehler nicht erkannt werden und womöglich einen sicheren Betrieb verhindern. Die Gefahr, dass Fehler während des späteren Betriebs auftreten, ist sehr hoch. Dieser Vorgang schließt damit alle Vorteile aus, die durch eine formale Spezifikation überhaupt erst möglich gemacht werden und ist damit keine sinnvolle Alternative zur indirekten Umsetzung einer Steuervorgabe.[45]

Der Weg der direkten Umsetzung ist mit einer gewissen Redundanz behaftet, da bei der Formulierung der Anforderungen sowie auch beim Steueralgorithmus der Gedankengang ähnlich ist. Die Festlegung der Anforderungen wird aus dem Blickwinkel des geschlossenen Steuerkreises heraus betrachtet. Im Gegensatz zum Steueralgorithmus, wo ein Teilsystem des Steuerkreises betrachtet wird. Dieses Teilsystem erzeugt die Stellbefehle aus Befehlen, Aktuator- und Prozess Rückmeldungen, siehe Abbildung 6. Diese Redundanz wird bewusst herbeigeführt, um ein System zu erschaffen, dass nicht nur zufällige Ausfälle des Systems, sondern auch systematische Fehler erkennen kann. Dieses Verhalten wird als Diversitäten-Redundanz bezeichnet und führt zu einer Steigerung der Qualität im Steuerungsentwurfsprozess, sowie zu einer Erleichterung in der Realisierung einer Steuervorgabe.[46]

4.6 Realisierung

Die Realisierung ist der wichtigste Vorgang beim Steuerungsentwurfsprozess und besteht aus der Kombination von Hard- und Software. Ausgehend von einer formalen Darstellung der informell gegebenen Aufgabenstellung, wird ein maschinenlesbarer Code in Form eines Steueralgorithmus realisiert. Involviert sind dabei mehrere Softwareebenen, die sich über das Echtzeitbetriebssystem, die Kommunikationsebene bis hin zur Algorithmenebene erstrecken. Nutzt man Standardsysteme mit definierten Funktionen wie etwa eine SPS, so ergibt die Realisierung die Software der Algorithmenebene. Neben der Hardware erlebten auch die Programmiersprachen eine umfassende Normierung.[47]

[45] Georg Frey (1999), S.147

[46] Vgl. Litz (2013), S. 194

[47] Vgl. Zander (2015), S. 107-108

5 MODELLBILDUNG

Modellierung ist eine formale Beschreibungstechnik für Systeme und deren Eigenschaften. Dargestellt werden Abläufe, Zusammenhänge und Spezifikationen. Je nach Anforderungen werden unterschiedliche Sprachen angewandt, die sich für unterschiedliche Phasen des Steuerungsentwurfs mehr oder weniger gut eignen. Dazu zählen Automaten-basierte Modelle, Petrinetze, Boolesche Gleichungen sowie eine Methode des Zustand-Zonen Modells (ZZM). In diesem Kapitel stehen der Modellierungsvorgang und die Gegenüberstellung der genannten Methoden im Vordergrund. Je nach Komplexität der zu lösenden Steuervorgabe ist das Auffinden eines Systemmodells im Steuerungsentwurf eine wichtige Aufgabe mit uneindeutigem Ausgang. Es kommt darauf an, welche Systemeigenschaften in Betracht gezogen werden und vor allem in welcher Granularität. Theoretisch ist es möglich, unendlich viele Lösungen für eine Steuervorgabe zu erhalten. Die Formung eines Systemmodells erlaubt zwar keine numerische Berechnung, hilft aber dabei grundlegende Abläufe und Zusammenhänge richtig darzustellen. Das Ziel der Modellbildung ist es, ein formales Hilfsmittel zur Verfügung zu stellen, um Systeme zu strukturieren und eine übersichtliche einheitliche Darstellung zu ermöglichen. Da es praktisch nie nur eine Lösung für eine Steuervorgabe gibt, sind die erstellten Modelle der verschiedenen Modellierungsmethoden nur schwer vergleichbar. Aus diesem Grund ist es schwierig beim Steuerungsentwurf applikationsunabhängige Standardstrukturen für individuelle Lösungen einzusetzen.[48]

Definition 5-1: Ein Modell ist ein vereinfachtes, formal dargestelltes Abbild eines physikalischen Gegenstandes oder eines anderen Modells. Es enthält relevante Eigenschaften und Merkmale, die zur Beschreibung der gestellten Aufgabe wichtig sind.[49]

Der Schlüsselfaktor zur Steigerung der Softwarequalität im Steuerungsentwurf liegt in der Modellierung des Steuerungssystems. Die Aufgabe der Modellierung eines Systems besteht darin, interne Strukturen sichtbar zu machen und systemtypische Eigenschaften zu definieren. Es ist eine sorgfältige Beschreibung des auszuführenden Systems, das die Steuerungsvorgabe ausführen soll, erforderlich. Mathematische Formeln sind für diesen Prozess nicht erforderlich, vielmehr stehen Abläufe und Funktionen im Mittelpunkt der Modellierung.[50]

5.1 Modelltheorie

Im Jahre 1847 entwickelte Georg Boole (1815 - 1864) die nach ihm benannte Boolesche Algebra, die zur Beschreibung von Kontaktschaltungen verwendet werden kann. Die speziell für diese Anwendung formulierte Ausprägung dieser Sprache wurde Schaltalgebra genannt. Damit wurde der Grundstein für die später durch D.A. Huffman, G.H. Mealy sowie E.F. Moore in den 1950er Jahren entwickelten Automatenmodelle gelegt. Später in den 1970er Jahren wurden spezielle Netztheorien basierend auf

[48] Vgl. Janschek (2010), S. 57

[49] Vgl. Marwedel (2021), S. 31

[50] Vgl. Janschek (2010), S. 58

vorangegangenen Entwicklungen vorgestellt. Die von Informatiker C.A. Petri erfundenen und nach ihm benannten Petrinetze revolutionierten die Modellbildung. Diese Petrinetze sind eine ausgehend von endlichen Automaten entwickelte Theorie zur Erstellung von Modellen diskreter Systeme. Dies gibt den Anstoß in diesem Kapitel die Grundlagen der Booleschen Gleichungen vorzustellen und an die wichtigen Kapitel der endlichen Automaten und Petrinetze anzuschließen.[51]

Zusätzlich wird im Kapitel 5.4 auf die Methode der Booleschen Algebra eingegangen, da diese für viele Methoden als Grundlage dient. Als praktikabel wird dabei eine Methode bezeichnet, die nicht nur zur Lösung einer akademischen Aufgabenstellung herangezogen werden kann, sondern bereits in Entwurfswerkzeuge integriert ist und die Realisierung einer Steuervorgabe mit einer automatisierten Umsetzung erfolgen kann.[52]

5.2 Modellierung typischer Systemeigenschaften

Die Basis stellt die informell gegebene Spezifikation der Steuervorgabe dar. Die darin enthaltenen Eigenschaften werden mithilfe von Modellierungsverfahren in eine formale Form gebracht, in der Wirkungen, Zusammenhänge und Vorgänge im Modell übersichtlich dargestellt werden. Systemtypische Eigenschaften werden häufig gruppiert, um diese besser zuordnen zu können. [53]

Folgende Eigenschafen sind in der Abbildung eines Modells wichtig:

- **Richtigkeit:** Syntaktisch ist ein Modell richtig, wenn es konsistent und vollständig gegenüber dem ihm zugrundeliegenden prozessbasierten Metamodell (beschreibt den Vorgang der Modellerstellung) definiert ist.
- **Relevanz:** Die Elemente eines Modells sind relevant, wenn ihr Fehlen einen sinkenden Nutzeffekt in der Modellverwendung hervorruft.
- **Vergleichbarkeit:** Als kompatibel gelten Modelle, die nach unterschiedlichen Methoden erstellt werden, aber dennoch syntaktisch vergleichbar bleiben. Hingegen werden inhaltliche Modellvergleiche semantisch diskutiert.
- **Wirtschaftlichkeit:** Die Modellierungsintensität wird durch den Grundsatz der Wirtschaftlichkeit begrenzt.
- **Systemverhalten:** Das Verhalten eines Systems muss einfach und verständlich dargestellt werden. Sowohl Beziehungen zwischen Elementen eines Systems als auch das sich ergebende neue Verhalten, soll vorhersehbar sein. [54]
- **Nebenläufigkeit:** Nebenläufigkeit ist die Eigenschaft eines Systems, mehrere Aufgaben, Berechnungen, Anweisungen oder Befehle gleichzeitig ausführen zu können. Daher ist eine einfache Spezifikation ein gutes Hilfsmittel, damit ein Mensch diese Vorgänge auch in ihrer

[51] Vgl. Zander (2015), S. 21-22

[52] Vgl. Georg Frey (1999), S. 150

[53] Vgl. Litz (2013), S. 310

[54] Vgl. Bernroider/Stix (2005), S. 4 ff.

Gesamtheit verstehen kann. Ein Grund, weshalb reale Systeme problembehaftet sind, ist die Folge mangelndes Verständnis für mögliche Abläufe in parallelen Systemen.

- **Synchronisation und Kommunikation:** Elemente eines Systems teilen sich dessen Ressourcen und müssen daher miteinander kommunizieren. Diese Kommunikation regelt die Benutzung und Verteilung der Ressourcen. Eine Doppelnutzung kann ausgeschlossen werden.
- **Zeitverhalten:** Die Echtzeitfähigkeit ist in Steuersystemen eine der wichtigsten Anforderungen und wird daher in einer expliziten Zeitbedingung definiert. Die Zeit ist in formalen Spezifikationen als Bedingung gegeben und ist eine ordnende Größe. In der allgemeinen Informatik existieren Methoden zur Berechnung der vergangenen Zeit, ob ein Algorithmus irgendwann terminiert und ob sich ein Prozess um eine bestimmte Zeit verzögern lässt.
- **Sicherheitseigenschaften:** Sicherheitseigenschaften sind Handlungs- oder Ereignisbeschreibungen, die niemals eintreten dürfen. Diese Eigenschaften stellen das höchste Gut in einem System dar, da Gefahren für Leib und Leben in technischen Systemen minimiert oder gar ausgeschlossen werden müssen.[55]

Definition 5-2: In einem Echtzeitsystem muss nachgewiesen werden, dass ein Algorithmus innerhalb einer bestimmten Zeitspanne terminiert oder nicht.

Die Modellierung von Systemeigenschaften ist eine herausfordernde Ingenieurstätigkeit, die bisher als nicht oder nur im Ansatz automatisierbar angesehen wird.[56]

5.3 Grundlegende Modellierungsverfahren

In diesem Teil der Arbeit werden ausgewählte Methoden zur Modellierung einer formalen Spezifikation vorgestellt. Diese Methoden gelten unter den bisher bekannten Methoden des systematischen Steuerungsentwurfs als die am vielversprechendsten, da es möglich ist, auch komplexere Sachverhalte darzustellen. Dabei gilt der allgemeine Grundsatz, je genauer und weitreichender die Aussagen sind, desto mehr Wissen muss einbezogen werden. Die Aussagen eines Modells werden qualitativer, je präziser ein zu steuerndes System durch eine Methodik beschrieben wird. Diese Werkzeuge des Steuerungsentwurfs unterscheiden sich durch die Art der Realisierung und auf welchen formalen Spezifikationen sie basieren.[57]

5.4 Boolesche Algebra

Durch Boolesche Gleichungen kann eine Steuereinrichtung beschrieben werden. Aussagen werden über Ausgänge dargestellt und in Form einer Wahrheitstabelle angezeigt. Daraus kann in weiterer Folge der Steueralgorithmus abgeleitet werden. In diesem Steueralgorithmus wird der Eingang u auf den Ausgang y abgebildet. Die Eingangswerte können in Form eines binären Eingangsvektors u mit dem Index m und die

[55] Vgl. Marwedel (2021), S. 31 ff.

[56] Vgl. Marwedel (2021), S. 33

[57] Vgl. Georg Frey (1999), S. 150 f.

Ausgangswerte in Form eine binären Ausgangsvektors y mit dem Index n dargestellt werden. Es werden den Elementen der Ein- bzw. Ausgangsvektors binäre Werte zugeordnet. Dieser Vorgang wird als Wertkombination bezeichnet. Anstelle des Begriffs Wertkombination kann auch der Begriff Schaltbelegung verwendet werden. Dies ist der Fall, wenn man die Eingangsvariablen mit den Wahrheitswerten 0 oder 1 belegt. Schaltsysteme haben die Aufgabe, Signale logisch zu verknüpfen oder zu speichern. Die wichtigsten Elemente zur Ausführung dieser Aufgaben sind die UND-, ODER-Verknüpfungen, siehe Formel (5-1) und Formel (5-3).[58]

$A \wedge B$	(5-1)	$\wedge$	Logisch und
$A \vee B$	(5-2)	$\vee$	Logisch oder

Das Verhalten eines Systems ist aus der Abfolge der Eingangskombinationen zu analysieren und in einem Ablaufdiagramm darzustellen. Innerhalb definierter Zeitintervalle ergeben Signaländerungen verschiedene Eingangsbilder, anhand derer eine Wertetabelle mit entsprechenden Ausgangskombinationen aufgestellt werden kann. Der innere Zustand des Speicherelementes kann nicht nur vom aktuellen, sondern auch vom Vorzustand der Eingangskombination beeinflusst werden. Dieses Zeitverhalten ist eine Eigenschaft des Systems und wird durch den inneren Zustand repräsentiert. Um ein solches System zu beschreiben, werden verschiedene Methoden der Modellierung angewandt. Neben steuerungstechnisch interpretierte Petrinetzen, zählen auch Modelle der Automatentheorie zu den bedeutendsten.

5.5 Modellierung mit endlichen Automaten

In Steuerungssystemen sind endliche Automaten als Teilgebiet der Automatentheorie interessant, da diese zur Modellierung eines bestimmten Verhaltens benutzt werden. Bei diesen Automaten handelt es sich um kombinatorische Modelle aus Zuständen und Zustandsübergängen. Einen Zustandsübergang charakterisiert dabei die Änderung eines Zustandes und die Bedingung, die dafür erfüllt sein muss. Darüber hinaus gibt es für endliche Zustandsautomaten vier Arten von Aktionen:

- Eingangsaktion: Beim Eintritt eines Zustands
- Ausgangsaktion: Beim Verlassen eines Zustands
- Eingabeaktion: Abhängig von der Eingabe und dem aktuellen Zustand
- Übergangsaktion: Abhängig vom Zustandsübergang

Wichtig für die Modellierung eines endlichen Automaten ist dessen Präzisierung und Darstellungsform. Dies stellt die Grundlage für eine einheitliche Beschreibung eines Steuerungssystems dar. Ein Automat ist in der Mengen-Funktionsdarstellung als 6-Tupel beschrieben, siehe Formel (5-3).

$A(Z, U, Y, G, H, z_0)$	(5-3)	Automat in Mengen-Funktionendarstellung

[58] Vgl. Zander (2015), S. 53-54

Folgende sechs Elemente sind Teil des Automaten:

$Z = (z_1, \dots z_n)$	(5-4)	Die endliche Menge an Zuständen
$U = (u_1, \dots u_i)$	(5-5)	Eingangsmenge (Eingangsalphabet)
$Y = (y_1, \dots y_j)$	(5-6)	Ausgangsmenge (Ausgangsalphabet)
$z(0) \in Z$	(5-7)	Ausgangszustand zu Beginn der Betrachtung
$G\colon\ Z \times U \to Z$	(5-8)	Zustandsübergangsfunktion
$H\colon\ Z \times U \to Y$	(5-9)	Ausgangsfunktion

Daraus ergibt sich die Automatengleichung, wenn jeder Schritt der Betrachtung mit k und jeder weitere mit $k + 1$ beschrieben wird.

$z(k+1) = G\big(z(k), u(k)\big)$	(5-10)	Automatengleichung
$y(k) = H\big(z(k), u(k)\big)$		$z(k) \in \mathrm{Z}, \quad u(k) \in U, \quad y(k) \in Y, \quad k = 0,1, \dots$

Erfolgt ein Eingang, so löst diese einen Schrittwechsel aus. Der genaue Ablauf, erfolgt über die Zuordnung der Eingänge zu den Signalen in diesem allgemein betrachteten Automatenmodell. Aus dieser Automatengleichung aus der Formel (5-10) lässt sich das Verhalten eines allgemein betrachteten Automaten wie in Abbildung 10 darstellen.

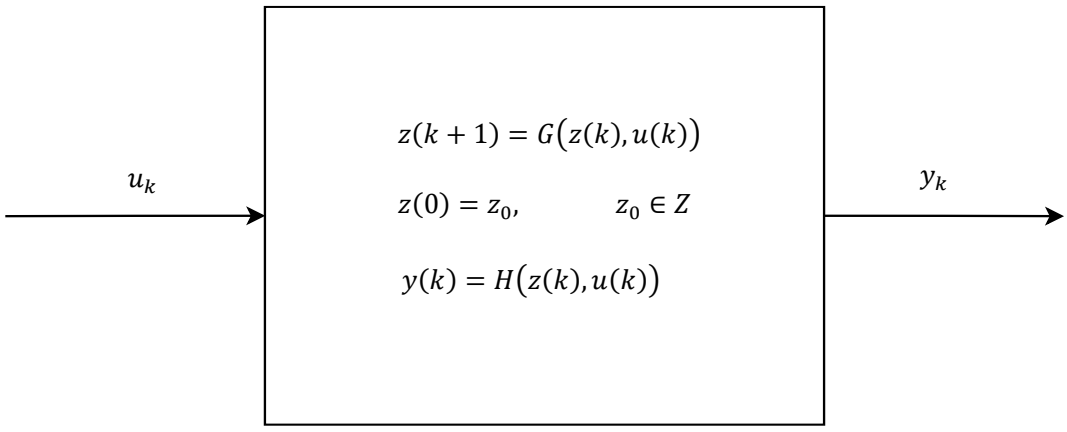


Abbildung 10: Ein Automat als ereignisdiskretes System, Quelle: Eigene Darstellung, in Anlehnung an Litz (2013). S. 207

Um nun einen Automaten für den Menschen leichter verständlich darzustellen, wird ein sogenannter Automatengraph angefertigt, siehe Abbildung 11. Für diesen gilt:

- Der Automatengraph ist ein gerichteter Graph, mit Zuständen als Knoten die als Elemente der Zustandsmenge Z gelten.
- Transitionen beschreiben Übergänge zwischen Zuständen und werden als Kanten bezeichnet, die aufgrund der Zustandsübergangsfunktion G vorkommen.
- Eine Kante wird mit den zugehörigen Ein-Ausgangsignalen u_i/y_j beschriftet, die aus der Zustandsübergangsfunktion G und der Ausgangsfunktion H resultieren.
- Ein unbeschrifteter Pfeil markiert den Ausgangszustand z_0.

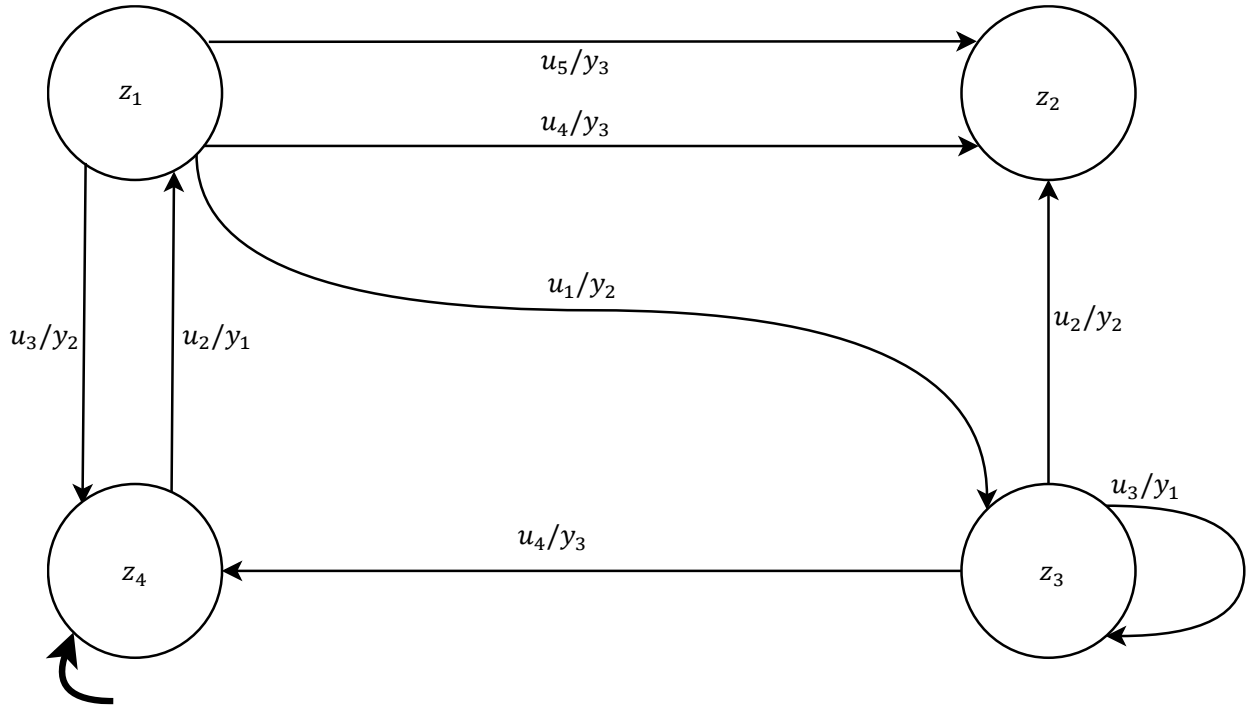


Abbildung 11: Ein Beispiel als Automatengraph, Quelle: Eigene Darstellung, in Anlehnung an Litz (2013), S. 210

5.5.1 Bedeutung von Automatenmodellen

Die Charakteristika der Automatentheorie und ihrer Modellierungsideen im Bereich der Technikwissenschaften sind die systematische Formulierung und Bearbeitung von Begriffen wie Zeichen, Ereignis oder Zustand. Dadurch werden Text-, Bild- und Tonsignale als Informationen verarbeitet und mittels Sender von einem Ort zu einem Empfänger an einen anderen Ort übertragen. Wird mit diesen Begriffen ein System anhand der Aufgabenstellung beschrieben, entsteht ein logisch, mathematisch definierbares System. Die System- oder Informationstheorie wird auch zur Steuerung und Regelung von Systemen und deren Programmierung verwendet. Konventionell wird bei der Analyse und Synthese von Schaltnetzen, diskret aufgebauten Systemen oder Relaisschaltungen seit dem Jahre 1938 die Boolesche Algebra benutzt. Die ersten Automaten wurden früher durch diese Logik entwickelt. Der Entwurf und die Analyse von Automaten werden als Verhältnis der Logik, Funktion und Algebra zueinander beschrieben.

Diese Begrifflichkeiten werden in der Modellbildung verwendet, um einen rechenbaren Prozess zu entwickeln.[59]

Definition 5-3: Ein **deterministischer endlicher Automat** ist ein Automat, der durch einen Eingang eines Zeichens des Eingangsalphabets, von einem Zustand, in dem er sich befindet, in einen eindeutig bestimmten Folgezustand wechselt. Die endliche Anzahl von Zuständen beinhaltet einen Start- sowie Endzustand. Ausgehend vom Startzustand, muss ein Übergang vom aktuellen Zustand zum Folgezustand existieren, bis der Endzustand erreicht wird. Erfolgt auf den gleichen Eingang immer der gleiche Ausgang, spricht man vom deterministischen Automaten. Bei diesem wird die Kausalität in Form von Ursache-Wirkungs-Beziehungen ausgedrückt.

In der Vergangenheit hatte die Beschreibung sequentieller Steuerungen über deterministische endliche Automaten einen hohen Stellenwert, da die Minimierung der Hardware einer damals hauptsächlich eingesetzten Verbindungsprogrammierbaren Steuerungen (VPS) im Vordergrund stand. Damit auch alle Wertekombinationen der Ein- bzw. Ausgänge in der Automatenfunktion vorkamen, wurden die Beschreibungen sehr umfangreich und unübersichtlich. Erst durch den Einsatz der SPS wurde es möglich, auch Automatenfunktionen mit unvollständig definierten Wertekombinationen einzusetzen. Später verloren Automatenmodelle durch die Einführung von steuerungstechnisch interpretierte Petrinetze (SIPN) an Bedeutung, da diese Beschreibungsmethoden weitreichendere Entwurfsmöglichkeiten bieten. Heute sind Automatentabellen hauptsächlich in der Beschreibung sequentiellen Verhaltens von Steuerungen durch Überführungstabellen relevant, da ein Petrinetz diese Form der Darstellung nicht unterstützt. Aus diesem Grunde wird in dieser Arbeit nach einer kurzen Übersicht über wichtige Eigenschaften der Automatenmodelle und ihr Verhalten, der Fokus auf Petrinetze und das Zustand-Zonen Modell gelegt. [60]

5.5.2 Automaten und ihr Verhalten

Das Verhalten von Automaten ist ein Oberbegriff für die Beschreibung einzelner oder voneinander getrennte, unabhängigen Eigenschaften. Betrachtet werden insbesondere die Lebendigkeit sowie das Antwortverhalten des Automaten. [61]

Seit den 1970er Jahre verlor die Hardware-Minimierung an Relevanz, weshalb Automatenmodelle zur Beschreibung von Steueralgorithmen für SPS eingesetzt wurden. Durch die Einführung der steuerungstechnisch interpretierte Petrinetze (SIPN) verloren Automatenmodelle jedoch zunehmend an Bedeutung, da diese weitreichendere Beschreibungsmöglichkeiten bieten. Automatenmodelle spielen in der Steuerungstechnik jedoch weiterhin eine gewisse Rolle, da diese in der Lage sind, sequentielles Verhalten von Steuerungen durch Überführungs- und Ausgangsfunktionen auszudrücken. Dies gewährt Automatenmodellen einen gewissen Vorteil gegenüber anderen Modellarten wie z.B. Petrinetzen. Für

[59] Lunze (2012), S. 23 ff.

[60] Vgl. Litz (2013), S. 218 ff.

[61] Vgl. Litz (2013), S. 224

zukünftige Anwendungen wird an einer Methode zur Minimierung von Systemzuständen sowie des Beschreibungsaufwands auf Basis von Automatenmodellen im Steuerungsentwurf geforscht.[62]

5.6 Deterministisch endliche Automatenmodelle

Endliche Automaten sind ein Modellierungsansatz und stellen eine der wichtigsten Grundlagen zur Beschreibung ereignisdiskreter Systeme dar. Die wichtigsten Eigenschaften sind der flexible Modellierungsansatz sowie die Hard- und Softwareunabhängigkeit in der Anwenderumgebung. In dieser Arbeit sind besonders die deterministisch endlichen Automatenmodelle von Mealy und Moore von Interesse. Beide Modelle sind dadurch gekennzeichnet, dass jeder Eingang einen eindeutig spezifizierten Ausgang mit definierter Endzustandsmenge bewirkt.[63]

5.6.1 Mealy- und Moore-Automat

Mealy definierte sein Modell einer sequentiellen Schaltung anhand folgender **Definition 5-4**.

Definition 5-4: Mealy-Automat: Ein Schaltkreis ist eine Schaltung mit einer endlichen Anzahl von Eingängen, Ausgängen und (internen) Zuständen. Die aktuelle Ausgangskombination und der nächste Zustand werden eindeutig durch die aktuelle Eingangskombination und den aktuellen Zustand bestimmt.[64]

Abbildung 12 und Abbildung 13 sind der Mealy- sowie Moore Automat in Form eines Automatengraphs dargestellt. Häufig unterscheiden sich die Bezeichnungen für die Elemente des 5- bzw. 6-Tupel. In der grafischen Darstellungsform werden Eingänge des Eingangsalphabets als $u_0, u_1, \dots, u_n$ bezeichnet, sowie Ausgänge des Ausgangsalphabets als $y_0, y_1, \dots, y_n$. Es existiert keine einheitliche standardisierte Normung für diese Bezeichnungen. So sind die Zustände in Kreisen dargestellt, in deren Mitte die Bezeichnung des Zustandes steht. Der Ausgangszustand wird mit einem Pfeil ohne definierten Anfang markiert. In Abbildung 12 und Abbildung 13 ist der Ausgangszustand als z_0 sowie ein Übergang (Transition) durch einen Pfeil dargestellt. Die notwendige Bedingung (Eingang/Ausgang) für eine Transition wird ober- oder unterhalb der Pfeile beschriftet.

[62] Vgl. Zander (2015), S. 108

[63] Vgl. Köhler-Bußmeier (2015), S. 73

[64] Vgl. Zander (2015), S. 110

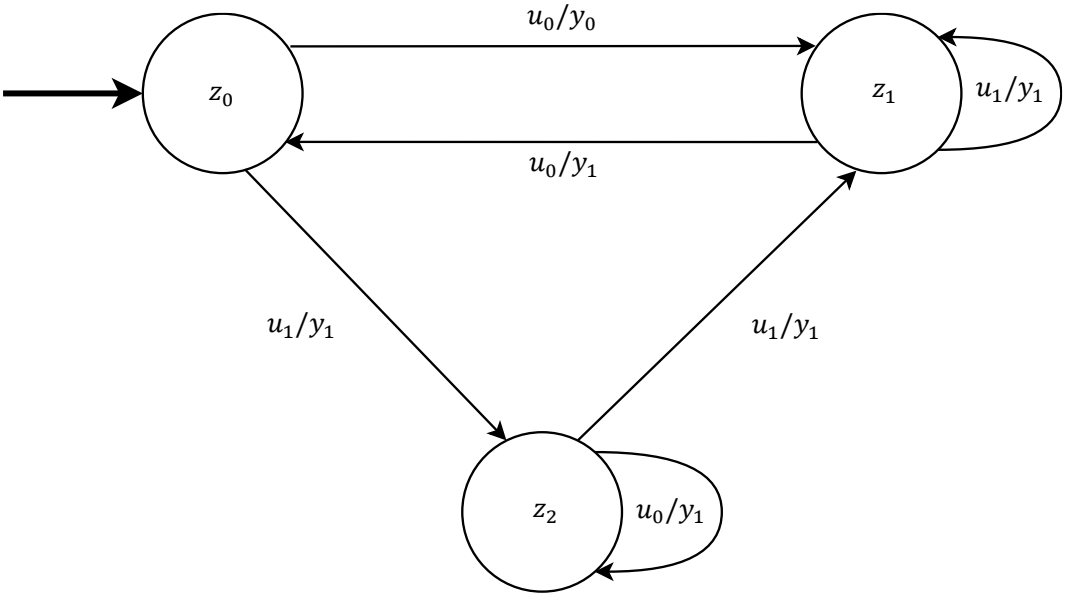


Abbildung 12: Mealy-Automat, Quelle: Eigene Darstellung, in Anlehnung an Jena (2021), Online-Quelle [22.03.2022]

Ein Automat, dessen Ausgang von Zustand und Eingang des Systems abhängt, wird Mealy-Automat genannt, siehe Abbildung 12. Im Gegensatz dazu hängt bei einem Moore-Automat der Ausgang nur vom aktuellen Zustand des Systems ab.[65]

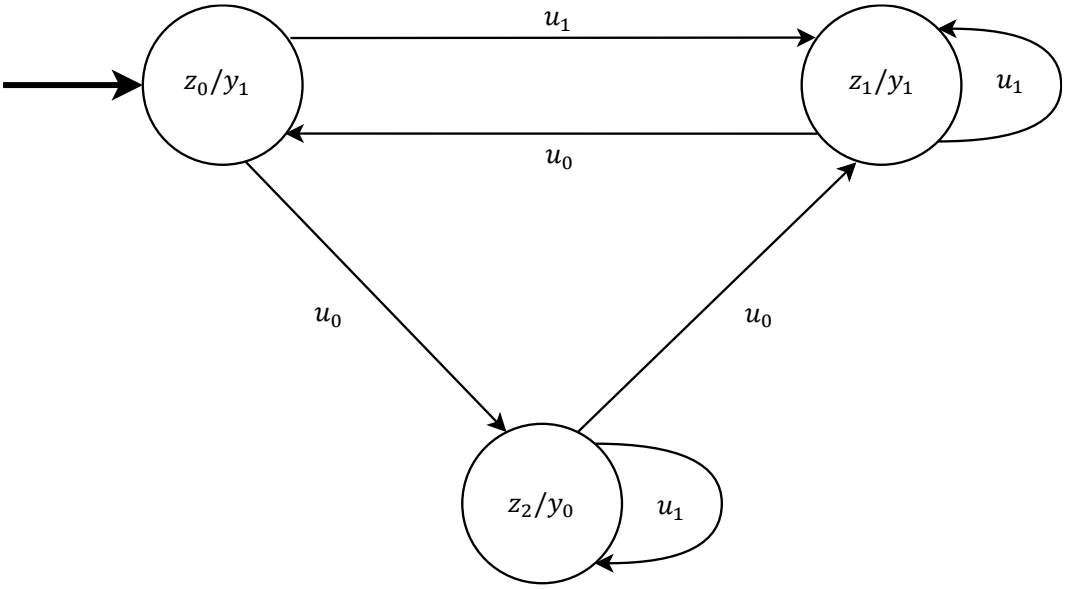


Abbildung 13: Moore-Automat, Quelle: Eigene Darstellung, in Anlehnung an Jena (2021), Online-Quelle [22.03.2022]

Aus der Abbildung 13 geht hervor, dass es sich beim Moore-Automaten um eine getaktete Schaltung handelt, da der Ausgang vom aktuellen Zustand abhängt und nicht vom Vorzustand. Moore definierte sein Modell anhand folgender Definition:

[65] Vgl. Georg Frey (1999), S. 212 ff

Definition 5-5: Moore-Automat: Das Verhalten von sequentiellen Maschinen ist streng deterministisch, da der aktuelle Zustand einer Maschine nur von ihrem vorherigen Eingang und ihrem vorherigen Zustand abhängt. Der aktuelle Ausgang hängt folglich nur vom gegenwärtigen Zustand der Maschine ab.

Es ist möglich einen Mealy-Automat in einen Moore-Automat umzuwandeln und umgekehrt. Für die Darstellung einer Steuervorgabe kommt der Mealy-Automat aufgrund seiner einfacheren Struktur mit weniger Zuständen aus, da es Eingängen direkt möglich ist, Ausgänge zu beeinflussen. Für dieselbe Funktion werden mithilfe des Moore-Automat weitere Zustände benötigt, da der Ausgang immer vom aktuellen Zustand des Systems abhängt.[66]

5.6.2 Aussagen zur Zustandsreduktion

Bei Zustandsautomaten wird versucht, die erforderlichen Speicherelemente sowie die Zustände minimal zu halten. Es gilt der Grundsatz, dass zwei Automaten als ähnlich gelten, wenn sie das Gleiche tun. Ähnliche Zustände werden als verträglich bezeichnet und können zur Optimierung kombiniert werden. Ein System hat eine Verträglichkeitsklasse, wenn alle Zustände mit einer ähnlichen Leistung in einer Klasse zusammengefasst sind. Ein System kann mehrere Verträglichkeitsklassen enthalten. Diese Ansammlungen zielen nicht nur auf die Reduktion der Zustände ab, sondern auch darauf, die Anzahl der logischen Elemente eines Systems zu reduzieren. Wenn in Sammlungen ähnliche Zustände mehrfach auftreten, können die nicht benötigten Zustände weggelassen werden. Aus den ähnlichen Zuständen die übrig bleiben, können Einheiten gebildet werden, unter der Voraussetzung, dass das System in sich geschlossen bleibt und in jeder Verträglichkeitsklasse mindestens ein Zustand auftritt.[67]

5.7 Petrinetze

Petrinetze sind Modelle für diskrete, vorwiegend verteilte Systeme. Der Informatiker Carl Adam Petri entwickelte sie in den 1960er Jahren, ausgehend von endlichen Automaten. Waren Petrinetze gerade am Anfang aber eher noch Betrachtungsgegenstand von Theoretikern, erfolgte der Übergang in die Praxis erst Mitte der 1980er Jahre. Dieser Umstand geht daraus hervor, da Ein- und Ausgänge bei Petrinetzen nicht definiert sind. Erst durch die Einführung spezieller Netze, den sogenannten interpretierten Petrinetzen, gelang der Einzug in die Steuerungstechnik. Die Vorteile eines Petrinetzes liegen in der Fähigkeit, parallele Abläufe und Nebenläufigkeiten einfach und übersichtlich darstellen zu können. Ein Steueralgorithmus der Petrinetz-basierend generiert ist, bietet umfassende Analysemethoden, sowohl aus dem netzspezifischen Bereich als auch aus dem Bereich der Automatenmodelle. Weiters sind Petrinetze in Automatenmodellen nach Moore umwandelbar. Im Vergleich zu Automatenmodellen verringern Petrinetze die Modellkomplexität, indem Prozesse durch parallele Strecken in der grafischen Darstellung repräsentiert werden. Da der Graph mehrere markierte Knoten enthalten kann, wird der Zustand des Systems nicht mehr durch einen einzelnen markierten Knoten, sondern durch eine Menge markierter Knoten beschrieben. Da

[66] Vgl. Zander (2015), S. 111

[67] Vgl. Litz (2013), S. 122 f.

es mehrere sich unterschiedlich schnell bewegende Markierungen im Graphen geben kann, ist es einfacher parallele Prozesse darzustellen.[68]

5.7.1 Beschreibung der Netzelemente

In diesem Abschnitt werden die Netzelemente des Petrinetzes beschrieben.

Definition 5-6: Ein Petrinetz ist als bipartiter Digraph mit Plätzen, Transitionen und Knoten definiert.

In Abbildung 14 ist ein autonomes Petrinetz ohne Eingangsgrößen dargestellt. Als bipartit ist die Eigenschaft des Netzes bezeichnet, zwei überschneidungsfreie Teilmengen von Knoten zu besitzen. Diese Teilmengen werden wie folgt dargestellt:

- Plätze oder Stellen werden als Kreise dargestellt und beschreiben den Systemzustand
- Transitionen sind durch schwarze Balken oder Rechtecke dargestellt und beschreiben die Zustandsübergänge (Ereignisse) [69]

Die Mengen dieser Netzelemente werden mit P und T bezeichnet. Die markierten Stellen sind in den Abbildungen durch eine Marke gekennzeichnet. Das Verhalten des Petrinetzes wird durch die Bewegung der Markierungen beschrieben, die den nachstehend beschriebenen Regeln folgt.[70]

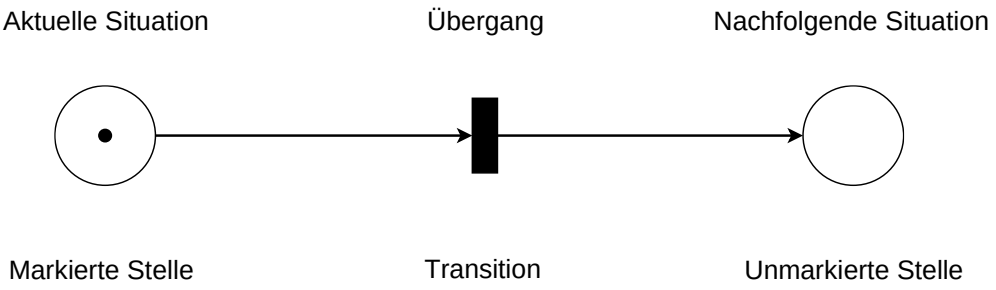


Abbildung 14: Netzelemente, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2020), S. 389

[68] Vgl. Lunze (2020), S. 389

[69] Vgl. Georg Frey (1999), S. 234

[70] Vgl. Lunze (2020), S. 389

Ein Petrinetz wird durch einen bipartiten Graphen dargestellt. Einen Graphen, in dem sich die beiden Knotentypen immer abwechseln, wenn man den Graphen entlang der gerichteten Kanten durchläuft. Aus der Sicht der Transitionen unterscheidet man zwischen:

- **Präkanten:**
  Sind Kanten von den Stellen $p \in P$ zu den Transitionen $t \in T$.
  Die Menge dieser Kanten wird mit $Prae$ gekennzeichnet.
- **Postkanten:**
  Sind Kanten von den Transitionen $t \in T$ zu den Stellen $p \in P$.
  Die Menge dieser Kanten wird mit $Post$ gekennzeichnet.

Der Systemzustand wird nicht durch eine einzelne Stelle beschrieben, sondern durch die Menge aller gleichzeitig markierten Stellen. Daher kann jede Stelle nicht einem Zustand, sondern nur einer Situation zugeordnet werden. Mehrere Situationen, die in unterschiedlichen Kombinationen auftreten können, charakterisieren den Systemzustand. Gerichtete Kanten, die von einer Stelle über einen Übergang zu einer anderen Stelle führen, können eine Markierung nach bestimmten Regeln von einer Stelle zur nächsten führen. Dieses Markierungskonzept ermöglicht die Modellierung dynamischer Vorgänge. Eine Stelle hat entweder keine oder genau eine Marke. Der Besitz einer Marke im einzigen Vorplatz einer Transition kann als **Bedingung** gedeutet werden, damit die Transition als **Ereignis** weiterschalten kann. Dies bedeutet für das System, dass eine bestehende Situation durch eine nachfolgende Situation ersetzt wird. Wichtig ist, dass mehrere Stellen gleichzeitig markiert sein können, siehe Abbildung 15.

Definition 5-7: Das **Markierungskonzept** besagt, dass Stellen Marken besitzen können, die von Transitionen durch das Netz geschalten werden können.

Die Markierung des Netzes zum Zeitpunkt $k = 0$ wird als M_0 bezeichnet und bildet zusammen mit den Beziehungen $Prae \subseteq p \times t$ und $Post \subseteq t \times p$ ein 5-Tupel, welches formal beschrieben werden kann, siehe Formel (5-11).

$$PN = (p, t, Prae, Post, M_0) \quad (5\text{-}11)$$

PN	Petrinetz
p	Stellen
t	Transitionen
$Prae$	Präkanten
$Post$	Postkanten
M_0	Markierung zum Zeitpunkt $k = 0$

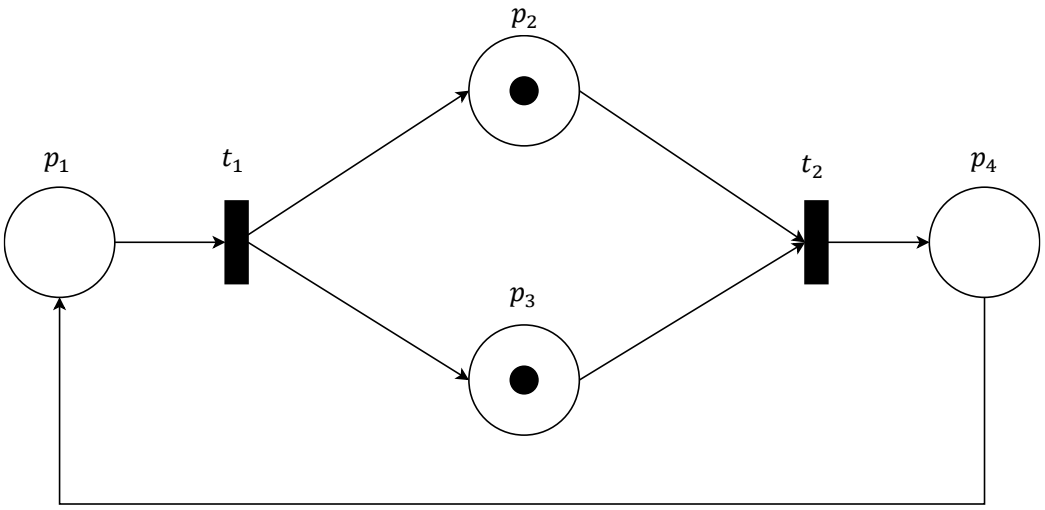


Abbildung 15: Petrinetz mit zwei parallelen Prozessen, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2020), S. 390

Die Prästellen (Vorgängerstellen) einer Transition sind die Menge der Stellen, von denen eine gerichtete Kante zu dem jeweiligen Übergang führen. Poststellen (Nachfolgerstellen) sind die Stellen, zu denen gerichtete Kanten von dem Übergang führen. Das Verhalten des Petrinetzes wird durch einen Markenfluss definiert. Das Netz hat eine Anfangsmarke mit der Bezeichnung M_0. Die Marken bewegen sich entlang der gerichteten Kanten von ihren Positionen über Transitionen zu Nachfolgepositionen. Dies wird als Schalten oder Feuern von Transitionen bezeichnet und ändert die Markierung M. Die folgende Schaltregel muss beachtet werden:

- Eine Transition ist aktiviert, wenn alle ihre Prästellen markiert sind.
- Eine Transition ist aktiviert, wenn alle ihre Poststellen nicht markiert sind.

Durch das Schalten oder Feuern einer aktiven Transition, werden allen Prästellen die Marke entzogen und alle Poststellen markiert.[71]

5.7.2 Modellbildung mit Petrinetzen

Die Beschreibung von technischen Systemen durch Petrinetze wird durch die Tatsache erleichtert, dass wichtige Elemente der Modellierung explizit durch entsprechende Netzelemente ausgedrückt werden können:

- **Prozessorientierte Modellbildung:**
  Das Systemverhalten wird durch eine Abfolge von Teilprozessen dargestellt. Das dynamische Verhalten des Systems besteht aus Abschnitten, in denen ein oder mehrere parallellaufende Teilprozesse aktiviert werden. Ereignisse markieren den Start oder das Ende eines oder mehrerer Teilprozesse. Die Teilprozesse können direkt als parallele Pfade eines Petrinetzes dargestellt werden.

[71] Vgl. Lunze (2020), S. 390 f.

- **Zustandsraum:**
  Der Systemzustand wird durch die Markierungen des Petrinetzes bestimmt. Der Zustandsraum wird durch die Menge aller möglichen Markierungen bestimmt, die von der Netzstruktur abhängen.
- **Ereignisse:**
  Die Ereignisse, die das Systemverhalten charakterisieren, werden anhand schaltender Transitionen nachgebildet.
- **Kausalität:**
  Ereignisse können nur auftreten, wenn alle notwendigen Bedingungen erfüllt sind. Diese Bedingungen werden in Petrinetzen durch Stellen repräsentiert. Eine Bedingung ist erfüllt, wenn die entsprechende Stelle markiert ist.
- **Nichtdeterminismus:**
  Ereignisse in parallellaufenden Teilprozessen können in beliebiger Reihenfolge zueinander auftreten, was zu einem nichtdeterministischen Systemverhalten führt. Petrinetze beschreiben diesen Nichtdeterminismus.
- **Dynamik:**
  Das dynamische Verhalten des Netzes wird durch den Markenfluss charakterisiert, der zu einer Folge von Ereignissen führt.[72]

5.7.3 Steuerungstechnisch interpretierte Petrinetze (SIPN)

Bei endlichen Automaten sind die Ein- und Ausgänge von Anfang an Teil der Definition, siehe die Arbeiten von Mealy und Moore aus Kapitel 5.6.1. Wie bereits erwähnt, ist dies in der Theorie der Petrinetze nicht der Fall. So ist in Petrinetzen der Informationsfluss zum Netz (Eingänge) und vom Netz weg zur Umwelt (Ausgänge) nicht allgemein definiert. Dies hat dazu geführt, dass sich viele Möglichkeiten entwickelt haben. Eine Zuordnung von Eingängen und Ausgängen zu den Netzelementen wurde eingeführt, da diese sich sehr gut für die Modellierung ereignisdiskreter Systeme eignet. Daraus entstand das sogenannte interpretierte Petrinetz (IPN). Die Bezeichnung weist darauf hin, dass dem herkömmlichen Petrinetze eine geeignete Semantik hinzugefügt wird, die den Netzelementen Transition und Stelle eine wohldefinierte Bedeutung im Sinne des Informationsflusses (Eingänge, Ausgänge) zuweist. In diesem Kapitel wird das IPN gleich in einer besonders für die Steuerungstechnik geeigneten Form, das steuerungstechnisch interpretierte Petrinetze (SIPN), beschrieben. Die Zuordnung von Ein- und Ausgängen zu den Netzelementen resultiert aus den Eigenschaften der Netzelemente. Eine Transition kann keine Marke halten. Sie nimmt nur Marken von Prästellen auf und fügt Marken zu Poststellen hinzu. Es gibt also zwei Möglichkeiten: Die Markierung einer bisher unmarkierten Stelle kann als Ereignis interpretiert werden, indem man ein Ausgangsereignis zuordnen kann, oder Signalzustände werden direkt einer markierten Stelle zugeordnet. In beiden Fällen geht der Informationsfluss nach außen von den Stellen aus.

[72] Vgl. Lunze (2020), S. 393

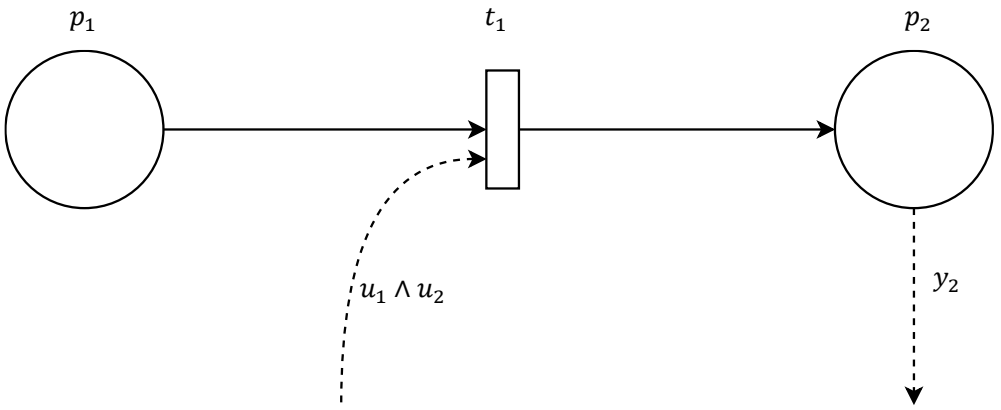


Abbildung 16: Elemente eines SIPN, Quelle: Eigene Darstellung, in Anlehnung an Lunze (2020), S. 401

Abbildung 16 zeigt die Zuordnung der Eingangssignale zu den Transitionen in Form von Schaltbedingungen, sowie die Zuordnung der Ausgangssignale zu den Stellen. Die Erweiterung der Petrinetze um Eingänge und Ausgänge soll es erlauben, das Schalten der Transitionen von außen durch Eingänge zu beeinflussen und den Stellen Ausgänge zuzuordnen. Man spricht hier auch vom steuerungstechnischen interpretierten Petrinetz (SIPN), weil man den abstrakten Netzelementen, den Stellen und Transitionen, eine steuerungstechnische Bedeutung gibt.

Der Transition t_1 werden die Eingangssignale $u_1 \wedge u_2$ zugewiesen, was durch den unterbrochenen Pfeil in Abbildung 16 dargestellt ist. Weiters ist eine Darstellung ohne unterbrochenen Pfeil möglich, siehe Abbildung 29. Die Transitionen, die durch eine Steuerung beeinflusst werden, sind durch ein nicht ausgefülltes Rechteck hervorgehoben, siehe Abbildung 16. Der Übergang t_1 kann nur dann schalten, wenn er gemäß der Schaltregel aktiviert wird und wenn gleichzeitig das Argument $u_1 \wedge u_2$ einen gültigen Wert zum Schalten liefert. Die zusätzliche Bedingung erweitert die Schaltregel wie folgt:

Schaltregel für Transitionen mit Eingang: Eine Transition mit Eingang ist aktiviert, wenn

- alle ihre Prästellen markiert sind,
- alle Poststellen nicht markiert sind und
- die zugeordnete Eingangsgröße einen gültigen Wert aufweist, z.B. den Booleschen Wert $u_1 = true$.

Wenn eine Transition keine Eingangsgröße besitzt, wird in ihrer grafischen Darstellung der Pfeil weggelassen und für sie gilt die alte Schaltregel. Das entsprechende Element im Vektor u hat dann stets den Booleschen Wert $true$. Zur Beschreibung der zum Zeitpunkt k vorhandenen Markierung $M(k)$ führt man einen N-dimensionalen Vektor $p(k)$ ein, wobei N die Anzahl der Stellen im Petrinetz bezeichnet. Eine aktivierte Transition kann schalten, muss es aber nicht. Die Menge der bei der Markierung $M(k)$ aktivierten Transitionen wird mit $T_{akt}(M(k))$ bezeichnet. Enthält diese Menge mehr als eine Transition, so verhält sich das Netz nichtdeterministisch. Welche der aktivierten Transitionen schaltet, kann nicht mit Hilfe des Petrinetzes bestimmt werden. So wie nichtdeterministische Automaten in mehr als einen Nachfolgezustand übergehen können, können Petrinetze von einer Markierung in mehr als eine Nachfolgemarkierung

wechseln. Die Menge der aktivierten Transitionen ist jetzt nicht nur vom Markierungsvektor $p(k)$, sondern auch vom Eingang $u(k)$ abhängig, siehe Formel (5-12).

$$p(k+1) = p(k) + N\,t(k)\ \text{für}\ t(k) \in T_{akt}(M(k)) \quad (5\text{-}12)$$

p	Markierungsvektor
y	Ausgangssvektor
H	Ausgangsfunktion
k	Zeitpunkt
N	Anzahl der Stellen im Petrinetz
T_{akt}	Menge bei Markierung $M(k)$ aktivierter Transitionen

Zur Verkürzung der Schreibweise werden ein Eingangs- und ein Ausgangsvektor gebildet, siehe Formel (5-13) und Formel (5-14).

$$u^T = [u_1 \quad u_2 \dots] \quad (5\text{-}13)$$

u^T	Eingangsvektor (Transponiert)
u	Eingangssignal

$$y^T = [y_1 \quad y_2 \dots] \quad (5\text{-}14)$$

y^T	Ausgangssvektor (Transponiert)
y	Ausgangssignal

Jeder Transition wird eine Schaltbedingung zugeordnet, die eine Boolesche Funktion aller Eingangssignale sein kann. Beispielsweise wird in Formel (5-15) die Schaltbedingung für Transition t_1 aus Abbildung 16 dargestellt.[73]

$$SB(t_1) = u_1 \wedge u_2 \quad (5\text{-}15)$$

SB	Schaltbedingungsc
t	Transition
u	Eingangssignal

Durch die Wahl des Eingangs kann man nicht nur festlegen, ob eine Transition schalten kann, sondern auch den Zeitpunkt bestimmen, an dem die Transition schaltet. Transitionen, denen ein Schaltausdruck zugeordnet ist, werden deshalb steuerbar genannt. Dann heißt auch das durch die Transition repräsentierte Ereignis steuerbar.

Die zweite Erweiterung des Petrinetzes betrifft den Ausgang. Jeder Stelle p_i wird ein Ausgangssignal y_j zugeordnet. Stimmt der Vektor aller Ausgangsgrößen p_i mit dem Markierungsvektor p_i überein, so kann der Markierungszustand des Netzes anhand des Ausgangs vollständig erkannt werden, siehe Formel

[73] Vgl. Litz (2013), S. 254 ff.

(5-16). In der grafischen Darstellung wird der Ausgang mit einer gestrichelt eingetragenen Kante, die von der betreffenden Stelle ausgeht, gekennzeichnet, siehe Abbildung 16. Alternativ kann der Ausgang auch ohne Pfeil in Klartext neben der Stelle platziert sein, siehe Abbildung 29.[74]

$$y(k) = H(p(k)) \qquad (5\text{-}16)$$

y	Ausgangssvektorfunktion
H	Zeitpunkt
k	

5.7.4 Prozessinterpretiertes Petrinetz (PIPN)

Wie im vorherigen Kapitel erwähnt, gibt es für beide Teile der formalen Spezifikation, Spezialisierungen des IPN-Begriffs. Im Kapitel 5.7.3 wurde bereits eine Spezialisierung, das SIPN vorgestellt. Die zweite Spezialisierung bezieht sich auf die Modellierung der Prozesse. Ein häufig verfolgter Ansatz zur Modellierung von Kontrollen und Prozessen durch Petrinetze ist die Verwendung von interpretierten Netzen. Kontrollen werden durch kontrollbezogene interpretierte Petrinetze modelliert, die zu kontrollierenden Prozesse durch prozessbezogene interpretierte Petrinetze. In Bezug auf den Aufbau sind SIPN und PIPN identisch. Beide sind Netze, die durch eine Kennzeichnung von Stellen und Transitionen erweitert werden. Die Übergänge erhalten einen Schaltausdruck, der in Abhängigkeit von Eingängen einen booleschen Wert liefert. Damit kann das Schalten der Transitionen von außen beeinflusst werden. Den Stellen wird eine Ausgangsfunktion zugewiesen, sodass die Markierung des Netzes nach außen wirken kann. Im Gegensatz zu den klassischen Petrinetzen, haben die Transitionen im SIPN eine definierte Schaltzeit. Sie schalten sofort bei Aktivierung, dem sogenannten erzwungenen Schaltvorgang oder das Feuern. Oft sind die SIPN mit einer Zeitgewichtung versehen, sodass auch das Zeitverhalten der Steuerung modelliert werden kann. Mit der PIPN können verschiedene Arten von Prozessmodellen erstellt werden. Ein Prozessmodell kann den ungeregelten Prozess mit allen möglichen Verhaltensweisen abbilden. Darüber hinaus können verschiedene Störungsmodelle einbezogen werden.[75]

5.8 Zustand-Zonen Modell (ZZM)

Das Modellierungskonzept zur Erstellung eines Zustand-Zonen Modells (ZZM) ist eine neuartige Methodik des Steuerungsentwurfs, dessen Grundlage in Zeiten der Relaistechnik entstanden ist. Ziel ist es, einen Standard für die Steuerung und eine Diagnose des schrittweisen Betriebs, von komplexen Maschinenanlagen zu etablieren. Die Spezifikation beinhaltet die Anbindung an das Human Maschine Interface (HMI), wo Diagnoseinformationen angezeigt werden können. Wurden damals Relaissteuerungen zur Lösung einer Steuervorgabe eingesetzt, sind oftmals Informationen zum Maschinenstatus anhand einer zweidimensionalen Lampen-Matrix signalisiert worden. Trat ein Fehlerzustand ein, so leuchtete diese Lampe und auf einem Display erschien in Klartext die entsprechende Fehlermeldung. Dieses Konzept

[74] Vgl. Lunze (2020), S.400 ff.

[75] Vgl. Wurmus (2002), S. 24

wurde für eine lange Zeit als Standard in der Automobilindustrie eingesetzt. Durch die optimierte Ausnutzung des verfügbaren Programmspeichers auch sehr erfolgreich. Die Entwicklung der SPS und das Anwachsen des zur Verfügung stehenden Programmspeicherplatzes, verdrängte das recht aufwändig zu entwerfende Steuerungskonzept. Um eine Steuerung zu entwerfen, überstieg mit den Mittel der damaligen Zeit der Aufwand den Nutzen um ein Vielfaches. Durch moderne Mittel der Automatisierungstechnik ist es gelungen, diese durchdachte Methodik des Steuerungsentwurfs wieder konkurrenzfähig einzusetzen. In dieser Arbeit wird dieses neu gedachte Konzept vorgestellt und Funktionsweise, Bestandteile sowie Vor- und Nachteile erläutert.

5.8.1 Philosophie des Steuerungssystems

Das ZZM ist grundlegend als Ablauf- bzw. Schrittsteuerung realisiert. Dabei wird der Grundsatz befolgt, dass jede Anlage oder Maschine in verschiedene Bereiche einteilbar ist, siehe Abbildung 17.

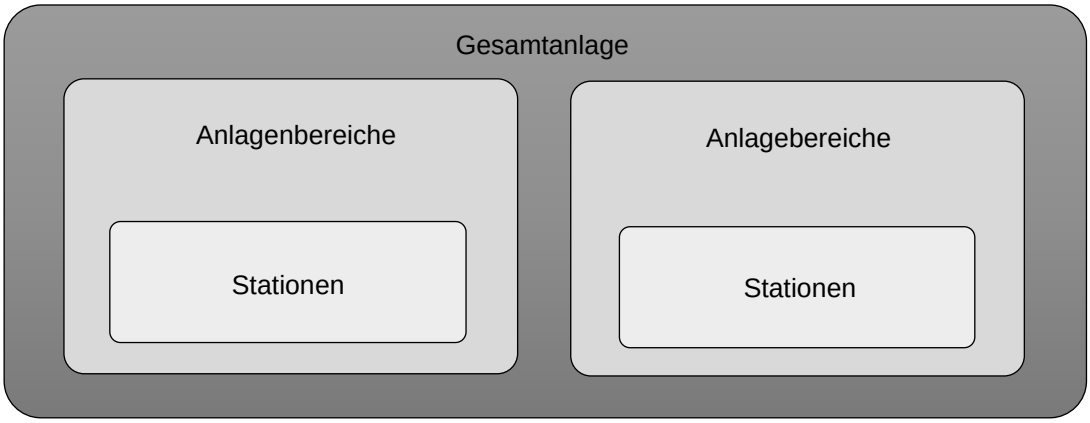


Abbildung 17: Anlagebereiche, Quelle: Eigene Darstellung

Die Maschine oder Anlage wird vollumfänglich als Gesamtanlage betrachtet. Diese kann in Anlagenbereiche unterteilt werden, die wiederum Stationen beinhalten können. Jede Station kann durch eine oder mehrere Ablauf- oder Schrittsteuerungen beschrieben werden. Daraus resultieren Signale und Zustände, die je nachdem auf einen oder mehrere Bereiche der Gesamtanlage wirken können. Beispielhaft kann ein Not-Aus Signal global über die Gesamtanlage wirken, aber Schutztürüberwachungen nur lokal auf die betreffende Station. Diese Unterteilung ist hilfreich, um komplexe Anlagen in überschaubare Bereiche einzuteilen. Die kleinste Einheit einer Anlage ist die Station, mit den dazugehörigen Ablaufsteuerungen. Weiter unterteilt befinden sich noch verschiedene Funktionen und ein Regelwerk, welche durch Schrittketten gesteuert sind, siehe Kapitel 7.3.4.1.

5.8.1.1 Anzeige- und Bedieneinrichtung

Die Schnittstelle zum Menschen ist das Human Machine Interface (HMI), welches zur Steuerung und Diagnose der Anlage verwendet wird, siehe Abbildung 18. Die Aufgabe des HMI ist es, den*die Bediener*in darüber zu informieren welcher Schritt der Schrittkette aktiv ist, in welchem Zustand sich die Anlage befindet und welche Bedingung (Eingang fehlt oder ist fehlerhaft) erfüllt sein muss, um in den Folgezustand zu wechseln. Weiters ist eine Umschaltung der Betriebsarten vorgesehen, der sogenannte Sequence Automatic Release. Um die Anlage im Handbetrieb zu bedienen, muss Single Step On aktiviert werden.

Um Schritte zu inkrementieren, wird Step Increment angewählt. Step Single startet die Schrittkette, Step Reset springt wieder and dessen Anfang. Alle weiteren Elemente dienen als Einstellungs- bzw. Parametriermöglichkeiten für den Automatik- bzw. Handbetrieb.[76]

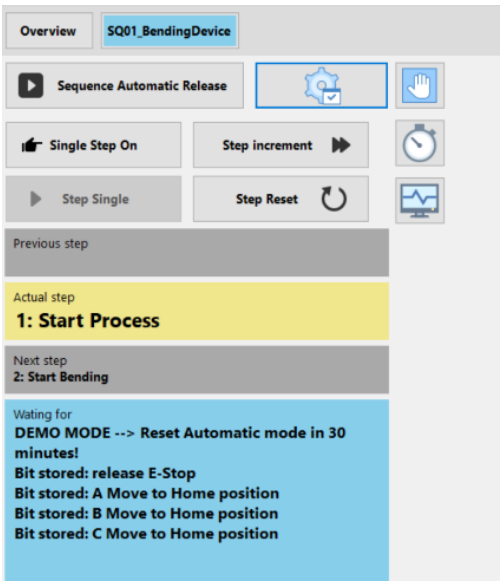


Abbildung 18: Human Maschine Interface, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

5.8.1.2 Betriebsarten

Die in Abbildung 17 dargestellten Anlagenbereiche können jeweils eigenen Betriebsarten zugewiesen werden. Die Betriebsarten können von Station zu Station variieren und beziehen sich auf einzelne Stationen, sowie Anlagenbereiche oder auf die Gesamtanlage. Man unterschiedet zwischen Hand- und Automatikbetrieb. Im Handbetrieb können alle Aktuatoren eines Systems gesteuert werden, in Abhängigkeit der manuellen Verriegelungen des Steuersystems. Die Anlagenautomatik der Gesamtanlage kann alle Aktuatoren automatisch nach entsprechendem Ablauf steuern, wenn sich alle untergeordneten Stationen im Automatikbetrieb befinden, der richtige Schritt passend zum Zustand der Station aktiv ist und keine Fehlermeldungen an der Gesamtanlage anliegen.

5.8.2 Struktur- und Modellaufbau

Das ZZM besteht aus sechs verschiedenen Ebenen, die miteinander in bestimmten Beziehungen stehen und verschiedene Aufgaben erfüllen.

[76] Gruber (2022), Online-Quelle [22.05.2022]

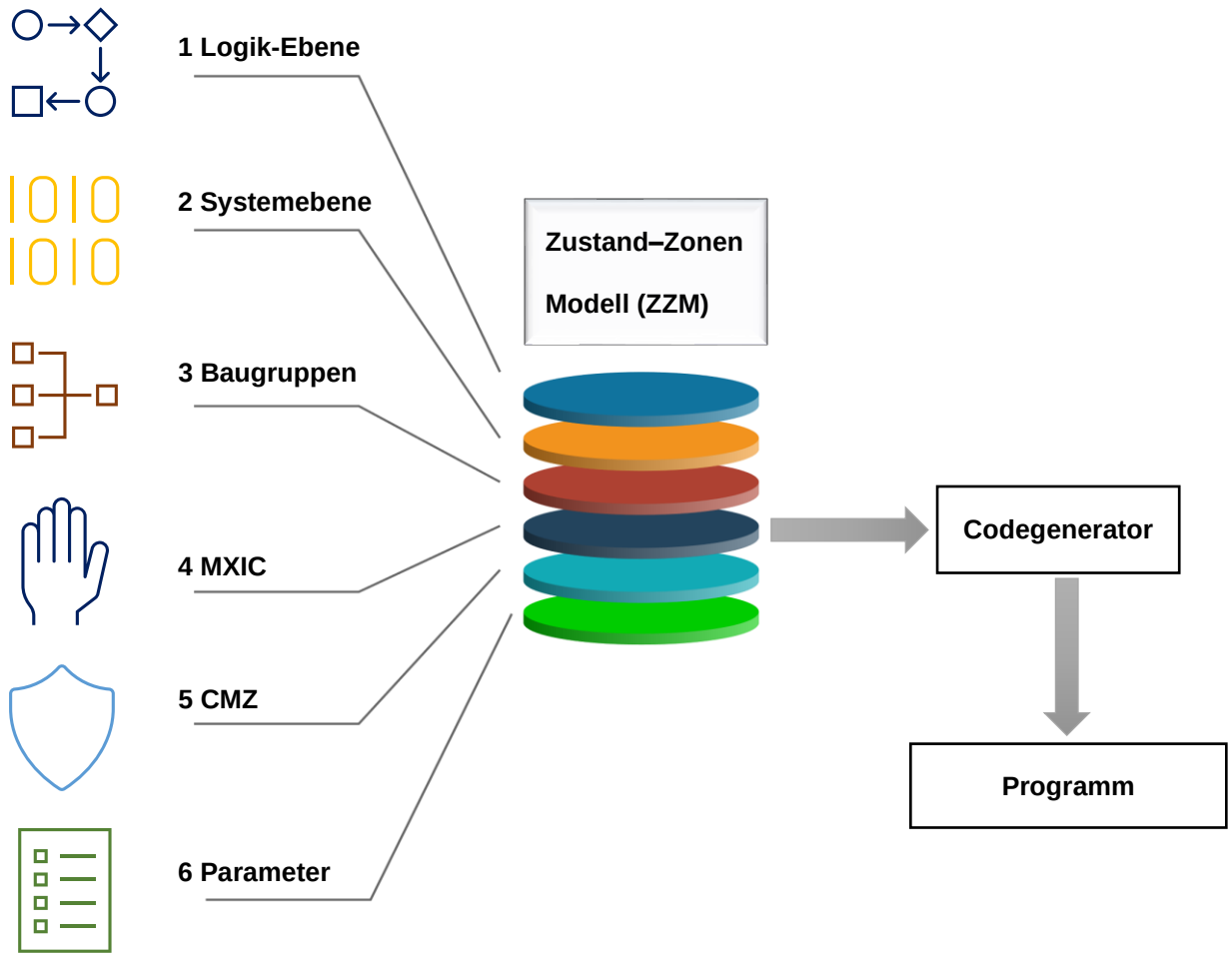


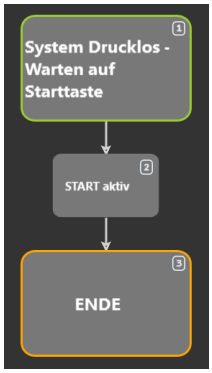
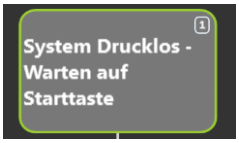
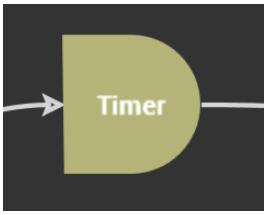
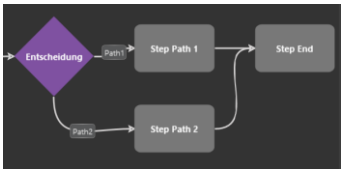
Abbildung 19: Modellierung des Zustand-Zonen Modells, Quelle: Eigene Darstellung

In Abbildung 19 wird der Aufbau sowie die weitere Verwendung des Zustand-Zonen Modells schematisch dargestellt. Die Modellbildung erfolgt ausgehend der informellen Prozessbeschreibung der Steuervorgabe und bildet das Schichtenmodell anhand der folgenden sechs Ebenen:

- **Logikebene:** Beschreibt die ereignisdiskrete Schrittabfolge des gewünschten Prozessablaufs. Durch logische Elemente werden die Beziehungen der einzelnen Schritte zueinander beschreiben. In Abschnitt 5.8.3 werden Details und Zusammenhänge der Logikebene vorgestellt. Die logischen Elemente dieser Ebene werden in Tabelle 2 dargestellt.
- **Systemebene:** In dieser Ebene wird die Zustand-Zonen Matrix definiert. Zu jedem Schritt des Prozessablaufs wird für jede Zone ein Zustand definiert, siehe Tabelle 5. Eine detaillierte Beschreibung der Systemebene wird in Kapitel 5.8.3 vorgenommen.
- **Baugruppen:** Die Baugruppen stellen eine Bibliothek vorgefertigter Funktionsbausteine zur Verfügung. Dadurch werden bereits vorhandene Lösungswege zur Funktionsbeschreibung wiederverwendet. Es ist möglich spezielle Baugruppen mit definierten Funktionalitäten als Vorlage zu erstellen.
- **Manual Cross Interlock Checks** (**MXIC**) sind manuelle Kreuzverriegelungen, die ermöglichen im Handbetrieb bestimmte Bewegungen zu sperren. Korrelieren zwei Bewegungen miteinander, so

kann verhindert werden, dass diese Bewegungen ausgeführt werden. Dieser Vorgang erhöht die Betriebssicherheit sowie die Bedienbarkeit der Anlage im Betrieb.

- Die **Constantly monitored Zone** (**CMZ**) sind Zonen, die unabhängig von der Schrittkette ständig überwacht werden. Damit können sicherheitskritische Zonen überwacht und bei einer Störung entsprechende Handlungen durchgeführt werden, z.B. die Anlage außer Betrieb setzen.
- **Parameter:** Um den Prozessablauf zu modellieren, sind Prozessparameter notwendig. Diese Ebene definiert in welcher Form bestimmte Kenngrößen im Modell notwendig sind.

Logische Elemente	Symbol	Beschreibung
Schritt		Ein Schritt der Schrittkette symbolisiert einen Systemzustand im logischen Prozessablauf. Der erste Schritt der Schrittkette ist grün umrandet dargestellt. Der letzte Schritt in der Schrittkette wird Orange umrandet dargestellt. In jeder Schrittkette existiert nur ein Anfangs- bzw. Endschritt. Schritte können beliebig eingefärbt werden, um z.B. logische Zusammenhänge zu symbolisieren (Motor ein = grün, Motor aus = rot). Die Farbe sowie auch die Bezeichnung haben jedoch keinen Einfluss auf dessen Funktionalität.
ID		Die Position des Schrittes wird oben rechts innerhalb eines Schritts angezeigt und als ID bezeichnet.
Pfeil		Mithilfe von Pfeilen werden die Elemente logisch miteinander verknüpft. Diese Pfeile können mit einem Zusatz versehen werden, um die Art der logischen Verknüpfung zu symbolisieren z. B. Loop, Path, Jump.
Timer		Mit dem Timer können zeitliche Verzögerungen zwischen den einzelnen Schritten realisiert werden. Durch Parameter kann die zeitliche Dauer der Verzögerung definiert werden.
Entscheidung		Die Entscheidung kann durch ein Entscheidungskriterium (z.B. der Zustand einer Zone oder einen Parameter) den weiteren Verlauf zwischen zwei Pfaden wählen.

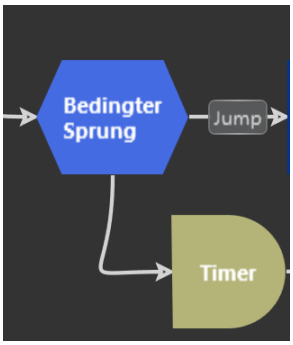
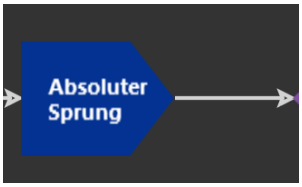
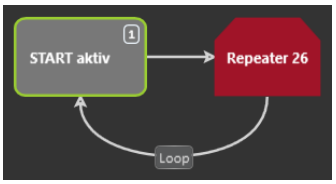
Sprung	 	Ein Sprung dient dazu, an einen bestimmten Schritt der Schrittkette zu springen. Es gibt zwei Arten von Sprüngen, die sich in Ihrer Form und Funktion unterscheiden. Der **Bedingte Sprung** entscheidet nach einem Entscheidungskriterium (z.B. der Zustand einer Zone oder ein Parameter), ob ein Sprung durchgeführt werden soll. Fällt diese negativ aus, wird kein Sprung durchgeführt. Fällt diese jedoch positiv aus, wird an diese bestimmte Stelle in der Schrittkette gesprungen. Symbolisiert wird dies anhand der Pfeilüberschrift Jump. Der **Absolute Sprung** springt ohne Entscheidung an die gewünschte Stelle in der Schrittkette.
Schleife		Eine Schleife wird verwendet, wenn ein bestimmter Teil einer Schrittkette wiederholt werden soll. Ein Parameter entscheidet dabei, wie oft dies erfolgen soll. Symbolisiert wird die zu wiederholende Schrittkette durch die Pfeilüberschrift Loop.

Tabelle 2: Elemente der Logikebene, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Zusammengefasst ergeben diese sechs Ebenen den Aufbau des ZZM. Diese Ebenen sind miteinander durch Beziehungen verbunden. Beispielsweise wird der Prozessablauf in der Logikebene definiert und der entsprechende Systemzustand in der Systemebene zugeordnet. Um alle Beziehungen und Vorgänge vollständig zu beschreiben werden umfassendere Beschreibungen benötigt. Deshalb beschränkt sich diese Arbeit auf die Betrachtung der Modellbildung, nicht dessen Programmerstellung oder Implementierung in das Zielsystem.

5.8.3 Funktionsprinzip des Zustand-Zonen Modells

Das Kernstück des ZZM bildet die Zustand-Zonen Matrix, die jene Schrittkette (oder auch Sequenz genannt) der Anlage oder Maschine abbildet, die durchlaufen werden soll. Die ZZM bildet dabei zu jedem Schritt der Schrittkette, einen definierten Zustand ab, siehe Formel (5-17).

$$z_s = [\mathrm{sz}_{\mathrm{s1}} \quad \dots \quad \mathrm{sz}_{\mathrm{sm}}] \tag{5-17}$$

z_s	Zustand im jeweiligen Schritt s
$\mathrm{sz}_{\mathrm{sm}}$	Operand der Sequenzzone mit Index m im jeweiligen Schritt s
m	Index der Sequenzzonen
s	Schritt

Ein Zustand bildet sich durch Operanden der Sequenzzonen mit Index 1 bis m im jeweiligen Schritt s.

Definition 5-8: Eine **Sequenzzone** oder auch einfach als **Zone** bezeichnet**,** ist ein logisches Teilsystem, das zur **Steuerung und Überwachung von Signalen** eingesetzt wird.

So werden Zustände für jeden Schritt der Schrittkette definiert. Die daraus entstehende Matrix wird Zustand-Zonen Matrix ZZM genannt, siehe Formel (5-18).

$$ZZM = \begin{bmatrix} z_1 \\ \cdots \\ z_s \end{bmatrix} \qquad (5\text{-}18)$$

ZZM Zustand-Zonen Matrix

z_s Zustand im jeweiligen Schritt s

s Schritt

Ein Zustand wird über die Operanden der jeweiligen Sequenzzonen gebildet. Dabei können die Operanden der Zahlenwerte annehmen, siehe Tabelle 3.

2	Sequence-Checks
1	Interlock-Checks
0	Don‘t care

Tabelle 3: Zahlenwerte der Operanden der Sequenzzone, Quelle: Eigene Darstellung

Die Operanden der Sequenzzonen können ebenso spezielle Werte annehmen, siehe Tabelle 4.

D1, D2	Zuweisung der Entscheidungspfade
J	Sprungbedingung

Tabelle 4: Spezielle Werte der Operanden der Sequenzzone, Quelle: Eigene Darstellung

Definition 5-9: Der **Sequence-Check** überwacht den **Zustandswechsel** von Sequenzzonen und stellt die Grundlage zur Steuerung und Überwachung des automatischen Betriebs eines Steuerungssystems dar. Der Sequence-Check wird durch der Zahl **2** symbolisiert.

Der Sequence-Check ist eine Funktion des Steuerns und Überwachens. Sprich der Sequence-Check ist eine Funktion, die einen überwachten Zustandswechsel der entsprechend verknüpften Sequenzzone realisiert. Damit soll sichergestellt werden, dass die Aktion durchgeführt und abgeschlossen worden ist. Erfolgte der Sequence-Check mit positivem Ergebnis, so können nacheinander weitere Sequenzzonen desselben Schrittes überprüft werden. Niemals werden zwei Zonen eines Schrittes zur selben Zeit überprüft. Ist das System nicht wie gefordert im modellierten Zustand, so wird der Interlock-Check aktiv. Dies ist eine Verriegelungskontrolle, die prüft, ob sich das System im aktuellen Schritt der Schrittkette im richtigen Zustand befindet.

Definition 5-10: Der **Interlock-Check** prüft den **Zustand** einer Sequenzzone und ist ein grundlegendes Konzept zur Überwachung und Steuerung von Bewegungen sowie zur Festlegung einer Schrittkette. Der Interlock-Check wird durch die Zahl **1** symbolisiert.

Der Interlock-Check dient zur Überwachung und Steuerung von Bewegungen, damit diese in einer vorgedachten richtigen Reihenfolge ablaufen. Weiters ermöglicht dieser eine einfache Detektion einer Zustandsfolge im Systemablauf. Um eine korrekte Diagnose zu gewährleisten, ist es wichtig, dass jeder Systemzustand zum jeweiligen Schritt der Schrittkette überwacht wird.

Definition 5-11: Ist es im Systemablauf **nicht relevant eine Zone zu überwachen oder zu steuern**, so wird diese egalisiert. Dies wird mit der Zahl **0** ausgedrückt und bedeutet, dass in diesem Schritt des Ablaufs, diese Zone für das Systemverhalten ignoriert wird.

5.8.4 Die Signalumgebung des Zustand-Zonen Modells

Die in Abbildung 6 dargestellte Struktur eines Steuerkreises gilt auch beim Zustand-Zonen Modell als schematische Beschreibung des Steuerkreises. Um zu verstehen, wie die Signalumgebung mit dem Steueralgorithmus ineinandergreift, ist der Aufbau der Struktur des Steueralgorithmus in Abbildung 20 dargestellt.

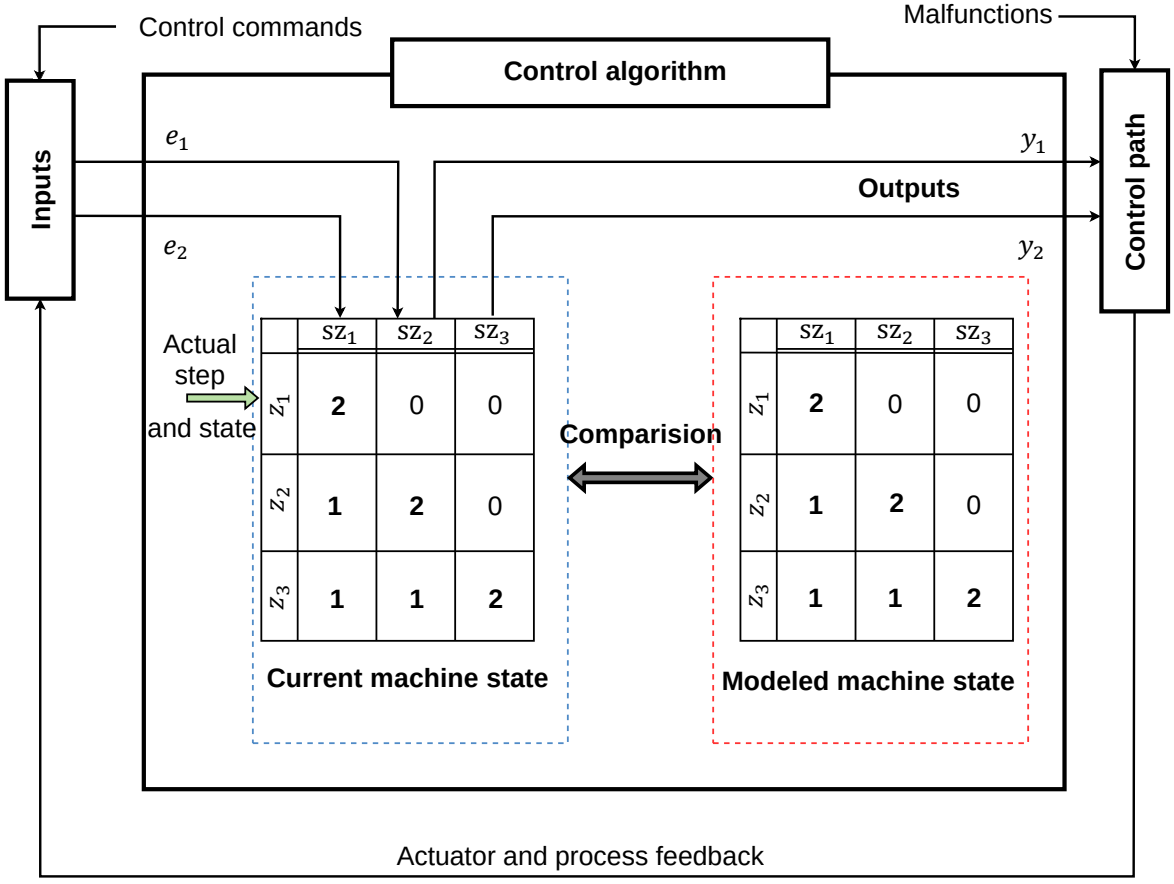


Abbildung 20: Struktur des Steueralgorithmus ausgehend von einem Zustand-Zonen Modell, Quelle: Eigene Darstellung

Die Abbildung 20 besteht aus den Eingängen, dem Steueralgorithmus und der Steuerstrecke. Die Eingänge e_1 und e_2 werden aus den Steuerbefehlen und den Aktuator- bzw. Prozessrückmeldungen der Steuerstrecke gebildet. Diese Eingänge werden mit den jeweiligen Zonen z_1 und z_2 des Steueralgorithmus verbunden. Je nachdem ob Eingänge und- oder Ausgänge mit den Zonen verbunden werden, spricht man von Eingangszonen, wie beispielsweise z_1. Man spricht von einer gemischten Eingangs-Ausgangszone, wenn Eingänge und Ausgänge mit einer Zone verbunden sind, wie z.B. z_2 und von reinen Ausgangszonen, wenn nur Ausgänge mit der Zone verbunden sind, wie z.B. z_3.

Es fehlt noch die Betrachtung, wie Ausgänge über den Steueralgorithmus hin zur Steuerstrecke erzeugt werden. Nachdem die Eingänge erfolgt sind und der Steueralgorithmus den positiven Vergleich der Prozessabbildungen verzeichnet, werden entsprechende Ausgänge erzeugt. Diese Ausgänge, wie z.B. y_2 aus der Abbildung 20, wirken zusammen mit unvorhersehbaren Störungen in das Verhalten der Steuerstrecke ein. Durch Aktuator- bzw. Prozessrückmeldungen werden etwaige Abweichungen der Steuerstrecke ständig überwacht.

Aus den Steuerbefehlen und Aktuator- bzw. Prozessrückmeldungen bildet sich das aktuelle Prozessabbild im Steueralgorithmus im jeweiligen Schritt ab. Das Gegenstück bildet das erwünschte, modellierte Prozessabbild im Steueralgorithmus. Ein Vergleich zwischen dem aktuellen und dem modellierten Prozessabbild gibt Auskunft darüber, ob und welche Zustände eingetroffen sind. Ist dieser Vergleich ohne Abweichungen bestanden, spricht also nichts gegen einen Zustandswechsel, wird der Schrittzähler inkrementiert und der Steueralgorithmus wechselt in den folgenden Zustand. Zusammenfassend wird dieses Verhalten in **Definition 5-12** beschrieben.

Definition 5-12: Ein **ZZM** wird modelliert, um daraus ein SPS-Programm zu generieren. Dieses SPS-Programm ist in der Lage, jeden Signalzustand zu jedem Schritt der Schrittkette zu überprüfen, ob und wie sich das Prozessabbild des Modells von dem Prozessabbild der Realität unterscheidet. Diese Betrachtung stellt einen **Vergleich** zwischen dem **aktuellen** und dem erwünschten**, modellierten Prozessabbild** im Steuerkreis dar. Dieser Vergleich erleichtert die Diagnosefähigkeit bei Abweichungen, sichert die Überwachung der Systemzustände und steuert das System fehlerfrei.

5.8.5 Zonentypen

Im ZZM können drei Zonentypen, die Eingangszone, Eingangs/Ausgangszone sowie Ausgangszonen, verwendet werden. Diese Zonentypen unterscheiden sich nicht nur durch die Verknüpfung mit Ein- oder Ausgängen, sondern auch durch die Aktionen, die über Operanden ausgelöst werden können. Die Operanden können in einer Zone einen Sequence- oder Interlock-Check, sowie Aktionen durch spezielle Operanden hervorrufen. Die Aktion dieser speziellen Operanden wird durch die praktische Umsetzung in Kapitel 7.3.4.2 beschrieben, da sich spezielle Operanden auf mehrere Ebenen des ZZM auswirken die erst im Fortschreiten der Arbeit beschrieben werden. Nachfolgend werden die Auswirkungen des Sequence- bzw. Interlock-Checks, sowie des Don't cares bezogen auf die drei Zonentypen sowie deren Eigenschaften, durch ein Beispiel erläutert, siehe Tabelle 5.

Voraussetzung für dieses Beispiel ist, dass in der Grundstellung der Schalter im Zustand Aus steht, das Ventil geschlossen ist und die Lampe ausgeschalten ist.

Schritt bzw. Zustand	**Schalter** Eingangszone	**Ventil** Eingangs/Ausgangszone	**Lampe** Ausgangszone	**Beschreibung**
	$\mathbf{sz_1}$	$\mathbf{sz_2}$	$\mathbf{sz_3}$	
z_1	**2**	0	0	**Schalter betätigen** Sequence-Check, Zustandswechsel der Eingangszone $\mathbf{sz_1}$
z_2	**1**	**2**	0	**Schalter betätigt, Ventil öffnen** Interlock-Check, Eingangszone $\mathbf{sz_1}$ aktiv Sequence-Check, Zustandswechsel der Eingangs/Ausgangszone $\mathbf{sz_2}$
z_3	**1**	**1**	**2**	**Schalter betätigt, Ventil offen, Lampe betätigt** Interlock-Check, Eingangszone $\mathbf{sz_1}$und Eingangs/Ausgangszone $\mathbf{sz_2}$ Sequence-Check, Zustandswechsel der Ausgangszone $\mathbf{sz_3}$

Tabelle 5: Beispiel Zustand-Zonen Matrix, Quelle: Eigene Darstellung

Die Eingangszone kann verwendet werden, um Informationen von z.B. Sensoren zu verarbeiten. Ist in einem Schritt der Sequence-Check in einer Eingangszone gesetzt, so wird ein Zustandswechsel dieser Zone in diesem Schritt erwartet. Ein Interlock-Check einer Eingangszone stellt sicher, dass der erwartete Zustand dieser Zone in diesem Schritt erhalten bleibt. In der Praxis kann z.B. ein Schalter durch eine Eingangszone überwacht werden. Wird ein Schaltvorgang gefordert, muss das Signal des Schalters mit der Eingangszone verknüpft werden und im entsprechenden Schritt ein Sequence-Check gesetzt sein, siehe sz_1 im Schritt 1 aus Tabelle 5. Erfolgt die entsprechende Aktion (Schalter wird betätigt) ist der

Sequence-Check erfolgreich. Wird ein Interlock-Check in einem Schritt gesetzt, ist gefordert, dass der Schalter betätigt ist, siehe sz_1 im Schritt 2 aus Tabelle 5. Wird ein don't care über den Wert 0 des Operanden in einer Eingangszone gesetzt, werden Informationen, die vom verknüpften Eingang erfolgen ignoriert und sind damit für das Modell nicht von Relevanz.

Ein weiterer Zonentyp stellt die Ausgangszone dar. Diese kann über den Sequence-Check die mit der Ausgangszone verknüpften Ausgänge steuern. Beispielsweise kann eine Lampe eingeschalten werden, siehe sz_3 im Schritt 3 aus Tabelle 5. Ein Interlock-Check ist in einer Ausgangszone nicht möglich, da es keine Möglichkeit der Überwachung gibt. Wird ein don't care über den Wert 0 des Operanden in einer Ausgangszone gesetzt, wird der verknüpfte Ausgang.

Die Eingangs/Ausgangszone ist als ein Ausgang mit entsprechendem Feedback über einen Eingang definiert. Dadurch können Ausgänge so lange angesteuert werden, bis ein entsprechendes Feedback über den Eingang rückgemeldet wird. Beispielsweise kann ein Ventil mit Positionserkennung dadurch angesteuert werden. Die Positionserkennung des Ventils ist so ausgeführt, dass die Position offen oder geschlossen erkannt werden kann. Um die Aktion (Ventil öffnen) zu starten, wird der Sequence-Check gesetzt, siehe sz_2 im Schritt 2 aus Tabelle 5. Nun wird der mit der Eingangs/Ausgangszone verknüpfte Ausgang des Ventils so lange angesteuert, bis die Zielposition des Ventils erreicht wird und das Feedback über den verknüpften Eingang der Positionserkennung erfolgt. Über den Interlock-Check kann überprüft werden, ob die mit der Eingangs/Ausgangszone verknüpften Eingänge im Zustand bleiben, siehe sz_3 im Schritt 3 aus Tabelle 5. Wird ein don't care über den Wert 0 des Operanden in einer Eingangs/Ausgangszone gesetzt, werden Informationen des verknüpften Eingangs ignoriert und der verknüpfte Ausgang nicht verwendet.

5.8.6 Zustandsüberwachung und Schrittweiterschaltung

In die Methodik des ZZM ist eine Einrichtung zur Überwachung der Systemzustände in jedem Schritt des Prozessablaufs integriert. In jedem Schritt wird der Zustand aufgrund der gesetzten Operanden der Zonen überprüft, wobei weitergeschalten wird, wenn nichts dagegenspricht. Dies bedeutet, dass im aktuellen Schritt alle Sequence- und Interlock-Checks erfolgreich sein müssen, um in den nächsten Schritt weiterzuschalten. Ist ein Interlock-Check in einem Schritt nicht erfolgreich, wird der Schritt angehalten, der Automatikbetrieb deaktiviert und dem*der Bediener*in eine Fehlermeldung auf dem HMI angezeigt. Ein Sequence-Check hingegen hält die Schrittweiterschaltung so lange an, bis dieser erfolgreich durchgeführt wird. Die Bedingungen für die Weiterschaltung werden den*dem Benutzer*in Klartext auf dem HMI angezeigt. Dadurch lässt sich einfach erkennen, welcher Schritt aktiv ist. Wird der Operand mit dem Wert 0 (don‘t care) gesetzt, ist es möglich bestimmte Zonen nicht zu überwachen.

Definition 5-13: Über Aktuator- bzw. Prozessrückmeldungen und Stellsignale zeichnet sich das aktuelle Prozessabbild ab. Ist der Vergleich zwischen dem aktuellen- und dem erwünschten Prozessabbild nicht korrekt, so wird der Schritt angehalten und wechselt nicht in den darauffolgenden Schritt. Dieses Verhalten zeigt, dass nur definierte Zustände im Prozessablauf eine Transition hervorrufen können.

5.8.7 Die Fehlermatrix des Zustand-Zonen Modells

Im Mittelpunkt des Mechanismus zur Erfassung und Anzeige von Fehler-/Diagnoseinformationen, steht die Fehlermatrix M_f. Diese Matrix stellt das Kernstück der Diagnose bzw. Überwachungs-Fähigkeit des ZZM dar und wird in verschiedene Bereiche eingeteilt, siehe Formel (5-19).

$$M_f = \begin{bmatrix} Cmzf_{11} & \cdots & Cmzf_{1m} \\ \vdots & \ddots & \vdots \\ Cmzf_{s1} & \cdots & Cmzf_{sm} \\ Seqf_{11} & \cdots & Seqf_{1m} \\ \vdots & \ddots & \vdots \\ Seqf_{s1} & \cdots & Seqf_{sm} \\ \mathrm{sz}_{11} & \cdots & \mathrm{sz}_{1m} \\ \vdots & \ddots & \vdots \\ \mathrm{sz}_{\mathrm{s1}} & \cdots & \mathrm{sz}_{\mathrm{sm}} \end{bmatrix} \quad (5\text{-}19)$$

M_f	Fehlermatrix
$Cmzf$	Constantly Monitored Zone-Fehler
$Seqf$	Sequenzfehler
s	Schritt
m	Index der Sequenzzonen
$\mathrm{sz}_{\mathrm{sm}}$	Operand der Sequenzzone mit Index m im jeweiligen Schritt s

Die Matrix M_f wird in verschiedene Bereiche eingeteilt. Der Zeilen-Index s symbolisiert einen Schritt in der Schrittkette und der Spalten-Index m symbolisiert eine Zone des Systems. Dadurch können Signale und Zustände schrittweise sowie auch schrittunabhängig überwacht werden. Dieser Vorgang ist ein zentrales Ziel der Schrittsteuerung. Im oberen Bereich der Matrix sind die Constantly Monitored Zone-Fehler $Cmzf$ angeordnet. Diese Zonen werden ständig, unabhängig von der Schrittkette, überwacht. Im Unterschied zu den Sequenzfehlern $Seqf$, diese können nur für die schrittbezogene Überwachung eingesetzt werden. Der unterste Bereich der Matrix definiert eindeutige Startbedingungen, die durch die Operanden der Sequenzzonen sz_{11} bis $\mathrm{sz}_{\mathrm{sm}}$ abgebildet werden.

5.8.8 Einsatzzweck und Vorteile des Zustand-Zonen Modells

In der Diagnose wird versucht, ein Fehlverhalten innerhalb einer definierten Zeitspanne zu erkennen und entsprechend darauf zu reagieren. Wesentlich ist dabei die Funktion, einen Fehler oder ein Fehlverhalten eines Systems, als solches auch eindeutig zu detektieren. Verbotene Zustände sowie verbotene Zustandsübergänge können nur als solche erkannt werden, wenn diese vorab im Steuerungsentwurf berücksichtigt sind, siehe Kapitel 4.2. Da der Anstieg der Komplexität eines Steuersystems mit der Anzahl an Eingangssignalen exponentiell steigt, kann folgende Definition formuliert werden.

Definition 5-14: Mit steigender **Komplexität der Steuervorgabe** steigt der Aufwand exponentiell an, ein Steuerungssystem zu realisieren, dass die Steuervorgabe fehlerfrei beschreibt. Der in der Praxis am häufigsten verwendete Weg zur Umsetzung einer Steuervorgabe, ist auf direktem Weg von der informellen Spezifikation zur Realisierung, siehe Kapitel 4.3. Dieser Widerspruch führt zwangsläufig dazu, dass durch die direkte Umsetzung einer Steuervorgabe **verbotene Zustände** sowie verbotene **Zustandsübergänge** enthalten sein werden. Um alle möglichen Systemzustände zu berücksichtigen, muss man dem exponentiellen Anstieg der Komplexität in einem Steuersystem entgegenwirken.

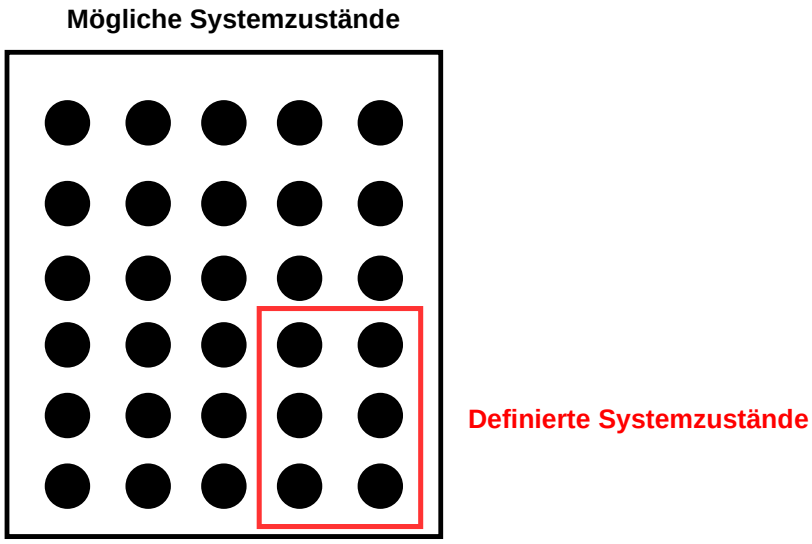


Abbildung 21: Schematische Darstellung der Systemzustände, Quelle: Eigene Darstellung

Weiters wird in der Abbildung 21 der Anstieg der Komplexität eines ZZM anders definiert als in anderen Modellierungsverfahren. Folgende Definition kann daraus abgeleitet werden, siehe Definition 5-15.

Definition 5-15: Das ZZM muss nicht alle möglichen Systemzustände abbilden, sondern nur die definierten Zustände, die das System annehmen soll.

6 SOFTWAREQUALITÄT

Der Begriff der Softwarequalität spielt sowohl für die Entwicklung als auch für die Akzeptanz einer Software eine wichtige Rolle. Zur Bewertung der Softwarequalität werden verschiedene Merkmale bewertet und herangezogen. Dieses Kapitel widmet sich den Merkmalen von Softwarequalität, der Einführung von Metriken zur Messung von Qualitätseigenschaften sowie einem Bewertungsschema von Softwarequalität.

Um zu verstehen, wie Softwarequalität entsteht, wird zuallererst die Frage geklärt, wie die Begrifflichkeiten rund um das Thema Software definiert sind. Zentral ist der Begriff des Programms.

Definition 6-1: Ein **Programm** ist ein Satz von Anweisungen, die in einer Programmiersprache geschrieben sind, um eine bestimmte Aufgabe oder eine bestimmte Funktion auszuführen.

Programme ermöglichen die Nutzung von Hardwaresystemen für unterschiedlichste Aufgaben. Wenn eine Aufgabe durch Programmierung gelöst ist, kann dieses Programm immer wieder in ein geeignetes Hardwaresystem eingesetzt werden.

Definition 6-2: Software ist ein Oberbegriff für ausführbare Programme und die dazu gehörigen Daten. Mit ihrer Hilfe ist ein softwaregesteuertes System in der Lage, Aufgaben zu erfüllen. Software wird oft mit den Begriffen Anwendung oder Programm beschrieben.

Software besteht aus Bündeln von Programmen und Datendateien. Programme einer bestimmten Software verwenden diese Datendateien, um eine bestimmte Art von Aufgaben auszuführen. Software gilt als der entscheidende Faktor zur Automatisierung.[77]

6.1 Bedeutung von Softwarequalität

Unter der Qualität einer Software versteht man ein Maß für die Güte oder Beschaffenheit der Software. Um eine Bewertung zu ermöglichen, müssen wichtige Bereiche oder Merkmale definiert werden. Die Definition der Softwarequalität kann anhand verschiedener Merkmale erfolgen. Die wohl gängigste und in den meisten Bereichen geläufigste Definition findet sich in der Norm ISO/IEC 25010.[78]

Definition 6-3: Unter dem Begriff der Softwarequalität versteht man die gesamten Merkmale und Eigenschaften eines Softwareprodukts. Diese Merkmale beziehen sich auf deren Eignung und müssen im Vorfeld definierte Aufgaben erfüllen.[79]

[77] Vgl. Manfred Broy (2021), S. 16 ff.

[78] Vgl. Georg Frey (1999), S. 108 ff.

[79] Vgl. ISO/IEC 25010 (2011)

Programme sind in einer Form beschrieben, die zur Ausführung auf einer Maschine bestimmt ist. Ein Programm besteht aus maschinenlesbarem Code, der für den Menschen kaum nachvollziehbar ist. Für das Verständnis von Softwaresysteme, sind daher strukturierte Darstellungsformen notwendig. Dabei geht es um die Frage, wie zuverlässig und korrekt ein Programm seine Funktionen ausführt. Werden Softwaresysteme in kritischen Anwendungen eingesetzt, muss deren Funktionalität gewährleistet sein, ohne eine Gefahr für den Menschen und anderen Systemen darzustellen. Es ist deutlich erkennbar, dass eine Vielzahl von unterschiedlichen Eigenschaften und Qualitätsmerkmale von Softwaresystemen von Interesse sind und dass spezielle Strukturen notwendig sind, um diese Eigenschaften angemessen zu beschreiben.[80]

6.2 Wichtige Merkmale der Softwarequalität

Die Vorgehensweise des Steuerungsentwurfs hat viele Ähnlichkeiten mit einem allgemeinen Softwareentwurf. Für den Softwareentwurf gelten eine Reihe von Einschränkungen: Der*Die Endanwender*in, der*die die Software in Auftrag gibt, definiert durch die informelle Spezifikation, was getan werden soll. Der*Die Endanwender*in ist nur in seltensten Fällen auch der*die Entwickler*in der Software und ist in der Regel nicht mit allen notwendigen technischen Randbedingungen vertraut. Dies erhöht die Wahrscheinlichkeit, dass Anpassungen in der Entwicklung notwendig werden, um das Ziel zu erreichen. Mit allen damit verbundenen Risiken und Aufwänden, die durch dieses Vorgehen begünstigt werden.[81]

Für Steuerungssoftware gibt es verschiedene Untersuchungen und Definitionen, was unter ihrer Qualität zu verstehen ist. Besonders umfangreich ist dieses Thema in der internationalen Norm ISO/IEC 25010 definiert. Diese Norm definiert die wichtigsten acht Qualitätsmerkmale, die im Softwareentwicklungsprozess zu berücksichtigen sind, siehe Abbildung 22. Zweck und Ziel dieser Norm ist es, die Qualität in der Softwareentwicklung und im Betrieb von Softwaresystemen zu sichern.[82]

[80] Vgl. Manfred Broy (2021), S. 41

[81] Vgl. Litz (2013), S. 329-330

[82] Vgl. ISO/IEC 25010:2011 (2011)

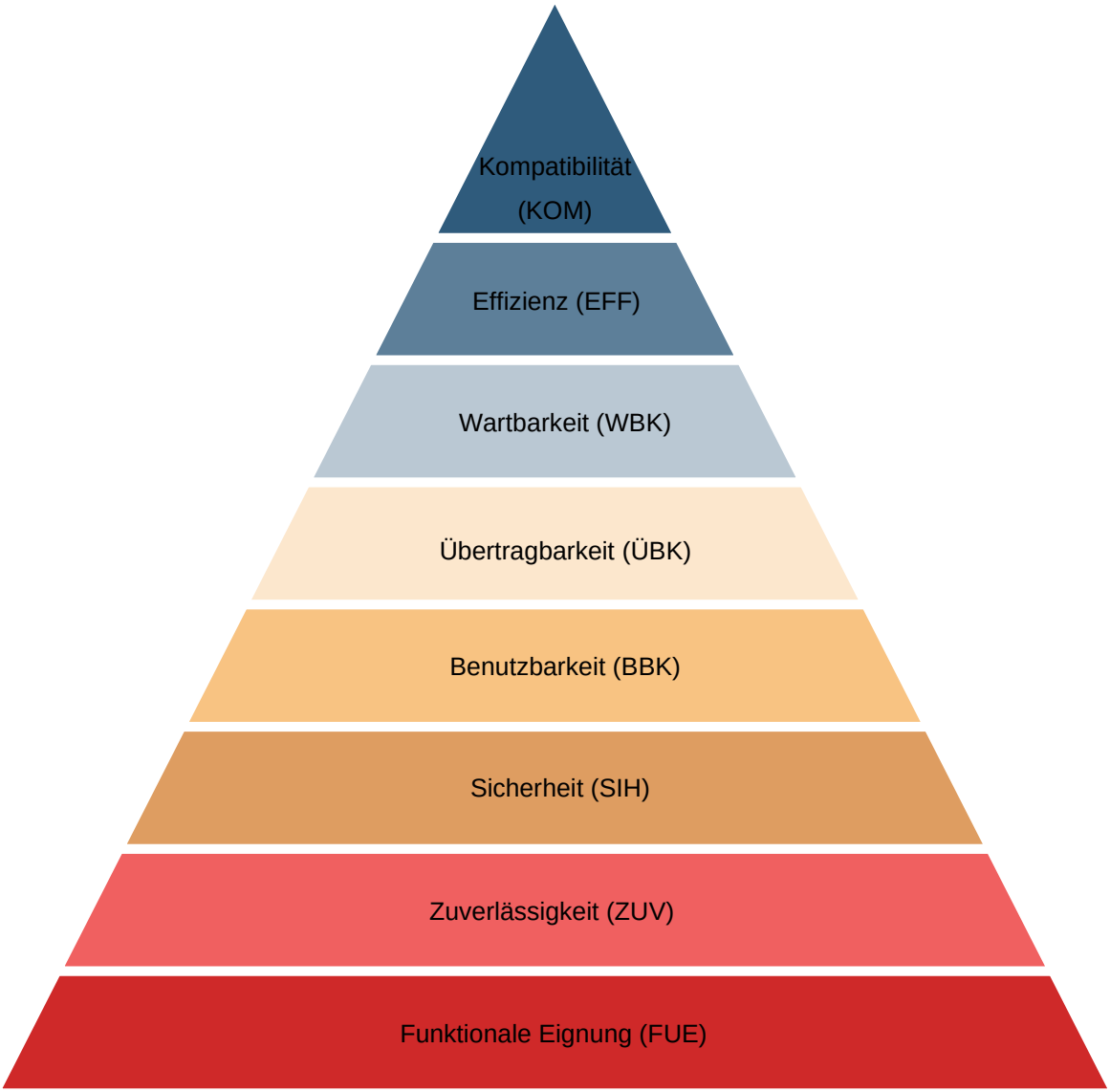


Abbildung 22: Acht Qualitätsmerkmale für Softwarequalität, Quelle: Eigene Darstellung, in Anlehnung an ISO/IEC 25010:2011 (2011)

Die ISO/IEC 25010:2011 unterscheidet zwischen Produktqualität und Gebrauchsqualität und beschreibt ein Modell der Produktqualität und ein Modell der Nutzungsqualität. Es werden Qualitätskriterien unterschieden, die das Produkt in der Entwicklung betreffen und solche, die sich auf die Nutzung eines Produkts fokussieren. Qualität in der Nutzung ist eine Sammlung von Eigenschaften, die all jene Qualitätsmerkmale umfasst, die den Grad beschreiben, in dem ein System die verschiedenen Anforderungen der Endnutzer*innen erfüllt. Aus den beiden Modellen der Nutzungsqualität sowie der Produktqualität lassen sich acht Hauptmerkmale ableiten. Abbildung 22 stellt diese Hauptmerkmale in Form einer Pyramide dar, die in ihrer Gesamtheit die Qualität von Software repräsentiert. Jedes dieser Merkmale besteht aus vielschichtigen Kriterien, von deren sich manche auch gegenseitig ausschließen können. Die Betrachtung der Qualitätsmerkmale und deren Kriterien richtet sich in dieser Arbeit auf das Aufgabengebiet der Steuerungstechnik aus. Wichtig ist zu vermerken, dass die Reihenfolge, in der die acht Qualitätsmerkmale aufgelistet werden, in der Regel keinen Rückschluss auf deren Bedeutsamkeit

zulassen. Jedoch ist in dieser Arbeit bewusst eine Reihung nach der Relevanz der Qualitätsmerkmale, bezogen auf das Aufgabengebiet der Steuerungstechnik, dargestellt. [83]

Die Qualitätsmerkmale von Softwarequalität aus Abbildung 22 sind wie folgt zu interpretieren:

- Die Relevanz eines Qualitätsmerkmals kann individuell je nach gestellten Anforderungen an die Software variieren.
- Den Qualitätsmerkmalen der **FUE**, **ZUV** und **SIH** werden für Software im steuerungstechnischen Umfeld eine hohe Relevanz zugesprochen, weshalb diese das Fundament der Pyramide bilden.
- Die Qualitätsmerkmale **BBK**, **ÜBK**, **WBK**, **EFF**, **KOM** repräsentieren dabei spezielle Aspekte, die besonders die Nutzung bzw. Erstellung der die Software betreffen. Die Relevanz dieser Qualitätsmerkmale hängt stark vom Umfeld, in der Software eingesetzt wird, ab.[84]
- Alle Qualitätsmerkmale können miteinander korrelieren, so kann ein Qualitätsdefizit eines Qualitätsmerkmals, auf die Qualität der andere Qualitätsmerkmale wirken.

Die wichtigsten acht Qualitätsmerkmale sowie deren Kriterien werden in folgender Tabelle 6 beschrieben.

Qualitätsmerkmale und deren Kriterien (K)	**Beschreibung**
Funktionale Eignung (FUE)	Die Funktionale Eignung der Software ist ein wesentlicher Aspekt der Softwarequalität und beschreibt in welchem Maß die Software den geforderten Funktionsumfang tatsächlich erfüllt. Im Rahmen der funktionalen Eignung soll sichergestellt werden, dass alle Anforderungen der Spezifikation korrekt und vollständig erfüllt werden. Funktionale Fehler könne dabei durch vielfältige Ursachen auftreten. Einige Fehler sind auf Fehlinterpretation oder fehlende Spezifikationen zurückzuführen. In anderen Fällen werden bestimmte Programmfunktionen ganz weggelassen. In allen Fällen gilt es diese Fehler zu vermeiden.
(K1) Funktionale Korrektheit	Die funktionale Korrektheit beschreibt den Grad, zu dem das System korrekte Ergebnisse, in angemessener Präzision liefert.
(K2) Vollständigkeit	Die Vollständigkeit beschreibt den Grad, zu dem das System die geforderte Funktionalität aus der informellen Spezifikation vollständig abdeckt.
(K3) Angemessenheit	Die Angemessenheit einer Software beschreibt ihre Fähigkeit bestimmte Aufgaben zu erfüllen oder Zielsetzungen zu erreichen, indem geeignete Funktionalitäten geliefert werden. Ein Beispiel für Angemessenheit: Wenn

[83]Vgl. Manfred Broy (2021), S. 41 ff.

[84] Vgl. Litz (2013), S. 331 ff.

	90 % der Fehlersuche der gesamten Software einen Tag benötigt. Ist die Fehlersuche für 100 % in 10 Tagen dann angemessen?
Zuverlässigkeit (**ZUV**)	Das Qualitätsmerkmal der Zuverlässigkeit beschreibt die Wahrscheinlichkeit, mit der das System die geforderte Funktionalität unter bestimmten Rahmenbedingungen erfüllen kann.
(K4) Ausgereiftheit	Die Ausgereiftheit ist ein Grad, der angibt, wie anfällig die Software auf Fehlerzustände durch ein Versagen reagiert.
(K5) Verfügbarkeit	Unter dem Begriff Zuverlässigkeit versteht man die Fähigkeit einer Hardware oder Software, dauerhaft und entsprechend den jeweiligen Spezifikationen, im Einsatz zu bleiben. Die Zuverlässigkeit unterscheidet sich von der Verfügbarkeit nur dadurch, dass bei der Zuverlässigkeit nur Ausfälle berücksichtigt werden, während die Verfügbarkeit sowohl von Ausfällen als auch von der Wiedereinstellung abhängt.
(K6) Fehlertoleranz	Fehlertoleranz bedeutet, dass die Software auf Fehler des Benutzers *der Benutzerin reagiert. Randbedingungen können oftmals nicht präzise vorhergesagt werden, deshalb muss eine gewisse Fehlertoleranz akzeptiert werden. Wenn möglich, behebt die Software die Fehler selbst. Wenn dies nicht möglich ist, gibt es eine Rückmeldung über das HMI und ermöglicht eine einfache Fehlerkorrektur, bei der die Software nach Möglichkeit hilft.
(K7) Wiederherstellbarkeit	Die Wiederherstellbarkeit bezieht sich auf die Fähigkeit, die Software an dem Punkt wiederherzustellen, an dem der Fehler aufgetreten ist.
Sicherheit (**SIH**)	Sicherheit beschreibt den Grad, mit dem die Software unter festgelegten Rahmenbedingungen keine Gefahr für Leib und Leben oder Ressourcen darstellt und nicht zu einer Beeinträchtigung der Umwelt führt.
(K8) Vertraulichkeit	Die Vertraulichkeit ist der Grad, mit dem die Software die Daten vor unautorisierten Zugriffen schützt.
(K9) Integrität	Die Integrität ist der Grad, mit dem die Software das unautorisierte Lesen und Modifizieren von Daten unterbindet. Außerdem bewertet Integrität wie sicher und unverändert Daten oder Informationen zugestellt werden und ob Prozesse wie beabsichtigt ablaufen.
(K10) Nachweisbarkeit	Nachweisbarkeit ist der Grad, mit dem es möglich ist, nachzuweisen, dass Aktionen oder Ereignisse tatsächlich stattgefunden haben, sodass diese später nicht in Frage gestellt werden können.
(K11) Verantwortlichkeit	Verantwortlichkeit ist der Grad, zu dem die Aktion einer Software-Einheit zu genau dieser Software-Einheit zurückverfolgt werden kann.

Benutzbarkeit (**BBK**)	Die Benutzbarkeit beschreibt den Aufwand, der zur Benutzung der Software erforderlich ist.
(K12) Erlernbarkeit	Unter Erlernbarkeit versteht man den Aufwand für den*die Benutzer*in, die Anwendung zu erlernen (z. B. Bedienung, Ein- bzw. Ausgang).
(K13) Bedienbarkeit	Die Bedienbarkeit beschreibt den Aufwand, die Anwendung zu bedienen.
(K14) Fehlertoleranz	Die Fehlertoleranz beschreibt die Eigenschaft von Software, wie gut eine Fehlbenutzung durch falsche Eingaben verhindert werden kann.
(K15) Verständlichkeit	Die Verständlichkeit beschreibt den Aufwand, das Konzept und die Anwendung zu verstehen.
Übertragbarkeit (**ÜBK**)	Übertragbarkeit ist der Grad der Effektivität und Effizienz, mit der ein Produkt, ein System oder eine Komponente von einer Hardware-, Software- oder Nutzungsumgebung auf eine andere übertragen werden kann.
(K16) Anpassbarkeit	Anpassbarkeit ist der Grad, mit dem ein Produkt oder System effektiv und effizient an andere oder neu entstehende Hardware, Software oder Nutzungsumgebungen angepasst werden kann.
(K17) Installierbarkeit	Installierbarkeit ist der Grad der Effektivität und Effizienz, mit der Software in einer bestimmten Umgebung erfolgreich installiert und/oder deinstalliert werden kann.
(K18) Austauschbarkeit	Austauschbarkeit ist der Grad, mit dem ein Softwareprodukt ein anderes spezifisches Softwareprodukt mit dem gleichen Zweck, in der gleichen Umgebung ersetzen kann.
Wartbarkeit (**WBK**)	Die Wartbarkeit beschreibt den Grad, mit dem eine effiziente und effektive Wartung der Software möglich ist. Die Wartung umfasst Fehlerkorrekturen, Verbesserungen zur Vermeidung zukünftiger Fehler und Modifikationen aufgrund geänderter Anforderungen.
(K19) Modularität	Die Modularität ist ein Maß für die effektive und effiziente Aufteilung von Software in Modulen, Komponenten, Baugruppen oder Bausteinen und ob eine klare Aufgabenteilung zwischen diesen definiert ist.
(K20) Wiederverwendbarkeit	Wiederverwendbarkeit beschreibt den Grad, zu dem die Software als Ganzes oder in Teilen effizient und effektiv, wiederverwendet werden kann.
(K21) Analysierbarkeit	Die Analysierbarkeit beschreibt den Grad, mit dem die Software eine strukturierte Untersuchung der Bedeutung und des Schwerpunkts einer Änderung unterstützt und eine klare Strukturierung bei Entwurf,

	Implementierung, Test und Umsetzung einer Änderung ermöglicht. Dazu gehört insbesondere die Lesbarkeit.
(K22) Modifizierbarkeit	Die Modifizierbarkeit von Software beschreibt, mit welchem Aufwand dieselbe an neue, zukünftige Anforderungen angepasst werden kann.
(K23) Verifizierbarkeit	Die Verifizierbarkeit beschreibt den Grad, mit dem die Software die Verifizier- und Testbarkeit unterstützt, um die gestellten Anforderungen zu erfüllen. Insbesondere die Testbarkeit ist von Relevanz.
Effizienz (**EFF**)	Die Effizienz bestimmt das Verhältnis zwischen dem Leistungsumfang der Software und dem Umfang der eingesetzten Mittel unter festgelegten Bedingungen.
(K24) Antwortverhalten	Für ein sinnvolles Arbeiten sollte die Software innerhalb einer bestimmten Zeit auf Benutzer*innenaktionen oder Ereignisse reagieren können.
(K25) Verbrauchsverhalten	Das Verbrauchsverhalten beschreibt den Grad, wie hoch der Bedarf an Betriebsmitteln für die Erfüllung der Funktionalitäten ist.
Kompatibilität (**KOM**)	Kompatibilität beschreibt die Fähigkeit, wie einfach Systeme mit unterschiedlicher Hardware- oder Softwareumgebungen kommunizieren können. Dieser Qualitätsfaktor umfasst Einfachheit sowie System- und Geräteunabhängigkeit.
(K26) Interoperabilität	Interoperabilität ist der Grad, der angibt, ob zwei oder mehr Software-Systeme Informationen störungsfrei austauschen können und ihre Funktionen ausführen können, während beide dieselbe Hardware- oder Softwareumgebung nutzen.[85]

Tabelle 6: Software-Qualitätsmerkmale und deren Kriterien, Quelle: Eigene Darstellung in Anlehnung an Andreas Maier (2014), S. 10 ff.

6.3 Die Entstehung von Softwarequalität

Die Definition 6-3 sagt aus, dass der Begriff Softwarequalität eine multikausale Größe beschreibt. Es gibt demnach kein einzelnes Kriterium, mit dem Softwarequalität direkt und vor allem quantitativ in Verbindung gebracht werden kann. Stattdessen bildet sich der Begriff durch eine Menge von vielschichtigen Merkmalen

[85] Vgl. ISO/IEC 25010:2011 (2011)

ab, von denen sich einige auch noch gegenseitig ausschließen. Es gilt nun zu ermitteln, in welchem Umfeld, Qualität in der Software entsteht.[86]

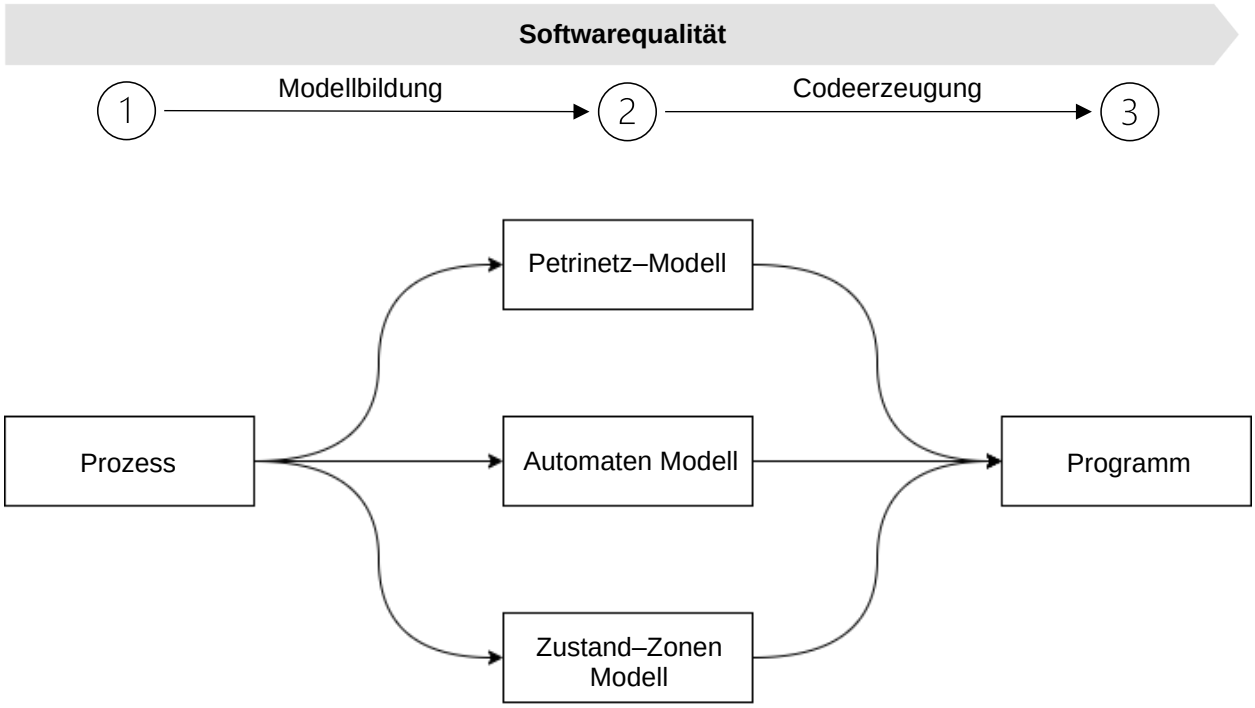


Abbildung 23: Softwareentwicklungsprozess, Quelle: Eigene Darstellung

In Abbildung 23 ist der Softwareentwicklungsprozess in drei Phasen dargestellt. In diesem Kapitel soll die Frage geklärt werden, wie Softwarequalität entsteht. Analysiert werden speziell die Aspekte, die für Anwendungen, die in einem steuerungstechnischen Umfeld eingesetzt werden, gelten. Der Aufbau jeder Phase wird analysiert und in Zusammenhang mit den acht Merkmalen der Softwarequalität gebracht, siehe Kapitel 6.2.

Die Analyse der drei Phasen des Softwareentwicklungsprozesses kann folgende Erkenntnisse liefern:

- **Phase 1: Prozess**
  Softwarequalität entsteht im Prozess über die vollständige Erfassung aller zeitlich parallelen oder nacheinander ablaufenden Vorgänge in einem Automatisierungssystem. Es gilt diese Vorgänge möglichst effizient zu beschreiben, um die gestellten Aufgaben zuverlässig erfüllen zu können. Im Prozess muss festgehalten werden, welche Anforderungen gelten, um die funktionale Korrektheit zu gewährleisten. Somit ist eine Prozessbeschreibung von hoher Qualität, wenn diese alle Vorgänge des zu automatisierenden Systems, funktional korrekt und zuverlässig beschreibt. Die Schwierigkeit besteht darin, alle Eigenschaften der zu beschreibenden Vorgänge in einem Automatisierungssystem vollständig zu erfassen. Je qualitativer die funktionalen Anforderungen in der Prozessbeschreibung definiert werden, desto qualitativere Software kann aus dieser Prozessbeschreibung gebildet werden.

[86]Vgl. Hoffmann (2008), S. 6

Definition 6-4: Die Prozessbeschreibung bildet den geforderten Leistungsumfang der zu erstellenden Software ab. Die Qualität der Prozessbeschreibung wird vorwiegend durch das Qualitätsmerkmal der **FUE** bewertet. Fehlerhafte oder unvollständige Prozessbeschreibungen führen zu qualitativen Einbußen im Modell.

- **Phase 2: Modell**
  Die Modellbildung stellt ein formales Hilfsmittel dar, um die geforderten Abläufe der Prozessbeschreibung respektive zugehöriger Eigenschaften als Modell formal zu spezifizieren. Softwarequalität wird demnach in der Modellbildung aus der informellen Spezifikation, in eine formale Spezifikation, übertragen. Daraus entsteht der Kompromiss zwischen Modellgenauigkeit und Modellkomplexität. Je genauer sich das Modell der Wirklichkeit nähert, desto höher der Aufwand ein solches zu beschreiben.

Definition 6-5: Die **Qualität eines Modells** hängt von der zugrunde liegenden **Qualität der Prozessbeschreibung** ab. Besonders das Qualitätsmerkmale der **FUE**, **ZUV** und **SIH** müssen bei der Modellbildung ohne qualitative Einbußen übertragen werden. Zusätzlich werden durch die Modellbildung die Qualitätsmerkmale **BBK**, **WBK** und **ÜBK** gebildet, siehe Abbildung 24.

- **Phase 3: Programm**
  Ein Programm kann durch einen sogenannten Codegenerator aus dem Modell heraus erzeugt werden. Ein Codegenerator ist ein Programm, welches Modelle oder andere formale Sprachen übersetzt und daraus einen Quellcode erzeugt. Dieser Quellcode dient als Vorgabe, um in einem weiteren Schritt verschiedene Programmteile zu einem Programm zu verbinden. Die Softwarequalität wird über die Gesamtheit der Merkmale, die sich auf die Fähigkeit beziehen, vorausgesetzte Anforderungen zu erfüllen, definiert. Bei der Programmierung sind wesentliche Aspekte der Softwarequalität zu berücksichtigen und durch die Gestaltung des Quellcodes umzusetzen.[87]

Die Qualitätsmerkmale **KOM** und **EFF** werden durch die Codeerzeugung maßgeblich beeinflusst, siehe Abbildung 24. Ob die Software nun von Qualität ist, geht nicht nur von der Codeerzeugung aus, sondern wie präzise und vollständig der Prozess beschrieben ist. Ein qualitativ hochwertiges Modell bildet alle Anforderungen und Eigenschaften ab, um die geforderten Aufgaben zuverlässig erfüllen zu können.

[87]Vgl. Manfred Broy (2021) S. 443 ff.

Definition 6-6: Soll ein **Code** aus einem **Modell** heraus erzeugt wurde, kann ein **Codegenerator** nicht qualitativ verbessernd auf das wichtigste Qualitätsmerkmal von Software, der FUE, einwirken. Wird beispielsweise die Funktion nicht vollständig in der Prozessbeschreibung definiert, so wird dies bei der Modellbildung und in weiterer Folge in der Software nicht berücksichtigt. Die Codeerzeugung kann folglich dafür sorgen, **die Qualität der FUE zu erhalten, diese aber nicht erhöhen.**

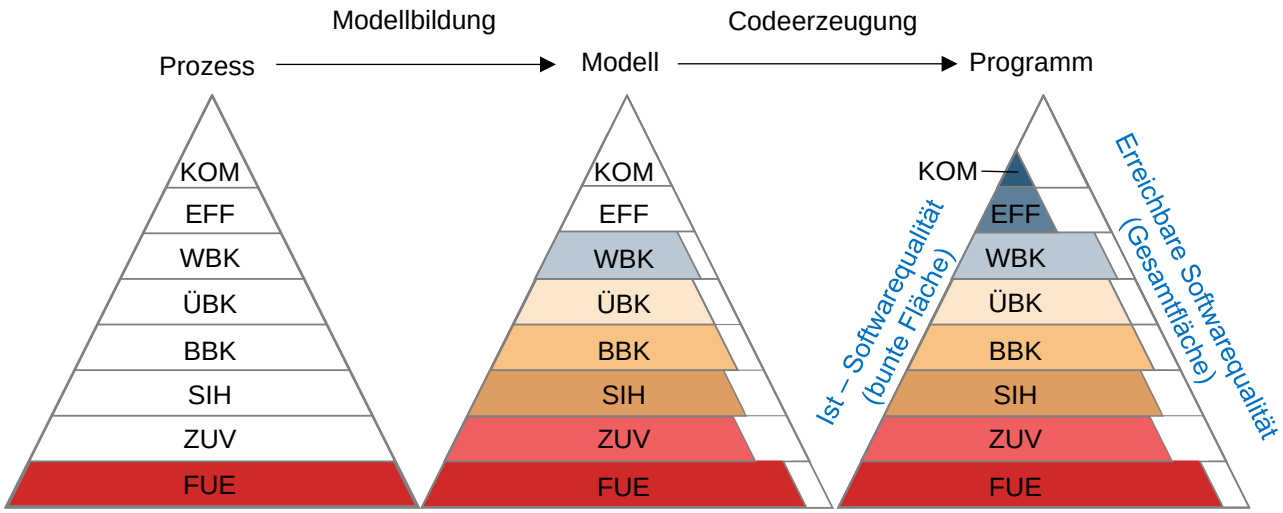


Abbildung 24: Qualitätspyramide über die Phasen der Softwareentwicklung, Quelle: Eigene Darstellung

Abbildung 24 zeigt den Aufbau der Qualitätspyramide über die drei Phasen der Softwareentwicklung. Diese Pyramide baut sich von Phase zu Phase auf und nimmt in Form der erstellten Software seine vollendete Größe (IST-Softwarequalität) an. Erkennbar ist, dass die erreichte, von der erreichbaren Größe abweichen kann, obwohl alle Merkmale der Qualitätskriterien abgebildet sind. Dies liegt daran, dass Qualität im Übergang zwischen den Phasen verloren gehen kann und somit die Gesamtqualität mindert.

Definition 6-7: Um die **Softwarequalität** in Bezug auf den Softwareentwicklungsprozess zu gewährleisten, liegt der Fokus auf einer qualitativ hochwertigen **Prozessbeschreibung** und **Modellbildung**. Damit wird ein Großteil der Softwarequalität im Modell gesichert und kann durch eine entsprechende Programmerstellung auch vollwertig in das Programm übertragen werden.

6.4 Techniken der Qualitätssicherung

Obwohl die beschriebenen Qualitätskriterien aus Kapitel 6.2 für allgemeine Anwendersoftware definiert wurden, gelten insbesondere die ersten sechs Kriterien in vollem Umfang für die Software, die den Steueralgorithmus abbildet. Daher wird ein methodisches Werkzeug benötigt, das möglichst alle diese Kriterien in Summe erfüllen kann. Die Besonderheiten aus der Welt der Automatisierungstechnik müssen dabei berücksichtigt werden. In den folgenden Kapiteln 6.4.1 und 6.4.2 werden besonders diejenigen Methoden skizziert, die im Stande sind, diese Leistung zu erbringen.

6.4.1 Simulation und Test

Ein gängiges Methodenpaar in der Softwaretechnik, auf das der Steuerungsentwurf nicht verzichten kann, sind Simulation und Test. Diese Begriffe werden in der Steuerungstechnik teilweise als Synonyme verwendet, dennoch gibt es einen Unterschied: Test ist der umfassendere Begriff und schließt die Simulation, also die Berechnung von Ausgängen auf der Grundlage geeigneter Eingänge, als Sonderfall ein. Sowohl die Simulation als auch der Test untersuchen das Verhalten einer Steuerung in einer definierten Umgebung, auf der Grundlage bestimmter Szenarien. Die Szenarien können so gewählt werden, dass vor allem die Kriterien **FUE**, **ZUV** und **SIH** überprüft werden. Die Hauptaufgabe beider ist die versuchsweise Überprüfung, ob die Spezifikation der Steuerungsvorgabe durch das realisierte System erfüllt wird. Dennoch ist es sinnvoll, zwischen beiden zu unterscheiden. Der Begriff Test stammt ursprünglich aus der Softwaretechnik. Er wird als Sammelbezeichnung für verschiedene Methoden verwendet, die sich in die beiden Gruppen White-Box-Test und Black-Box-Test unterteilen lassen. Beim **White-Box-Test** werden die internen Elemente eines Programms betrachtet. Für den Steueralgorithmus sind dies die Startbedingung, die Zustandsübergänge mit den Eingängen, die Zustände und den entsprechenden Ausgängen. Der **Black-Box-Test** handelt vom Input-Output-Verhalten des Steuerungssystems, ohne die internen Details zu berücksichtigen. Durch definierte Eingänge werden Ausgänge vom Steueralgorithmus berechnet. Die Ergebnisse werden gegen die informelle Spezifikation geprüft. Wenn dieses Verfahren auf der Grundlage von Modellen erfolgt, ist die Simulation ein Black-Box-Test. In der Praxis wird bei Steuerungssystemen die Simulation auch verwendet, wenn der entworfene Steueralgorithmus auf dem Zielsystem, z. B. der SPS, ausgeführt wird und nur die Prozessrückführung oder Teile davon durch manuelle Eingänge simuliert werden.[88]

Durch die Simulation kann in der Steuerungstechnik das Ein-/Ausgangsverhalten eines Systems auf der Basis eines Systemmodells berechnet werden. Typische Systemmodelle sind endliche (deterministische) Automaten mit Ein-/Ausgang, steuerungstechnisch interpretierte Petrinetze (**SIPN**) oder Modelle der Zustand-Zonen Modellierung (**ZZM**). Das große Problem der Simulation ist die Zustandsexplosion. Das bedeutet durch eine Modellierung von komplexen Systemen kann die Anzahl der zu überprüfenden Zustände exponentiell steigen, siehe Kapitel 2.4. Der Nutzen einer Simulation hängt wesentlich vom Simulationsszenario ab. Als ein Simulations- oder Testszenario wird eine spezielle Eingangsfolge u_k für den Automaten oder e_k für die SIPN, mit den jeweiligen Ausgangszuständen oder Ausgangsmarkierungen bezeichnet. Um den Aufwand für eine solche Überprüfung zu minimieren, ist die Anzahl der Testfälle oftmals zu minimieren. Dabei soll die Wahrscheinlichkeit, Fehler zu entdecken, trotzdem hochgehalten werden. Ein Simulations- oder Testszenario kann prinzipiell mit oder ohne Berücksichtigung der Struktur des zu testenden Algorithmus ein Ergebnis erzielen. Im Sinne des Black-Box-Tests sollten die Testfälle aus der gegebenen Spezifikation abgeleitet werden. Dies hat den Vorteil, dass sich eventuell vorhandene Fehler in der Struktur, nicht auf die Testfälle auswirken können. Dies gilt auch für jene Szenarien, die nur selten im Betrieb zu erwarten sind oder im Extremfall gar nicht eintreten werden.[89]

[88] Vgl. Litz (2013), S. 334

[89] Vgl. Litz (2013), S. 335 ff.

6.4.2 Model-Checking

Model-Checking ist eine Methode, die sich immer auf Modelle der Realität stützt. Streng genommen erfordert sie sogar zwei Modelle ein und desselben Sachverhalts, z.B. die Beschreibung des Steueralgorithmus, als endlicher Automat oder als Petrinetz und die Beschreibung des Zielverhaltens in Form einer temporalen Logik. Was unter dem Gesichtspunkt des Aufwands als Nachteil angesehen werden könnte, erweist sich auf der anderen Seite als ein nicht zu unterschätzender Vorteil: Es werden zwei unterschiedliche Modelle gegeneinander geprüft, was insbesondere den Nachweis von Funktionalität ermöglicht. Im Gegensatz zu Simulation und Test lässt sich die Modellprüfung relativ einfach auf die Zuverlässigkeit ausweiten. Das liegt daran, dass an bestimmten Stellen, Abstraktionen der ursprünglichen Modelle verwendet werden können. Diese decken dann wesentlich mehr Verhalten ab als die ursprünglichen Modelle. Das große Problem bei Model-Checking ist die Zustandsexplosion, denn mit Modellierung von komplexen Systemen kann die Anzahl der Zustände exponentiell wachsen, siehe Kapitel 2.4. Man bezeichnet das Verfahren des Model-Checking daher auch als erschöpfend. Unter großen Bemühungen wird versucht, den Zustandsraum möglichst zu verringen.[90]

6.4.3 Verifikation und Validierung

Die Begriffe Verifizierung und Validierung überschneiden sich manchmal inhaltlich oder werden sogar als Synonyme verwendet. Man spricht oft pauschal von V&V-Methoden. Dennoch ist es sinnvoll, zwischen beiden Begriffen zu unterscheiden. Dazu wird folgende Fragestellung zur Unterscheidung dieser Begriffe verwendet:

- Validierung: Erzeugen wir das richtige Produkt?
- Verifikation: Erzeugen wir das Produkt richtig?

Hinter diesen einprägsamen Fragestellungen verbergen sich zwei grundverschiedene Problemstellungen. Bei der Validierung geht es darum, das Richtige zu tun. Sie stellt die Frage in den Vordergrund, ob die Aufgabenstellung erfüllt wird. Damit bezieht sich Validierung auf den aller ersten Prozessschritt des Steuerungsentwurfsprozess, die informelle Spezifikation aus Abbildung 8. Es ist durch die Komplexität der Steuervorgabe schwierig, eine vollständige, in sich konsistente informelle Spezifikation zu definieren. Die Validierung bedient sich vor allem der Methoden der Simulation und des Tests sowie dem Model-Checking. Bei der Verifizierung hingegen geht es darum, es richtig zu machen. Ziel ist es, die während des Entwurfsprozesses gemachten Fehler aufzudecken oder zu beweisen, dass alle Fehler korrigiert wurden. Der Fall, dass Fehler von vornherein während des Entwurfsprozesses vermieden wurden, ist eher theoretischer Natur. Typisch für die Verifikation ist, dass formale Spezifikationen vorhanden sind, um auf formale Methoden angewandt zu werden.[91]

[90] Vgl. Litz (2013), S. 342

[91] Vgl. Litz (2013), S. 353

6.5 Bewertung von Softwarequalität

Die Bewertung der Qualität von Software hängt von den ermittelten oder vorausgesetzten Anforderungen an die Qualität ab. Insofern kann man die Qualität einer Software nicht generell beschreiben, sondern muss sie immer in Hinblick auf ihre konkreten Anforderungen in ihrem Umfeld betrachten. In diesem Kapitel werden geeignete Mittel ausgewählt, um die Qualität von Software zu beschreiben, die zur Lösung einer Steuervorgabe für Speicherprogrammierbare Steuerungen entwickelt worden sind. Dazu wird der Bewertungsvorgang in drei Phasen näher beschrieben:

- **Qualitätsanforderungen festlegen**
  In der ersten Phase des Bewertungsprozesses werden die Qualitätsanforderungen definiert. Hier gilt es vor der Entwicklung zu spezifizieren, welche Anforderungen an die zu entwickelnde Software bestehen. Für die in dieser Arbeit betrachteten Modelle für Steuerungssysteme, ist besonders das Qualitätsmerkmal der **FUE** von Bedeutung.
- **Softwarequalitäts-Metriken auswählen**
  Die Qualität der untersuchten Software ist nicht an ihren Anforderungen messbar, weshalb eine Funktion notwendig ist, die Eigenschaften von Software anhand einer Maßzahl abbilden kann. Dadurch entsteht die Möglichkeit, eine formale Bewertung einer Eigenschaft von Software durchzuführen. Zur Bewertung der Softwarequalität wird folgende Produkt-Metrik ausgewählt: Die definierten Softwarequalitätskriterien gelten als Anforderungen, die über funktionale Tests oder Reviews überprüft werden. Die Umsetzung des praktischen Beispiels aus Kapitel 7 dient dabei als Bewertungsgrundlage. Die daraus resultierenden Ergebnisse werden somit messbar. Die Maßzahl gibt das Verhältnis zwischen Ist- und Soll-Funktionalität an.
- **Bewertungsmethodik festlegen**
  In Kapitel 6.3 wird beschrieben, welche Phasen des Softwareentwicklungsprozesses wichtig in Bezug auf die verschiedenen Qualitätsmerkmale sind. Werden Modelle aufgrund der Qualitätsmerkmale gegenübergestellt und verglichen, muss eine einheitliche Prozessbeschreibung für alle Modelle vorliegen, siehe **Definition 6-5**. Eine informelle Prozessbeschreibung soll möglichst vollständig, funktionell korrekt und konsistent dargestellt werden, siehe **Definition 6-4** Man kann davon ausgehen, dass die Übersetzung eines Modells in Software durch, z. B. einen Codegenerator, keinen qualitativen Mehrwert erzeugt, sondern nur Qualität in dieser Phase sichert. Dieser Sachverhalt wird in der **Definition 6-6** zusammengefasst.

7 PRAKTISCHE UMSETZUNG

In diesem Kapitel wird ein praktisches Beispiel vorgestellt, um anschließend durch ausgewählte Modellierungsmethoden formale Modelle dieser Steuervorgabe zu erstellen. Diese Modelle dienen als Basis, um die Modellierungsmethoden auf deren Qualität zu bewerten und gegenüberzustellen.

7.1 Das Steuerungsbeispiel Windkessel

In Abbildung 25 ist eine Anwendung dargestellt, die einer industriellen Anlage entstammt und zu Darstellungszwecken geringfügig vereinfacht wurde. Ein Windkessel ist eine Bezeichnung für einen Druckluftbehälter, der versucht ein kostantes Druckniveau für den angeschlossenen Verbraucher zu halten. Dabei kann nicht vorhergesehen werden, wie hoch die Luftabnahme durch die Verbraucher sein wird.

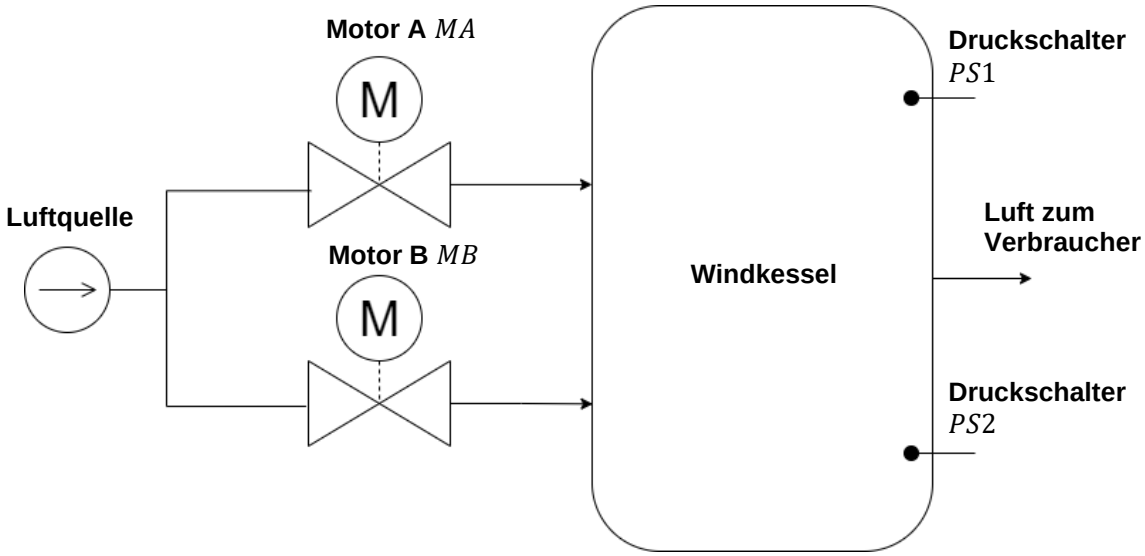


Abbildung 25: Schaubild des Beispiels Windkessel, Quelle: Eigene Darstellung, in Anlehnung an Litz (2013), S. 182

Aufgabe dieser Steuerung ist es, den Druck im Windkessel möglichst konstant zu halten. Dabei wird Druckluft durch zwei motorbetriebene Kompressoren in den Windkessel gefördert. Die Motoren können sowohl im Einzelbetrieb, als auch im Parallelbetrieb arbeiten. Die Verbraucher entnehmen Luft aus dem Windkessel in einem nicht vorhersehbaren Rhythmus. Die Herausforderung besteht nun darin, ein konstantes Druckniveau zu gewährleisten. Dafür sind zwei Sensoren angebracht, die das Unterschreiten eines bestimmten Druckniveaus signalisieren. Die Druckschalter werden als $PS1$ und $PS2$ bezeichnet. Um die Abnützung der beiden Motoren möglichst gering zu halten, soll bei Unterschreiten des Druckniveaus von $PS1$ abwechselnd nur ein Motor betrieben werden. Wird auch das Druckniveau $PS2$ unterschritten, sollen beide Motoren betrieben werden. Beide Motoren sind mit je einer Störüberwachung zur Detektion von Störungen ausgestattet. Wird eine solche Störung eines Motors erkannt, soll der jeweils andere Motor den Betrieb aufrecht erhalten. Um diese Steuerung letzlich auch zu betreiben, sind zwei zusätzliche Befehle von außen notwendig, der Start- und Stoppbefehl der von dem*der Benutzer*in betätigt werden kann. Der **Ausgangszustand** ist dabei, dass der Windkessel über den Druckniveaus der Druckschalter $PS1$ und $PS2$

gefüllt ist. Ein seperater Startvorgang wird ausgeführt, um diesen Systemzustand herbeizuführen. Dieser Vorgang wird in dieser Arbeit nicht behandelt.[92]

7.2 Informelle Spezifikation

Das Beispiel Windkessel aus Kapitel 7.1 wird in Abbildung 25 dargestellt und kann durch folgende vier **Anforderungen** beschrieben werden:

a) Beide Motore sollen betrieben werden, wenn der Druckschalter $PS2$ anspricht (**Druckniveau unter 590000 Pa, entspricht ca. 5,9 bar***).
b) Ein Motor soll betrieben werden, wenn der Druckschalter $PS1$ anspricht (**Druckniveau unter 610000 Pa, entspricht 6,1 bar***).
c) Zur Optimierung des Verschleißes sollen die Motoren abwechselnd betrieben werden.
d) Beim Auftreten einer Störung eines Motors soll der jeweils andere betrieben werden.

Eine informelle Spezifikation ist eine Beschreibung aus verschiedenen Sichtweisen. In dieser Beschreibung sind in der Regel das Eingangs-Ausgangs Verhalten sowie eine Störungsbehandlung vorgesehen. Eine Vollständigkeit der Beschreibung wird so gut wie nie erreicht, siehe Kapitel 4.4.[93]

7.3 Formale Spezifikation

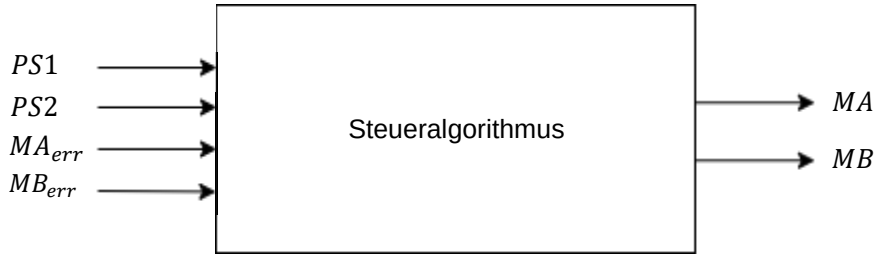


Abbildung 26: Signalumgebung des Steueralgorithmus, Quelle: Eigene Darstellung, in Anlehnung an Litz (2013), S. 183

In Abbildung 26 ist die Signalumgebung der Steuerung aus dem Beispiel dargestellt. Informationen aus den insgesamt vier Eingangssignalen werden verarbeitet und über zwei Stellsignale MA und MB werden Startbefehle zu den jeweiligen Motoren übermittelt. Die laufenden Motoren werden überwacht und etwaige Laufzeitstörungen MA_{err} und MB_{err} an die Steuerung rückgemeldet.

Aus Darstellungsgründen wird in dieser Arbeit nur ein partieller Teil der Aufgabenstellung betrachtet. Dies wäre für eine praktische Umsetzung der Steuervorgabe nicht korrekt ausgeführt, dennoch wird konkret auf den Start- sowie Stoppbefehl verzichtet. Angenommen wird, dass der Startbefehl bereits erfolgte und die Anwendung im Dauerbetrieb laufen soll. Dennoch hat diese Abgrenzung der Aufgabenstellung einen

[92] Vgl. Litz (2013), S. 179 ff.

[93] Vgl. Georg Frey (1999), S. 148-149

* Da in einigen Quellen die Einheit bar anstatt Pa für Druck verwenden, wird in dieser Arbeit die Einheit bar weiterhin verwendet.

Einfluss auf die Lösung. Wie groß dieser Einfluss ist, wird am Ende dieser Arbeit thematisiert, siehe Kapitel 9.

7.3.1 Boolesche Gleichung

In der Booleschen Gleichung werden die Aussagen über die Ausgänge dargestellt. Über eine Wahrheitstabelle wird die Logik erstellt, aus der sich der Steueralgorithmus ableiten lässt, siehe Tabelle 7. Der Steueralgorithmus bildet die Eingänge ($PS1, PS2, MA_{err}, MB_{err}$) auf die Ausgänge ($MA, MB$) ab. Da der Drucksensor $PS2$ physikalisch nicht vor dem Drucksensor $PS1$ ansprechen kann, sind die Zustände z_5 bis z_8 Fehlerzustände. Es kann nicht ermittelt werden welcher Sensor eine Fehlfunktion aufweist, jedoch kann generell eine Fehlfunktion festgestellt werden. Deswegen wird kein Motor eingeschaltet und der Betrieb ausgesetzt.

Zustand	$PS1$	$PS2$	MA_{err}	MB_{err}	MA	MB	Beschreibung
z_1	**0**	**0**	**0**	**0**	0	0	Windkessel voll, keine Störung
z_2	**0**	**0**	**0**	**1**	0	0	Windkessel voll, Störmeldung Motor B
z_3	**0**	**0**	**1**	**0**	0	0	Windkessel voll, Störmeldung Motor A
z_4	**0**	**0**	**1**	**1**	0	0	**Fehler** Störmeldung Motor A & B, Windkessel voll,
z_5	**0**	**1**	**0**	**0**	0	0	Sensor **Fehler**
z_6	**0**	**1**	**0**	**1**	0	0	Sensor **Fehler**, Störmeldung Motor B
z_7	**0**	**1**	**1**	**0**	0	0	Sensor **Fehler**, Störmeldung Motor A
z_8	**0**	**1**	**1**	**1**	0	0	Sensor **Fehler**, Störmeldung Motor A & B
z_9	**1**	**0**	**0**	**0**	1	0	Motor A ein
z_{10}	**1**	**0**	**0**	**1**	1	0	Störmeldung Motor B, Motor A ein
z_{11}	**1**	**0**	**1**	**0**	0	1	Störmeldung Motor A, Motor B ein
z_{12}	**1**	**0**	**1**	**1**	0	0	**Fehler** Störmeldung Motor A & B
z_{13}	**1**	**1**	**0**	**0**	1	1	Motor A und B ein
z_{14}	**1**	**1**	**0**	**1**	1	0	Motor A ein, Störmeldung Motor B
z_{15}	**1**	**1**	**1**	**0**	0	1	Motor B ein, Störmeldung Motor A
z_{16}	**1**	**1**	**1**	**1**	0	0	**Fehler** Störmeldung Motor A & B

Tabelle 7: Wahrheitstabelle Beispiel Windkessel, Quelle: Eigene Darstellung

Um die Anforderung c) aus 7.2 zu realisieren, muss eine zusätzliche logische Verknüpfung im Steueralgorithmus realisiert werden. Dies betrifft den Zustand z_9 der in Tabelle 7 nur mit Motor A ausgeführt wurde. Auch Motor B wäre in der Lage den Betrieb im Zustand z_9 durchzuführen. Somit können die Anforderungen a) b) und d) in folgenden Booleschen Ausdrücken beschrieben werden.

$$MA = (PS1 \wedge \neg MA_{err})$$

$$MB = (PS1 \wedge \neg PS2 \wedge MA_{err} \wedge \neg MB_{err}) \vee (PS1 \wedge PS2 \wedge \neg MB_{err})$$

7.3.2 Endlicher Automat

7.3.2.1 Deterministisch endlicher Automat als Automatentabelle

Die Automatentabelle beschreibt die Zustandsübergänge an den entsprechenden Eingangsvektoren siehe Formel (7-1). Die Ausgangsvektoren sind direkt von den Zuständen abhängig und werden ihnen auch in der Darstellung zugeordnet, siehe Formel (7-2).

$$u_n = [PS1\ \ PS2\ \ MA_{err}\ \ MB_{err}] \quad (7\text{-}1)$$

u_n	Eingangsvektor
$PS1$	Eingang Druckschalter (< 6.1 bar)
$PS2$	Eingang Druckschalter (< 5,9 bar)
MA_{err}	Eingang Störmeldung Motor A
MB_{err}	Eingang Störmeldung Motor B

$$y_n = [MA\ \ MB] \quad (7\text{-}2)$$

y_n	Ausgangssvektor
MA	Ausgang Motor A (Ein/Aus)
MA	Ausgang Motor B (Ein/Aus)

Die im vorherigen Kapitel behandelte Wahrheitstabelle wird in der weiteren Betrachtung um all jene Zustände verringert, die Fehler im System verursachen (Zustand z_4 bis z_8, z_{12} und z_{16}) da in den Anforderungen kein Handlungsbedarf vorgesehen ist. Weiters sind in den Zuständen z_2 und z_3 keine Ausgänge aktiv und kann damit vernachlässigt werden, siehe Tabelle 7. Die übrig gebliebenen Zustände werden nun neu angeordnet, um dadurch eine gute Übersicht zu erlangen. Tabelle 8 stellt die Zustände vor, die zur Lösung der informellen Spezifikation aus Kapitel 7.2 benötigt werden. Dabei sind die Zustände z_1, z_2, z_4 und z_6 notwendig um die Anforderungen a), b) und d) darzustellen. Die Zustände z_3 und z_5 sind notwendig, um die Anforderung c) zu realisieren. Um die Motoren abwechselnd zu betreiben, wird je nachdem welcher Motor vorher lief, entschieden welcher danach bei Bedarf folgt. Deshalb sind in der weiteren Betrachtung die Zustände z_2 und z_3 sowie z_1 und z_5 notwendig, siehe Tabelle 8.

* Der Zustand z_4 aus Tabelle 8 wird aus den Zuständen z_9, z_{10} und z_{14} aus Tabelle 7 gebildet.

** Der Zustand z_6 aus Tabelle 8 wird aus den Zuständen z_9, z_{11} und z_{15} aus Tabelle 7 gebildet.

	Eingänge					Ausgänge		
Zustand	$PS1$	$PS2$	MA_{err}	MB_{err}	y_n	MA	MB	Beschreibung
z_1	**0**	**0**	**0**	**0**	y_1	**0**	**0**	Ausgangszustand (Grundstellung), Windkessel voll (> 6.1 Bar), keine Störung, vorher Motor B
z_2	**1**	**1**	**0**	**0**	y_2	**1**	**1**	$PS1$ und $PS2$ aktiv, Motor A und B ein, vorher Motor B
z_3	**1**	**1**	**0**	**0**	y_3	**1**	**1**	$PS1$ und $PS2$ aktiv, Motor A und B ein, vorher Motor A
z_4*	**1**	**0**	**0**	**0**	y_4	**1**	**0**	$PS1$ aktiv, keine Störung, vorher Motor B, Motor A ein
z_4*	**1**	**0**	**0**	**1**				$PS1$ aktiv, Störmeldung Motor B, Motor A ein
z_4*	**1**	**1**	**0**	**1**				$PS1$ und $PS2$ aktiv, Störmeldung Motor B, Motor A ein
z_5	**0**	**0**	**0**	**0**	y_5	**0**	**0**	Windkessel voll (> 6.1 Bar), keine Störung, vorher Motor A
z_6**	**1**	**0**	**0**	**0**	y_6	**0**	**1**	$PS1$ aktiv, keine Störung, vorher Motor A, Motor B ein
z_6**	**1**	**0**	**1**	**0**				$PS1$ aktiv, Störmeldung Motor A, Motor B ein
z_6**	**1**	**1**	**1**	**0**				$PS1$ und $PS2$ aktiv, Störmeldung Motor A, Motor B ein

Tabelle 8: Zustandstabelle Beispiel Windkessel, Quelle: Eigene Darstellung

Um die Lesbarkeit zu erhöhen, wird aufgelistet, in welchen Zuständen es welche Ausgänge y_n gibt und welche Folgezustände sich durch die Eingänge u_n ergeben, siehe Tabelle 9.

	y_n			y_{n+1}		u_n				
Zustand	MA	MB	Folgezustand	MA	MB	Bezeichnung	$PS1$	$PS2$	MA_{err}	MB_{err}
z_1	**0**	**0**	z_2	**1**	**1**	u_1	**1**	**1**	**0**	**0**
z_1	**0**	**0**	z_4	**1**	**0**	u_2	**1**	**0**	**0**	**0**
z_1	**0**	**0**	z_4	**1**	**0**	u_{14}	**1**	**0**	**0**	**1**
z_1	**0**	**0**	z_6	**0**	**1**	u_{10}	**1**	**0**	**1**	**0**
z_2	**1**	**1**	z_4	**1**	**0**	u_7	**1**	**0**	**0**	**0**
z_2	**1**	**1**	z_6	**0**	**1**	u_{17}	**1**	**0**	**1**	**0**
z_2	**1**	**1**	z_4	**1**	**0**	u_{18}	**1**	**0**	**0**	**1**
z_3	**1**	**1**	z_6	**0**	**1**	u_5	**1**	**0**	**0**	**0**
z_3	**1**	**1**	z_4	**1**	**0**	u_{19}	**1**	**0**	**0**	**1**
z_3	**1**	**1**	z_6	**0**	**1**	u_{20}	**1**	**0**	**1**	**0**
z_4	**1**	**0**	z_5	**0**	**0**	u_{15}	**0**	**0**	**0**	**1**
z_4	**1**	**0**	z_3	**1**	**1**	u_8	**1**	**1**	**0**	**0**
z_4	**1**	**0**	z_5	**0**	**0**	u_3	**0**	**0**	**0**	**0**
z_4	**1**	**0**	z_6	**0**	**1**	u_{12}	**1**	**0**	**1**	**0**
z_4	**1**	**0**	z_5	**0**	**0**	u_{23}	**0**	**0**	**1**	**0**
z_5	**0**	**0**	z_6	**0**	**1**	u_4	**1**	**0**	**0**	**0**
z_5	**0**	**0**	z_4	**1**	**0**	u_{11}	**1**	**0**	**0**	**1**
z_5	**0**	**0**	z_6	**0**	**1**	u_{21}	**1**	**0**	**1**	**0**
z_5	**0**	**0**	z_3	**1**	**1**	u_{22}	**1**	**1**	**0**	**0**
z_6	**0**	**1**	z_4	**1**	**0**	u_{13}	**1**	**0**	**0**	**1**
z_6	**0**	**1**	z_1	**0**	**0**	u_{16}	**0**	**0**	**1**	**0**
z_6	**0**	**1**	z_1	**0**	**0**	u_6	**0**	**0**	**0**	**0**
z_6	**0**	**1**	z_2	**1**	**1**	u_9	**1**	**1**	**0**	**0**
z_6	**0**	**1**	z_1	**0**	**0**	u_{24}	**0**	**0**	**0**	**1**

Tabelle 9: Automatentabelle, Quelle: Eigene Darstellung

7.3.2.2 Deterministisch endlicher Automat als Zustandsgraph

Zu Beginn werden aus der informellen Spezifikation die entsprechenden Zustände erarbeitet, siehe Tabelle 9. Die informelle Spezifikation aus Kapitel 7.2 wird durch Zustände und Zustandsübergänge beschrieben. Ziel ist es, eine formale Spezifikation zur Erfüllung der informellen Spezifikation zu finden. Dabei gilt es, alle geforderten Funktionalitäten inklusive aller relevanten Fehlermöglichkeiten zu berücksichtigen. Die Komplexität, diese Aufgabenstellung formal in einem Modell darzustellen, steigt exponentiell mit der Anzahl an möglichen Zuständen an. Je mehr mögliche Zustände das System annehmen kann, desto schwieriger wird es, alle Zustände korrekt abzubilden. Der Fokus liegt dabei auf den Zuständen, die für die korrekte Abhandlung aller Funktionalitäten zuständig sind. Alle Zustände, die nicht eintreffen dürfen oder sollen, sind nicht in den Anforderungen behandelt und werden daher nicht berücksichtigt. Anzumerken ist jedoch, dass nur vorgedachte und im Modell einbezogene Fehlerzustände im weiteren Verlauf auch als solche

erkannt werden können. Dies lässt diese Methodik der Modellierung zu einer, mit steigender Größe der zu bewältigenden Aufgabe, herausfordernden Tätigkeit werden.

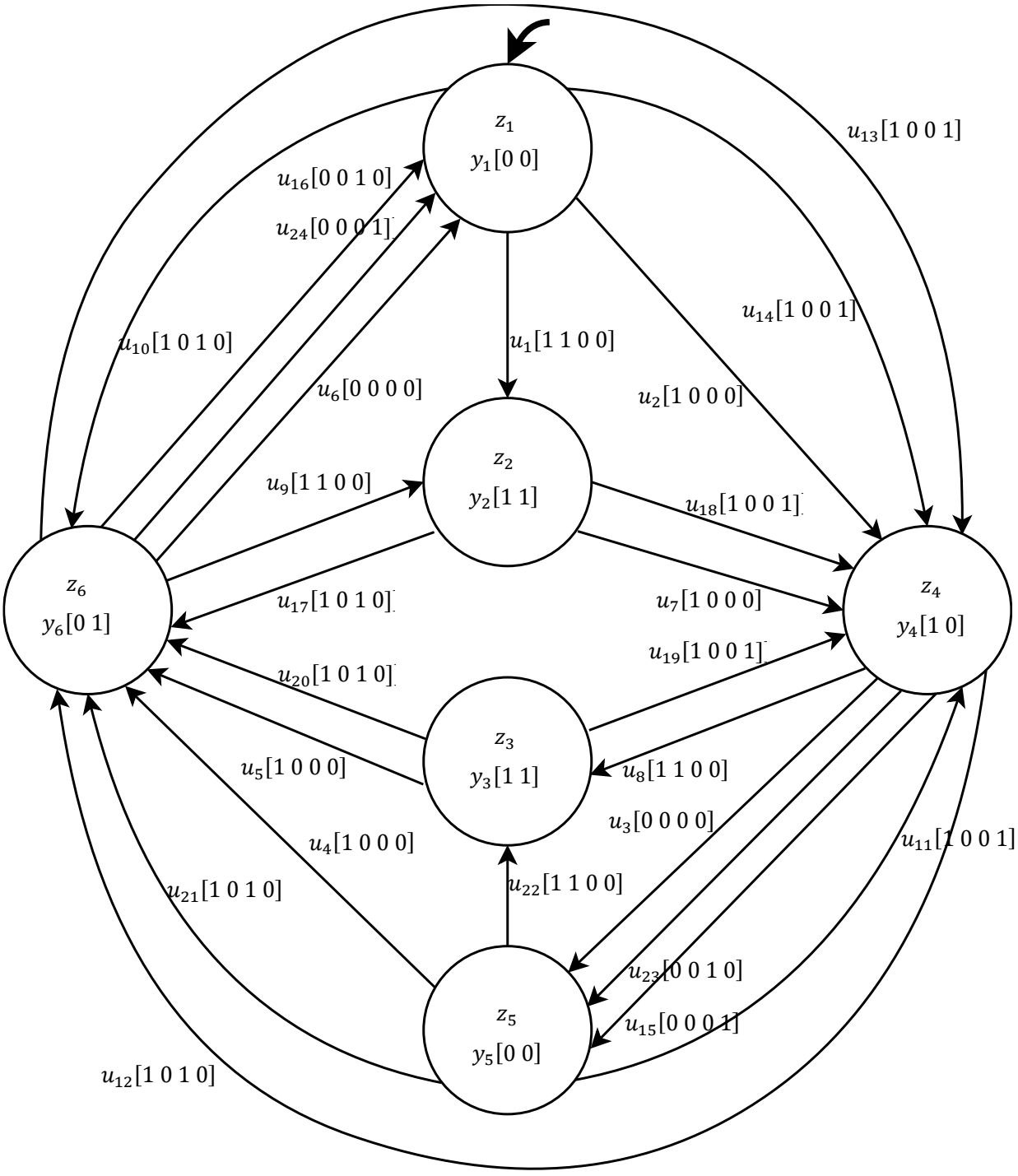


Abbildung 27: Zustandsgraph, Quelle: Eigene Darstellung

Abbildung 27 stellt eine Lösungsvariante der informell spezifizierten Aufgabenstellung aus Kapitel 7.2 als Zustandsgraph dar.

7.3.3 Das Petrinetz

Im Folgenden wird die informelle Spezifikation aus Kapitel 7.2 schrittweise in die formale Spezifikation des Steueralgorithmus überführt. Zwei Motoren, die unabhängig voneinander laufen können, benötigen zwei Stellen. Mindestens eine dritte Stelle wird für die Aktion Null benötigt, für den Fall, dass gerade kein Motor läuft. Es geht aber nicht darum, mit möglichst wenigen Stellen auszukommen, sondern immer darum, den Ablauf so transparent wie möglich darzustellen. Daher können vier, fünf oder mehr Stellen für die Lösung verwendet werden. Zwei Anforderungen der Auflistung nach Kapitel 7.2, nämlich b) und c), lassen sich durch ein steuerungstechnisch interpretiertes Petrinetz (SIPN) nach Abbildung 28 darstellen.

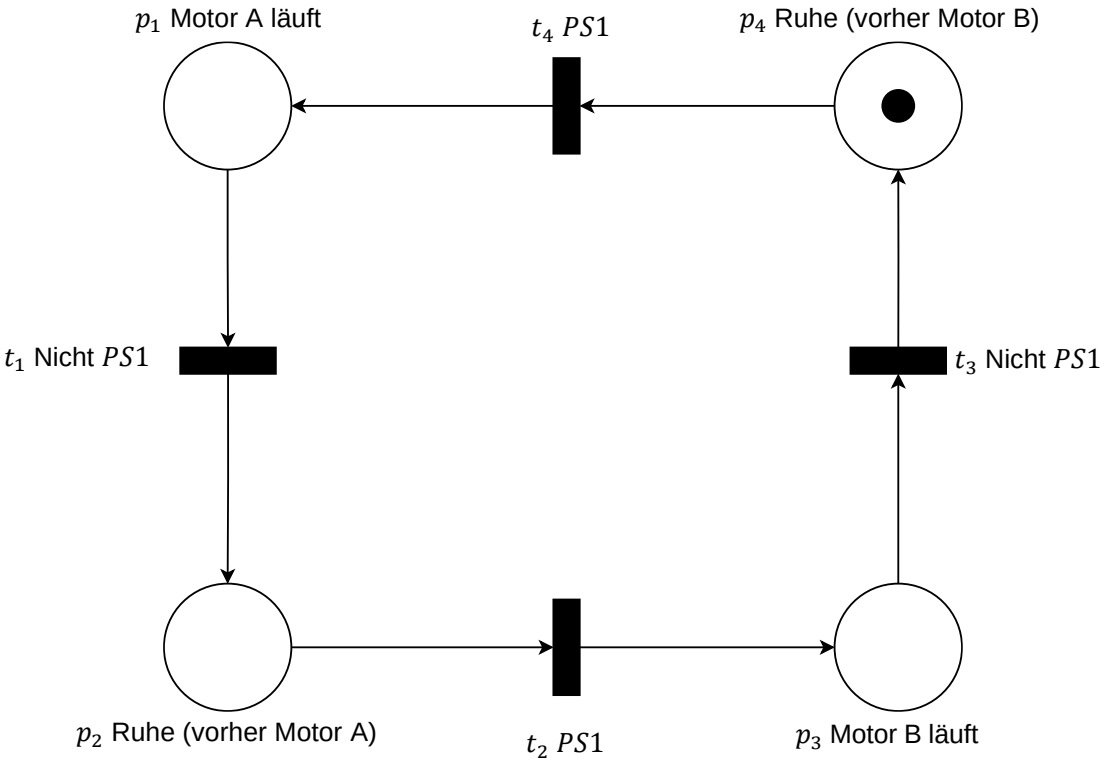


Abbildung 28: SIPN, Quelle: Eigene Darstellung, in Anlehnung an Georg Frey (1999), S. 149

Die Verwendung von vier, statt der unbedingt notwendigen drei Stellen, ermöglicht eine leicht verständliche Struktur, die den Verlauf des Grundzyklus ohne Störungsbehandlung und ohne Ansprache des Sensors $PS2$ berücksichtigt. An der zunächst markierten Stelle p_4 (keiner der beiden Motoren läuft), wartet der Algorithmus auf das Drucksignal $PS1$, das die Markierung nach p_1 (Motor A läuft) überträgt. Die entsprechende Aktion wird so lange ausgegeben, bis das Signal des Drucksensors $PS1$ nicht mehr anliegt. Zu diesem Zeitpunkt wird der Motor A wieder abgeschaltet. Jetzt ist p_2 aktiv, was bedeutet, dass kein Motor läuft. Ein Anliegen des Signals $PS1$ aktiviert p_3, was den Motor B einschaltet. Liegt das Signal $PS1$ nicht mehr an, wird das Netz wieder in den Ausgangszustand mit aktiver Stelle p_4 versetzt. Diese Schaltung wird

nun für den Fall modifiziert, dass neben dem Drucksignal $PS1$ auch $PS2$ anspricht. Es ist zu beachten, dass $PS2$ aus physikalischen Gründen erst anliegen kann, nachdem $PS1$ anliegt.[94]

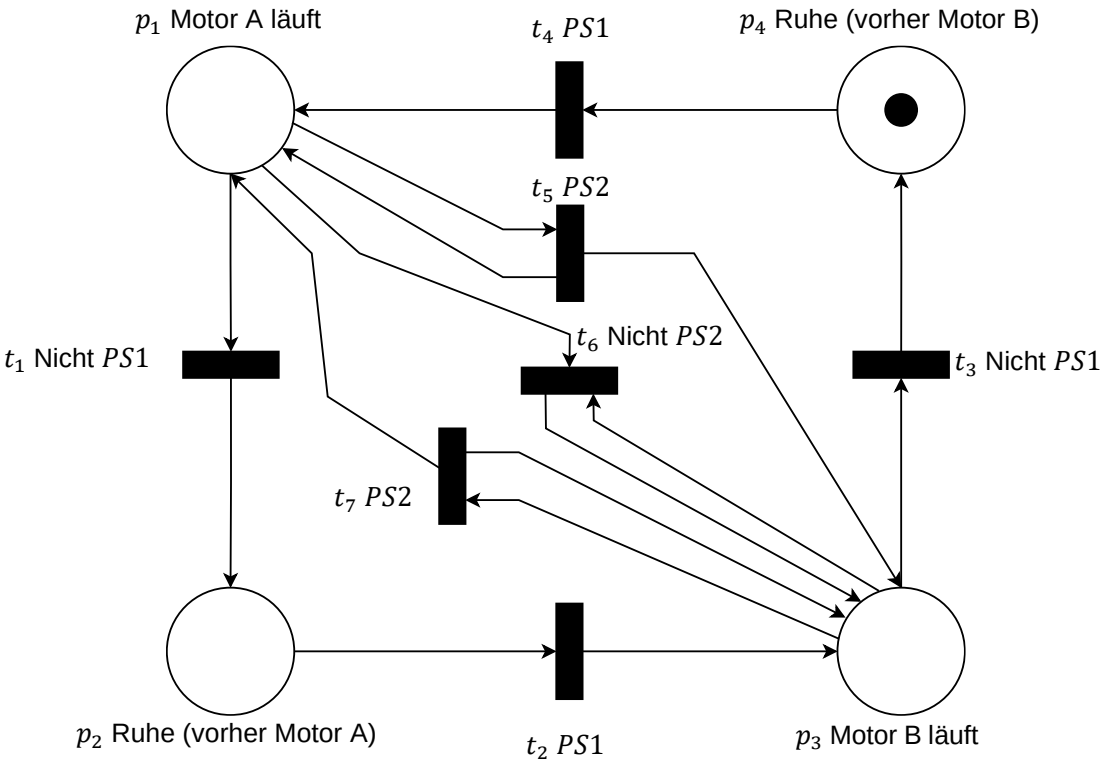


Abbildung 29: SIPN für die Anforderungen a), b) und c), Quelle: Eigene Darstellung, in Anlehnung an Georg Frey (1999), S. 151

Abbildung 29 bildet die erforderliche Erweiterung damit die Anforderungen a), b) und c) der informellen Spezifikation erfüllt werden können.

[94] Vgl. Georg Frey (1999), S. 151

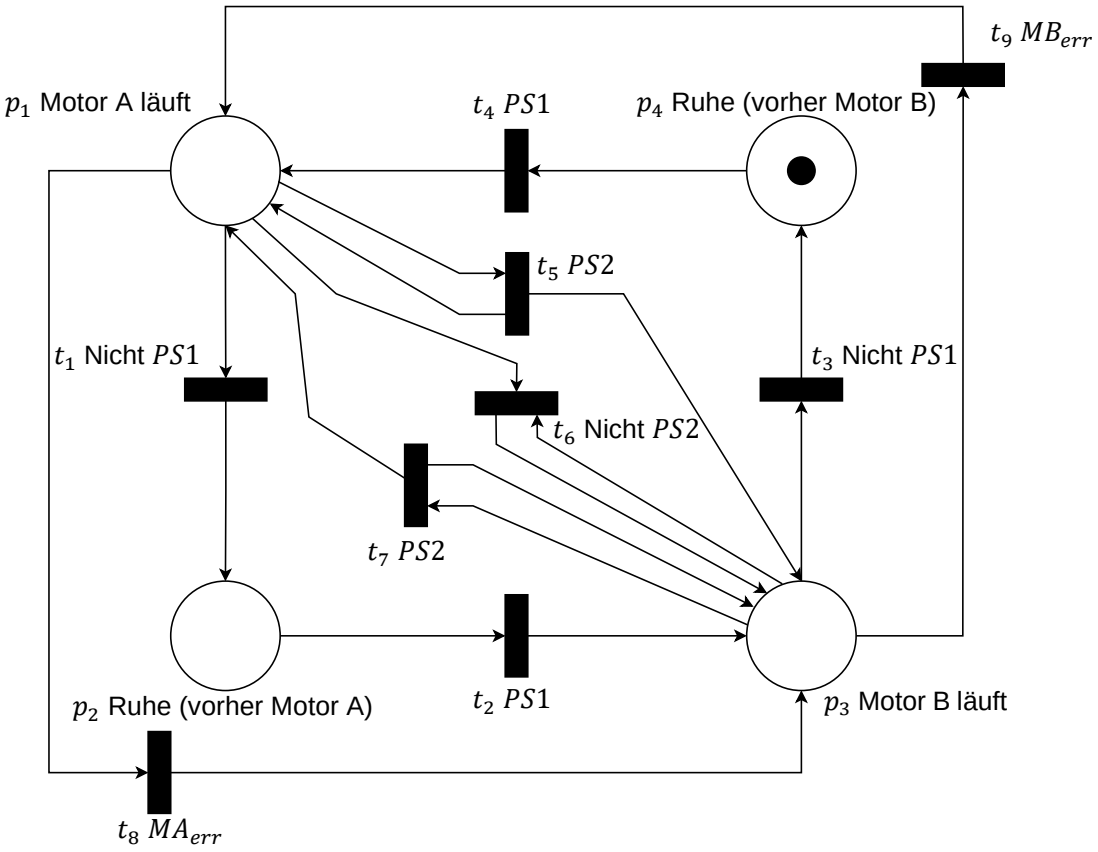


Abbildung 30: Ausgangsmarkiertes SIPN für die Anforderungen a), b), c) und d), Quelle: Eigene Darstellung, in Anlehnung an Georg Frey (1999), S. 149

In einer abschließenden Erweiterung wird nun die Anforderung d) der informellen Spezifikation bearbeitet, sodass dann alle Anforderungen a), b), c) und d) erfüllt sind, siehe Abbildung 30. Die Störungsbearbeitung nach d) hat zu zwei neuen Transitionen t_8 und t_9 geführt. Damit ist sichergestellt, dass eine Störmeldung für den aktuell laufenden Motor, zu einer Umschaltung auf den jeweils anderen Motor führt.[95]

Um Modelle miteinander vergleichen zu können, muss auch das Verhalten des Prozesses modelliert werden. Dies kann auf vielfältige Weise geschehen, wobei insbesondere verschiedene Möglichkeiten für Störungen berücksichtigt werden können. Im vorliegenden Beispiel erfolgte die Modellierung durch ein PIPN, siehe Abbildung 31. Als Störgrößen wurden Motorstörungen (entweder für Motor A oder B), jedoch keine Sensorstörungen vorgesehen. Das PIPN enthält auch implizit die Information, dass das Sensorsignal $PS2$ erst ansprechen kann, nachdem $PS1$ angesprochen hat. Wird die SIPN nach Abbildung 30 mit der PIPN nach Abbildung 31 gekoppelt, dann steht eine formale Spezifikation der gestellten Aufgabenstellung zur Verfügung.

[95] Vgl. Georg Frey (1999), S. 151

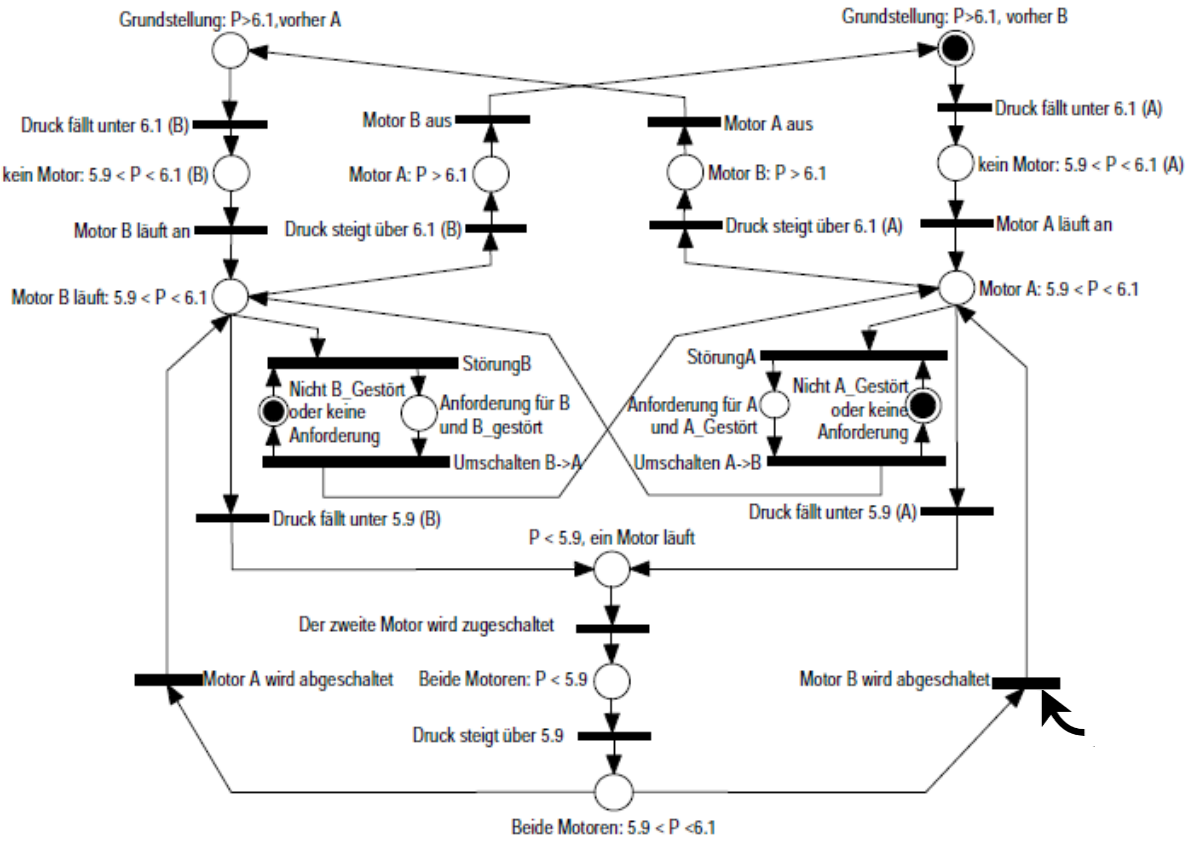


Abbildung 31: Prozessinterpretiertes Petrinetz des Beispiels Windkessel, Quelle: Georg Frey (1999), S. 153

7.3.4 Zustand-Zonen Modell

Das ZZM besteht aus mehreren Ebenen, die nach der Modellierungsreihenfolge vorgestellt werden, siehe Kapitel 5.8.2. Die Anwendung Selmo Studio wird zur Lösung dieses Beispiels als Modellierungsumgebung eingesetzt. In dem folgenden Kapitel wird der Entstehungsprozess eines Zustand-Zonen Modells vorgestellt.

7.3.4.1 Logik-Ebene

In der Logikebene wird der logische Schrittablauf und dessen Verknüpfungen modelliert. Die Logik-Ebene dient dazu, die Schritte des Modells auf einer grafischen Oberfläche zu modellieren. In dieser Ebene können verschiedene logische Elemente in einer Reihenfolge und somit in Beziehung zueinander gestellt werden. Diese grafische Darstellung erleichtert die Modellierung eines Prozesses erheblich.

Das ZZM bildet die möglichen Zustände sowie Zustandsübergänge ab, die entweder vom Druckniveau des Windkessels und/oder den Fehlermeldungen der Motoren abhängen. Wie genau die Elemente der Modellierung eines ZZM eingesetzt werden, wird in Kapitel 5.8.2 detailliert beschrieben. Da diese Methodik der Modellierung Übung und Erfahrung benötigt, wird die Entscheidungs- bzw. Aktionslogik in Worten beschrieben, um Fehlinterpretationen oder Falschannahmen zu vermeiden. Die vorgestellte Schrittkette ist möglicherweise nicht die einzige Lösung für die gestellte Aufgabe, sondern stellt einen möglichen Lösungsweg dar. Da die Modellierung ein Werkzeug mit definierten Regeln, jedoch ohne strikte Vorgabe

ist, können mehrere Lösungen zum gleichen Ergebnis führen. Das Ziel der Modellierung ist jedoch immer dasselbe, die Anforderungen der gestellten Steuervorgabe funktional korrekt und effektiv zu erfüllen.

Die **Vorgehensweise** wie die Logik-Ebene des ZZM modelliert wird, folgt einem einfachen vier Schritte Prinzip:

1. Definiere den Ausgangszustand. Dieser ergibt sich aus der Steuervorgabe und muss eindeutig definiert sein.
2. Ausgehend vom Ausganszustand der Steuervorgabe, werden alle möglichen Zustände definiert, die von diesem Zustand aus erreichbar sind.
3. Sind der aktuelle Zustand und die möglichen Folgezustände klar definiert, so wird der Zustandsübergang definiert. Welche Bedingungen müssen erfüllt werden, um vom aktuellen Zustand auf den Folgezustand zu wechseln. Wenn mehrere Folgezustände möglich sind, muss definiert werden, wie diese Möglichkeiten unter- und entschieden werden.
4. Definiere einen Endzustand. Je nachdem ob es sich um einen kontinuierlichen oder diskontinuierlichen Prozess handelt, kann der Endzustand nicht immer definiert werden. Handelt es sich um einen kontinuierlichen Prozess, so markiert der Endzustand nur das Ende der Schrittfolge und die Schrittkette beginnt von vorne.

Diese Vorgehensweise wird nun schrittweise angewandt, um das ZZM besser zu verstehen. Modelliert wird das ZZM Windkessel. Im ersten Schritt muss der Ausgangszustand ermittelt werden. Da es sich beim Beispiel Windkessel um einen kontinuierlichen Prozess handelt, muss der aktuelle Systemzustand ermittelt werden. Dazu werden Entscheidungen definiert werden, um das aktuelle Druckniveau und somit den Systemzustand am Anfang der Schrittkette zu realisieren.

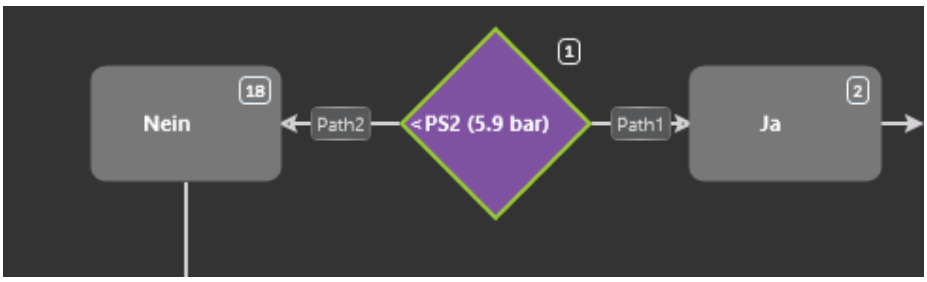


Abbildung 32: Entscheidung PS2, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Abbildung 32 zeigt die erste Entscheidung, ob sich das Druckniveau im Windkessel unter 5,9 bar befindet. Dieser Zustand tritt ein, wenn der Drucksensor $PS2$ anspricht. Der Ausgang dieser Entscheidung kann einen Zustandswechsel auslösen, da im Falle des positiven Ausgangs der Entscheidung, beide Motoren eingeschaltet werden könnten. Damit kann Anforderung a) der Steuervorgabe erfüllt werden. Falls die Entscheidung negativ ausfällt, muss weiter entschieden werden. Bevor beide Motoren betrieben werden können, müssen beide Motoren auf Störungen überprüft werden, siehe Anforderung d). Dazu sind drei Entscheidungen mit der ID 3, ID 5 und ID 13 vorgesehen, siehe Abbildung 33. Nach diesen Entscheidungen steht definitiv fest, ob ein Motor gestört ist und somit der jeweils andere betrieben wird, oder ob beide Motoren betrieben werden. Außerdem werden beide Motoren ausgeschalten, falls diese gestört sind.

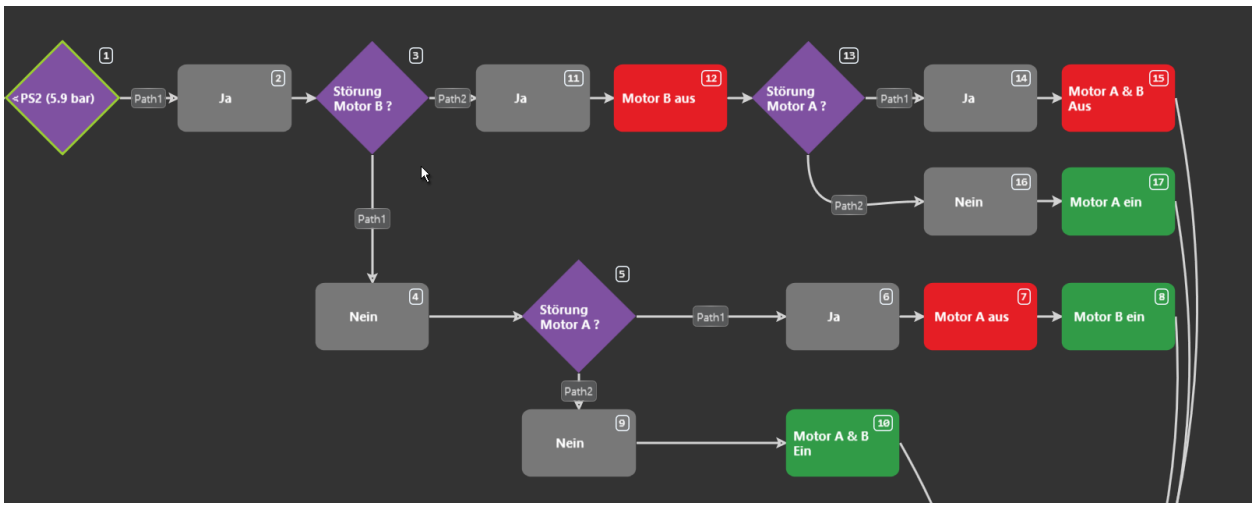


Abbildung 33: Beide Motoren einschalten, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Abbildung 34 zeigt die zweite Entscheidung, ob sich das Druckniveau im Windkessel unter 6,1 bar befindet. Dieser Zustand tritt ein, wenn der Drucksensor $PS1$ anspricht. Der Ausgang dieser Entscheidung kann direkt einen Zustandswechsel auslösen, da im Falle des negativen Ausgangs der Entscheidung beide Motoren ausgeschaltet werden, siehe Schritt mit der ID 45 aus Abbildung 34.

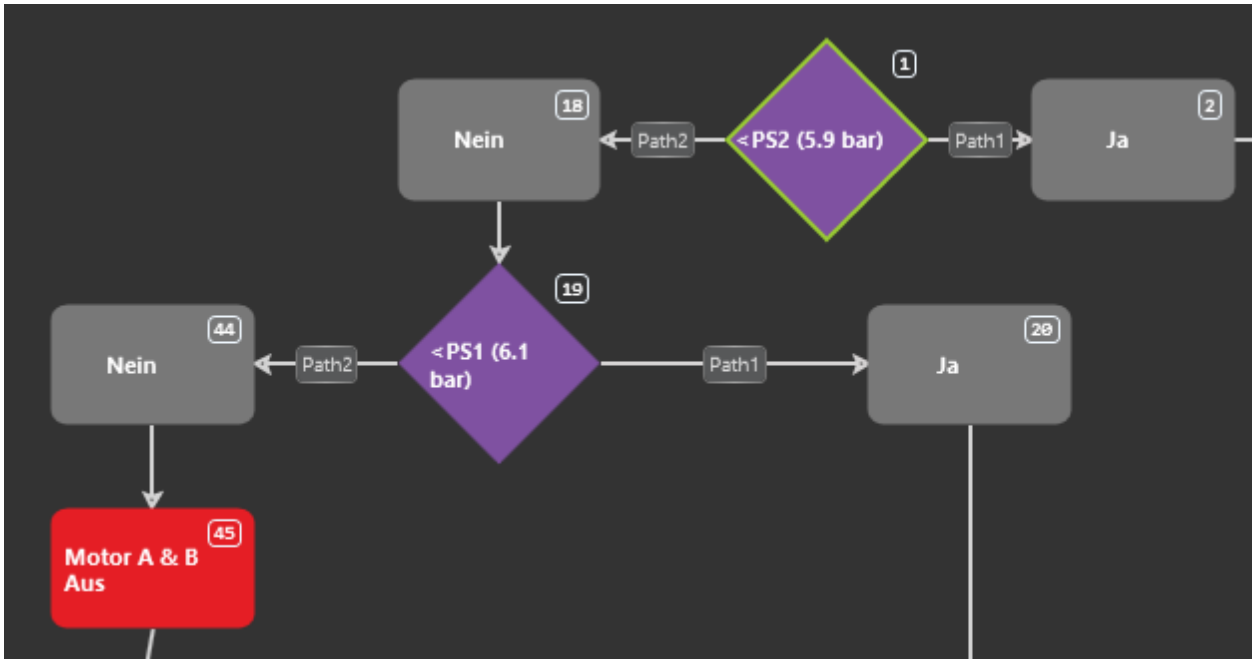


Abbildung 34: Entscheidung PS1, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Kombiniert man die beiden Entscheidungen, so kann man das Druckniveau (5,9 < P < 6,1 bar) ermitteln, siehe Abbildung 35. Damit kann Anforderung a) und b) der Steuervorgabe erfüllt werden. Der Systemzustand kann dadurch eindeutig bestimmt werden. Den Anfang der Schrittkette bildet die grün umrandete Entscheidung, die als Raute dargestellt wird.

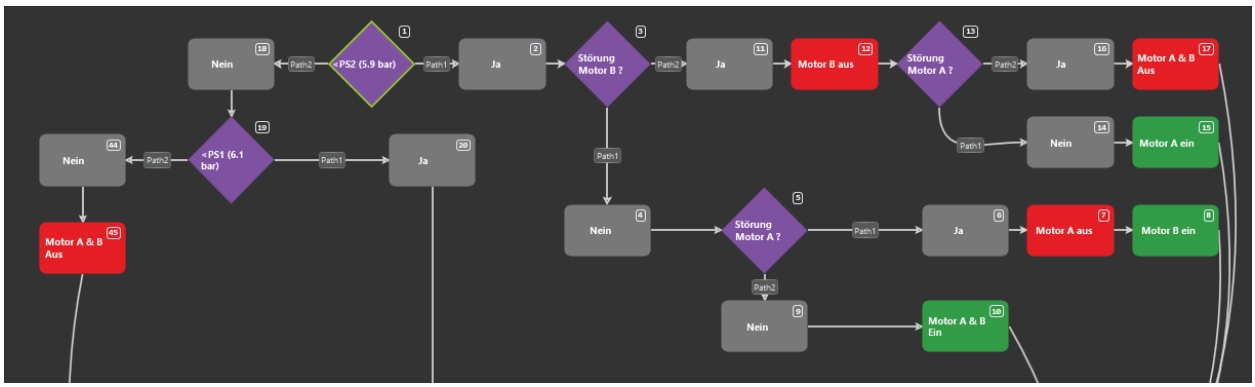


Abbildung 35: Entscheidung PS1 und PS2, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Bevor eine endgültige Entscheidung getroffen werden kann, welcher Motor eingeschaltet wird, ist Anforderung c) von Bedeutung. Diese besagt, dass beide Motoren abwechselnd laufen sollen. Dafür ist die Entscheidung Motor wechseln notwendig, siehe Abbildung 36.

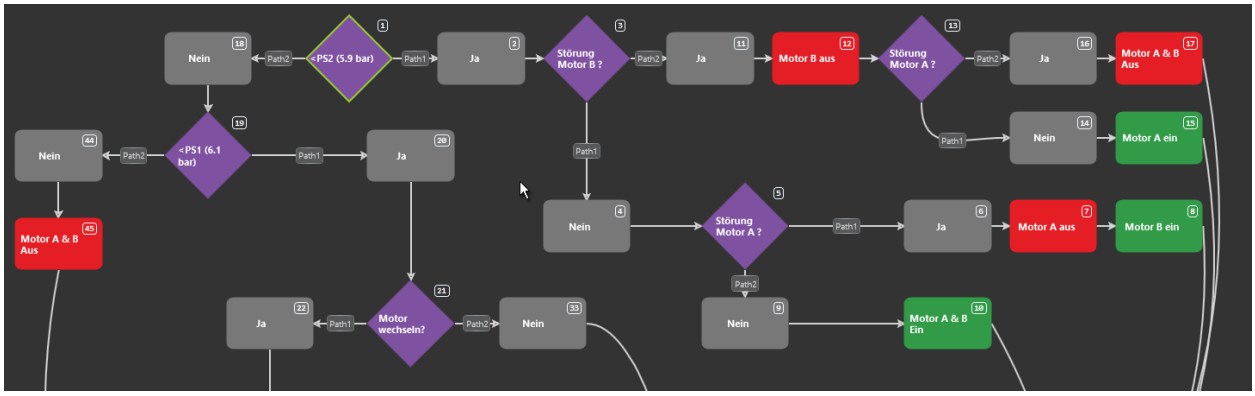


Abbildung 36: Entscheidung Motor wechseln, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Aus Sicht der Aktuatoren, die im Beispiel enthalten sind, existieren nur vier beeinflussbare Zustände. Beide Motoren ein, beide Motoren ausschalten oder jeweils ein Motor ein, siehe Abbildung 37. Durch die Steuervorgabe definiert ist der Ausgangszustand als Windkessel voll (Druck > 6.1 bar) angenommen.

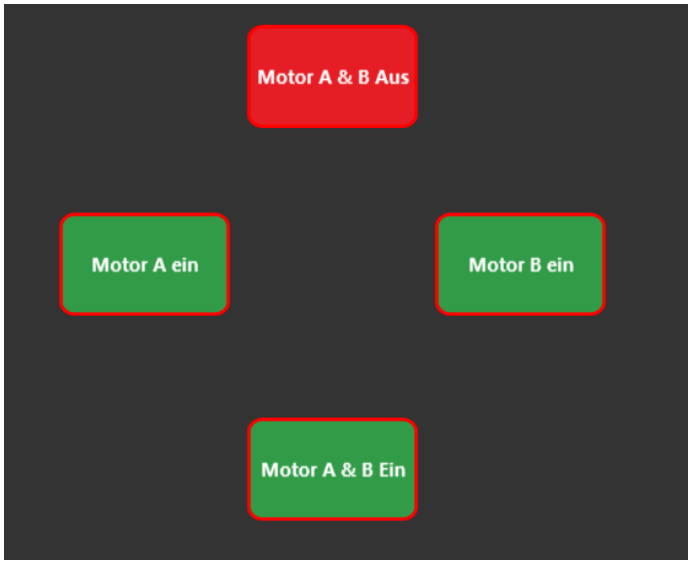


Abbildung 37: Zustände im System, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Um die Anforderung d) zu realisieren ist es notwendig den jeweiligen Motor, der eingeschaltet werden soll auf Störungen zu überprüfen. Dafür sind die Entscheidungen Motor A oder B Störung nach der Entscheidung Motor wechseln notwendig, siehe Abbildung 38. Mit diesem Entscheidungsbaum kann definitiv ein Motor ermittelt werden, der eingeschaltet werden soll. Ist ein Motor gestört, so wird der jeweils andere eingeschaltet. Zusätzlich wird nach dem Einschalten eines Motors auch der jeweils andere ausgeschaltet.

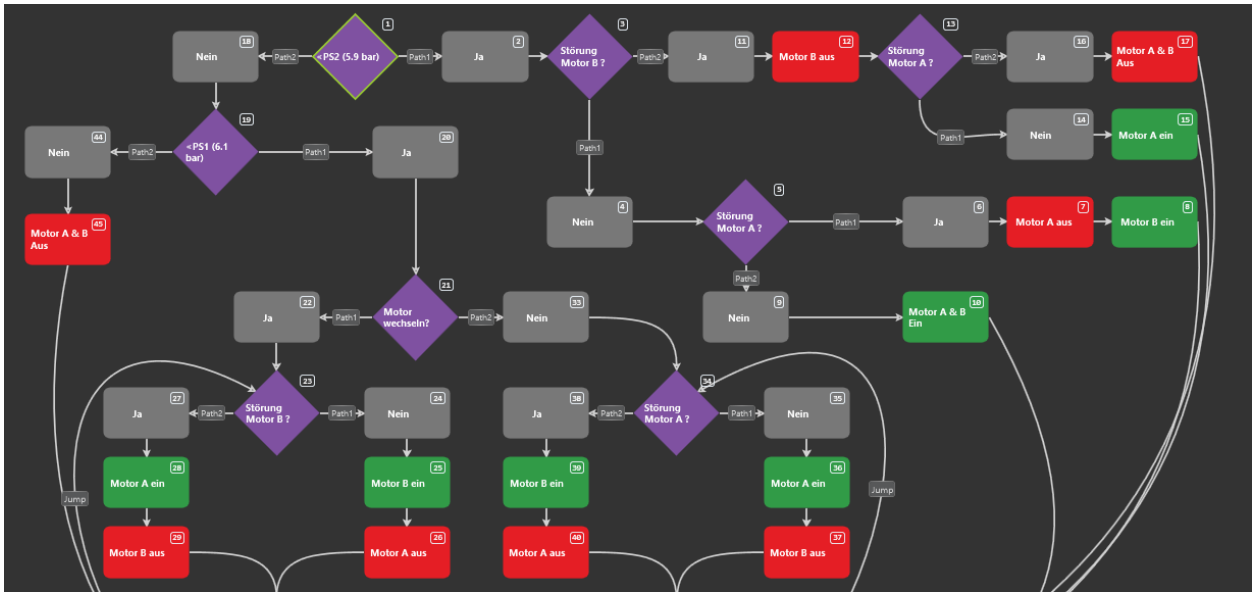


Abbildung 38: Entscheidung Motor Störung, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Wesentlich ist das Kriterium, wann die Entscheidung Motor wechseln getroffen wird. Dazu wird eine Set-Reset Funktion verwendet, siehe Abbildung 43.

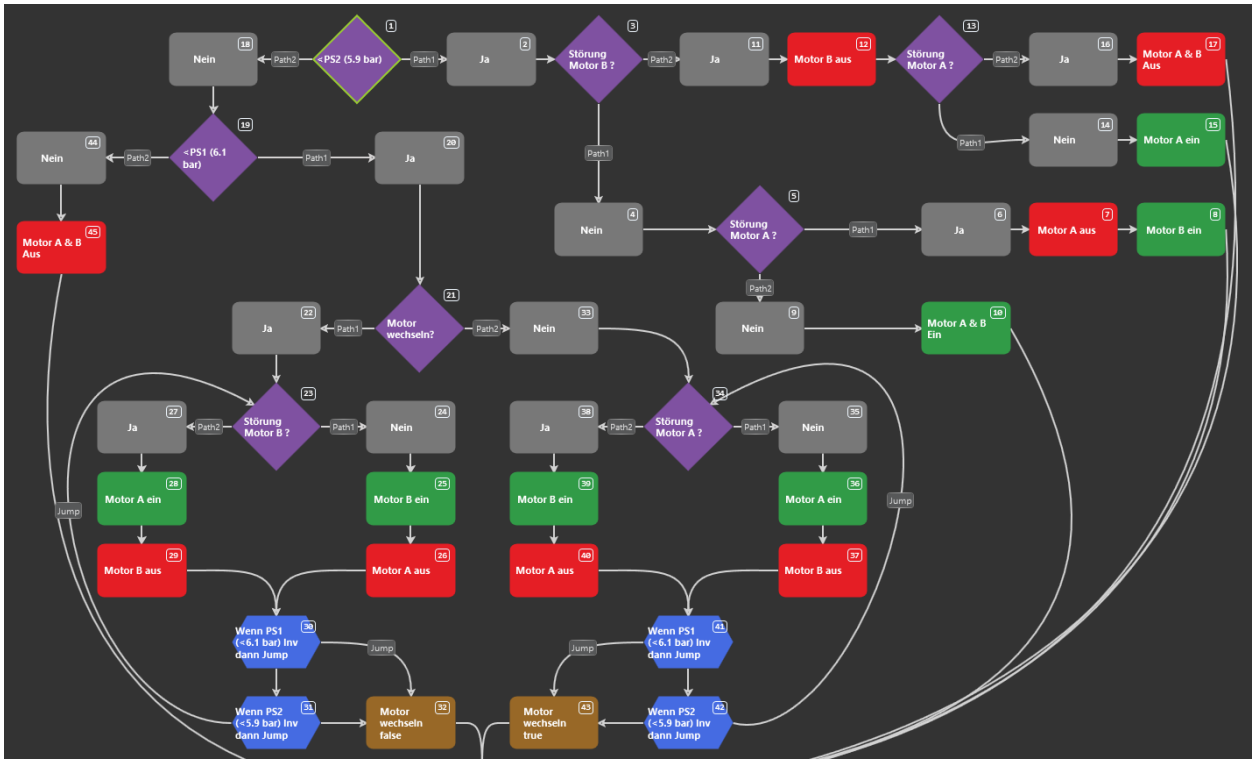


Abbildung 39: Motor wechseln, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Wenn ein einzelner Motor betrieben wird, ist es wichtig sich diesen Zustand zu merken. Merken bedeutet in diesem Fall, das eine Schleife in der Schrittkette vorgesehen ist. Innerhalb dieser Schrittkette wiederholen sich dessen Schritte so lange, bis ein Zustandswechsel erforderlich ist. Dieser kann mehrere Gründe haben. Entweder fällt der Druck aus dem Druckniveau (5,9 < P < 6,1 bar) und der Sensor $PS2$ spricht an, oder der Druck steigt und die Sensoren $PS1$ und $PS2$ sprechen nicht an. Weiters kann noch eine Störung des bereits laufenden Motors auftreten. Dies führt dazu, dass der jeweils andere Motor eingeschaltet werden muss. Deshalb ist in jeder Schleife eine Entscheidung (ID 23 und ID 34) vorgesehen, die bei einer Störung eines Motors, den jeweils anderen in Betrieb nimmt, siehe Abbildung 39.

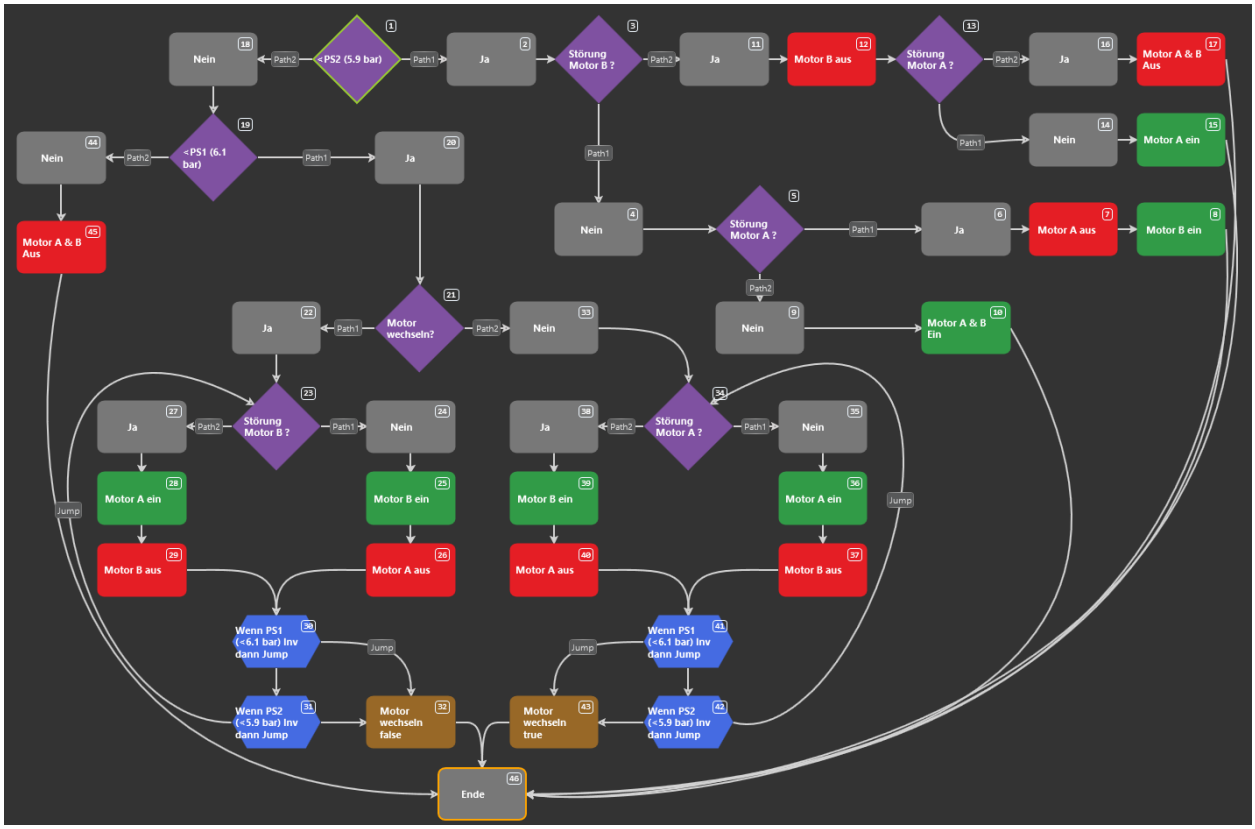


Abbildung 40: Logik-Ebene des ZZM Windkessel, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

In Abbildung 40 ist die Logik-Ebene des ZZM dargestellt. Das Modell erfüllt nun alle notwendigen Anforderungen auf der Logik-Ebene. Abschließend ist noch ein Ende der Schrittkette zu definieren. Wird dieser Endzustand erreicht, fängt die Schrittkette wieder von vorne an. Das Ende der Schrittkette bildet der orange umrandete Schritt, mit der ID 46 und der Bezeichnung Ende. Ein solcher Schritt wird modelliert, um die Lesbarkeit des Modells zu erhöhen und muss keine funktionale Bedeutung haben. Erkennbar ist dies nicht an der Bezeichnung, z.B. Ja, Nein oder Ende, sondern den Operanden in der Systemebene, siehe Abbildung 42.

Dieses Modell ist mit der untergeordneten Systemebene verbunden und kann so zu den entsprechenden Schnittstellen, z.B. Zonen, Parameter etc. verknüpft werden, siehe Abbildung 42. Dabei stellt die Logikebene des ZZM die Elemente dieser Ebene in definierter Reihenfolge dar. In die Systemebene werden Schritte, Sprünge, Entscheidungen etc. aus der Logikebene übernommen und über die eindeutig vergebene ID gereiht.

7.3.4.2 Systemebene

In diesem Bereich erfolgt die Modellierung der jeweiligen Zustände des Prozessablaufs. Dazu werden in jedem Schritt die geforderten Zustände den jeweiligen Zonen zugeordnet. Das Funktionsprinzip ist in Kapitel 5.8.3 näher beschrieben worden. Die wichtigsten Elemente der Systemebene und dessen Verknüpfung zu den anderen Ebenen des ZZM werden kurz vorgestellt, um die Lösung des Beispiels Windkessel besser zu verstehen.

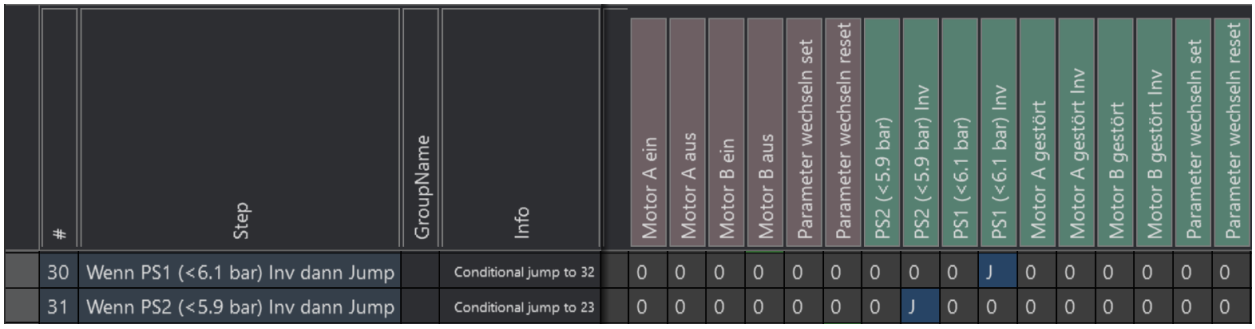
#	Schritte (Steps)	GroupName	Info	Zone 1	Zone 2	…
ID1	Bezeichnung			Operand	Operand	
ID2	Bezeichnung			Operand	Operand	
…	…			…	…	

Tabelle 10: Aufbau Systemebene, Quelle: Eigene Darstellung

Der Aufbau der Systemebene ist schematisch in Tabelle 10 dargestellt und nachfolgend erklärt. Alle Elemente der Logikebene wie z.B. der Schritt, die Entscheidung, der Sprung usw. werden in die Systemebene übernommen, verknüpft und in der Spalte Schritte (Steps) angeführt. In der Spalte mit der Bezeichnung # werden die Schritte anhand ihrer ID angereiht. Die Spalte GroupName dient zur Gruppierung einzelner Schritte. Das Spalte Info dient z.B. bei Entscheidungen zur Übersicht, welche Entscheidungspfade im jeweiligen Fall eintreten. Die Zonen repräsentieren das jeweilige Ein- und/oder Ausgangssignal der Steuerung. Eine Zone steuert und überwacht den jeweiligen Zustand eines bestimmten Signales im jeweiligen Schritt. Es existieren drei Arten von Zonen, siehe Kapitel 5.8.3. In der Lösung des Beispiels Windkessel, werden allerdings nur zwei Arten verwendet. Farblich grün markierte Eingangszonen und rot (leicht gräulich) markierte Ausgangszonen, siehe Abbildung 42.

Die Schrittkette startet mit einer Entscheidung mit ID 1, siehe Abbildung 42. Die Entscheidung wird über zwei spezifische Operanden, D1 und D2 gesteuert. Die Zone, in der der Operand D1 gesetzt wird, entscheidet, ob der Pfand 1 nach erfolgreicher Prüfung gewählt wird. Die Zone, in der der Operand D2 gesetzt wird, entscheidet, ob der Pfand 2 nach erfolgreicher Prüfung gewählt wird. Geprüft werden die mit den Zonen verknüpften Eingangssignale, wobei das Ergebnis dieser Prüfung eindeutig sein muss. In diesem Fall das Signal des Drucksensors $PS2$ über die Zone PS2 (< 5.9 bar) und das invertierte Signal des Drucksensors $PS2$ über die Zone PS2 (< 5.9 bar) Inv. Eingangssignale werden invertiert, um einen Pfad eindeutig bestimmen und zuzuordnen zu können. Eindeutigkeit ist bei der Entscheidung relevant, da sich der Systemzustand eindeutig feststellen lassen muss. Steigt der Druck im Windkessel über 5,9 bar, liefert der Drucksensor $PS2$ über das Eingangssignal den binären Wert 0. Die Zone PS2 (< 5.9 bar) ist somit 0 und die die Zone PS2 (< 5.9 bar) Inv ist folglich 1. Die Entscheidung fällt damit in diesem Beispiel für D2 und somit den Pfad 2 aus. Es folgt der Schritt mit ID 18, siehe Abbildung 42.

Eine Entscheidung ähnelt einer zweiten Funktion der Systemebene, dem Sprung. Der Sprung unterscheidet sich von der Entscheidung dadurch, dass kein zweiter Pfad erstellt wird, sondern nur an eine andere Stelle in der Schrittkette gesprungen wird und diese von dort aus weiterverfolgt wird, ansonsten die Schrittkette direkt weiterverfolgt wird. Das entscheidende Kriterium des Sprungs ist, wie bei der Entscheidung, die Zone, in der der spezielle Operand J gesetzt wird. Der Sprung mit der ID 30 entscheidet deshalb anhand der Zone PS1 (< 6.1 bar) Inv ob der Sprung stattfinden soll, siehe Abbildung 41. Wohin gesprungen werden kann, lässt sich aus der Info des Sprungs entnehmen, in diesen Fall auf den Schritt mit der ID 32. Die Definition und die Kennzeichnung des Sprungs erfolgt in der Logikebene, durch einen wegführenden Pfeil mit der Aufschrift Jump, siehe Abbildung 40.



#	Step	GroupName	Info	Motor A ein	Motor A aus	Motor B ein	Motor B aus	Parameter wechseln set	Parameter wechseln reset	PS2 (<5.9 bar)	PS2 (<5.9 bar) Inv	PS1 (<6.1 bar)	PS1 (<6.1 bar) Inv	Motor A gestört	Motor A gestört Inv	Motor B gestört	Motor B gestört Inv	Parameter wechseln set	Parameter wechseln reset
30	Wenn PS1 (<6.1 bar) Inv dann Jump		Conditional jump to 32	0	0	0	0	0	0	0	0	0	J	0	0	0	0	0	0
31	Wenn PS2 (<5.9 bar) Inv dann Jump		Conditional jump to 23	0	0	0	0	0	0	0	J	0	0	0	0	0	0	0	0

Abbildung 41: Der Sprung in der Systemebene, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Die wichtigsten Funktionen der Systemebene wurden somit vorgestellt. Nachfolgend ist die Gesamtübersicht des ZZM Windkessel in der Systemebene dargestellt, siehe Abbildung 42. Damit sind die Modellierung und Zuweisung der Systemebene abgeschlossen.

#	Step	GroupName	Info	Motor A ein	Motor A aus	Motor B ein	Motor B aus	Parameter wechseln set	Parameter wechseln reset	PS2 (<5.9 bar)	PS2 (<5.9 bar) Inv	PS1 (<6.1 bar)	PS1 (<6.1 bar) Inv	Motor A gestört	Motor A gestört Inv	Motor B gestört	Motor B gestört Inv	Parameter wechseln set	Parameter wechseln reset
1	<PS2 (5.9 bar)		Path 1 jump to 2 Path 2 jump to 18	0	0	0	0	0	0	D1	D2	0	0	0	0	0	0	0	0
2	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	Störung Motor B ?		Path 1 jump to 4 Path 2 jump to 11	0	0	0	0	0	0	0	0	0	0	0	0	D2	D1	0	0
4	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Störung Motor A ?		Path 1 jump to 6 Path 2 jump to 9	0	0	0	0	0	0	0	0	0	0	D2	D1	0	0	0	0
6	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	Motor A aus			0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	Motor B ein			0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
9	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Motor A & B Ein			2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
11	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	Motor B aus			0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
13	Störung Motor A ?		Path 1 jump to 14 Path 2 jump to 16	0	0	0	0	0	0	0	0	0	0	D2	D1	0	0	0	0
14	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	Motor A ein			2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	Motor A & B Aus			0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0
18	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	<PS1 (6.1 bar)		Path 1 jump to 20 Path 2 jump to 44	0	0	0	0	0	0	0	0	D1	D2	0	0	0	0	0	0
20	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	Motor wechseln?		Path 1 jump to 22 Path 2 jump to 33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	D1	D2
22	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	Störung Motor B ?		Path 1 jump to 24 Path 2 jump to 27	0	0	0	0	0	0	0	0	0	0	0	0	D2	D1	0	0
24	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	Motor B ein			0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
26	Motor A aus			0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	Motor A ein			2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	Motor B aus			0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
30	Wenn PS1 (<6.1 bar) Inv dann Jump		Conditional jump to 32	0	0	0	0	0	0	0	0	0	J	0	0	0	0	0	0
31	Wenn PS2 (<5.9 bar) Inv dann Jump		Conditional jump to 23	0	0	0	0	0	0	0	J	0	0	0	0	0	0	0	0
32	Motor wechseln false			0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
33	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	Störung Motor A ?		Path 1 jump to 35 Path 2 jump to 38	0	0	0	0	0	0	0	0	0	0	D2	D1	0	0	0	0
35	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	Motor A ein			2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
37	Motor B aus			0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0
38	Ja			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
39	Motor B ein			0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0
40	Motor A aus			0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
41	Wenn PS1 (<6.1 bar) Inv dann Jump		Conditional jump to 43	0	0	0	0	0	0	0	0	0	J	0	0	0	0	0	0
42	Wenn PS2 (<5.9 bar) Inv dann Jump		Conditional jump to 34	0	0	0	0	0	0	0	J	0	0	0	0	0	0	0	0
43	Motor wechseln true			0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
44	Nein			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
45	Motor A & B Aus			0	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0
46	Ende			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Abbildung 42: Systemebene Beispiel Windkessel, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

7.3.4.3 Baugruppen

Die Baugruppen stellen eine Bibliothek von vorgefertigten Funktionsbausteinen zur Verfügung. Für die Lösung des ZZM Windkessel wurden drei Funktionsbausteine gleichen Typs verwendet, der sogenannten Set-Reset Funktion siehe Abbildung 43. Motor wechseln, Motor A und Motor B bedienen sich dieser logischen Funktion.

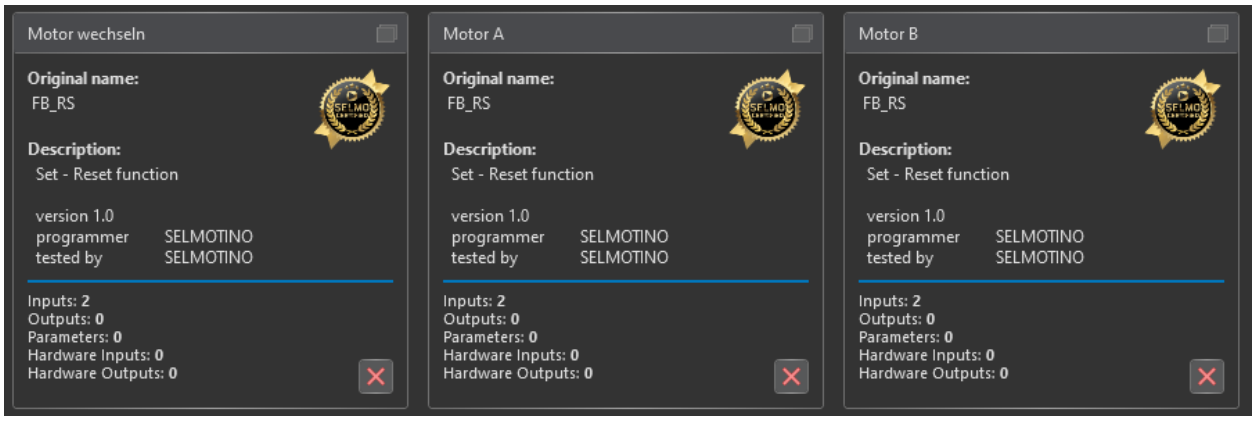


Abbildung 43: Set-Reset Funktion: Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Das Funktionsprinzip der Set-Reset Funktion ist folgendermaßen. Zwei Zonen werden auf ein Ausgangssignal der Steuerung verbunden. Wie z.B. die Ausgangszonen Motor A ein und Motor A aus, siehe Abbildung 42. Diese Zonen werden in der jeweiligen Set-Reset Funktion ausgewählt und sind somit verbunden, siehe Abbildung 44. Wird auf Systemebene nun der entsprechende Operand gesetzt, so wird diese Funktion wirksam.

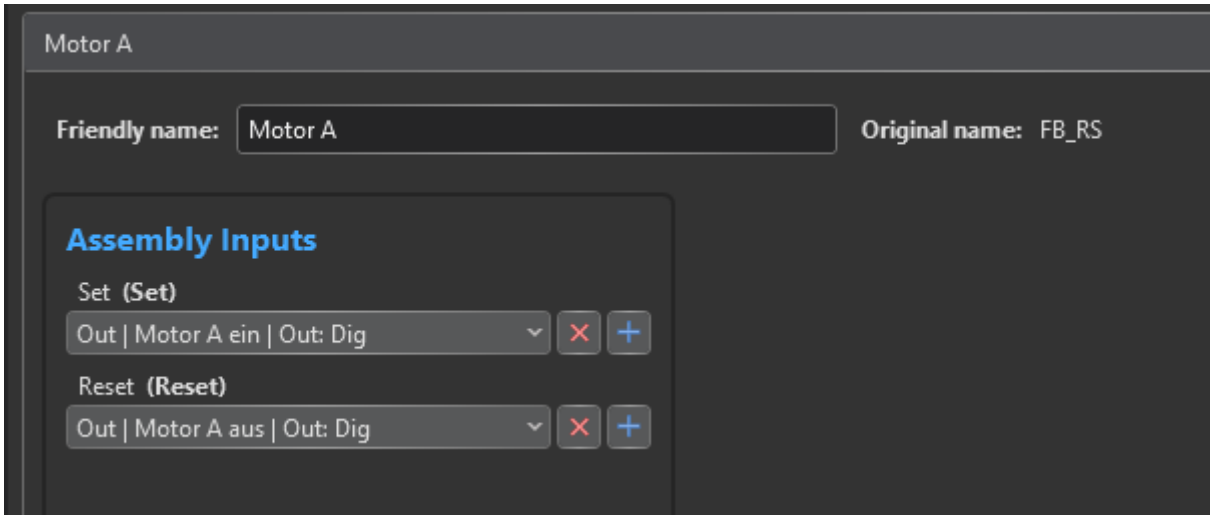


Abbildung 44: Set-Reset Funktion Motor A, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Im Schritt mit der ID 15 wird über den Operanden 2 die Zone Motor A ein aktiviert, siehe Abbildung 42. In der Set-Reset Funktion ist diese Zone als (Set) Zone ausgewählt worden, weshalb der Zustand Motor ein gesetzt wird. Der Zustand Motor ein bedeutet, dass das Ausgangssignal für den Motor A über die Steuerung aktiv bleibt, bis die Zone Motor A aus (Reset) aktiviert wird. Diese Set-Reset Funktion wird genauso auch für Motor B eingesetzt. Die Motor wechseln Funktion nutzt dieselbe Set-Reset Funktion wie Motor A und

Motor B, siehe Abbildung 43. In der Systemebene wird im Schritt mit der ID 32 über die Zone Parameter wechseln reset der Zustandswechsel deaktiviert und im Schritt mit der ID 43 über die Zone Parameter wechseln set aktiviert, siehe Abbildung 42.

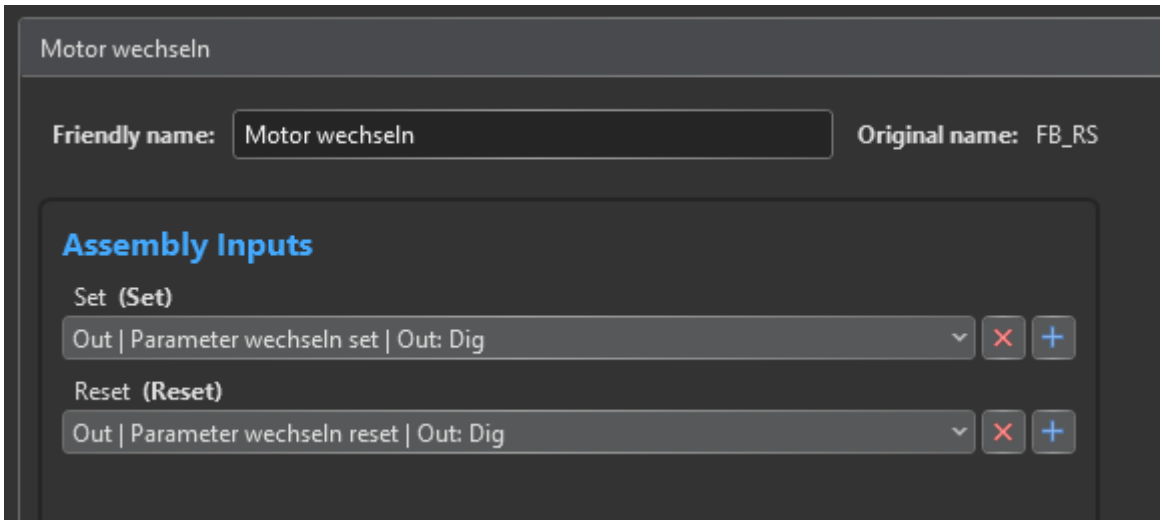


Abbildung 45: Motor wechseln Funktion, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

Die Funktion Motor wechseln hat die Aufgabe, zwischen den beiden Motoren A und B umzuschalten. Die Ausgangszone Parameter wechseln set und Parameter wechseln reset sind dabei die Eingänge der Funktion Motor wechseln, siehe Abbildung 46. Die Funktion Motor wechseln gibt, je nachdem welches Signal gesetzt ist, ein Ergebnis aus. Dieses Ergebnis wird über die Eingangszonen Parameter wechseln set und Parameter wechseln reset abgefragt und wird benötigt, um zwischen den Motoren umzuschalten. Diese Entscheidung wird im Schritt mit der ID 21 über die vorhin erwähnten Eingangszonen Parameter wechseln set und Parameter wechseln reset getroffen. Notwendig ist diese Zuordnung von Ausgangszonen zu Eingangszonen immer dann, wenn eine Entscheidung aufgrund einer Ausgangszone getroffen werden soll. Die Eingangszonen repräsentieren den Wert der entsprechend gesetzten Ausgangszonen. Wobei die Ausgangszone gezielt im Programm gesetzt wird, z.B. im Schritt mit der ID 32 wird die Ausgangszone Parameter wechseln reset durch den Wert 2 des Operanden gesetzt. Im Schritt mir der ID 43 wird die Ausgangszone Parameter wechseln set durch den Wert 2 des Operanden gesetzt, siehe Abbildung 42. Wird die Schrittkette kontinuierlich durchlaufen, so wechseln sich beide Motoren durch die Entscheidung mit der ID 21 ab. Vorausgesetzt beide Motoren sind störungsfrei.

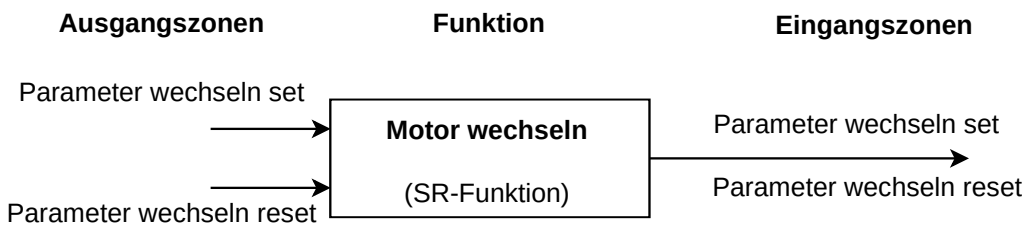


Abbildung 46: Motor wechseln Funktion mit zugewiesenen Zonen, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

7.3.4.4 Manual Cross Interlock Checks (MXIC)

Die Manual Cross Interlock Checks (MXIC) wurden in diesem Beispiel dazu verwendet, den Betrieb der Motoren A und B bei einer Störung zu blockieren. Da das ZZM durch dessen Betriebsarten auch manuell gesteuert werden kann, ist es wichtig den Manuellen Betrieb abzusichern. Dazu werden Ausgangszonen blockiert, wenn definierte Zonen aktiviert sind. Wenn z.B. das Eingangssignal des Motor A gestört aktiv ist, wird im manuellen Betrieb das Einschalten des Motors A über die Ausgangszone Motor A ein blockiert. Sprich diese Kreuzverriegelungen verhindern im Handbetrieb, die ausgewählten Zonen manuell zu betätigen, siehe Abbildung 47.

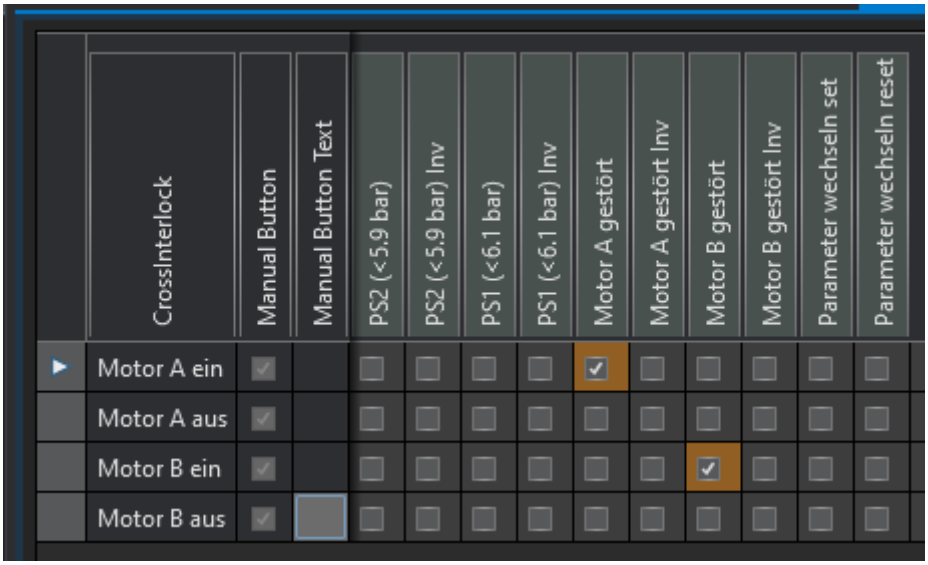


Abbildung 47: MXIC, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

7.3.4.5 Constantly monitored Zone (CMZ)

Constantly Monitored Zone wurden in diesem Beispiel nicht benötigt. Könnten aber eingesetzt werden, um beispielsweise den Not-Aus Schalter der Anlage unabhängig vom aktuellen Schritt und Systemzustand zu überwachen.

7.3.4.6 Parameter

Um den Prozessablauf zu modellieren, sind Prozessparameter notwendig. In dieser Ebene werden bestimmte Entscheidungsparameter oder auch Prozessgrößen definiert. In diesem Modell wurden keine Parameter benötigt, zur Veranschaulichung sind jedoch zwei Parameter Druck 1 und Druck 2 dargestellt, siehe Abbildung 48.

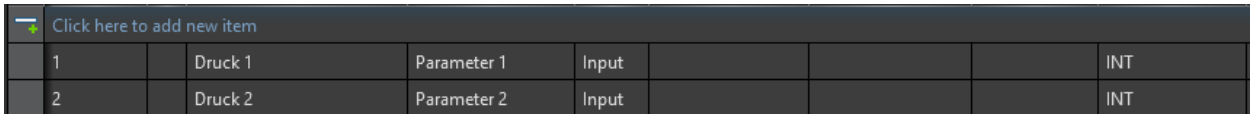
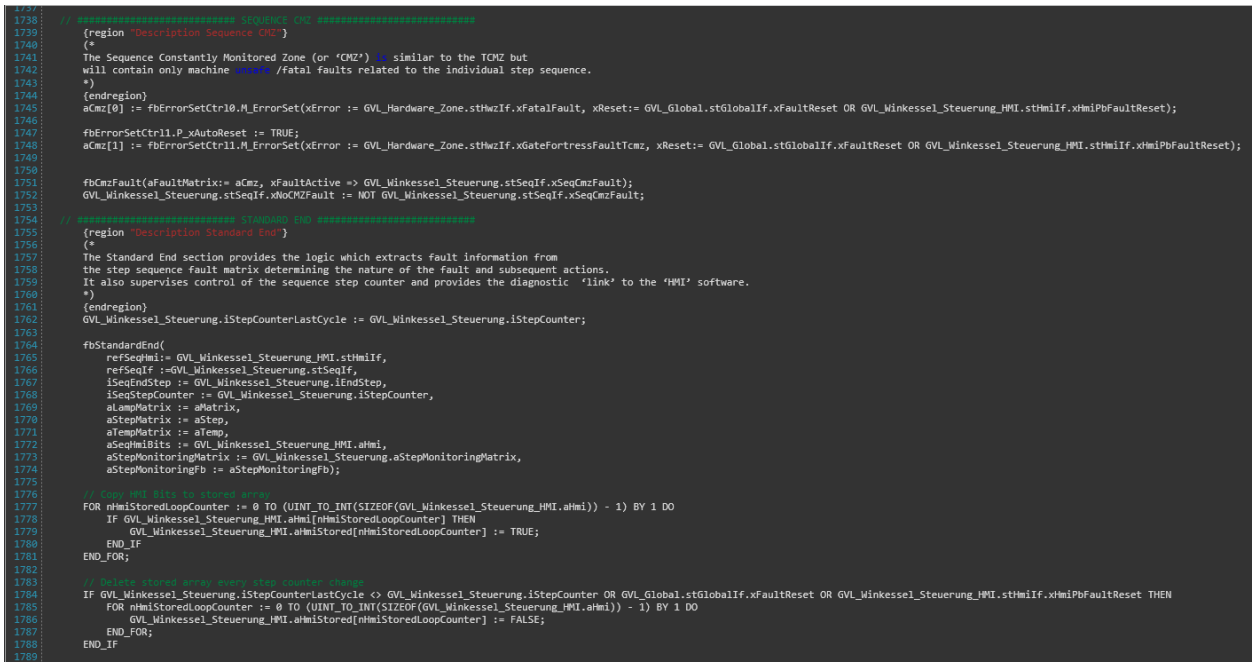


Abbildung 48: Nicht verwendete Parameter, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

7.3.4.7 Das fertige Modell

Um dieses Modell nun einzusetzen, ist kein weiterer Schritt mehr notwendig. Alle Ebenen sind miteinander verbunden. Das Modell ist somit vollständig beschrieben und zur Implementierung in die Steuerung bereit.

In der Modellierungsumgebung ist ein Codegenerator integriert, weshalb das Ergebnis direkt aus dem Modell ohne Zusatzaufwand generiert werden kann. Das aus dem Modell generierte Programm umfasst 1788 Zeilen Code, siehe auszugsweise Abbildung 49.



```
1737
1738    // ######################### SEQUENCE CMZ #########################
1739        {region "Description Sequence CMZ"}
1740        (*
1741        The Sequence Constantly Monitored Zone (or 'CMZ') is similar to the TCMZ but
1742        will contain only machine unsafe /fatal faults related to the individual step sequence.
1743        *)
1744        {endregion}
1745        aCmz[0] := fbErrorSetCtrl0.M_ErrorSet(xError := GVL_Hardware_Zone.stHwzIf.xFatalFault, xReset:= GVL_Global.stGlobalIf.xFaultReset OR GVL_Winkessel_Steuerung_HMI.stHmiIf.xHmiPbFaultReset);
1746
1747        fbErrorSetCtrl1.P_xAutoReset := TRUE;
1748        aCmz[1] := fbErrorSetCtrl1.M_ErrorSet(xError := GVL_Hardware_Zone.stHwzIf.xGateFortressFaultTcmz, xReset:= GVL_Global.stGlobalIf.xFaultReset OR GVL_Winkessel_Steuerung_HMI.stHmiIf.xHmiPbFaultReset);
1749
1750
1751        fbCmzFault(aFaultMatrix:= aCmz, xFaultActive => GVL_Winkessel_Steuerung.stSeqIf.xSeqCmzFault);
1752        GVL_Winkessel_Steuerung.stSeqIf.xNoCMZFault := NOT GVL_Winkessel_Steuerung.stSeqIf.xSeqCmzFault;
1753
1754    // ######################### STANDARD END #########################
1755        {region "Description Standard End"}
1756        (*
1757        The Standard End section provides the logic which extracts fault information from
1758        the step sequence fault matrix determining the nature of the fault and subsequent actions.
1759        It also supervises control of the sequence step counter and provides the diagnostic  'link' to the 'HMI' software.
1760        *)
1761        {endregion}
1762        GVL_Winkessel_Steuerung.iStepCounterLastCycle := GVL_Winkessel_Steuerung.iStepCounter;
1763
1764        fbStandardEnd(
1765            refSeqHmi:= GVL_Winkessel_Steuerung_HMI.stHmiIf,
1766            refSeqIf :=GVL_Winkessel_Steuerung.stSeqIf,
1767            iSeqEndStep := GVL_Winkessel_Steuerung.iEndStep,
1768            iSeqStepCounter := GVL_Winkessel_Steuerung.iStepCounter,
1769            aLampMatrix := aMatrix,
1770            aStepMatrix := aStep,
1771            aTempMatrix := aTemp,
1772            aSeqHmiBits := GVL_Winkessel_Steuerung_HMI.aHmi,
1773            aStepMonitoringMatrix := GVL_Winkessel_Steuerung.aStepMonitoringMatrix,
1774            aStepMonitoringFb := aStepMonitoringFb);
1775
1776        // Copy HMI Bits to stored array
1777        FOR nHmiStoredLoopCounter := 0 TO (UINT_TO_INT(SIZEOF(GVL_Winkessel_Steuerung_HMI.aHmi)) - 1) BY 1 DO
1778            IF GVL_Winkessel_Steuerung_HMI.aHmi[nHmiStoredLoopCounter] THEN
1779                GVL_Winkessel_Steuerung_HMI.aHmiStored[nHmiStoredLoopCounter] := TRUE;
1780            END_IF
1781        END_FOR;
1782
1783        // Delete stored array every step counter change
1784        IF GVL_Winkessel_Steuerung.iStepCounterLastCycle <> GVL_Winkessel_Steuerung.iStepCounter OR GVL_Global.stGlobalIf.xFaultReset OR GVL_Winkessel_Steuerung_HMI.stHmiIf.xHmiPbFaultReset THEN
1785            FOR nHmiStoredLoopCounter := 0 TO (UINT_TO_INT(SIZEOF(GVL_Winkessel_Steuerung_HMI.aHmi)) - 1) BY 1 DO
1786                GVL_Winkessel_Steuerung_HMI.aHmiStored[nHmiStoredLoopCounter] := FALSE;
1787            END_FOR;
1788        END_IF
1789
```

Abbildung 49: Codeausschnitt aus dem Steuerprogramm, Quelle: Eigene Darstellung, mithilfe der Anwendung Selmo Studio

8 GEGENÜBERSTELLUNG DER MODELLIERUNGSMETHODEN

In diesem Kapitel werden die Modelle anhand den Software-Qualitätskriterien bewertet, siehe Kapitel 6 und Kapitel 7. Diese Bewertung erfolgt in drei Schritten. Im ersten Schritt werden Qualitätsanforderungen definiert, im zweiten Schritt die Qualitäts-Metriken festgelegt und im dritten und letzten Schritt werden die Modelle bewertet und gegenübergestellt.

8.1 Bewertungsgrundlage

Die Untersuchungen aus Kapitel 7 haben gezeigt, dass besonders zwei Modellierungsverfahren für die Modellbildung geeignet sind. Ein Zustand-Zonen Modell (ZZM) und ein Petrinetz-Modell werden somit gegenübergestellt, da die Vergleichbarkeit eines Prozessinterpretierten Petrinetzes mit einem ZZM höher ist als es zwischen dem Zustandsgraph und dem ZZM der Fall wäre. Die Qualitätsanforderungen für diese Gegenüberstellung richten sich nach den Qualitätsmerkmalen sowie deren Kriterien von Software, siehe Kapitel 6.2. Um dieselbe Ausgangslage zu schaffen, diente als Grundlage die Prozessdefinition des informell spezifizierten Beispiels aus Kapitel 7.1. In der folgenden Gegenüberstellung werden somit die Modellierungsverfahren, die daraus resultierenden Modelle sowie deren Eigenschaften und Funktionalitäten bewertet und gegenübergestellt.

8.2 Die Qualitätsanforderungen

Um eine Bewertung und Gegenüberstellung zweier Modelle zu gewährleisten, müssen geeignete Anforderungen definiert werden. Die Basis dafür bilden die acht Qualitätsmerkmale sowie deren Kriterien von Softwarequalität, siehe Kapitel 6.2. Untersucht werden die Aspekte dieser Qualitätsmerkmale und die Kriterien, die sich auf die Modellbildung und das Modell selbst beziehen. Die Fragestellungen zum jeweiligen Kriterium (K1-K26) sind in Tabelle 11 dargestellt.

Qualitätsmerkmale	**Fragestellung**
(K1) Funktionale Korrektheit	Wie funktional korrekt konnte die informell spezifizierte Aufgabenstellung umgesetzt werden?
(K2) Vollständigkeit	Wie vollständig wurde die informelle Spezifikation formal im Modell beschrieben? Wie sind alle möglichen Systemzustände abgebildet? Wie konnten Fehlzustände berücksichtigt werden? Wie ist das Modellierungsverfahren für umfangreiche Aufgabenstellungen geeignet?
(K3) Angemessenheit	Wie geeignet ist die Methodik des Modellierungsverfahren, um ein Modell funktional korrekt zu beschreiben?
(K4) Ausgereiftheit	Schützt das Modell gegen unvorhersehbare Störungen des Prozessverhaltens?

(K5) Verfügbarkeit	Wie gut schützt das Modell gegen Ausfälle? Kann ein Wiedereinstieg nach einem Ausfall gewährleistet werden?
(K6) Fehlertoleranz	Werden Fehler im Prozessverhalten toleriert, erkannt und an die entsprechenden Schnittstellen weitergeleitet? Können alle gewollten sowie nicht gewollten Systemzustände abgebildet werden?
(K7) Wiederherstellbarkeit	Kann durch auftretende Fehler im Modell, die Entstehungsstelle erkannt werden?
(K8) Vertraulichkeit	Ist das Modell vor unautorisierten Zugriffen geschützt?
(K9) Integrität	Werden im Modell ausreichende Vorkehrungen getroffen, um die Sicherheit in angrenzenden Systemen sowie die eigens genutzten Ressourcen zu sichern? Kann das Modell gewährleisten, dass das gewünschte Prozessverhalten sicher abläuft?
(K10) Nachweisbarkeit	Wird im Modell überprüft, ob Aktionen oder Ereignisse tatsächlich stattgefunden haben? Ist es möglich die Ausführung nachzuweisen?
(K11) Verantwortlichkeit	Wie gut kann rückverfolgt werden, welche Elemente des Modells auftretende Ereignisse hervorrufen?
(K12) Erlernbarkeit	Wie hoch ist der Aufwand das Modellierungsverfahren zu erlernen? Erfordert das Modellierungsverfahren spezielle Vorkenntnisse?
(K13) Bedienbarkeit	Wie intuitiv und einfach sind die Methoden des Modellierungsverfahrens anzuwenden?
(K14) Fehlertoleranz	Werden Fehler in der Modellierung erkannt, oder deren Erkennung erleichtert?
(K15) Verständlichkeit	Sind die Methoden und das Konzept des Modellierungsverfahren klar strukturiert und verständlich aufgebaut?
(K16) Anpassbarkeit	Können nachträgliche Änderungen effektiv und effizient in das Modell einfließen?
(K17) Installierbarkeit	Wie integrierbar ist das Modell in eine definierte Zielumgebung?
(K18) Austauschbarkeit	Wie hoch ist der Aufwand, das Modell durch ein anderes in derselben Umgebung zu ersetzen?
(K19) Modularität	Bestehen die Modelle aus logischen Einheiten oder Komponenten mit klar abgegrenzter Aufgabenteilung?
(K20) Wiederverwendbarkeit	Können Teile oder ganze Modelle effizient und effektiv wiederverwendet werden?

(K21) Analysierbarkeit	Wie lässt sich die Bedeutung und der Schwerpunkt einer Änderung eines Modells erkennen? Unterstützt das Modellierungsverfahren einen klar strukturierten Entwurf, eine unkomplizierte Implementierung sowie eine einfache Testmöglichkeit?
(K22) Modifizierbarkeit	Wie einfach ist es das Modell an neue oder zukünftige Anforderungen anzupassen?
(K23) Verifizierbarkeit	Wie gut unterstützt das Modell die Methoden der Verifikation sowie des Tests, um gestellte Anforderungen zu erfüllen?
(K24) Antwortverhalten	Kann das Modell entsprechend auf Ereignisse reagieren?
(K25) Verbrauchsverhalten	Wie hoch ist der Bedarf an Betriebsmittel zur Erfüllung der Funktionalität?
(K26) Interoperabilität	Ist das Modell in der Lage, Informationen mit anderen Modellen oder Systeme störungsfrei auszutauschen?

Tabelle 11: Fragestellung zu den Bewertungskriterien, Quelle: Eigene Darstellung

8.3 Bewertungs-Metrik

In Tabelle 12 ist das Bewertungsschema dargestellt, das in Bezug auf die Bewertung der Qualitätsmerkmale in dieser Arbeit gilt.

Bewertungsschema	Note	Bedeutung
Sehr gut	**1**	Die gestellten Anforderungen sind weit über dem wesentlichen Ausmaß erfüllt worden.
Gut	**2**	Die gestellten Anforderungen sind in einem über den wesentlichen Bereichen hinausgehendem Ausmaß erfüllt worden.
Befriedigend	**3**	Die gestellten Anforderungen sind in den wesentlichen Bereichen zur Gänze erfüllt worden.
Genügend	**4**	Die gestellten Anforderungen sind in den wesentlichen Bereichen überwiegend erfüllt worden.
Nicht genügend	**5**	Die gestellten Anforderungen sind in den wesentlichen Bereichen überwiegend nicht erfüllt worden.

Tabelle 12: Bewertungsschema, Quelle: Eigene Darstellung

8.4 Die Gegenüberstellung und Bewertung der Modellierungsmethoden

Die Gegenüberstellung sieht vor, aufgrund der Fragestellungen zum jeweiligen Kriterium aus Tabelle 11 eine Bewertung abzugeben. Jedes Kriterium wird einmal bewertet. Die Bewertung der Kriterien des jeweiligen Qualitätsmerkmals wird zu einer Bewertung jedes einzelnen Qualitätsmerkmals

zusammengefasst. Die Gesamtbewertung am Ende bildet sich schließlich aus dem Mittelwert der bewerteten Qualitätsmerkmale. Um einen Bewertungsschnitt am Ende zu erlangen, bildet sich die Gesamtbewertung aus der Summe der bewerteten Qualitätsmerkmale, dividiert durch die Anzahl der Qualitätsmerkmale.

Qualitätsmerkmale	**Zustand-Zonen Modell**	**Petrinetz-Modell**
(K1) Funktionale Korrektheit	2	4
(K2) Vollständigkeit	2	3
(K3) Angemessenheit	1	3
Funktionale Eignung (FUE)	**2**	**3**
(K4) Ausgereiftheit	3	3
(K5) Verfügbarkeit	1	4
(K6) Fehlertoleranz	1	4
(K7) Wiederherstellbarkeit	2	3
Zuverlässigkeit (**ZUV**)	**2**	**4**
(K8) Vertraulichkeit	1	1
(K9) Integrität	1	4
(K10) Nachweisbarkeit	1	3
(K11) Verantwortlichkeit	1	2
Sicherheit (**SIH**)	**1**	**3**
(K12) Erlernbarkeit	2	2
(K13) Bedienbarkeit	3	3
(K14) Fehlertoleranz	2	3
(K15) Verständlichkeit	2	2
Benutzbarkeit (**BBK**)	**2**	**2**
(K16) Anpassbarkeit	1	1
(K17) Installierbarkeit	1	4
(K18) Austauschbarkeit	1	3
Übertragbarkeit (**ÜBK**)	**1**	**3**
(K19) Modularität	1	2
(K20) Wiederverwendbarkeit	3	4
(K21) Analysierbarkeit	3	3
(K22) Modifizierbarkeit	1	2
(K23) Verifizierbarkeit	3	3

Wartbarkeit (**WBK**)	**2**	**4**
(K24) Antwortverhalten	1	3
(K25) Verbrauchsverhalten	3	3
Effizienz (**EFF**)	**2**	**3**
(K26) Interoperabilität	1	3
Kompatibilität (**KOM**)	**1**	**3**
Gesamtbewertung	13/8 = 1,625 **Gut**	25/8 = 3,125 **Befriedigend**

Tabelle 13: Gesamtbewertung, Quelle: Eigene Darstellung

Die Gegenüberstellung aus Tabelle 13 zeigt ein klares Ergebnis. Das ZZM kann in Summe über alle Qualitätsmerkmale, die dem Modell zuzuordnen sind, qualitativ überzeugen. Das Vergleichsmodell, das nach der Methodik der Petrinetze erstellt wurde, erzielte im Wesentlichen eine befriedigende Gesamtbewertung, lag jedoch im direkten Vergleich mit dem ZZM etwas zurück. Anhand der Qualitätsmerkmale werden die wichtigsten Ergebnisse aus der Bewertung und Gegenüberstellung zusammengefasst:

- **FUE**: Die Methodik des ZZM erlaubt es, ein vollständiges und funktional korrektes Modell mit einer minimalen Anzahl an Systemzuständen zu definieren. Durch ein einzigartiges Funktionsprinzip können alle geforderten Systemzustände, inklusive aller Fehlermöglichkeiten, abgebildet werden, siehe Kapitel 5.8. Im Petrinetz-Modell finden sich ebenfalls alle geforderten Systemzustände wieder. Lediglich die Fehlerzustände können nur mit großem Aufwand vollständig abgebildet werden. Wird der Umfang einer Steuervorgabe größer, wird der Aufwand ein Petrinetzes vollständig zu beschreiben, durch den Anstieg der möglichen Systemzustände, größer.
- **ZUV**: Das ZZM ist so konzipiert, dass in jedem Schritt der Schrittkette, das modellierte Prozessabbild mit dem aktuellen Prozessabbild verglichen wird, siehe Abbildung 20. Abweichungen oder Störungen des Prozessverhaltens werden somit zuverlässig in jedem Schritt und zu jedem Zeitpunkt des Prozessablaufs erkannt. Dagegen werden im Petrinetz-Modell nur jene Abweichungen vom Prozessverhalten detektiert und verhindert, die im Vorhinein als solche definiert wurden, siehe Kapitel 7.3.3.
- **SIH**: Das Petrinetz-Modell sieht keine Überprüfung, ob Ereignisse oder Aktionen wirklich stattgefunden haben, vor. Speziell wenn der Umfang der Steuervorgabe steigt, wird die Sicherheit im Petrinetz-Modell durch unvorhersehbare Fehlerzustände beeinträchtigt. Mit steigendem Umfang der Steuervorgabe, steigt die Komplexität diese Steuervorgabe durch ein ZZM zu beschreiben, linear an, siehe Kapitel 7.3.4.2. Unabhängig vom Umfang der Steuervorgabe kann somit das ZZM die Sicherheit gewährleisten.
- **BBK**: Der Aufwand, um die Methodik einer Modellierung zu meistern, ist eine herausfordernde Tätigkeit, die einiges an Übung abverlangt. Beide Methoden sind nicht gänzlich ohne systemtheoretischem Basiswissen anwendbar. Die Methodik des ZZM ist durch ihre klar aufgeteilte Ebenen-Struktur, optisch ansprechenden Elemente und einfachen Schaltregeln, jedoch ohne

großen Aufwand erlernbar. Die Methodik der Petrinetze ist ebenfalls klar strukturiert und besitzt klar definierte Schaltregeln. Dem entgegen wird ein Petrinetz mit größerem Modellierungsaufwand derart unübersichtlich, dass der Vorteil der einfachen Schaltregeln nicht zur Geltung kommen kann.

- **ÜBK**: Beide Modellierungsmethoden erfordern bei einer Änderung des Zielsystems oder durch abgeänderte Nutzungsanforderungen eine gewisse Überprüfung. Da das ZZM so aufgebaut ist, dass gezielt nur die betroffene Ebene betrachtet werden kann, erleichtert dies die Modellüberprüfung ungemein. Das Petrinetz-Modell kommt hingegen nicht ohne eine komplette Modellüberprüfung aus. Wird einem Petrinetz-Modell ein Signal zu einem bestehenden Modell hinzugefügt, so müssen alle möglichen Systemzustände, sowie das bereits bestehende Modell erneut überprüft werden. Wird dem ZZM ein weiteres Signal hinzugefügt, kann die Systemebene angepasst werden, ohne dabei das bereits vorhandene System abändern zu müssen. Dieser Vergleich zeigt, wie unterschiedlich die Wirkung einer Änderung auf beide Modelle wirkt.
- **WBK**: Die Struktur des ZZM besteht aus Ebenen, die durch Verknüpfungen miteinander in Beziehung stehen. Die Abgrenzung des Aufgabengebiets ist somit klar definiert. Das Petrinetz-Modell besteht zu einem großen Teil aus einer klar definierten Struktur, mit geregelter Aufgabenteilung. Werden Teile oder gar ganze Petrinetz-Modelle kopiert, leidet die gesamte Softwarequalität erheblich unter diesen Umständen. Fehler im Prozessverhalten können oft nur durch praktische Tests oder zeitaufwändige Simulationsverfahren aufgedeckt werden. Da jede Änderung einen Einfluss auf das Systemverhalten hat, kann und wird praktisch nie das gesamte Modell nach einer Änderung verifiziert und validiert. Auch wenn die Verifizier- und Validierbarkeit beim Petrinetz-Modell gegeben ist, führt jede Modelländerung zu einer Änderung des gesamten Prozessverhaltens. Auch das ZZM ist vollständig verifizier- und validierbar. Der Unterschied ist jedoch, dass Änderungen nur auf ein bestimmtes Systemverhalten Einfluss nehmen. Somit kann gezielt verifiziert und validiert werden, ohne dabei nach jeder Änderung das gesamte Modell verifizieren bzw. validieren zu müssen.
- **EFF**: Das ZZM ist so aufgebaut, dass auf Ereignisse jeglicher Art entsprechend reagiert werden kann. Dies resultiert aus der einzigartigen Schaltlogik, siehe Kapitel 5.8.6. Der Aufwand ist durch die standardisierte Modellierungsmethode des ZZM ohne erheblichen Mehraufwand, im Vergleich zum Petrinetz-Modell, realisierbar. Das Petrinetz-Modell reagiert auf Ereignisse nur wenn dies auch vorgesehen ist. Dadurch kann die Qualität des Modells nicht gänzlich erfüllt werden.
- **KOM**: Das ZZM ist plattformunabhängig und kann in jeglicher Softwareumgebung eingesetzt werden. Das Petrinetz-Modell kann nur eingeschränkt oder durch Änderungen des Modells in verschiedenen Softwareumgebungen eingesetzt werden.

9 ZUSAMMENFASSUNG UND AUSBLICK

Die Ausarbeitung und die Gegenüberstellung des Zustand-Zonen Modells und des Petrinetz-Modells haben gezeigt, wie Softwarequalität entsteht und in einem Modell abgebildet werden kann. Betrachtet man den Steuerungsentwurfsprozess, kann man feststellen, dass sich Qualität über drei Phasen hinweg bilden kann. In der Prozessdefinition wird eine informelle Spezifikation der Steuervorgabe gebildet. Diese umfasst besonders die zu erfüllenden Funktionalitäten sowie alle gestellten Anforderungen an die zu erstellende Software. Da alle Qualitätsmerkmale von Software miteinander korrelieren können, wird bereits in der Prozessdefinition der Rahmen der erreichbaren Softwarequalität festgelegt. Weder die Modellbildung noch die Programmerstellung können diesen Rahmen erweitern. Sie dienen dazu, die in der Prozessdefinition enthaltene Qualität zu sichern und zu übertragen. Anhand dieser Erkenntnisse kann der Fokus auf die Prozessdefinition und die Modellbildung gelegt werden. Über die systemtheoretische Betrachtung der Grundsätze vieler Modellierungsmethoden konnte ermittelt werden, welche etablierte Modellierungsmethode sich am besten zur formalen Beschreibung einer informell spezifizierten Steuervorgabe eignet. Die Methodik der Petrinetze wurde anhand dieser Vorgehensweise für eine Gegenüberstellung mit der des ZZM ausgewählt. Da diese neue Methodik zur Modellierung des Zustand-Zonen Modells kaum bekannt ist, wurde diese in einem umfangreichen Kapitelabschnitt vorgestellt.

Die Gegenüberstellung erforderte bestimmte Anforderungen, an denen die Qualität des Modells bewertet worden ist. Es wurden ausgewählte Fragen zu den einzelnen Kriterien der Qualitätsmerkmale von Software definiert und eine eigene Softwarequalitäts-Metrik zur Messung von Qualitätseigenschaften definiert. Damit ist das Bewertungsschema voll definiert. Um ein messbares und verifizierbares Ergebnis zu erlangen, wurde dasselbe praktische Beispiel als Steuervorgabe ausgewählt. Eine Windkesselsteuerung soll dafür sorgen, dass das Druckniveau im Windkessel trotz unvorhersehbarer Luftentnahme stabil bleibt. Dafür können zwei Motoren, zwei separate Kompressoren antreiben, um der Druckabnahme entgegenzuwirken. Dieses Beispiel wurde mit allen Anforderungen informell spezifiziert und dient als Steuervorgabe, um daraus ein formales Modell abzubilden. Die Modellbildung im praktischen Teil der Arbeit erforderte besonderes Augenmerk, da eine formale Beschreibung eines Modells eine gewisse Erfahrung benötigt und eine herausfordernde Tätigkeit darstellt. In dieser Phase zeichnet sich bereits ein klares Bild zur Anwendbarkeit und dem Umgang der Modellierungsmethoden ab. Das Petrinetz-Modell konnte mit annährend gleichem Aufwand, gegenüber dem ZZM, die Steuervorgabe darstellen. Mit dem Unterschied, dass das Petrinetz-Modell aufwendiger zu beschreiben ist, je mehr Systemzustände im Modell zu berücksichtigen sind. Dieser Umstand kann dazu führen, dass das Modell unvollständig beschrieben wird und sich daraus Fehlerzustände ergeben können. Besonders die Überwachung dieser Fehlerzustände haben dazu beigetragen, die Komplexität der Modellbildung stark zu erhöhen. Dem sogenannten State-Explosion-Problem ist es zuzuschreiben, weshalb mit jedem weiteren Signal, das dem System hinzugefügt wird, die Anzahl der möglichen Systemzustände exponentiell ansteigt. Die in ihrem Kern einfach umzusetzende Methodik des Petrinetzes, wird durch diese Umstände zu einem versteckten und besonders hohen Risiko für die Modellqualität und der daraus abgeleiteten Softwarequalität. Mit der einzigartigen Schaltlogik des ZZM ist es gelungen, ein Modell zu modellieren, das nicht nur steuern, sondern auch überwachen kann. Im Konzept des ZZM sind verschiedene Ebenen dafür zuständig, das gewünschte Prozessverhalten mit dem aktuellen Prozessverhalten zu vergleichen und dadurch jede Abweichung oder

Störung erkennbar zu machen. Die Methodik des ZZM konnte über alle Qualitätsmerkmale hinweg überzeugen. Die Fragestellung dieser Arbeit konnte somit vollständig geklärt werden, wobei es dennoch Raum für weitere Überlegungen gibt.

Es besteht der Anreiz, an die Erkenntnisse und gemachten Erfahrungen dieser Arbeit anzuknüpfen und eine Grenzwertbetrachtung durchzuführen. Es ist von Interesse, bis zu welchem Umfang der Steuervorgabe die Aussagen zur Qualität des ZZM gültig sind. Weiters gilt es diejenigen Qualitätsmerkmale herauszufinden, die für qualitative Einbußen verantwortlich wären. Da der Rahmen dieser Arbeit mit diesen Untersuchungen überschritten worden wäre, gilt es diese Thematik in zukünftigen Arbeiten zu behandeln.

10 LITERATURVERZEICHNIS

Gedruckte Werke (20)

Andreas Maier, Harmut (2014): *PQ4Agile-Qualitätsmodell*, KMU-innovativ, Sulzbach

Bernroider, Edward; Stix, Volker (2005): *Grundzüge der Modellierung*, WUV Universitätsverlag, Wien

BOCK, PROF. (2018): *Grundlagen der SPS-Pogrammierung / Prozessinformatik*, Regensburg

Deutsches Institut für Normung, DIN (2014): *DIN IEC 60050-351*, 63069 Offenbach am Main

Dirk Vallée, Waldemar (2018): *Infrastruktur*, in: Achim Kampker, Prof. (Hrsg.): *Elektromobilität*, 2. Auflage, Springer Berlin Heidelberg, Heidelberg, S. 387

Döring, Daniela (2011): *Eine kurze Einführung in die Systemtheorie*, Vieweg+Teubner Verlag, Leipzig

Gruber, Markus (2008): *Forschungspraxis am CAMPUS 02*, Leykam Buchverlagsgesellschaft m.b.H. Nfg. & Co. KG, Graz, Graz

Hoffmann, Dirk (2008): *Software-Qualität*, Springer Berlin Heidelberg, Heidelberg

IEC61131-3 (2014): *Grundlagen Speicherprogrammierbarer Steuerungen - Teil 3*

Janschek, Professor (2010): *Systementwurf mechatronischer Systeme*, Springer Heidelberg Dordrecht London New York, Dresden

Karaali, Prof. (2018): *Speicherprogrammierbare (SPS)-Steuerungen*, in: Springer Professional "Wirtschaft+Technik", Springer (Hrsg.): *Grundlagen der Steuerungstechnik*, Springer Fachmedien Wiesbaden, S. 155 - 177

Kleuker, Stephan (2009): *Formale Modelle der Softwareentwicklung*, GWV Fachverlage GmbH, Wiesbaden

Linke, Petra (2021): *Speicherprogrammierbare Steuerungen*, Springer FachmedienWiesbaden GmbH, ein Teil von Springer Nature 2021, Wiesbaden

Litz, Prof. (2013): *Grundlagen der Automatisierungstechnik*, Oldenburg

Lunze, Jan (2020): *Automatisierungstechnik - Methoden für die Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme*, 5 Auflage, De Gruyter Oldenbourg, Bochum

Lunze, Prof. (2012): *Ereignisdiskrete Systeme*, Oldenbourg Wissenschaftsverlag GmbH, München

Manfred Broy, Marco (2021): *Einführung in die Softwaretechnik*, Springer Verlag GmbH Deutschland, Bayern

Marwedel, Peter (2021): *Eingebettete Systeme*, Springer Vieweg, Dortmund, Deutschland

Plenk, Valentin (2018): *Grundlagen der Automatisierungstechnik*, Springer Vieweg, Hof, Bayern

Prof. Dr. Dietmar Göhlich, Dr. (2021): *Energie- und Mobilitätssysteme der Zukunft*, in: Göhlich, Prof. (Hrsg.): *Mobility2Grid - Sektorenübergreifende Energie- und Verkehrswende*, Springer Berlin Heidelberg, Berlin, S. 282

R. Bliesener, F. (1997): *Speicherprogrammierbare Steuerungen*, Springer Berlin Heidelberg, Heidelberg

Wurmus, Dipl.-Ing. (2002): *CNet - Komponentenbasierter Entwurf verteilter Steuerungssysteme mit Petri-Netzen*, Hannover

Zander, Hans-Joachim (2015): *Steuerung ereignisdiskreter Systeme*, Springer Science+Business Media, Bannewitz bei Dresden, Deutschland

Wissenschaftliche Artikel (3)

Feess, Prof. (2018): *System*, in: Gabler Wirtschaftslexikon, Springer Fachmedien Wiesbaden GmbH, S. 1

Georg Frey, Lothar (1999): *Methoden und Werkzeuge zum industriellen Steuerungsentwurf - Historie, Stand, Ausblick*, in: at - Automatisierungstechnik, S. 12

Köhler-Bußmeier, Michael (2015): *Formale Grundlagen der Informatik*, in: Automaten, Grammatiken und Formale Sprachen, Universität Hamburg, S. 232

Online-Quellen (3)

Gruber, Markus (2022): *Selmo Helpcenter*
https://helpcenter.selmo.at/DE/ [Stand: 22.05.2022]

Gruber, Markus (2022): *Selmo Helpcenter*
https://helpcenter.selmo.at/DE/hmi.html [Stand: 17.05.2022]

Jena, Günther (2021): *https://www.semiversus.com/dic/grundlagen_der_digitaltechnik*
https://www.semiversus.com/dic/grundlagen_der_digitaltechnik/automatentheorie.html [Stand: 22.03.2022]

Normen (2)

DIN (Hrsg.) (2011): *ISO/IEC 25010: ISO/IEC 25010:2011-03 Software-Engineering - Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) - Qualitätsmodell und Leitlinien*

ISO (Hrsg.) (2011): *ISO/IEC 25010:2011: ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

FORMELVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

ZZM	Zustand-Zonen Modell
IPN	Interpretiertes Petrinetz
SIPN	Steuerungstechnisch interpretierte Petrinetz
PIPN	Prozessinterpretiertes Petrinetz
SPS	Speicherprogrammierbare Steuerung
VPS	Verbindungsprogrammierbare Steuerung
AWL	Anweisungsliste
ST	Strukturierter Text
AS	Textbasierte Ablaufsprache
FBS	Funktionsbaustein-Sprache
KOP	Kontaktplan
AS	Grafische Ablaufsprache
HMI	Human Machine Interface
MXIC	Manual Cross Interlock Check
CMZ	Constantly monitored Zone
FUE	Funktionale Eignung
ZUV	Zuverlässigkeit
SIH	Sicherheit
BBK	Benutzbarkeit
ÜBK	Übertragbarkeit
WBK	Wartbarkeit
EFF	Effizienz
KOM	Kompatibilität