Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in der gleichen oder ähnlichen Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 31. Juli 2009

Wittmann Thomas

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Regelungstechnik

Prof. Dr.-Ing. G. Roppenecker **Prof. Dr.-Ing. T. Moor**

Studienarbeit

Ereignisdiskreter Reglerentwurf - eine Laborstudie

Als Studienarbeit
vorgelegt von

Thomas Wittmann

Betreuer:	Prof. Dr.-Ing. Thomas Moor	Ausgabedatum:	01.03.2009
		Abgabedatum:	31.07.2009

Ereignisdiskreter Reglerentwurf – eine Laborstudie

Aufgabenstellung:

Es soll eine Laborstudie zum ereignisdiskreten Reglerentwurf vorbereitet und durchgeführt werden.

Zur Vorbereitung der Laborstudie ist

1. eine Analogschaltung zu realisieren, welche extern die ereignisdiskrete Dynamik eines einfachen Aufzugs ausweist;
2. die am Lehrstuhl entwickelte DES Library an einen Industrie PC der Firma WAGO anzupassen, insb. hinsichtlich digitaler IO Komponenten.

Zur Durchführung der Laborstudie ist

1. ein ereignisdiskretes Regelstreckenmodell zu erstellen;
2. eine formale Spezifikation herzuleiten;
3. der Reglerentwurf anhand bekannter Algorithmen durchzuführen;
4. der geschlossene Regelkreis experimentell hinsichtlich des gewünschten Verhaltens zu überprüfen.

Als Ergebnis der Arbeit sollte ein Dokument entstehen, an Hand dessen sich ein mit der klassischen Regelungstechnik vertrauter Anwender einen Überblick über den ereignisdiskreten Reglerentwurf und seiner praktische Umsetzung verschaffen kann.

(Prof. Dr.-Ing. Thomas Moor)

Thomas Wittmann

Inhaltsverzeichnis

Abschnitt 1

Einleitung

Im Entwicklungsprozess technischer Systeme steigen die Kosten mit zunehmender Komplexität und der Anzahl für Testzwecke benötigter Prototypen. Dieser Umstand bedingt den Einsatz moderner Simulationstechniken.
Besonders in frühen Entwicklungsstadien hat sich der Digitalrechner als Mittel der Wahl durchgesetzt. Im Bereich der Regelungstechnik erlaubt er den Vorabentwurf eines Reglers gemäß vorgegebener Spezifikation, sowie die Simulation des geschlossenen Kreises zur Validierung des erzielten Streckenverhaltens. Erfüllt dieses die Anforderungen nicht, beginnt der Entwurf von Neuem.
Ist das Entwurfsstadium abschlossen, schließt sich der Phase der Implementierung des Reglers auf realer Hardware an. Vor allem mit Blick auf möglichst geringe Entwicklungskosten, besteht hier der Wunsch die reale Stelleinrichtung an einem geeigneten Simulationsaufbau zu testen. Gemeinhin fallen Simulationstechniken dieser Art in die Kategorie der Hardware in the Loop (HIL) - Simulation.
In der Forschungsgruppe „Ereignisdiskrete Systeme“ (DES) der Universität Erlangen-Nürnberg wurde in den letzten Jahren ein umfangreiches Softwarepaket zum Entwurf ereignisdiskreter Regler entwickelt, die sogenannte libFAUDES [1]. Aufbauend auf den dort implementierten Algorithmen, soll in dieser Arbeit für eine geeignete Simulationshardware ein Regler entworfen und auf Hardware der Firma WAGO implementiert werden.
Im Folgenden werden zunächst die Grundlagen der Supervisory Control Theorie (SCT) von einem sehr pragmatischen Standpunkt aus dargelegt. Es folgt die Beschreibung der Simulationshardware und der industriellen Plattform, sowie der zur Implementierung des Stellgesetzes notwendigen Software. Eine kurze Beschreibung des gesamten Hardwareaufbaus schließt dieses Kapitel ab. Anschließend werden die zum Reglerentwurf be-

nötigten Schritte beschrieben und dieser durchgeführt. Ergebnisse und Ausblick bilden den Kern des letzten Kapitels.

Abschnitt 2

Grundlagen der ereignisdiskreten Systemtheorie

Im Folgenden wird es immer wieder nötig sein auf die Begriffe der SCT zurückzugreifen. Deshalb sollen im ersten Kapitel die dazu notwendigen Grundbegriffe eingeführt werden. Auf Grund der großen Praxisnähe dieser Arbeit, wird dabei auf mathematische Strenge verzichtet. Stattdessen werden die Sachverhalte an einfachen Beispielen motiviert und erläutert. Ein genaue mathematische Beschreibung findet sich in [3]

2.1 Begriffsklärung - Ereignisdiskretes System

Im Bereich der konventionellen Regelungstheorie beschäftigt man sich mit kontinuierlichen Systemen. Kennzeichnend für diese Systemklasse ist, dass sowohl ihr Zustandsraum, als auch der ihn bestimmende Ordnungsparameter, die Zeit, kontinuierlich sind. Im Gegensatz dazu besitzen ereignsisdiskrete Systeme einen diskreten Zustandsraum und der Ordnungsparameter „Zeit“ spielt nur eine untergeordnete Rolle. Vielmehr werden Zustandsänderung einzig und allein durch das diskrete Auftreten von Ereignissen bestimmt. Will man diese charakterisieren so kann dies auf zwei Ebenen geschehen. Einerseits werden Ereignisse auf physikalischer Ebene mit Signalflanken assoziert, andererseits verursachen sie, auf der abstrakten Ebene der ereignisdiskreten Systemtheorie, Übergänge im Zustandsraum.
Man stelle sich als Beispiel die Ankunft eines Aufzuges in einem bestimmten Stockwerk vor. Auf physikalischer Ebene wird die Ankunft des Aufzuges von einem Sensor detektiert, was eine Änderung im Pegel des Sensorausgangssignals zur Folge hat (siehe Abb.

2.1),

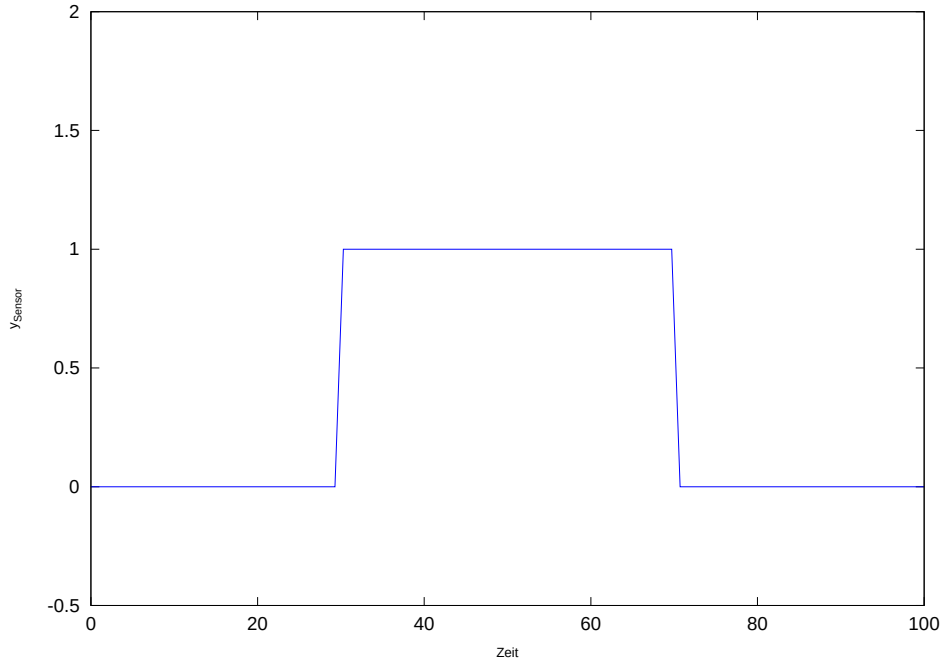

Bild 2.1: Signalflanken

während im abstrahierten Verhalten der Aufzug den Zustand „Aufzug fährt nach oben“ verlässt und in „Aufzug hat obere Position erreicht“ übergeht.
Gleiches gilt für die Zustände eines DES. Während auf physikalischer Ebene jedem Zustand eine bestimmte Sensor-Aktor-Konfiguration entspricht, geben sie auf abstrakter Ebene Aufschluss über die im System bereits abgelaufende Kette von Ereignissen.
Im Hinblick auf den späteren Reglerentwurf muss zudem ein Partitionierung der Menge aller Ereignisse in *steuerbare* und *nicht-steuerbare* Ereignisse vorgenommen werden. Erstere können von einem externen Benutzer in der Strecke aktiv ausgelöst werden, wie zum Beispiel das Einschalten eines Motors. Letztere dagegen, sind von außen nicht zu beeinflussen und können lediglich passiv registriert werden.

2.2 Automaten und formale Sprachen

Während die Regelungstheorie kontinuierlicher Systeme zur Beschreibung der Dynamik von Strecke und Regler Differentialgleichungen verwendet, sind *Automaten* bzw. *formale Sprachen* das Mittel der Wahl zur Modellierung ereignisdiskreter Dynamik.
Unter einer formalen Sprache versteht man eine Menge von Strings, welche die Dynamik

eines DES modellieren. Allerdings ist die Repräsentation ereignisdiskreter Dynamik als formale Sprache in der Praxis wenig effizient. Vorteilhafter ist die Darstellung als Automat, wobei anzumerken ist, dass jede formale Sprache eine Automatenrepräsentation besitzt, solange nur jeder in ihr enthaltene String endlich lang, also die Sprache regulär ist.

Automaten sind von der Struktur her gerichtete Graphen, welche einigen Modifikationen und Erweiterungen unterworfen wurden. So heißen die Knoten des Graphen *Zustände* und die Kanten *Ereignisse*. Zudem nennt man die Menge aller Ereignisse das *Alphabet* des Automaten. Zur Beschreibung der in dieser Arbeit herangezogenen Automaten, genügen fünf Attribute. Neben den bereits genannten, die *Menge der Anfangszustände*, die der *markierten Zustände* sowie die *Transitionenrelation*. Letztere ist ein Abbildung, die jeweils einem Zustand und einem Ereignis eine Menge von Folgezuständen zuordnet.
Nachfolgende Abbildung soll diese Definition verdeutlichen.

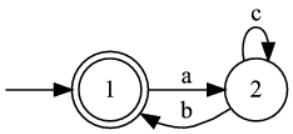

Bild 2.2: Einfacher Automat

Der oben dargestellte Automat besitzt die Zustandsmenge $Q = \{1,2\}$, die Menge der Anfangszustände $Q_0 = \{1\}$ und die Menge der markierten Zustände $Q_m = \{1\}$. Im gesamten Automaten kommen nur die Ereignisse *a*, *b* und *c* vor, so dass sich das Alphabet des Automaten zu $\Sigma = \{a,b,c\}$ ergibt. Dabei wird angenommen, dass nur *a* steuerbar ist. Nachfolgende Abbildung gibt tabellarisch die Transitionenrelation wieder.

(Ausgangszustand,Ereignis)	Folgezustand
(1,*a*)	2
(2,*b*)	1
(2,*c*)	2

Tabelle 2.1: Transitionenrelation des einfachen Automaten

Diese sagt aus, dass der Anfangszustand “1“, nachdem das Ereignis *a* aufgetreten ist, verlassen wird und der Automat in den Zustand „2“ übergeht. Tritt dort das Ereignis *c* auf, wird der Automat seinen Zustand beibehalten, und erst nachdem das Ereignis *b* ausgeführt wurde, zurück in den Anfangszustand wechseln. Behält man im Hinerkopf, dass Automaten eine Sprache representieren, stellen markierte Zustände eine Möglichkeit dar,

bestimmte Ereignisfolgen als gültig zur deklarieren. Dies ist dann der Fall, wenn eine Ereigniskette in einem markierten Zustand endet. Die Menge aller solcher Strings nennt man die *markierte Sprache* des Automaten. Im obigen Beispiel wären das alle Strings, die im Anfangszustand enden.
Im Framework der SCT werden Automaten zudem verschiedene Eigenschaften zugeordnet. Die hier benötigten werden nachfolgend kurz wieder gegeben.

2.2.1 Grundlegende Eigenschaften von Automaten

Bedenkt man, dass Automaten in diesem Kontext zur Modellierung ereignisdiskreter Dynamik dienen, also jeder Zustand einer bestimmten Systemkonfiguration entspricht, stellt sich die Frage, ob auch wirklich alle gewünschten Systemkonfiguration im System auftreten können. Dazu führt man Eigenschaften ein, welche mit dem Überbegriff *Erreichbarkeit* überschrieben werden können. Ein Zustand heißt *erreichbar*, falls es einen String in der von dem jeweiligen Automaten repräsentierten Sprache gibt, welcher im Anfangszustand beginnt und im betreffenden Zustand endet. Weiterhin heißt ein Zustand *co-erreichbar*, falls ein String in der von dem jeweiligen Automaten repräsentierten Sprache existiert, der in dem betreffenden Zustand beginnt und in einem markierten Zustand endet. Ist jeder erreichbare Zustand gleichfalls co-erreichbar heißt der Automat *blockierungsfrei*. Falls jeder Zustand des Automaten erreichbar und co-erreichbar ist so nennt man den Automaten *trimm*.
Das folgende Beispiel zeigt einen nicht blockierungsfreien Automaten.

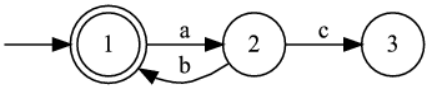

Bild 2.3: Beispiel für einen nicht blockierungsfreien Automaten

Daneben stellt sich die Frage nach der Eindeutigkeit der Transitionenrelation. Besitzt die Menge der Anfangszustände genau ein Element und ist die Transitionenrelation dahingehend eindeutig, dass jedem Paar bestehend aus einem Zustand und einem Ereignis, genau ein Folgezustand zugeordnet wird, heißt der Automat *deterministisch*. Der folgende Automat erfüllt diese Eigenschaft offensichtlich nicht.

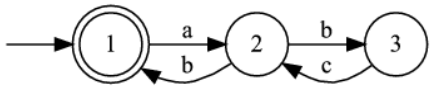

Bild 2.4: Beispiel für einen nicht deterministischen Automaten

Neben den genannten Eigenschaften sind für den späteren Reglerentwurf noch die nachfolgenden Operationen auf Automaten bzw. formalen Sprachen wichtig.

2.2.2 Inverse natürliche Projektion und Parallelkomposition

Im Weiteren wird es notwendig sein, in eine Sprache alle möglichen Zeichenketten, die mit Hilfe eines bestimmten Alphabets geformt werden können, an willkürlichen Stellen einzufügen. Dies ermöglicht formal die *Inverse Projektion* bzgl. des betreffenden Alphabets. Praktisch umgesetzt wird dies mit Hilfe von sogenannten *Selfloops*, also Transitionen, welche den gleichen Ausgangs- und Folgezustand haben und in jedem Zustand der Automatenrepresentation der betreffenden Sprache eingefügt werden.
Die folgende Abbildung zeigt die Inverse Projektion des Automaten in Abb. 2.2 bezüglich des Alphabets $\Sigma = \{x, y\}$.

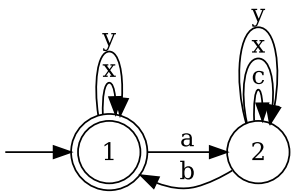

Bild 2.5: Beispiel zur inversen naürlichen Projektion

Außerdem interessiert man sich für das Verhalten zweier Systeme im gekoppelten Betrieb. Ein dazu geeignetes Hilfsmittel ist die *Parallelkomposition*.
Formal bezeichnet sie die Schnittmenge beider Sprachen, nachdem beide der inversen Projektion bezüglich des Alphabets der jeweils anderen Sprache unterworfen wurden. Praktisch berechnet man das Kreuzprodukt der jeweiligen Automatenrepresentationen, das sogenannte *synchrone Produkt*. Bei der Berechnung der Transitionenrelation des synchronen Produkts unterscheidet man zwei Fälle. Ist ein Ereignis Teil beider Alphabete, so wird es nur in den Zuständen des synchronen Produkts zugelassen, welche aus Zuständen

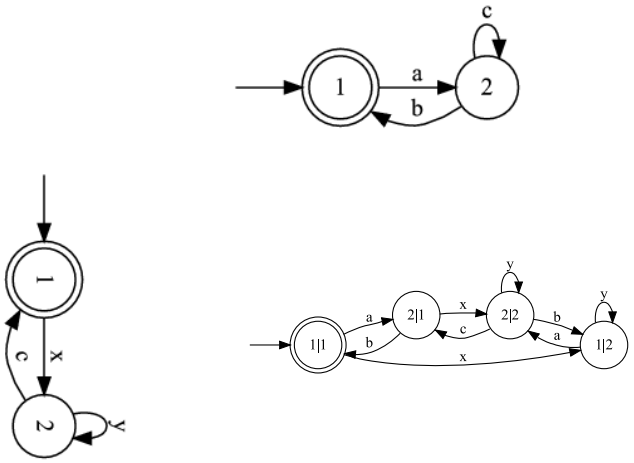

Bild 2.6: Beispiel zum synchronem Produkt

beider Automaten entstanden sind, die beide zugleich die Ausführung des betreffenden Ereignisses ermöglichen. Im anderen Fall ist ein Ereignis nur Bestandteil eines der beiden Alphabete. Es wird folglich in all den Zuständen des synchronen Produkts zugelassen werden, welche aus Zuständen entstanden sind, für die nach dem Auftreten des betreffenden Ereignisses ein Nachfolgezustand definiert ist. Markierung finden sich im synchronen Produkt nur dann wieder, falls beide Ausgangszustände markiert sind. Die nachfolgende Abbildung soll diesen Zusammenhang verdeutlichen. Mit Hilfe des synchronen Produkts ist man nun in der Lage, komplexe Systeme aus relativ einfachen Teilautomaten zu synthetisieren. Auf die Grundlagen der Modellierung ereignisdiskreter Dynamik wird im nächsten Abschnitt eingegangen.

2.2.3 Automatenbasierte Modellierung ereignisdiskreter Dynamik

Da man bei der Modellierung kontinuierlicher Systeme auf Naturgesetze wie den Newton'schen Bewegungsaxiome zurückgreifen kann, wird, solange man sich über die zu berücksichtigenden Phänomene geeinigt hat, ein Modell eindeutig in Form eines Differentialgleichungssystems erzielt. Nicht so bei ereignisdiskreten Systemen. Die Modellierung ihrer Dynamik erfolgt auf einer höheren Abstraktionsebene und unterliegt daher weit wenigeren Restriktionen. Im Hinblick auf den Reglerentwurf sollte das Modell zumindest dem Umstand Rechnung tragen, dass in jedem Zustand des Modells prinzipiell alle steuerbaren Ereignsisse auftreten können.
Als Beispiel sei eine einfache Maschine herangezogen, welche ein Werkstück aufnehmen, bearbeiten und ausgeben kann. Darüberhinaus wird berücksichtigt, dass die Maschine während des Betriebs möglicherweise Störungen unterliegt und in diesem Falle repariert

werden muss, bevor sie ihre Arbeit wieder aufnehmen kann. Folgende Abbildung zeigt den aus der Modellierung resultierenden Automaten.

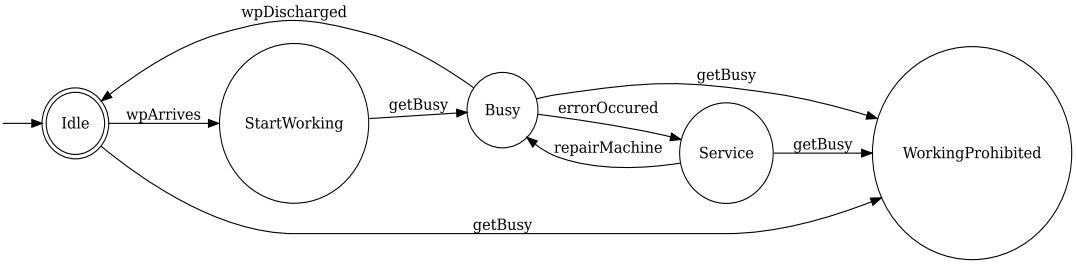

Bild 2.7: Modell einer einfachen Maschine

Im Anfangszustand „Idle“ gekennzeichnet durch einen Pfeil ohne Ausgangspunkt, ist die Maschine in der Lage eine Werkstück aufzunehmen. Beginnt die Maschine ohne ein Werkstück zu arbeiten (getBusy), soll das System in den Fehlerzustand „WorkingProhibited“ übergehen. Wird von der Maschine, ein Werkstück aufgenommen (wpArrives), ist die Maschine in der Lage dieses zu bearbeiten und geht in den Zustand „Busy“ über. Wird die Bearbeitung erfolgreich abgschlossen, wirft die Maschine das bearbeitete Werkstück aus (wpDischarged) und geht in den Ausgangszustand über. Tritt dagegen während der Bearbeitung ein Fehler (errorOccured) auf, so geht das System in den Fehlerzustand „Service“ über, den es nur nach abgeschlossener Reparatur wieder verlassen kann. Außerdem führt eine doppelte Bearbeitung eines Werkstücks in den Fehlerzustand „WorkingProhibited“. Die Markierung des Anfangszustandes drückt aus, dass nur vollständige Fertigungszyklen als gültig erachtet werden.

2.3 Grundgedanken der Supervisory Control Theory

Ziel der SCT ist die Bereitstellung von Methoden, die einen modellbasierten Reglerentwurf analog zur kontinuierlichen Regelungstechnik erlauben. Diesem Grundgedanken folgend ist der erste Schritt im Zuge des ereignisdiskreten Reglerentwurfs die Bereitstellung eines ausreichend detaillierten Streckenmodells. Formal ist anschließend eine Zuordnung, ein sogenannter Supervisor, zu entwerfen, welche jedem möglichen String aus der Sprache der Strecke ein Ereignis zuordnet, so dass die Sprache des geschlossenen Kreises vorher festgelegten Anforderung genügt.
Für die praktische Umsetzung ist an erster Stelle die Automatenrepräsentation des Streckenmodells zu erstellen. Anschließend muss das gewünschte Verhalten des geschlossenen Kreises spezifiziert werden. Dieses wird wiederum in Form eines Automaten festge-

halten und entsprechend *Spezifikation* genannt.
Wie im vorhergehenden Abschnitt gezeigt liefert die Parallelkomposition zweier Automaten jenes Verhalten, welches zwei Systeme im Verbund an den Tag legen. Im Besonderen liefert die Parallelkomposition von Strecke und Spezifikation einen ersten Kandidaten für die Dynamik des geschlossenen Kreises. Tatsächlich kann es jedoch sein, dass dieser Automat nicht blockierungsfrei ist, oder aber in einigen Zuständen nicht-steuerbare Ereignisse verboten werden, obwohl sie im Streckenmodell erlaubt sind. Da jedoch nicht-steuerbare Ereignisse nicht dem Einfluss des Supervisors unterliegen, muss dies vermieden werden. Dazu entfernt man die kritschen Zustände aus dem gesteuerten System, einen nach dem anderen bis entweder kein Zustand mehr kritisch oder aber die Transistionenrelation leer ist. In diesem Fall muss die Spezifikation angepasst oder gegebenfalls das Modell überarbeitet werden um zu einem brauchbaren Ergebnis zu kommen.
Will man im obigen Beispiel erzwingen, dass jedes aufgenommene Werkstück auch wieder die Maschine verlässt, so könnte man die Spezifikation wie nachfolgend dargestellt angeben.

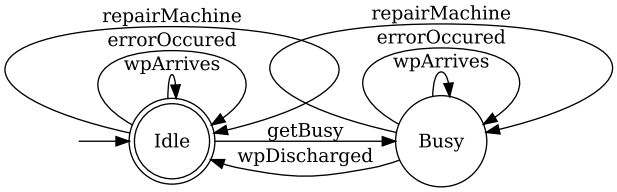

Bild 2.8: Beispiel für eine einfache Spezifikation

Ohne weiter auf die Struktur des verwendete Algorithmuses einzugehen, zeigt die nächste Abbildung die Automatenrepräsentation des geregelten Streckenverhaltens.

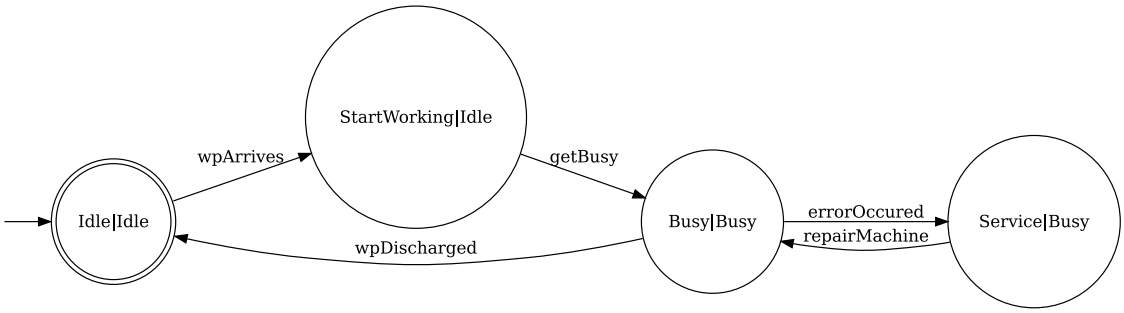

Bild 2.9: Automatenrepräsentation des geregelten Streckenverhaltens

Obiger Automat repräsentiert jene Teilsprache der Strecke, welche einerseits möglichst groß ist, andererseits jedoch nur Strings enthält, die gefolgt von einem nicht-steuerbaren

Ereignis aus dem Streckenalphabet noch immer in der markierten Sprache der Strecke liegen. Besitzt eine Sprache diese Eigenschaft heißt sie *steuerbar* bzgl. der Strecke und deren Alphabet. In diesem Fall kann man zeigen, dass ein Supervisor mit geforderten Eigenschaften existiert. Das bedeutet, die Implementierung eines Stellgesetzes in Form des obigen Automaten auf geeigneter Hardware, liefert im geschlossenen Kreis das gewünschte Streckenverhalten. An dieser Stelle ist jedoch anzumerken, dass dieser sogenannte *monolithische* Regler nur den ersten Ansatz im Framework der SCT darstellt und auf Grund des hohen Speicher- und Rechenaufwands, hohe Ansprüche sowohl an die Reglerplattform als auch an die zur Berechnung des Automaten eingesetzten Rechner stellt. Dieser Ansatz ist daher nur für relativ kleine Systeme praktikabel, gleichwohl aber für diese Arbeit ausreichend.

Abschnitt 3

Aufbau des Simulationsystems

Zum Test der libFAUDES-basierten Steuerungssoftware ist der Aufbau eines geschlossenen Regelkreises notwendig. Darin wird mit dem Begriff *Strecke* im Folgenden die Simulaionshardware, eine Analogsimulation der Dynamik eines Aufzugs, bezeichnet. Dagegen bezieht sich *Regler* oder *Stelleinrichtung* auf den WAGO-IPC inklusive operationeller Software. Die Automatenrepräsentation des geregelten Streckenverhaltens wird im Weiteren als *Supervisor* bezeichnet.
Alle Ergebnisse des Hardwareaufbaus wurden photodokumentiert und auf der CD im Anhang hinterlegt.

3.1 Beschreibung der Simulationshardware

Auf der Suche nach geeigneter, also leicht in Hardware zu simulierenden Systemen, findet man in einem einfachen Aufzug einen guten Kandidaten. Nicht zuletzt auf Grund seines hohen Bekanntheitsgrades wurde als Strecke die Simulation einer vereinfachten Fahrstuhldynamik gewählt. Nachfolgender Abschnitt untergliedert sich in zwei Teile. Zunächst wird das reale Vorbild der Strecke vorgestellt und anschließend deren Nachbildung in Analogtechnik beschrieben.

3.1.1 Das nachzubildenede System

Man stelle sich einen Bergwerksaufzugs vor. Im Zuge des Schichtwechsels befördert er die Belegung der neuen Schicht von der Oberfläche zu ihrer Arbeitsstätte und ermöglicht umgekehrt der Arbeiterschaft die Rückkehr ans Tageslicht.

Hauptbestandteil einer solchen Einrichtung ist die Passsagierkabine mit Tür sowie entsprechende Motoren. Motor 1 dient der Bewegung der Kabine, Motor 2 öffnet und schließt

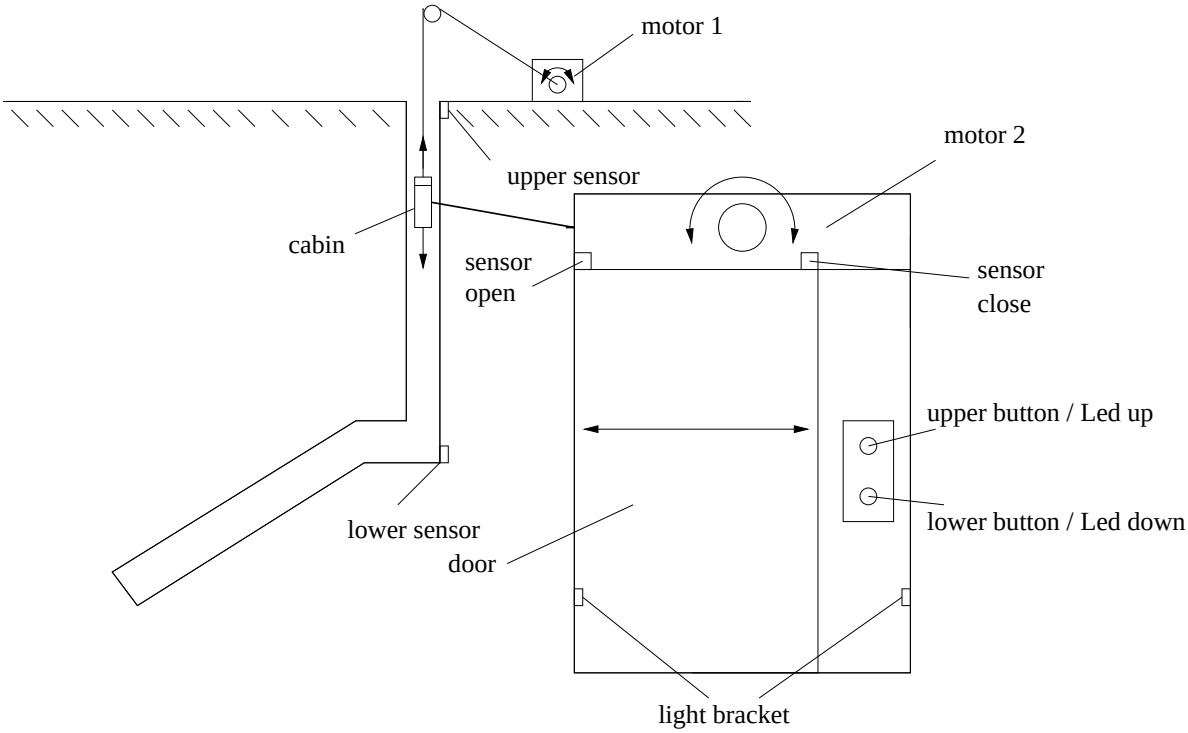

Bild 3.1: Prinzipbild des realen Systems

die Tür. Zudem besitzt ein Aufzug dieser Art wenigstens zwei Knöpfe, mit denen die Bewegungsrichtung der Kabine gesteuert werden kann. Weil Bergbauunternehmen bekanntlich gut mit ihrem Geld umzugehen wissen, wurden auch genau zwei Knöpfe außen an der Kabine angebracht. In den Knöpfen befinden sich je eine LED die separat gesteuert werden. Um die Position von Kabine und Tür zu bestimmen, wurden je zwei Sensoren verbaut.
Die Bezeichnungen wurden mit Blick die später einzuführenden Ereignisnamen aus dem Englischen gewählt.

3.1.2 Implementierung in Analogtechnik

Aus der Perspektive der linearen Systemtheorie kann man sowohl die Kabine als auch die Tür als System mit integierendem Verhalten auffassen. Um den schaltungstechnischen Aufwand zu verringern, wurde jedoch darauf verzichtet Kabine und Tür als ideale Integratoren nachzubilden. Stattdessen wurde ihre Dynamik durch die eines PT_1-Gliedes angenähert. Da die tatsächliche reale kontinuierliche Dynamik von Kabine und Tür für

die Modellierung als ereignisdiskretes System keine große Rolle spielt, ist diese Näherung durchaus gerechtfertigt.Vereinfachend kommt hinzu, dass beide System unabhängig von einander und, von der Struktur der verwendeten Schaltung her, gleich aufgebaut werden können. Abbildung 3.2 zeigt die beiden Teilsysteme mit den entsprechenden Ein- und Ausgängen.

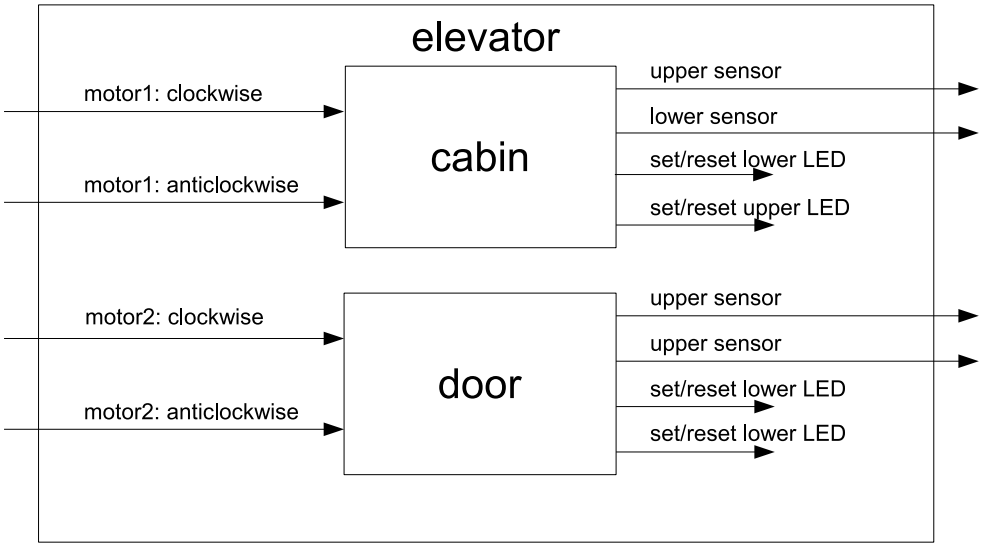

Bild 3.2: Ein- und Ausgänge der Aufzugssimulation

Zudem wurden zwei Knöpfe und eine Lichtschranke, in Form eines Schalters, vorgesehen. Um Systemfehler zu simulieren wurden weitere zwei Schalter verbaut, welche die Bewegung von Kabine und Tür verhindern können. Um den Zustand des Systems nach außen sichtbar zu machen, werden insgesamt 11 LEDs verwendet. Jeweils zwei davon entfallen auf Tür und Aufzug. Sie zeigen an ob der Aufzug sich an der Oberfläche oder unter der Erde befindet, bzw. die Tür offen oder geschlossen ist. Zwei weitere entfallen auf die Anzeige von benutzerbedingten Fehlern. Im Gegensatz zu allen anderen LEDs werden diese sechs direkt von der Hardware gesteuert und unterliegen somit nicht der Kontrolle des Supervisors. Die verbleibenden LEDs zeigen den Zustand der Lichtschranke und paarweise die Bewegungsrichtung von Aufzug und Tür an. Im Gegensatz zur Realität ist

es in der Simulation nicht offensichtlich, ob die Tür sich öffnet oder schließt. Daher wurde der Streckenaufbau um die entsprechenden LEDs ergänzt. Analoges gilt für die der Lichtschranke zugeordneten LED. Mit Blick auf die Testphase wurde ein weiterer Schalter verbaut, mit dem sich die Geschwindigkeit des Aufzugs in zwei Stufen einstellen lässt. Um diese Erweiterungen, soweit notwendig, der Steuerung von Außen zugänglich zu machen, musste die Schaltungsschnittstelle entsprechend erweitert werden. Die nachfolgende Abbildung zeigt die Schaltung als Blackbox mit allen vorgesehen Ein- und Ausgängen.

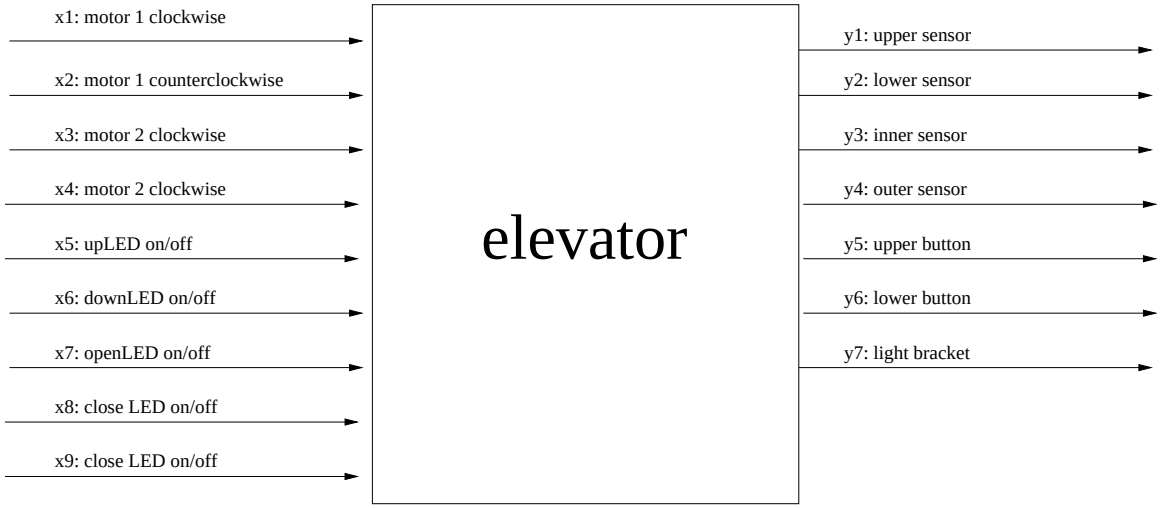

Bild 3.3: Ein- und Ausgänge der Aufzugssimulation

Schaltungstechnisch werden Kabine und Tür als einfache RC-Glieder umgesetzt, deren Zeitkonstanten ihre Bewegungsgeschwindigkeit bestimmen. Zudem muss eine Logik implementiert werden, die dafür Sorge trägt, dass die Kondensatoren der RC-Glieder, einmal bis zu einer gewissen Schranke geladen, ihren Ladungszustand solange beibehalten, bis sie wieder aktiv entladen werden. Abb. 3.4 zeigt die Ladelogik der Aufzugsschaltung.
Der ihr zu Grunde liegende Gedanke ist, die Spannung über dem Kondensator auf Vcc zu legen, solange die Kabine sich an der Oberfläche befindet ($y_1 = Vcc$), oder Motor 1 die Kabine nach oben befördert ($x_1 = Vcc, x_2 = GND$). Die dabei verwendeten UND und OR - Gatter wurden in Widerstand-Dioden - Technik ausgeführt, während als Komperatoren ICs des Typs LM339 zum Einsatz kamen. Ihr Datenblatt findet sich im Anhang. Zudem wurden mit ihrer Hilfe, bei Bedarf, Teile der Schaltung, sowie betreffende Ein- und Ausgänge entkoppelt. Schaltungstechnische Details entnehme man dem Eagle - Projekt auf der CD im Anhang.

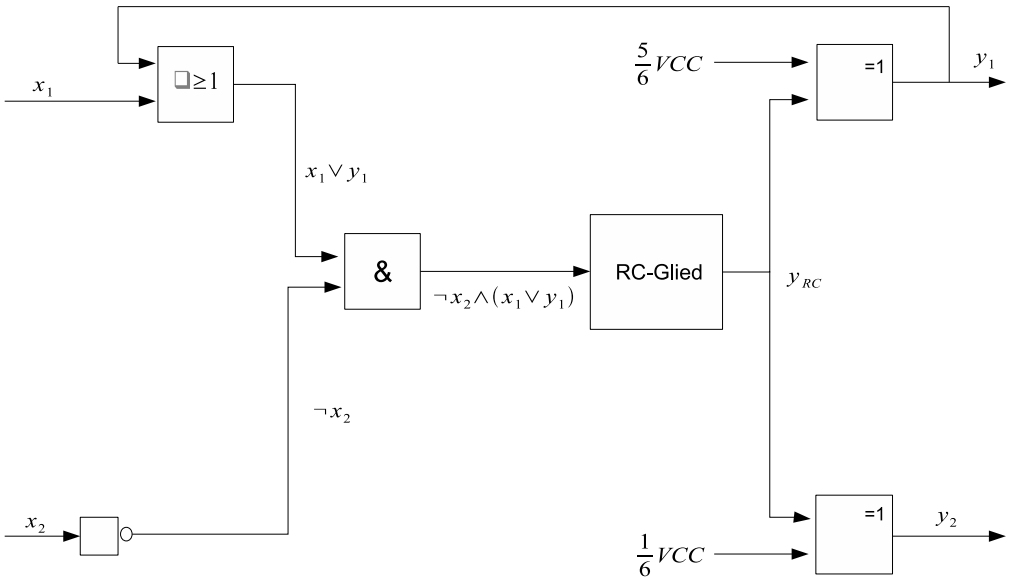

Bild 3.4: Ladelogik für die Aufzugsschaltung

3.1.3 Äußere Schnittstellen

Um den Simulationsaufbau flexibel an verschiedenen Stelleinrichtungen betreiben zu können, wurde als Schnittstelle ein 25-poliger D-SUB – Anschluss gewählt. Die nächste Tabelle zeigt dessen Belegung.

Pin-Nummer	Signal	Pin-Nummer	Signal
1	motor1 clockwise	10	lower sensor
2	motor1 anticlockwise	11	upper sensor
3	motor2 clockwise	12	inner sensor
4	motor2 anticlockwise	13	outer sensor
5	LED up on/off	14	upper button
6	LED dw on/off	15	lower button
7	LED open on/off	16	light bracket
8	LED close on/off	24	GND
9	LED lb on/off	25	VCC

Tabelle 3.1: D-SUB 25 Buchsenbelegung

3.2 Beschreibung der Stelleinrichtung

Im Folgenden wird zunächst die Reglerplattform vorgestellt und anschließend auf die benötigte Software eingegangen.

3.2.1 Die Reglerplattform

Bei der verwendeten Reglerplattform handelt es sich um eine Hutschienen-SPS der Firma WAGO, welche der Universität Erlangen-Nürnberg freundlicherweise kostenlos zur Verfügung gestellt wurde. Der WAGO-I/O-IPC-P14 ist Teil der Produktpalette WAGO-I/O-SYSTEM 750 und trägt die genaue Produktnummer 0758-0876/0000-0110. Im Wesentlichen handelt es sich dabei um einen vollwertigen Industrie-PC mit diversen I/O-Schnittenstellen. Genauere Informationen zur verbauten Hardware finden sich in Handbuch (CD im Anhang).
Was den WAGO-IPC für den hier vorgesehen Einsatzzweck besonders qualifiziert, ist der der Intel Pentium M 1.4Ghz, also eine x86-CPU. Diese lässt eine, auf die verwendete Hardware abgestimmte, Linux-Distribution als Betriebssystem zu und ermöglicht so den Einsatz von Programmen, welche mit Hilfe der libFAUDES erzeugt wurden. Da meistens x86-kompatible Rechner zum Erstellen der Steuerungssoftware benutzt werden, macht sie den Einsatz eines cross-compilers beim Übersetzen der libFAUDES-binaries überflüssig. Die Kommunikation mit der Außenwelt stellt der WAGO-IPC über den Klemmenbus (KBus) her. Dabei handelt sich um ein I/O-System bestehend aus Ein- und Ausgangsklemmen.
.

3.2.2 Softwarestruktur der Stelleinrichtung

Nach erfolgreichem Reglerentwurf liegt der Supervisor in Form eines Automaten vor. Die von ihm repräsentierte Sprache legt die Reihenfolge fest, in der Ereignisse in der Anlage auftreten dürfen. Ist man also in der Lage, die in dem Automaten enthaltene Information als Grundlage für ein Steuerprogramm zu nutzen, so erzielt man in der Strecke das gewünschte Verhalten. Genau dies ermöglicht das libFAUDES-plugin „simulator“ in Kombination mit dem plugin „iodevice“.

Das libFAUDES-Plugin simulator

Mit Hilfe des Plugins simulator ist man in der Lage eine Anwendung, einen sogenannten *Simulator*, zu erstellen, welche aus einem Automaten wieder die von ihm repräsentierte Sprache erzeugt.
Konkret muss dieser Anwendung ein Automat und eine Konfigurationsdatei übergeben werden, in welcher allen Ereignissen aus dem Alphabet des zu simulierenden Automaten, bestimmte Attribute zugeordnet werden können. Unter anderem kann jedem Ereignis eine bestimmt Priorität zugewiesen werden, an Hand derer man nicht nur die Reihenfolge der Ausführung gleichzeitig möglicher Ereignisse festlegen, sondern auch zwischen steuerbar und nicht-steuerbar unterscheiden kann. Dazu ordnet man steuerbaren Ereignissen eine positive Priorität und entsprechend nicht-steuerbaren eine negative Priorität zu. Einmal gestartet ermittelt das Programm den Anfangszustand des Automaten und prüft welche Ereignisse in diesem Zustand ausführbar sind. Aus der Menge dieser Ereignisse wird das jenige ausgewählte, welches die höchste Priorität aufweist. Besitzt keines der ausführbaren Ereignisse positive Priorität, verbleibt der Simulator solange im Anfangszustand bis das erste mögliche Ereignis mit negativer Priorität in der Strecke registriert wird und geht erst dann in den festgelegeten Folgezustand über. Dieser Vorgang wird nun in jedem Zustand wiederholt, solange bis die Simulation abgebrochen wird.

Das libFAUDES-Plugin iodevice

Um nun Ereignisse auf realer Hardware auszuführen und einzulesen zu können, wurde die libFAUDES um das Plugin „iodevice“ (IO-Device) erweitert. Gegliedert in verschiedene Abstraktionsebenen stellt es Schnittstellen bereit, die das Ausführen bzw. erkennen von Ereignissen ermöglichen. An Anwendungen welche diese Schnittenstellen nutzen möchten, muss eine Konfigurationsdatei übergeben werden, in der festgelegt ist, welches Ereignis mit welcher Signalflanke assoziiert wird. Um steuerbare Ereignisse auszuführen ist es ausreichend eben jene Signalflanken in der Anlage zu erzeugen. Schwieriger ist das Erkennen nicht-steuerbaren Ereignissen in der Strecke. Dazu erzeugen IO-Device gestützte Programme einen parallel laufenden Thread. Dieser ermittelt zyklisch die Sensorkonfiguration der Anlage und registiert eventuell aufgetretene Flanken. Mit Hilfe der in der Konfigurationsdatei hinterlegten Daten ist es möglich, den erkannten Flanken die entsprechenden Ereignisse zu zuordnen.
Um nun die Kommunikation mit unterschiedlicher, im Besonderen neuer Hardware flexibel handhaben zu können, wurde das IO-Device, im Sinne objektorientierten Program-

mierung, mit einer entsprechenden Vererbungsstruktur (Abb. 3.5) ausgestattet.

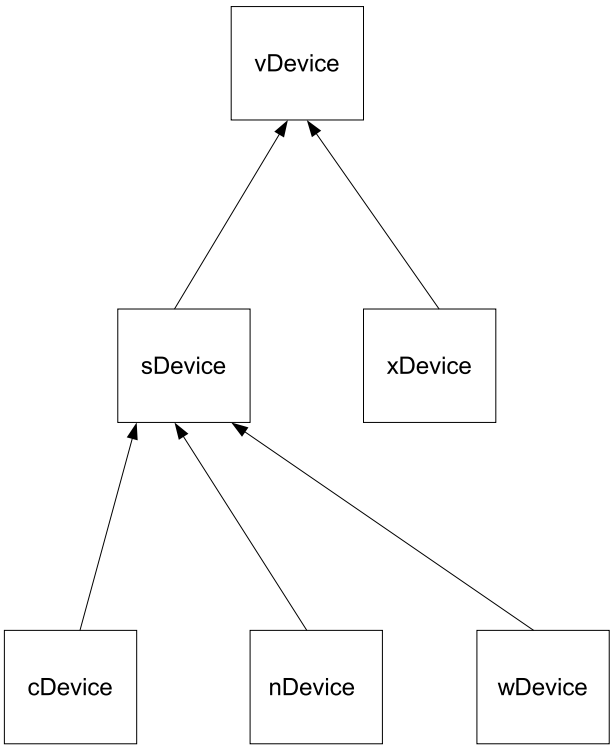

Bild 3.5: Vererbungsstruktur des IO-Device

In der Klasse „vDevice“ werden grundlegende Attribute und Funktionen definiert die in jeder Kindklasse benötigt werden. Davon ableitet werden die Klassen „sDevice“ und „xDevice“. Erstere erweitert das „xDevice“ um notwendige Funktionen zur Umsetzung physikalischer Signale in logische Ereignisse. In den Klassen „cDevice“ und „nDevice“ wird die Art und Weise der Erzeugung mit bestimmten Ereignissen assozierter Signale bestimmt. Während das „cDevice“ mit Hilfe des comedi-frameworks auf geeigneter I/O-Hardware physikalische Signale erzeugt, dient das „nDevice“ der Kommunikation über das Netzwerk

Im Zuge dieser Arbeit wurde dieses Plugin um die Klasse „wDevice“ erweitert. Sie stellt im Wensentlichen eine low-level Schnittstelle zur Kommunikation mit dem WAGO-IPC bereit. Eine genauere Behandlung der Software Struktur des IO-Devices würde an dieser Stelle zu weit führen. Zudem findet sich unter [1] eine genaue Beschreibung.

Struktur des verwendeten Steuerungsprogramms

Kombiniert man Simulator und IO-Device, so kann man eine Anwendung erzeugen, welche einerseits in der Lage ist, eine gegebenen Supervisor zu simulieren und gleichzeitig die erzeugte Sprache auf realer Hardware umzusetzen. Die folgende Abbildung verdeutlicht diesen Zusammenhang.

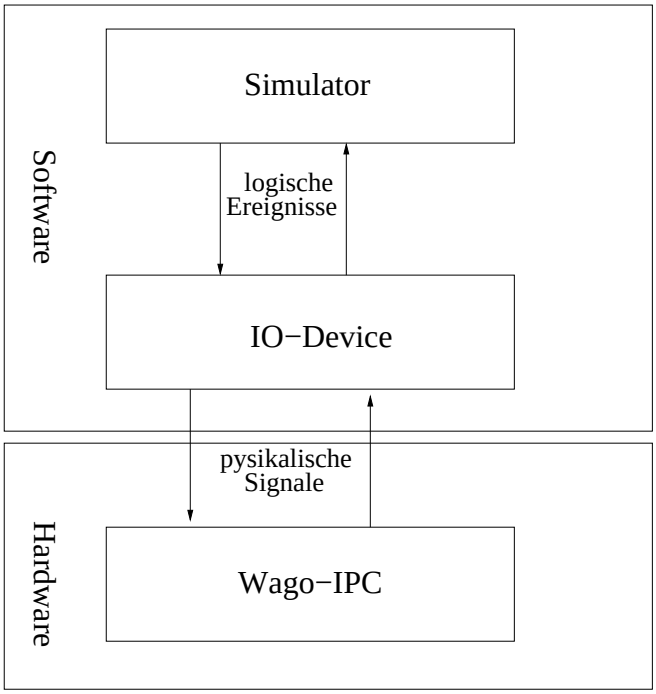

Bild 3.6: Zusammenspiel von Simulator und IO-Device

An dieser Stelle bleibt anzumerken, dass die Binärdatei des Steuerungsprogramms im Wesentlich auf einem beliebigen Rechner mit kompaitbel tool-chain erzeugt und per LAN auf den WAGO-IPC übertragen werden kann. Da der Anwendung der Supervisor und die beiden Konfigurationsdateien als Parameter übergeben werden, muss die Binärdatein auch nur einmal erzeugt werden, sollte sich der Supervisor im Zuge des Entwicklungsprozesses verändern.

Abschnitt 4

Supervisorentwurf

Der Entwurf ereignisdiskreter Regler besteht aus drei Teilschritten. Zunächst ist ein hinreichend genaues Automatenmodell der Regelstrecke zu konstruieren. Anschließend muss das gewünschte Regelstreckenverhalten spezifiziert werden. Ist diese Vorarbeit geleistet, kann der Supervisor unter Verwendung bereits implementierter Algorithmen berechnet werden. Dem entsprechend untergliedert, wird in diesem Kapitel der Reglerentwurf für die bereits beschriebene Aufzugssimulation geschildert.
Alle im Folgenden vorgestellten Automaten und Skripte werden auf der CD im Anhang mitgeliefert.

4.1 Definition der relevanten Ereignisse

Bevor mit der eigentlichen Modellierung der Streckendynamik begonnen werden kann, muss festgeleget werden, welchem Ereignis welcher Vorgang im System zugeordnet wird. Aus praktischen Gründen erfolgt die Bezeichnung mit Hilfe von Abbkürzungen, die wie nachfolgend erklärt sind.

Abbkürzung	Bedeutung
c	cabin
d	door
lb	light bracket
bup	button up
➥	

Abbkürzung	Bedeutung
bdw	button down
mv	move
stp	stop
lv	leave
ar	arrive
up	upper/upward
lw	lower
dw	downward

Tabelle 4.1: Abkürzungen

Mit Hilfe dieser Abkürzungen lassen sich die Ereignisnamen so einführen, dass auf Anhieb ersichtlich ist, welchem System und welchem Vorgang das Ereignis zugeordnet ist.

$$\underbrace{c}_{\text{System}} - \underbrace{mvup}_{\text{Vorgang}}$$

Das Ereignis „c_mvup“ wird also mit dem System „cabin“ und dem Vorgang „move up“ assoziiert.
Eine vollständige Auflistung aller verwendeter Ereignisse enthält die nächst Tabelle. Darin werden in der Spalte „Flag“ Eigenschaften der Ereignisse aufgeführt. Hier wird nur zwischen steuerbar (stb) und nicht-steuerbar unterschieden.

Ereignis	Flag	zugeordneter Vorgang
c_mvup	stb	Kabinenmotor (motor1) wird eingeschaltet. Die Kabine beginnt sich nach oben zu bewegen.
c_mvdw	stb	Kabinenmotor (motor1) wird eingeschaltet. Die Kabine beginnt sich nach unten zu bewegen.
c_stp	stb	Aufzugmotor(motor1) wird ausgeschaltet. Die Kabine beendet ihre Bewegung.
c_arup		Kabine erreicht obere Halteposition. Oberer Sensor (upper sensor) belegt.
c_lvup		Kabine verlässt obere Halteposition. Oberer Sensor (upper sensor) wird frei.
➥		

Ereignis	Flag	zugeordneter Vorgang
c_arlw		Kabine erreicht untere Halteposition. Unterer Sensor (lower sensor) belegt.
c_lvlw		Kabine velässt untere Halteposition. Unterer Sensor (lower sensor) wird frei.
d_open	stb	Türmotor (motor2) wird eingeschaltet. Die Tür öffnet sich.
d_close	stb	Türmotor (motor2) wird eingeschaltet. Die Tür schließt sich.
d_stp	stb	Türmotor (motor2) wird ausgeschaltet. Die Tür beendet ihre Bewegung.
d_opened		Tür ist vollständig geöffnet. Seitlicher Sensor (open sensor) belegt.
d_lvopened		Tür beginnt sich zu schließen. Seitlicher Sensor (open sensor) wird frei
d_closed		Tür ist vollständig geschlossen. Mittiger Sensor (close sensor) belegt.
d_lvclosed		Tür beginnt sich zu öffnen. Mittiger Sensor (close sensor) wird frei.
bup_pressed		Oberer Knopf (upper button) wird gedrückt.
bdw_pressed		Unterer Knopf (lower button) wird gedrückt.
LEDup_on	stb	Einschalten der LED „LEDup“.
LEDup_off	stb	Ausschalten der LED „LEDup“.
LEDdw_on	stb	Einschalten der LED „LEDdw“.
LEDdw_off	stb	Ausschalten der LED „LEDdw“.
LEDopen_on	stb	Einschalten der LED „LEDopen“.
LEDopen_off	stb	Ausschalten der LED „LEDopen“.
LEDclose_on	stb	Einschalten der LED „LEDclose“.
LEDclose_off	stb	Ausschalten der LED „LEDclose“.
LEDlb_on	stb	Einschalten der LED „LEDlb“.
LEDlb_off	stb	Ausschalten der LED „LEDlb“.

Tabelle 4.2: Ereignisdefinition

Mit Hilfe dieser Ereignisse kann nun ein gut lesbares Streckenmodell entworfen werden. Zudem erleichtert diese Nomenklatur die Suche nach Fehlern in der Spezifikation erheblich.

4.2 Konstruktion des Aufzugsmodells

Es erweist sich als überaus schwierig das Modell des kompleten Systems in einem Zug zu erstellen. Wesentlich einfacher ist es, das System in kleinere, von einander unabhängige Subsysteme aufzuteilen, diese einzeln zu modellieren und anschließend das vollständige Modell per synchronem Produkt zu synthetisieren. Die Aufteilung des Aufzugs in Kabine, Tür, Knöpfe, Lichtschranke und LEDs hat sich als praktikabel erwiesen.
Grundsätzlich wurde bei der Modellierung darauf geachtet, die Modelle möglichst nahe am realen Verhalten der Simulationshardware zu orientieren. Dies impliziert natürlich das Verhalten eines realen Aufzugs. Einige Details würden aber angepasst werden müssen, sollte man das Verhalten eines realen Aufzugs erfassen wollen.

4.2.1 Modellierung von LEDs und Knöpfen

Das IO-Device erzeugt Signale ohne Kenntnisse des aktuellen Signalpegels. Vielmehr wird davon ausgegangen, dass der Supervisor sicherstellt, dass das erzeugte Signal tatsächlich die gewünschte Flanke zur Folge hat. Hinzu kommt, dass alle Signale diskrete sind, also einzelne Aktoren ein- oder ausgeschaltete bzw. Ausgänge auf Masse oder Versorgungssapannung liegen. Diese beiden Gründe rechtfertigen die Aussage, dass die dem Ein- un Ausschalten der LEDs zugeordneten Signale in keinem kausalen Zusammenhang stehen müssen. Im Besonderen können die zugeordneten Ereignisse in beliebiger Reihenfolge auftreten. Ein probates LED–Model kann also als Automat mit nur einem einzigen Zustand und Ein - und Ausschaltereignissen als Selfloops angegeben werden(Abb. 4.1).

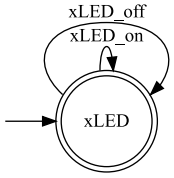

Bild 4.1: Automatenmodell für eine beliebige LED

Da sämtliche zu steuernde LEDs mit einem Modell dieser Struktur erfasst werden, wird hier nur ein Automat stellvertretend für alle anderen angegeben. Dementsprechend ist „x“ durch ein entsprechendes Kürzel zu ersetzen.
Obwohl die Dynamik eines Schalters erst durch zwei Ereignisse (gedrückt und wieder logelassen werden) vollständig beschrieben wird, ist es an dieser Stelle ausreichend sich auf

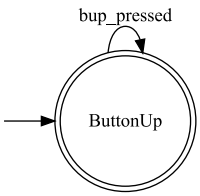

Bild 4.2: Automatenmodell für den oberen Knopf

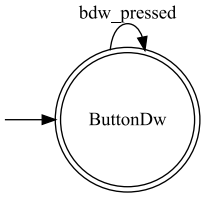

Bild 4.3: Automatenmodell für den unteren Knopf

nur auf das Drücken des Schalters zu beschränken. Dass der Schalter wieder losgelassen werden muss, bevor man ihn wieder drücken kann, ist nur eine physikalische Konsequenz des Schalteraufbaus und trägt zur Regelung des Systems keine relevante Information bei. In folgedessen kann das Automatenmodell strukturgleich dem LED–Modell angegeben werden (Abb. 4.10 und 4.11)

4.2.2 Modellierung der Lichtschranke

Im Gegensatz zu den LEDs, bestimmen nicht-steuerbare Ereignisse die Dynamik der Lichtschranke. Würde man in der Modellierung der Lichtschranke die Reihenfolge, mit der die Ereignisse auftreten können, nicht erfassen, so wäre es im anschließenden Spezifikationsentwurf möglich, ein Verhalten zu erzwingen, das das Streckenmodell zulässt, im physikalischem Aufbau jedoch nicht auftreten kann. Daher wird ein Automatenmodell gewählt, das vorsieht, dass sich Belegen und Freiwerden der Lichschranke abwechseln müssen (Abb. 4.4).

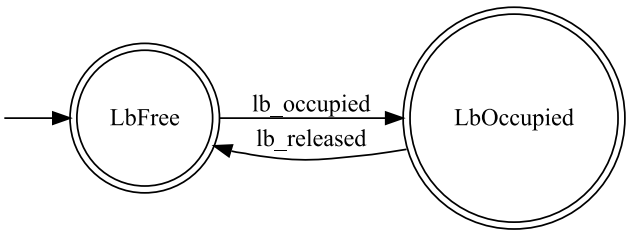

Bild 4.4: Automatenmodell für die Lichtschranke

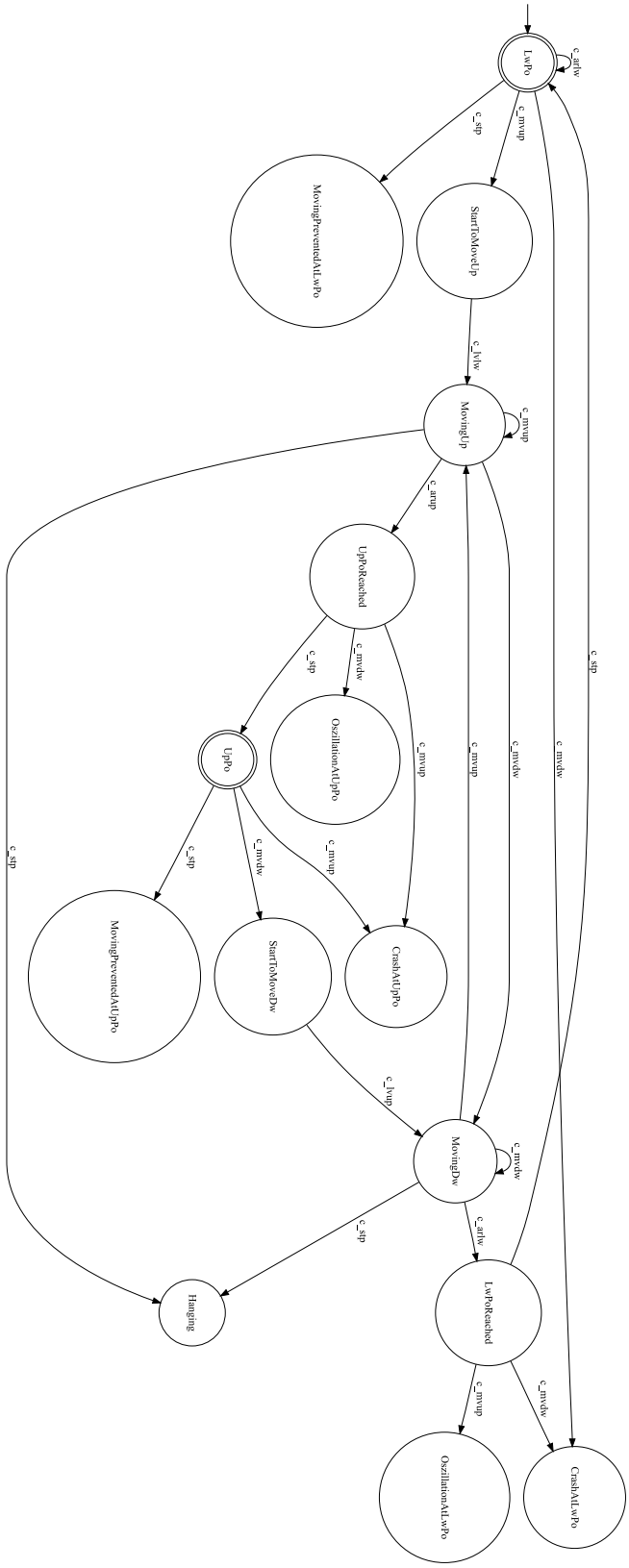

Bild 4.5: Automatenmodell der Kabine

4.2.3 Modellierung von Kabine und Tür

Wie bereits erwähntm sind Kabine und Tür, bis auf die Zeitkonstanten der verwendeten RC-Glieder, schaltungstechnisch identisch aufgebaut. Dementsprechend können sie auch analog modelliert werden. Es wird daher nur die Modellbildung der Kabine ausgeführt.
Neben der eigentlichen Dynamik der Kabine werden im Modell verschiedene Fehler berücksichtigt. Im Folgenden wird zuerst der mögliche Fahrtzyklus beschrieben, bevor auf die Fehlerbehandlung eingegangen wird.
Dem Aufbau der Simulationshardware gemäß wird der Anfangszustand als der Zustand definiert, in dem sich die Kabine im unteren der beiden Stockwerke befindet und nicht nicht bewegt wird („LwPo“). Erfolgt das steuerbare Ereignis „c_mvup“ , wird der Kabinenmotor eingeschaltet („StartToMoveUp“). Einmal in Bewegung gesetzt muss die Kabine das untere Stockwerk verlassen („c_lvlw“) und das System geht in den Zustand „MovingUp“ über. Nachdem eine gewisse Zeitspanne verstrichen und kein weiteres steuerbares Ereignis aufgetreten ist, wird die Kabine mit dem Auftreten des Ereignisses („c_arup“) das obere Stockwerk erreichen („UpPoReached“). Wird nun der Motor ausgeschaltet („c_stp“), so geht das System in den markierten Zustand („UpPo“) über. Mit dem Ereigniss „c_mvdw“ beginnt eine entsprechende Ereigniskette, die den Aufzug zurück in den markierten Zustand „LwPo“ führt.
Die im Modell berücksichtigten Fehler haben im Wesentlichen zwei Ursachen. Zum einem diejenigen die dem Verhalten eines realen Aufzuges zuwider laufen und zum anderen solche, die aus Eigenheiten des Simulationsaufbaus resultieren. In die erste Kategorie fallen alle Ereignissketten, die in den Zuständen „CrashAtLwPo“ „CrashAtUpPo“ und „Hanging“ enden. Dagegen fallen die Fehlerzustände „MovingPreventedAtLwPo“ „MovingPreventedAtUpPo“ sowie „OszillationAtLwPo“ und „OszillationAtUpPo“ in die letzere.
„CrashAtLwPo“ wird vom System dann eingenommen, falls im unteren Stockwerk ein weitere Befehl zum Nachuntenfahren erfolgt, was in der Realität zu schweren Schäden am System führen würde. Entsprechendes gilt für „CrashAtUpPo“. Der Zustand „Hanging“ resultiert aus der Vorstellung eines „vernünftig“ agierenden Fahrstuhls, der eine Aufgabe zu Ende führt, bevor er eine neue beginnt. Natürlich ist dieser Zustand auf den Fall eines Aufzugs zwischen zwei Ebenen gemünzt. Sollen mehr Stockwerke berücksichtigt werden, so müsste dieser Zustand auf jeden Fall überdacht werden.
Beim Erstellen der Spezifikation folgt man dem Paradigma, dem Supervisor so viele Optionen wie möglich zur Beeinflussung des Systems offenzuhalten. Bestimmte Eigenheiten der Schaltung würden einer Vorgehensweise, die diesem Vorsatz folgt im Wege stehen.

Aus diesem Grund wurden die Fehlerzustände der zweiten Kategorie eingeführt.

4.3 Aufstellen der Spezifikation

Mit Hilfe der Spezifikation kann die Streckensprache so geformt werden, dass sie bestimmten Vorgaben entspricht. Grundsätzlich wurde sich beim Entwurf der Spezifikation am Verhalten eines realen Bergwerksaufzugs orientiert. Dieses wird jedoch sicherlich nicht eindeutig definiert werden können. Entsprechend erhebt das nachfolgend angebene erwünschte Aufzugsverhalten keineswegs den Anspruch der Universalität, sondern entspringt vielmehr der Beobachtung des Erstellers.

4.3.1 Verbale Spezifikation des Anlagenverhaltens

Bevor eine formalisierte Spezifikation in Form eines Automaten erstellt werden kann, muss das Sollverhalten der Regelstrecke verbal und hinreichend genau festgehalten werden. Aus Gründen der Übersichtlichkeit und der besseren Referenzierbarkeit erfolgt dies in Form einer Aufzählung.

Kabine mit Knöpfen

1. Die Kabine soll, nachdem die obere/untere Position erreicht worden ist, anhalten.
2. Befindet sich die Kabine in der unteren/oberen Position und wird der untere/obere Knopf gedrückt, so soll dies ignoriert werden.
3. Befindet sich die Kabine in der unteren/oberen Position und wird der obere/untere Knopf gedrückt, so soll die Kabine untere/obere Position verlassen.
4. Befindet sich die Kabine auf dem Weg nach oben/unten und wird der untere/obere Knopf gedrückt, so soll die Kabine zunächst die obere/untere Position einnehmen und anschließend die untere/obere Position erreichen.
5. Einmal die untere/obere Position verlassen, muss die obere/untere eingenommen werden, bevor die Kabine ihre derzeitige Bewegungsrichtung umkehren kann.

Tür

1. Die Tür soll sich öffnen und wieder schließen sobald die Kabine die untere/obere Position eingenommen hat
2. Die Tür soll sich auf Knopfdruck öffnen und wieder schließen.
3. Die Kabine darf sich nicht bewegen, solange die Tür nicht geschlossen ist.
4. Die Tür darf nicht geöffnet werden, solange die Kabine in Bewegung ist

Lichtschranke

1. Ist die Lichtschranke belegt, darf die Tür nicht geschlossen und die Kabine nicht bewegt werden.

LEDs

1. upLED soll eingeschaltet werden, sobald der obere Knopf gedrückt wurde.
2. upLED soll ausgeschaltet werden, sobald die Kabine die obere Position erreicht hat.
3. dwLED soll eingeschaltet werden, sobald der untere Knopf gedrückt wurde.
4. dwLED soll ausgeschaltet werden, sobald die Kabine die untere Position erreicht hat.
5. openLED soll eingeschaltet werden, sobald sich die Tür sich öffnet.
6. openLED soll ausgeschaltet werden, sobald die Tür geöffnet ist.
7. closeLED soll eingeschaltet werden, sobald sich die Tür sich schließt.
8. closeLED soll ausgeschaltet werden, sobald die Tür geschlossen ist.
9. lbLED soll eingeschaltet werden, sobald die Lichtschranke belegt ist.
10. lbLED soll ausgeschaltet werden, wenn die Lichtschranke wieder frei wird.

Entsprechend der Modellierung gibt es auch bei der Spezifikation viele Möglichkeiten das oben spezifizierte Verhalten zu erzwingen. Das folgende Kapitel ist also nur eine von vielen möglichen Lösung für das gleiche Problem.

4.3.2 Formalisierung der Spezifikation

Im geregelten Verhalten werden Spezifikation und Strecke im synchronen Produkt betrieben. Das heißt, ausgehend von einem gemeinsamen Anfangszustand wird durch die Spezifikation bestimmt, welche Ereignisketten in der Strecke auftreten dürfen und welche nicht. Dies hat zwei wesentliche Konsequenzen. Zum einem dürfen durch die Spezifikation keine nicht-steuerbare Ereignisse verboten werden, sollten sie an entsprechender Stelle im Streckenverhalten möglich sein. Zum anderen sollte die Spezifikation möglichst wenige Restriktionen enthalten. Dazu fordert man nur die Reihenfolge bestimmter, möglichst weniger Ereignisse und führt irrelevante Ereignisse als Selfloops mit. In der Regel treten diese Ereignisse dann nicht unmittelbar hintereinander auf. In diesem Fall sorgt der Supervisor für die Einhaltung der Spezifikation.
Es gilt zu beachten, dass im unmarkierten (d.h. alle Zustände sind markiert) geregelten Streckenverhalten, keine speziellen Anlagenzustände auftreten müssen oder auch nur können. Stattdessen wird nur Blockierungsfreiheit gewährleistet, was beim Reglerentwurf ohne Markierungen im Bezug auf das gewünschte Streckenverhalten praktisch ohne Aussagekraft ist. Selbst mit Markierungen gewährleistet das Entwurfsverfahren nur, dass gültige Strings in endlicher Zeit auflaufen müssen. Über ihre Form, im Besonderen über ihre Länge, gibt das Verfahren keinen Aufschluss. Diesen Umstand gilt es beim Erstellen der Spezifikation zu berücksichtigen.
Entsprechend der Modellierung komplexer Systeme, wird der Spezifikationsentwurf zunächst nur für kleinere Subsysteme durchgeführt. Anschließend werden die Teilspezifikationen per synchronem Produkt zusammengesetzt. Als sinnvoll hat sich in diesem Fall die Aufteilung in Kabine inklusive Knöpfen, Tür und Lichtschranke erwiesen. Unberücksichtigt bleiben zunächst die LEDs.
In der Diskussion der Automaten wird im Folgenden immer wieder Bezug auf die oben erstellte Anforderungsliste genommen. Referenzen auf Gliederungspunkte beziehen sich immer auf diel, dem jeweiligen System zugehörige, Auflistung..

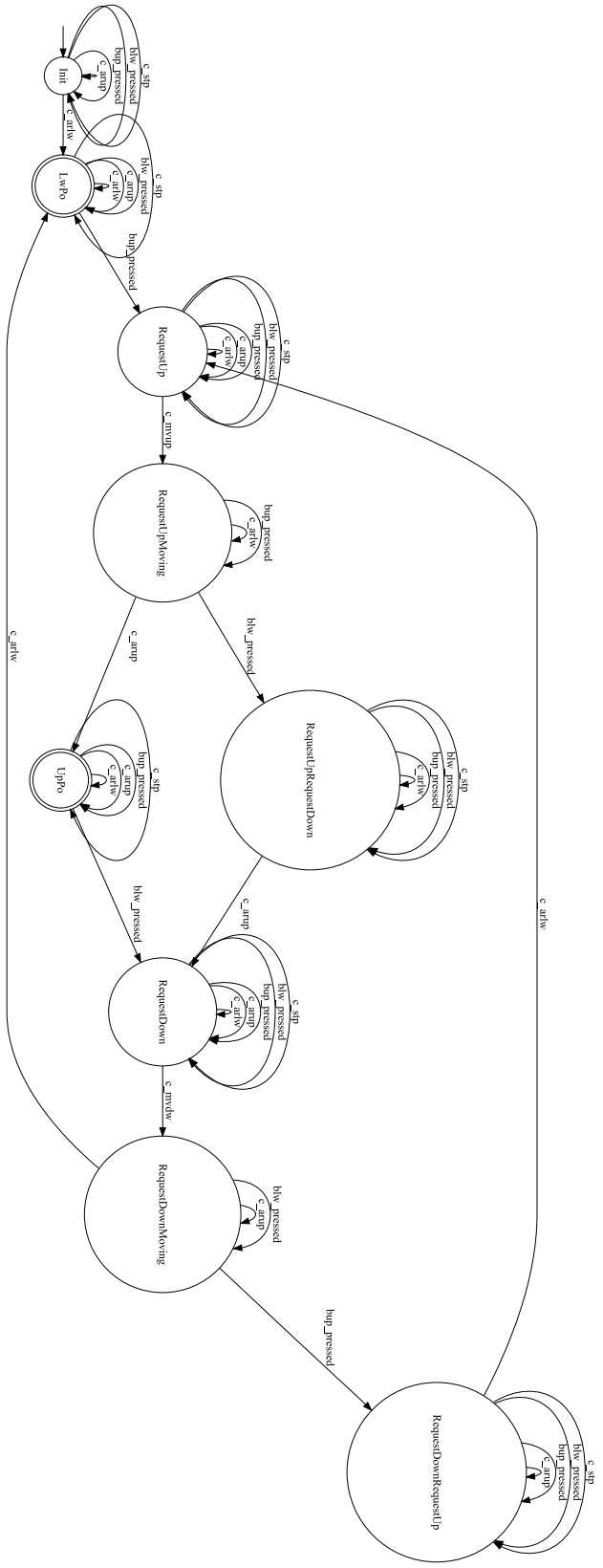

Bild 4.6: Spezifikation für Kabine mit Knöpfen

Spezifikation für die Kabine mit Knöpfen

Abbildung (4.6) zeigt die Spezifktation für die Kabine inklusive Knöpfen.
Zu Beginn befindet sich die Spezifikation in einem Zustand, der der Synchronisation mit der Anlage dient. In diesem Zustand sind alle steuerbaren Ereignisse verboten und die nicht-steuerbaren Ereignsisse ziehen als Selfloops ausgeführt keine Zustandsänderung nach sich. Erst wenn in der Strecke das Ereignis „c_arlw“ registiert wird, wird auf das Ereignis „bup_pressed“ hin das steuerbare Ereigniss „c_mvup“ erlaubt und ausgeführt, solange die Strecke dazu in Lage ist (vgl. Punkt 3). Dagegen wird das Ereignis „lwb_pressed“ ignoriert (vgl Punkt 2). Damit befindet sich die Spezifikation in dem Zustand „RequestUpMoving“Ẇird in diesem Zustand der untere Knopf gedrückt, wird mit dem Auftreten des Ereignisses „c_arup“ der markierte Zustand „UpPo“ übersprungen, und in den Zustand „RequestDown“ gewechselt. Da dieser nicht markiert ist, erzwingt dies einen Folgestring, der im markierten Zustand „LwPo“ endet (vgl. Punkt 4). Tritt in „RequestUpMoving“ dagegen das Ereginis „c_arup“ auf, geht die Spezifikation in den markierten Zustand „UpPo“ über. Mit dem Ereignis „c_mvdw“ wird ein, dem bereits geschilderten Mechanismus, analoger Vorgang in Gang gesetzt.
Zudem werden der Punkte 1 durch die Markierungen im Modell bedingt während Punkt 5 durch die im Automaten 4.6 geforderten Zyklen erzwungen wird.

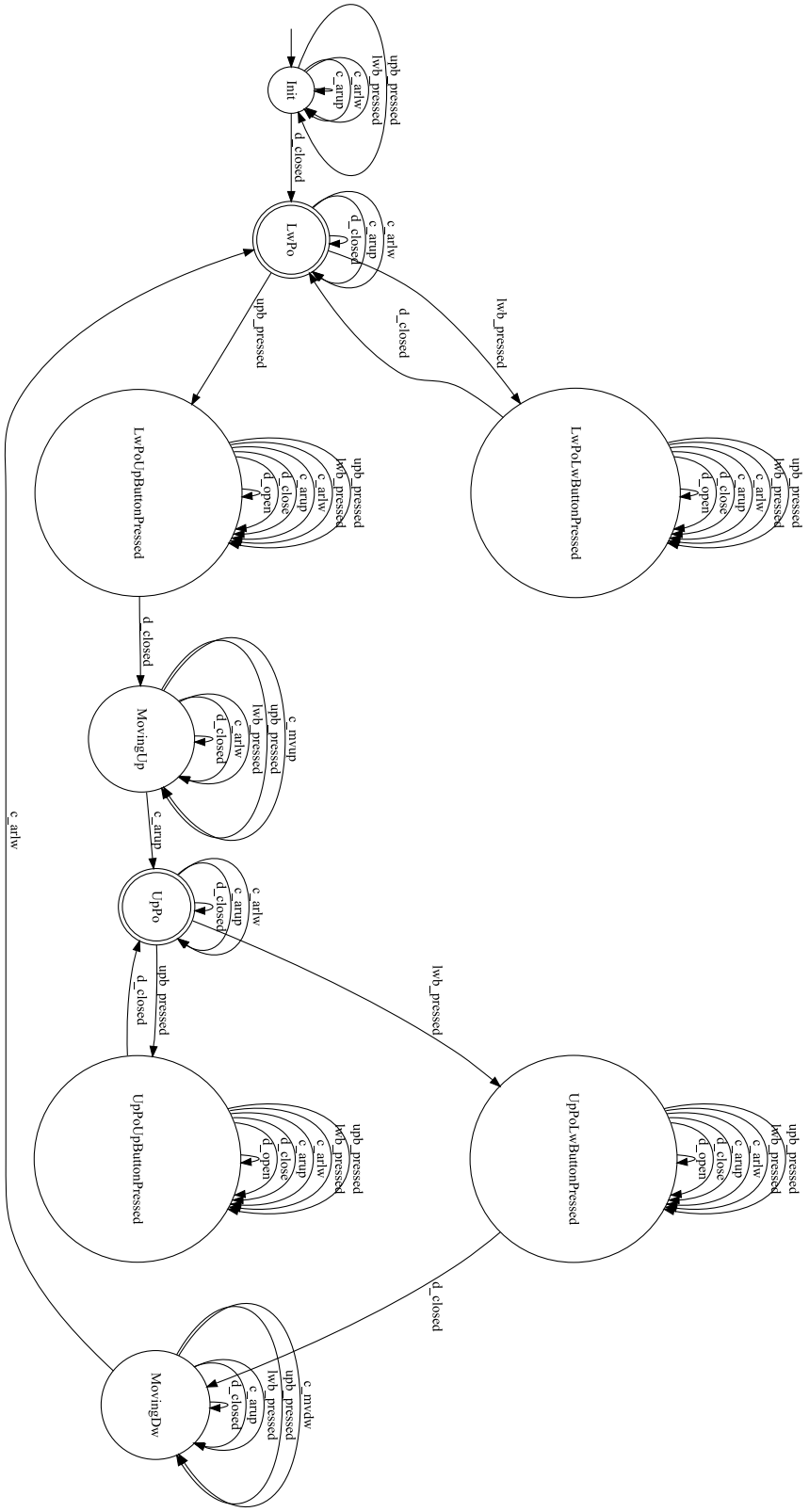

Bild 4.7: Spezifikation für die Tür - Teil1

Spezifikation für die Tür

Wie vorgeben stellt die Spezifikation in Abbildung (4.7) sicher, dass sich die Tür beim erreichen eines Stockwerkes bzw. auf Knopfdruck öffnet und wieder schließt.
Nachdem der Initialisierungszustand verlassen wurde, geht man davon aus, dass sich die Kabine in der unteren Position befindet und die Tür geschlossen ist. Nachdem einer der beiden Knöpfe gedrückt wurde, erzwingt das Erreignis „d_closed“ das Öffnen und Schließen der Tür (vgl. 1 und 2). Wurde der untere Knopf gedrückt, fällt die Spezifikation zurück in den Zustand „LwPo“İm anderen Fall wird einerseits „c_mvup“ (vgl. 3) erlaubt und andereseits durch das Fordern von „c_arup“ sichergestellt, dass die Kabine ins obere Stockwerk („UpPo“) befördert wird. In diesem Zustand greift eine, dem bereits geschilderten Mechnismus analoge, Wirkungskette.
In allen Zuständen, in denen die Ereignisse „d_open“ und „d_close“ als Selfloops eingebunden wurden, muss zudem die Reihenfolge ihrer Ausführung festgelegt werden. Wird dieser Schritt übergangen wird der Simulator diese Ereignisse alternierend ausführen, da beide steurbar sind und das Modell dieses Verhalten zulässt. In Abb. 4.8 wird spezifiziert, dass dem Befehl zum Öffnen der Tür eine Meldung über das vollständige Öffnen folgen muss. Entsprechendes gilt für den Schließvorgang.
Punkt 4 ist nun eine Folge des geforderten Zykluses.

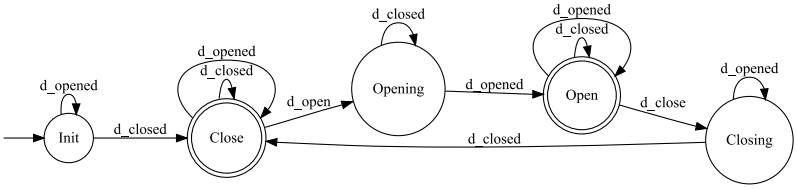

Bild 4.8: Spezifikation für die Tür - Teil2

Spezifikation für die Lichtschranke

Aus dem Alltag heraus würde man erwarten, dass solange die Lichtschranke belegt ist, sich weder Tür noch Kabine bewegen dürfen. Diesen Gedanken setzt die nachfolgende Spezifikation um.

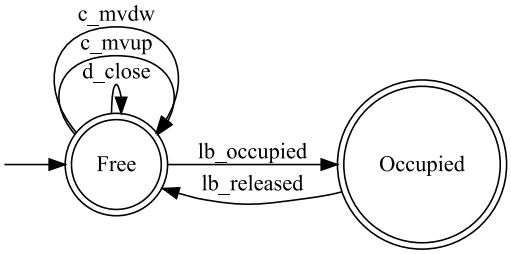

Bild 4.9: Spezifikation für die Lichtschranke

Im Anfangszustand soll die Lichtschranke nicht belegt sein, das heißt die Ereignisse, die eine Bewegung der Tür oder der Kabine zur Folge hätten, sind in diesem Zustand erlaubt. Wird der Lichtstrahl jedoch unterbrochen („lb_occupied“), geht die Spezifikation in einem Zustand über, in dem die vorher angesprochenen Ereignisse verboten sind. Erst nachdem die Lichtschranke wieder frei ist („lb_released“), geht der Autotmat in den Anfangszustand über.

4.4 Synthese des Supervisors

Nachdem nun Modell und Spezifikation für die meisten Subsysteme erstellt wurden, kann nun mit der Synthese des Supervisors begonnen werden. Außen vor bleiben wieder die LEDs.

4.4.1 Die luaFAUDES

Eingangs wurde schon erwähnt, dass die libFAUDES alle zum monolithischen Supervisorentwurf benötigten Werkzeuge zur Verfügung stellt. Um auf sie zurückgreifen zu können, müsste jedoch ein C++-Programm erstellt werden. Um diesen zusätzlichen Aufwand zu vermeiden, wurde die libFAUDES um das plugin „luabindings“ erweitert. Dieses stellt auf Basis der Skriptsprache Lua eine komfortable Möglichkeit zur Nutzung der

libFAUDES-Algorithmen bereit, in dem sie das Ausführen bestimmter, in der libFAUDES implementierter Funktionen per Konsole oder aber kompleter Lua-Skripten erlaubt.

4.4.2 Synthese des Supervisors ohne LEDs

Grundsätzlich gliedert sich das verwendete Lua-Skript in drei Teile. Zunächst werden die Teilmodelle und -spezifikationen geladen und anschließend das vollständige Streckenmodell und die Spezifikation für das System ohne LEDs erstellt.
Der Übersichlichkeit halber werden hier nur Auszüge des vollständigen Syntheseskripts angegeben.
Intern werden Automaten durch die Klasse „Generator“ und deren Deriverate repräsentiert. Diese bietet schon bei der Initialisierung die Möglichkeit einen Automaten aus einer Datei einzulesen.

Listing 4.1: Einlesen der Modelle

```
-- load models
m_elevator = faudes.Generator("../genFiles/models/m_elevator.gen")
m_lwButton = faudes.Generator("../genFiles/models/m_lwButton.gen")
m_upButton = faudes.Generator("../genFiles/models/m_upButton.gen")
...
```

Danach kann über den Bezeichner links des Zuweisungsoperators auf den entsprechenden Automaten zugegriffen werden.

Synthese des vollständigen Modells und der Spezifikation ohne LEDs

Aus oben eingelesen Automaten wird nun das vollständige Streckenmodell aufgebaut.

Listing 4.2: Aufbau des Streckenmodells

```
-- build basic model consisting of cabin, door and light-bracket
faudes.Parallel(m_door,m_cabin,plant)
-- reduce state space to minimum
faudes.StateMin(plant,plant_min)
faudes.Parallel(plant_min,m_lb,plant)
faudes.StateMin(plant,plant_min)
```

Der Aufruf „faudes.Parallel()“bildet das synchrone Produkt der beiden ersten Argumente und speichert dies in dem leeren Automaten „plant“. Anschließend wird mit dem Befehl „faudes.StateMin()“ die Zustandsmenge des Automaten „plant“ minimiert und in

„plant_min“ abgelegt. Diesem Automaten wird dann der nächste Teilautomat hinzugefügt usw. . Analog wird die Spezifikation erstellt und in dem Automaten „spec_min“Ḋie in der libFAUDES hinterlegeten Algorithmen zur Supervisorberechnung verlangen, dass die Alphabete von Streckenmodell und Spezifikation identisch sind. Daher muss in der Spezifikation fehlende Ereignisse in nachträglich per Selfloop eingefügt werden.

Listing 4.3: Aufbau des Streckenmodells

```
-- get alphabet of complete spec (except LEDs)
es_spec = spec_min:Alphabet()

-- add missing plant events to spec
faudes.EventSetDifference(es_plant,es_spec,es_diff)
faudes.SelfLoop(spec_min,es_diff)
```

Zur Berechnung eines monolithischen Supervisors stellt die libFAUDES die Funktion „SupConNB()“ zur Verfügung. Dieser werden Strecke und Spezifikation sowie ein leerer Automat, der als Container für den zu berechnenden Supervisor dient, übergeben.

Listing 4.4: Berechnung des Supervisors

```
-- synthesis
sup = faudes.Generator()
faudes.SupConNB(plant,spec,sup)
```

Resultat dieser Operation ist ein blockierungsfreier Supervisor, der das gewünschte Streckenverhalten sicherstellt.
Bevor mit Hilfe dieser Funktion der vollständige Supervisor für die Strecke ohne LEDs berechnet wird, muss den LEDs noch einige Aufmerksamkeit geschenkt werden.

4.4.3 Ein LED Supervisor

Entsprechend den Vorgaben kann beispielsweise eine Spezifikation für die LED „LEDup“ wie folgt angeben werden. Dabei werden der Übersichtlichkeit halber zwei Automaten angegeben. Erst deren synchrones Produkt liefert die vollständige Spezifikation.. Diese erfüllt nun offensichtlich die an sie gestellten Forderung. Aus Platzgründen werden hier nicht alle Automaten angegeben, sie sind jedoch im Anhang enthalten.
Man erinnere sich daran, dass als erster Kandidat für einen Supervisor das synchrone Produkt aus Strecke und Spezifiaktion dient. Auf Grund des einfachen Modells sind Spezifikation und synchrones Produkt jedoch identisch. Zudem ist dieser Kandidat offensichtlich

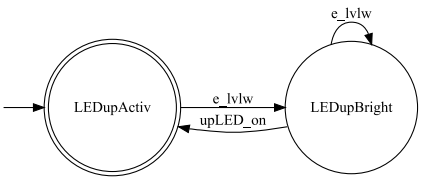

Bild 4.10: LED-Spezifikation Teil1

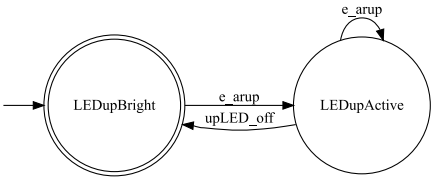

Bild 4.11: LED-Spezifikation Teil2

blockierungsfrei und steuerbar bzgl. des LED-Modells und dessen Alphabet. Der Supervisorentwurf ist daher schon nach der ersten Iteration abgeschlossen.

4.4.4 Synthese des vollständigen Supervisors

Der triviale Ansatz zum Entwurf des vollständigen Supervisors ist, für jede LED eine Spezifiaktion in die Gesamtspezifikation einzubauen. Dies führt jedoch zu einer immensen Zunahme des Zustandsraums. Entsprechend nehmen Speicher- und Rechenaufwand kaum noch akzeptable Dimensionen an.
Im zweiten Ansatz fasst man die LEDs als separate Systeme auf, entwirft für sie einen eigenen Supervisor und bindet diesen per synchronem Produkt in das bereits berechnete, geregelte Streckenverhalten ein. Dies ist aber auch zulässig, da nur steuerbare Ereignisse in die Sprache des geregelten Systems eingebracht werden und somit ihre Steuerbarkeitseigenschaften nicht kompromitiert werden.
Das nächste Listing verdeutlicht dieses Vorgehen.

Listing 4.5: Berechnung des Supervisors

```
-- state minimization has no effect since alphabets
--  are always disjoint
faudes.Parallel(sup_upLED1,sup_upLED2,sup_LEDs)
...
-- compute supervisor for plant WITHOUT LEDs
faudes.SupConNB(plant_min,spec_min,sup_partial)
-- and save result in a file
sup_partial:Write("../genFiles/supervisors/sup_partial.gen")

--build complete supervisor
faudes.Parallel(sup_LEDs,sup_partial,sup_complete)
-- and save result
sup_complete:Write("../genFiles/supervisors/sup_complete.gen")
```

Nachdem zunächst der Supervisor für alle LEDs per synchronem Produkt aufgebaut worden ist, wieder der Teilsupervisor für die Strecke ohne LEDs berechnet. Anschließend kann der vollständige Supervisor aus den eben berechneten zusammengesetzt werden. Der Container „sup_complete“ schließlich, enthält die Automatenrepräsentation des vollständigen, geregelten Streckenverhaltens.

Abschnitt 5

Simulationsergebnisse

In diesem letzten Kapitel sollen kurz die Ergebnisse von Simulationen auf verschiedenen Plattformen zusammengefasst werden.

5.1 Sofware in Loop Simulation

Bevor die Steuerungssoftware auf realer Hardware getestet wird, soll ihre Funktionstüchtigkeit mit Hilfe einer reinen Softwareumgebung geteset werden. Dazu stellt das Plugin „simulator“ eine entsprechende Umgebung bereit. Standartmäßig wird mit Hilfe dieses Plugins die Anwendung „simfaudes“ erstellt. Wie bereits erwähnt wird dieser eine Konfigurationsdatei übergeben, in der sämtliche verwendeten Ereignisse inklusive entsprechender Prioritäten aufgelistet werden. Zudem muss der Pfad zu dem zu simulierenden Automaten mitgeteilt werden und zwar relativ zum Arbeitsverzeichnis.

Listing 5.1: Übergeben des Automatenpfades

```
<Generators>
"supervisor.gen"
</Generators>
```

Im nachfolgenden Listing wird den nicht-steuerbaren Ereignsissen „c_arlw“ und „c_lvlw“ jeweils die negative Priorität „-1“ zugewiesen.

Listing 5.2: Auszug aus der Datei elevator.sim

```
...
"c_arlw"
<Priority>
–1
</Priority>

"c_lvlw"
<Priority>
–1
</Priority>
...
```

Wird „simfaudes“ ohne weitere Parameter aufgerufen, so wird ein Interface erstellt, welches zum einem Auskunft über den aktuellen Zustand der Strecke und der Spezifikation gibt, und andererseits die Ausführung eines bestimmten Ereignisses per Konsole ermöglicht.
Auf Linux/Unix-Systemen kann dieser Schritt mit Hilfe des Shell-Skripts „softwareInTheLoop.sh “ nachvollzogen werden.

5.2 Hardware in the Loop Simulation

Wurde im verhergehenden Schritt die grundlegende Funktionalität des erstellten Steuerungsprogramms sichergestellt, so folgt als Nächstes Schritt der Test auf Laborhardware.

5.2.1 Simulation auf einem Laborrechner

Im Labor „Ereignisdiskrete Systeme“ existiert ein dazu gegeignter Aufbau. Dieser besteht im Wesentlichen aus einem gewöhnlichem x86-Rechner, ergänzt durch digitale I/O-Karten der Firma Advantech und der entsprechender Verkabelung. Diese stellt über D-SUB - Anschlüsse die Möglichkeit bereit unterschiedliche Simulationshardware an diesen Rechner zu betreiben. Das nachfolgende Bild zeigt den vollständigen Aufbau der Testumgebung.

Die Anwendung „simfaudes“ kann über das Plugin „iodevice“ steuerbare Ereignisse auf realer Hardware ausführen und nicht-steuerbare Ereignisse registrieren. Dazu muss eine zusätzliche Konfigurationsdatei erstellt werden, in der die, den Ereignissen entsprechenden, physikalischen Reaktionen definiert werden. Ausserdem müssen zum Teil globale, aber auch hardwarespezifische Details angeben werden.

Listing 5.3: Standartkopf eines ComediDevice-Files

```
<DeviceContainer>

"LrtLabSignalIO"    % overall device name
10                  % ms/ftu (overwrites per device def)


<Devices>

<ComediDevice>

"LrtLabInputDevice"             % device name
500                             % fauDES-time scaling factor
64                              % max bitaddress
100                             % background sample period in nsec
"/dev/comedi0"                  % system device file

<EventConfiguration>
...
```

Das IO-Device kennt verschiedene Möglichkeiten Strukturen, die der Kommunikation mit externen Hardware dienen, zu verwalten. Hier wird in der ersten Zeile wird dem Simulator mitgeteilt, dass die Information in dieser Datei als „xDevice“ organisiert werden soll. Das Label „DeviceContainer“ zeigt schon an, diese Struktur in der Lage ist mehrere Devices zu verwalten. In diesem Fall wird zu nächst die Daten für eine Struktur vom Typ ComediDevice eingelesen. Bevor die einzelnen Unterstrukturen eingelesen werden, wird dem ganzen Gebilde ein Name zugewiesen sowie ein Zeitumrechnungsfaktor definiert. Nach dem Label „ComediDevice“ werden verschieden Attribute gesetzt, und anschließend die Ereigniskonfiguration definiert. Für ausführlichere Information sei auf [1] verwiesen.

Listing 5.4: Ereigniskonfiguration

```
...
"c_lvup"
<Sensor>
<Triggers>
6 +NegEdge+
</Triggers>
</Sensor>
...
"c_mvup"
<Actuator>
<Actions>
2 +Clr+ 0 +Set+
</Actions>
</Actuator>
...
```

Im obigen Listing wird dem nicht-steuerbaren Ereigniss „c_lvup“ eine negative Signalflanke am Eingangspin „6“ zugeordnet. Auf diese Weise müssen alle verbleibenden nichtsteuerbaren Ereignisse abgehandelt werden. Anschließend folgen die steuerbaren Ereignisse. Im Beispiel wird der Simulator angewiesen bei der Ausführung des logischen Ereignisses „c_mvup“ , die Ausgänge „0“ und „2 “ auf Vcc bzw. GND zu legen.
Auch dieser Schritt kann auf „toy“ mit Hilfe des Skripts „hardwareInTheLoop_toy.sh“ nachvollzogen werden.

5.2.2 Simulation auf Wago-Hardware

Da das Plugin “’iodevice”’ bereits um die Klasse „wDevice“ erweitert wurde, genügt es die .dev Datei anzupassen. Entsprechend abgeändert und umbenannt findet sich die Datei aus dem vorhergehenden Abschnitt als „wago_elevator.dev“ auf der CD im Anhang wieder.
Obwohl erste Tests mit dem „iomonitor“ einem dem IO-Device beiliegenden Programm, erfolgreich waren, war es bisher nicht möglich die bestehenden Fehler in der Wago-Software zu kompensieren. Daher blieb die Ausführung des vorgeschlagenen Steuerungsprogramms auf Wago-Hardware nur auf sehr einfache Automaten beschränkt. Eine Regelung der vollständigen Strecke war zum Zeitpunkt der Abgabe leider nicht möglich.

Abschnitt 6

Zusammenfassung und Ausblick

Rückblickend wurde in dieser Arbeit der Entwicklungszyklus eines HIL-Simulationssystems von der Planung und Fertigung der Simulationshardware, über deren Modellierung inklusive Reglerentwurf bis hin zum Betrieb an industrieller Hardware durchlaufen. Kernpunkte waren darin die hardwaremäßige Realisierung des Streckenaufbaus, die Modellierung und Regelung der Strecke mit Hilfe ereignisdiskreter Dynamik, sowie die Implementierung des Stellgesetztes auf einer Hutschienen-SPS der Firma Wago.
Dazu wurden Eingangs einige Begriffe der SCT eingeführt, darunter der Begriff des Automaten und der formalen Sprache, sowie einigen grundlegenden Operationen auf Automaten. Ausserdem wurde kurz auf die automatengestützte Modellierung technischer Systeme, sowie den darauf aufbauenden ereignisdiskreten Reglerentwurf eingegangen. Daran schloss sich die Beschreibung der Streckenhardware an, die bei ihrer Motivation begann und bei ihrer schaltungstechnischen Realisierung endete. Um die Hardwareseite des Simulationssystems zu kompletieren, wurde alsdann die Reglerplattform sowie die Struktur der Steuerungssoftware beschrieben. Nachdem im Eingangskapitel der Supervisorentwurf in Grundzügen vorgestellt worden ist, folgte im dritten Kapitel die Anwendung der vorgestellten Methodik auf das Beipspiel des Bergwerkaufzugs. Untergliedert in die drei wesentlichen Schritte Modellierung, Aufstellen der Spezifikation und Berechnung des Supervisors wurde der Reglerentwurf durchgeführt. Im darauffolgenden Kapitel wurden kurz die Ergebnisse der Software in the Loop - und HIL - Simulation zusammengefasst.
Nachdem sich diese Arbeit als eine Art Leitfaden zum Aufbau eines Simulationssystems versteht, sollen nun noch die aufgetreten Probleme diskutiert werden.
Da die Simulationshardware für Demonstrationszwecke ausgelegt ist, übertreffen hier die

Kosten den Nutzen erheblich. Grundlegende Probleme der konventionellen Regelungstheorie wie eventuelle Grenzzyklen etc. treten in dieser Form hier im Kontext ereignisdiskreter Systeme nicht auf. Der Aufbau einer Simulationshardware in der dargestellten Form ist daher für die bloße Funktionsvalidierung nicht notwendig. In industriellen Anwendungen wären digitale Input/Output-Karten in Kombination mit einer graphischen Representation des realen Systems eher praktikabel.
Bereits bekannt ist das Problem der Zustandsraumexplosion mit zunehmender Streckengröße. Hier wurden verschiedene Lösungsansätze vorgestellt, z.B. der hierarachisch-dezentrale Supervisorentwurf nach Schmidt [2]. Mit diesem Verfahren ist es möglich den Zustandsraum erheblich zu verkleinern.
Dank der durchgehenden objektorientieren Programmierung ist die Erweiterung der libFAUDES, speziell des IO-Device's, relativ leicht durchzuführen. Die in dieser Arbeit aufgetreten Probleme sind eher auf das frühe Entwicklungsstadium der Wago-Software als auf strukturelle Defizite der libFAUDES zurückzuführen. Im Besonderen die weitgehend unvollständige Dokumentation sowie der zeitliche Rahmen verhinderten die Lösung der verbleibenden Probleme.
Leider konnten in dieser Arbeit nicht alle interessanten Probleme behandelt werden. Es verbleiben Performanzanalysen im Bezug auf die Leistungsfähigkeit des Wago-IPC. Zudem ermöglicht die libFAUDES grundsätzlich den Betrieb der Wago-SPS im Netzwerk.

Literaturverzeichnis

[1] libFAUDES: Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg, http://www.rt.uni-erlangen.de/FGdes/faudes/, Stand Juli 2009

[2] K. SCHIMDT: Hierachical Control of Decentralized Discrete Event Systems - Theory and Application, Doktorarbeit, Lehrstuhl für Regelungstechnik, Universität Erlangen Nürnberg, 2005

[3] W. M. WONHAM: Notes on Control of Discrete Event Systems. Department of Electrical & Computer Enginieering, University of Toronto, 2006