Prototypische Entwicklung einer zentralen Steuerung in der Gebäudeautomation mittels SoftSPS

Ralph Freudrich
(Matr.-Nr.: 1828519)

Berlin, 09. Oktober 2012

Diplomarbeit im Fachbereich 03: Mathematik / Informatik der
Universität Bremen
unter der Betreuung von Prof. Dr. Karl-Heinz Rödiger
und
Jürgen Maaß, Hauptabteilungsleiter F&E (Kieback&Peter GmbH & Co. KG)

Gutachter: Prof. Dr. Karl-Heinz Rödiger
Dr. Dieter Müller

Inhaltsverzeichnis

1. Einleitung[1]

Diese Diplomarbeit wird im Rahmen einer Analyse der künftigen Entwicklungs- und Vermarktungsorientierung der Firma Kieback&Peter GmbH & Co. KG, einem der führenden Unternehmen in der Gebäudeautomation, angefertigt.

Das Unternehmen verfügt über eine breite Produktpalette für die Realisierung der unterschiedlichsten Anforderungen im Heizungs-, Lüftungs- und Klima-Bereich (HLK) und bietet Lösungen u. a. aus den Bereichen Energieeffizienz, Raumautomation, Gebäudemanagement und Systemintegration.

1.1. Motivation

Die Gebäudeautomation in ihrer Gesamtheit kann dazu beitragen, die Energieeffizienz von Gebäuden nachhaltig zu verbessern und den primären Energiebedarf um bis zu 50 Prozent zu reduzieren. Bedenkt man, dass Gebäude mit einem Anteil von rund 40 Prozent des weltweiten Primärenergiebedarfs zu den größten Energieverbrauchen überhaupt zählen, wird schnell klar, dass eine moderne Automation eine lohnende Investition darstellt.

Gerade auch sogenannte Nichtwohngebäude stellen mit rund zwei Dritteln des Energieverbrauchs einen erheblichen Anteil dar, durch deren Modernisierung der Gebäudeautomation bzw. nachhaltig geplante Automation bei einem Neubau ein gewaltiges Einsparpotenzial an Energie und CO_2-Emmissionen realisiert werden kann.

In diesem Zusammenhang und auf Grund der zunehmenden Verknappung fossiler Brennstoffe, damit verbundener Steigerung der Energiepreise und einer massiven Verschlimmerung der Umweltbelastung in den letzten Jahrzehnten wurde von der Europäischen Union eine Richtlinie zur Gesamtenergieeffizienz von Gebäuden (*Energy Performance of Buildings Directive*, kurz: EPBD) verabschiedet.

Auf Basis dieser EPBD wurde in Deutschland die Energieeinsparverordnung (EnEV), welche einen Energiepass für Gebäude abhängig von deren Energieverbrauch vorschreibt, gesetzlich eingeführt. Zudem diente die EPBD dem europäischen Normungskomitee (CEN TC247) als Grundlage für die Entwicklung von Normen zur Beurteilung verschiedener Automationskomponenten unter Berücksichtigung deren Energieeffizienz. Auf dieser Basis werden Geräte durch die eu.bac getestet und zertifiziert. Hier bietet das Unternehmen Kieback&Peter GmbH & Co. KG mit dem Einzelraumregler RCN aus der technolon® Produktreihe einen der ersten eu.bac zertifizierten Einzelraumregler überhaupt an.

Als einen weiteren Baustein in der Gebäudeautomation ist die Raumautomation zu nennen, mit deren Hilfe sämtliche Geräte und Systeme eines Raumes zu einem integrierten System zusammengefasst werden. Hierbei kann es sich sowohl um Einzelräume – in diesem Fall wird von Einzelraumregelung gesprochen – oder bei größeren Räumen wie Büroetagen um Zonen – in diesem Fall spricht man von Zonenregelung – handeln.

Einerseits dient die Raumautomation dazu, gewisse Umgebungsbedingungen zu schaffen, in denen sich die Nutzer des Raumes bzw. der Zonen wohl fühlen. Zum

[1] Grundlage für die in der Einleitung enthaltenen Daten und Aussagen bilden [K&P2009] und [K&PRA].

Einsatz kommt Raumautomation z. Bsp. in Klassenräumen, Hotelzimmern und Büros, aber auch in Räumen, in denen stabilere Umgebungsbedingungen gefordert werden wie Laboren oder Museen.

Andererseits ist Raumautomation auch unabdingbar, um die bereits erwähnte Energieeffizienz von Gebäuden zu verbessern, indem die genutzte Energie möglichst genau an den realen Bedarf angepasst und damit Energieverschwendung vermieden wird. Es soll also nur solange Energie für verschiedene Kreisläufe wie Heizung, Klimatisierung oder Beleuchtung genutzt werden, wie diese durch die Nutzer tatsächlich beansprucht werden. Ohne genutzte Räume negativ zu beeinflussen, kann mit Raumautomation dafür Sorge getragen werden, dass mittels Fensterüberwachung Heizung und Kühlung bei offenem Fenster umgehend abgeschaltet werden. Weiterhin bietet sich hier die Möglichkeit, Jalousien ungenutzter Räume so zu steuern, dass diese im Winter das Sonnenlicht zur Unterstützung der Heizung in den Raum hineinreflektieren und im Sommer die Kühlung entlasten, indem der entspr. Raum verschattet wird.

Diese so verfügbaren Daten zum Energiebedarf bei der Raumautomation werden zudem für die Steuerung von Primäranlagen wie z. Bsp. Kesseln und Kälteanlagen genutzt, wodurch die grundlegende Energiebereitstellung bedarfsgenau optimiert und angepasst werden kann. Ebenso wie die zu Beginn erwähnten Planungen zur Energieeffizienz bietet auch die optimale Raumautomation bedeutende Möglichkeiten zur Energieeinsparung und damit zur Verbesserung der allgemeinen Energieeffizienz von Gebäuden.

Damit eine Schnittstelle zwischen den Gebäudeautomationssystemen nebst der darin integrierten Anlagen und dem Menschen existiert, sind Werkzeuge, die sog. Gebäudemanagement-Systeme, notwendig. Diese stehen dem technischen Facility Management bei der Bedienung, Überwachung, Wartung und Betriebsoptimierung der Anlagentechnik zur Seite, indem sie alle Anlagendaten verarbeiten und speichern. Die so gewonnenen Daten stehen schließlich für Auswertungen, Analysen und Protokolle zur Verfügung.

Für diese Anforderungen ist es unabdingbar, dass ein solches Gebäudemanagement-System unter vollständiger Ausnutzung seines Potenzials effizient arbeitet. Um dies zu realisieren muss dieses System alle Anlagen integrieren sowie unterschiedliche Protokolle verstehen und verarbeiten können. Insbesondere ist dies auch bei den Fabrikaten bzw. Anlagen unterschiedlicher Hersteller von Bedeutung.

Da ein solches Gebäudemanagement-System ein zentrales Werkzeug zur Bedienung einer Anlage darstellt, sollte dieses unbedingt hohen Bedienkomfort bieten. Hierbei sind besonders Transparenz, nachvollziehbare Anlagenbilder und eindeutige Beschreibungen notwendig, damit auch komplexe, anspruchsvolle Anlagen in Gebäuden oder größeren Liegenschaften fehlerfrei überwacht und bedient werden können. Dies gilt sowohl bei der direkten Bedienung vor Ort als auch beim Fernzugriff auf das System.

Abschließend sei hier noch die Systemintegration erwähnt, unter der man die Integration unterschiedlicher betriebstechnischer Anlagen in ein einheitliches Netzwerk von Automatisierung, Information und Kommunikation versteht.

Die Systemintegration unter diesen Aspekten ist die grundlegende Voraussetzung für die bereits erwähnte sichere und effiziente Anlagennutzung sowie die effiziente

Durchführung von Wartungsarbeiten. Zudem schafft die Systemintegration somit die Grundlagen für eine Erhöhung der Betriebssicherheit, der Lebensdauer von Anlagen als auch für die schon mehrfach erwähnte Energieeffizienz, indem sämtliche Anlagenkomponenten intelligent interagieren und dadurch möglichst umweltschonend betrieben werden können.

1.2. Ziel der Arbeit

Diese vier, einleitend bereits erwähnten Bestandteile der Gebäudeautomation – Energieeffizienz, Raumautomation, Gebäudemanagement und Systemintegration – stellen die Grundlagen für die hier vorliegende Arbeit und die damit verbundene Forschung dar.

Im Rahmen dieser Ausarbeitung wird es darum gehen, eine zentrale Steuerung[2] für die Gebäudeautomation unter Verwendung einer SoftSPS[3] prototypisch zu entwickeln.

Hierbei wird der Fokus darauf liegen, mit firmenfremder Hardware sowie einer firmenfremden Entwicklungsumgebung die hardwareübergreifende zentrale Steuerung bzw. Regelung eines Heizkreises umzusetzen.

Ziel dieser Arbeit ist es zu zeigen, dass innovative Entwicklung mit unkonventionellen Mitteln und die Realisierung aktueller Marktanforderungen durchaus vereinbar sind. Dabei liegt der Fokus darauf zu zeigen, dass die vorwiegend in anderen Automationsbereichen eingesetzten Konzepte Speicherprogrammierbarer Steuerungen auch Anwendung in den Aufgabenbereichen der Gebäudeautomation finden können.

Letzen Endes stellt diese Arbeit den Versuch dar, bewährte Konzepte der Gebäudeautomation zur Regelung eines Heizkreises mittels SoftSPS umzusetzen und durch diese Möglichkeiten eine möglichst modulare Steuerung prototypisch zu entwickeln.

1.3. Gliederung der Arbeit

Damit ein grundlegendes Verständnis für die (elektro-) technischen Anforderungen entsteht, werden im zweiten Kapitel die wichtigsten Grundlagen dazu erläutert. Zusätzlich werden hier die mathematischen Grundlagen verschiedener Regler und Regelmechanismen für die spätere Umsetzung in der Steuerung hergeleitet. Zudem erfolgt in diesem Kapitel ein für das spätere Verständnis notwendiger Einblick in die Grundlagen von Rechnernetzen.

Das dritte Kapitel schließlich verdeutlicht den Stand der Technik, indem die unterschiedlichen Ebenen sowie Bussysteme und -protokolle der Gebäudeautomation näher erläutert werden. Hierbei wird verdeutlicht, für welche Ebene bzw. welchen Bereich die Entwicklung im Rahmen dieser Arbeit erfolgen soll.

Den eigentlichen Schwerpunkt dieser Arbeit stellt schließlich das vierte Kapitel dar. Hier erfolgen detaillierte Erläuterungen zum gesamten Vorgehen während der Entwicklung. Dazu folgen einer Beschreibung des genutzten Programmiersystems und

[2] Der Begriff „Steuerung“ wird sowohl hier als auch im Titel verallgemeinernd und losgelöst von der später folgenden Definition genutzt.

[3] SPS: SpeicherProgrammierbare Steuerung; Gerät zur Steuerung/Regelung einer Maschine/Anlage, programmiert auf digitaler Basis; im Gegensatz zu herkömmlicher SPS bei SoftSPS kein Systemaufbau mit unterschiedlichen Baugruppen sondern ein zentraler Computer

der eingesetzten Hardware die Erläuterungen zum Vorgehen, um das Projekt grundlegend zu konfigurieren. Neben der Darlegung des geplanten Programmablaufs werden in diesem Kapitel Details zur vorgenommenen Implementierung sowie zur Umsetzung der im zweiten Kapitel beschriebenen mathematischen Grundlagen ausgeführt. Den Abschluss dieses Kapitels bieten schließlich eine auf den erwähnten Implementierungsdetails aufbauende Beschreibung des tatsächlichen Programmablaufs sowie Erläuterungen zur Umsetzung der Visualisierung.

Aus diesen Betrachtungen wird zum Ende in Kapitel 5 das Ergebnis dieser Arbeit eingestuft, indem eine Bewertung der genutzten Komponenten und der daraus resultierenden Entwicklung erfolgt. Inwiefern das Ergebnis dieser Arbeit letztlich den einleitend aufgeführten Erwartungen entspricht, wird im Abschnitt *Ergebniszusammenfassung* dieses Kapitel erläutert.

Den Abschluss dieser Arbeit stellt schließlich das sechste Kapitel dar. Hier erfolgt ein Ausblick auf unerwartete und ungelöste Probleme während der Entwicklung sowie die Möglichkeiten zur Verbesserung der aktuell vorliegenden Steuerungsrealisierung. Zudem erfolgt eine Auflistung möglicher und notwendiger Weiterentwicklungen, die im Rahmen weiterführender Entwicklungsarbeit umgesetzt werden könnten.

2. Theorie und Grundlagen

In diesem Kapitel werden zunächst die zum weiteren Verständnis der angestrebten Entwicklung notwendigen technischen Grundlagen beschrieben. Das Hauptaugenmerk liegt hierbei auf dem Bereich der Mess-, Steuerungs- und Regelungstechnik als maßgeblicher Bestandteil der Automationsebene[4] in der Gebäudeautomation.

Im zweiten Abschnitt dieses Kapitels wird zur Nachvollziehbarkeit der Themen des dritten Kapitels („*Bussysteme und -protokolle in der Gebäudeautomation*“) ein theoretischer Einblick in die Netzwerktechnologie erfolgen.

2.1. Grundlagen der Mess-, Steuerungs- und Regelungstechnik

Bei der Mess-, Steuerungs- und Regelungstechnik[5] geht es im Wesentlichen darum, physikalische Größen wie z. Bsp. die (Raum-) Temperatur, auf einen gewünschten Wert zu bringen bzw. auf diesem zu halten. Dies kann erfolgen, indem die physikalische Größe *gesteuert* oder *geregelt* wird.

2.1.1. Aufbau von Automationsanlagen

Zur Einführung in das Gebiet der Regelungstechnik im Bereich der Gebäudeautomation wird zunächst ein einfacher Heizkreis samt der darin enthaltenen maßgeblichen Komponenten erläutert.

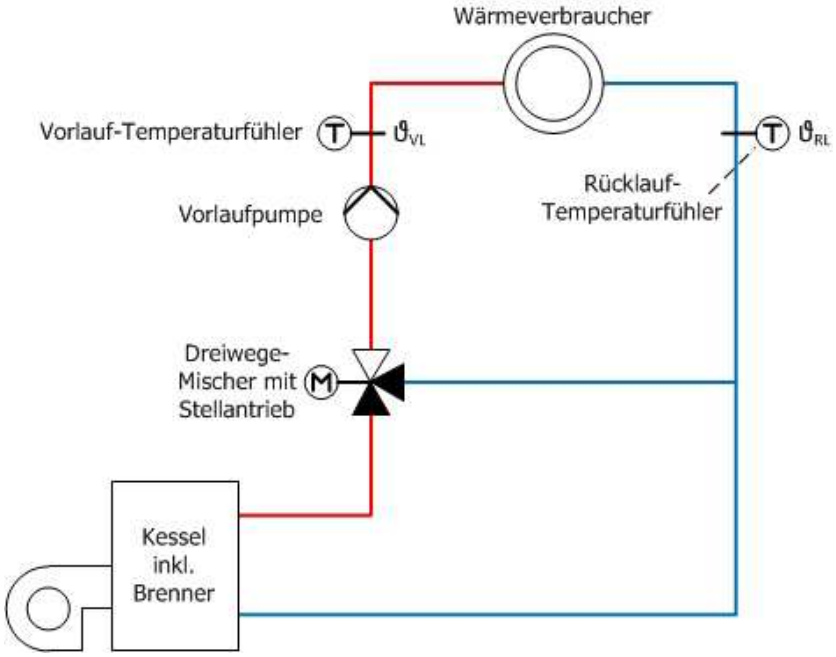

Abbildung 1: Ein einfacher Heizkreis mit dem Heizkessel inkl. Brenner, dem Dreiwege-Mischer mit Stellantrieb, der Vorlaufpumpe im Heizungsvorlauf, dem Temperaturfühler für die Vorlauftemperatur, dem Wärmeverbraucher sowie dem Temperaturfühler für die Rücklauftemperatur.

Grundlegendes Trägermedium der Wärmeenergie in einem Heizkreis ist das in den Leitungen des Kreislaufs bewegte Wasser. Dieses wird durch den Brenner des Kessel (in Abbildung 1 links unten dargestellt) erhitzt und dem Heizungsvorlauf (in Abbildung 1 rot) zugeführt. Über den in Abbildung 1 mittig dargestellten Dreiwege-Mischer mit

[4] Auf die Ebenenunterteilung in der Gebäudeautomation wird im Verlauf des folgenden Kapitels näher eingegangen.

[5] Regelungstechnik: Künftig wird dieser Begriff stellvertretend für den gesamten Bereich der Mess-, Steuerungs- und Regelungstechnik genutzt.

Stellantrieb erfolgt die ggf. notwendige Anpassung bzw. Absenkung (notwendig z. Bsp. bei Niedertemperaturheizkörpern oder Fußbodenheizung) der Heizungsvorlauftemperatur durch anteilige Beimischung des kühleren Heizmediums aus dem Heizungsrücklauf (in Abbildung 1 blau). Die Vorlaufpumpe im Heizungsvorlauf (in Abbildung 1 über dem Dreiwege-Mischer) sorgt für den notwendigen Transport des erhitzten Wärmeträgermediums zu den Wärmeverbrauchern[6] (in Abbildung 1 oben mittig, i. d. R. Heizkörper) und der Rückführung des Trägermediums zum Kessel, nachdem dieses die Wärmeenergie mittels der Wärmeverbraucher an die Umgebung abgegeben hat. Schließlich dienen noch zwei Temperaturfühler der Überwachung und Weiterverarbeitung einerseits der Vorlauftemperatur (in Abbildung 1 links oben: ϑ_{VL}) und andererseits der Rücklauftemperatur (in Abbildung 1 rechts oben: ϑ_{RL}) des Heizkreises.

Je nach erforderlicher bzw. erwünschter Temperatur für den Vorlauf werden folgende Arten von Heizungen unterschieden:

- Tieftemperaturheizung mit einer Vorlaufauslegung bis 40°C
- Niedertemperaturheizung mit einer Vorlaufauslegung bis 60°C
- Warmwasserheizung mit einer Vorlaufauslegung bis 100°C
- Heißwasserheizung mit einer Vorlaufauslegung zwischen 180°C und 220°C

Während Tief- und Niedertemperaturheizungen für den Betrieb als Fußbodenheizung oder mit größer dimensionierten Radiatoren genutzt werden, erfolgt der Einsatz von Warmwasserheizungen im Regelfall mit Vorlauftemperaturen um die 75°C. Im Gegensatz zu diesen drei Heizungsarten, welche vorwiegend im Privatbereich genutzt werden, erfolgt der Einsatz von Heißwasserheizungen überwiegend im industriellen Bereich bzw. bei älteren Fernwärmenetzen.

Zusätzlich zu dieser Differenzierung werden Heizungssysteme nach ihrer jeweiligen Art des Transportes des Wärmeträgermediums unterschieden und dabei eingeteilt in:

- Schwerkraftheizung
- Pumpenheizung

Bei einer Schwerkraftheizung erfolgt der Transport des Wärmeträgermediums durch eine geringe Druckdifferenz zwischen dem Vor- und dem Rücklauf. Diese entsteht, da das im Vorlauf erhitzte Wärmeträgermedium eine geringere Dichte als das bereits abgekühlte Wärmeträgermedium des Rücklaufes aufweist. Das abgekühlte Wärmeträgermedium ist somit schwerer, sinkt im Rohrsystem des Heizkreises nach unten und erzeugt so eine Schwerkraftzirkulation. Dies funktioniert allerdings nur, wenn der Heizkessel den bautechnisch tiefsten Punkt des Heizkreises darstellt und die am Heizsystem angeschlossenen Heizkörper möglichst hoch im zu beheizenden Gebäude montiert wurden. Aufgrund der relativ geringen Temperaturdifferenz, die für die Beheizung zu überbrücken ist (i. d. R. handelt es sich hier um einen Bereich von 20K), reagieren Schwerkraftheizungen zudem äußerst träge. Letztlich wird der wirksame Druck (Umtriebsdruck) maßgeblich durch die Differenz zwischen Vor- und

[6] Wärmeverbraucher, da die Wärmeenergie durch Abgabe an die Umgebung aus dem Heizkreis entnommen wird. Selbstverständlich wird diese Energie nicht „verbraucht“ sondern lediglich transferiert und steht dem Heizkreis letztlich nicht mehr oder nur noch in geringerem Maße zur Verfügung.

Rücklauftemperatur sowie die Anlagenhöhe und das allgemeine Temperaturniveau bestimmt, weshalb der Betrieb einer Schwerkraftheizung sowohl als Tief- oder Niedertemperaturheizung als auch als Etagenheizung nicht in Frage kommt.

Im Gegensatz zur Schwerkraftheizung werden bei einer Pumpenheizung die Höhenunterschiede sowie die Widerstände im Heizkreis durch den Einsatz einer elektrisch betriebenen Pumpe überwunden. Diese wird zumeist, wie in Abbildung 1 dargestellt, im Heizungsvorlauf des Heizkreises installiert. Eine solche Pumpenheizung bietet neben einer geringeren Trägheit und damit einhergehend einer besseren Regelbarkeit die Vorteile, dass der Heizkreis schneller aufgeheizt und für den Heizvorgang auch geringere Temperaturen für das Wärmeträgermedium nutzbar sind.

Im Normalfall kommen in heutigen Haushalten sog. PWWH-Anlagen (**P**umpen**W**arm**W**asser**H**eizungs-Anlagen) mit einer Vorlauftemperatur zwischen 60°C und 80°C zum Einsatz.

Für die folgenden Betrachtungen ist es zunächst noch notwendig, die Funktionsweise des in Abbildung 1 und in Abbildung 2 dargestellten Dreiwege-Mischers mit Stellantrieb zu erläutern. Wie bereits erwähnt, dient dieser zur ggf. notwendigen Anpassung bzw. Absenkung der Vorlauftemperatur ϑ_{VL} des Heizkreises. Hierbei wird über das Summentor (A) das von der Wärmequelle erhitzte Trägermedium in den Heizkreis geleitet und über den Bypass (C) wird mittels Stichleitung das abgekühlte Trägermedium vom Rücklauf des Heizkreises beigemischt. Das entspr. Mischverhältnis wird durch den (elektrischen) Stellantrieb reguliert, der im offenen Zustand 100% des erhitzten Trägermediums in den Heizkreis einspeist und die Zuleitung durch den Bypass vollständig blockiert. Das durch den Mischvorgang neu temperierte Trägermedium strömt schließlich über das Regeltor (B) in den Heizungsvorlauf. Dieser Dreiwege-Mischer stellt in einem Heizkreis das zentrale Element dar, da hierüber die Temperatur des Wärmeträgermediums im Heizungsvorlauf und damit die verfügbare Heizwärme reguliert wird.

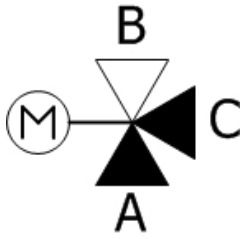

Abbildung 2: Dreiwege-Mischer mit Stellantrieb

A: Summentor
B: Regeltor
C: Bypass

2.1.2. Konzept der Steuerung

Das eigentliche Ziel eines wie in Abbildung 1 dargestellten Heizkreises besteht nun darin, die Temperatur in einem Raum mittels korrekt dosierter Wärmezufuhr unter Berücksichtigung äußerer Störeinwirkung (wie z. Bsp. der Außentemperatur) anzupassen bzw. möglichst konstant zu halten (vgl. [BOLLU], S.3).

Dies kann beispielsweise erfolgen, indem die aktuelle Außentemperatur (Führungsgröße) gemessen und anhand dieses gemessenen Wertes das Mischverhältnis des Dreiwege-Mischers eingestellt wird. So würde in diesem Fall eine niedrigere Außentemperatur zu einer größeren Öffnung des Mischers, also zu einer erhöhten Wärmezufuhr in den Heizkreis führen. Nicht berücksichtigt wird bei dieser Vorgehensweise also die eigentlich zu beeinflussende Größe, die Raumtemperatur (Steuergröße). Vielmehr bestimmt eine nicht von dieser Temperatur abhängige andere Größe (die Außentemperatur) die einzustellende Größe, also die Position des

Stellantriebes. Bei dieser Unabhängigkeit handelt es sich um das Hauptmerkmal einer **Steuerung** gem. DIN 19226:

> *„Das Steuern, die Steuerung, ist ein Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen. Kennzeichen für das Steuern ist der offene Wirkungsablauf über das einzelne Übertragungsglied oder die Steuerkette."*

Kennzeichnend für eine Steuerung gem. der obigen Definition ist demnach der nur in eine Richtung erfolgende Informationsfluss; die Steuergröße bleibt also unüberwacht. Der zentrale Bestandteil der Steuerung, das Steuerglied, erhält gem. Abbildung 3 den aktuellen über die Messeinrichtung erfassten Istwert der Führungsgröße und wertet diesen aus. In Abhängigkeit dieser Auswertung beeinflusst das Steuerglied den Stellantrieb, ohne jedoch eine rückgekoppelte Information darüber zu erhalten, ob diese Einflussnahme auf die Zielgröße im Sinne der gewünschten Zielparameter (einer konstanten Raumtemperatur) erfolgreich war oder nicht.

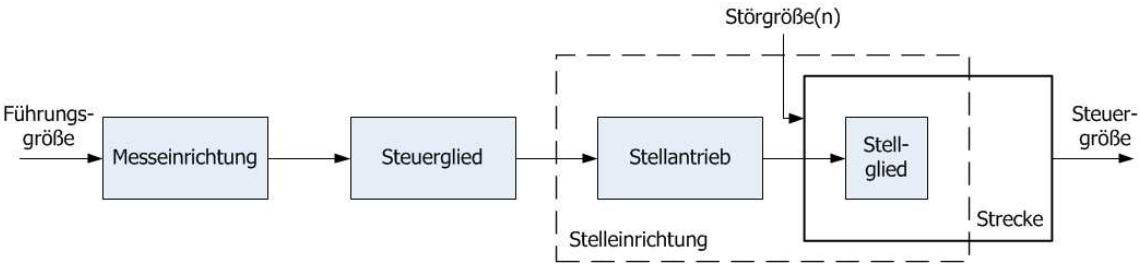

Abbildung 3: Blockschematische Darstellung des Wirkungsplans einer Steuerung (Quelle: in Anlehnung an [BOLLU], S. 4).

Diese in einer Steuerung fehlende Rückkopplung der Steuergröße auf das Steuerglied wirkt sich z. Bsp. dann nachteilig aus, wenn äußere Einflüsse – Störgrößen – auf die Strecke einwirken und somit die Steuergröße beeinflussen. Im Falle der Raumtemperatur würde beispielsweise die durch ein geöffnetes Fenster in den Raum eindringende kalte Luft als Störgröße auf die Steuergröße einwirken. Wird die Raumtemperatur ohne derartige Störeinflüsse in Abhängigkeit von der Außentemperatur konstant gehalten, so wird dies bei Einwirken solcher Störeinflüsse aufgrund des nicht vorhandenen Feedbacks nicht mehr gelingen. Die Steuerung arbeitet also erst dann wieder innerhalb normaler Parameter, wenn die bei der Planung nicht berücksichtigten Störeinflüsse beseitigt wurden.

2.1.3. Konzept der Regelung

Um den durch die fehlende Rückmeldung hervorgerufenen Mangel einer Steuerung zu beseitigen, ist es von Nutzen, die tatsächlichen Istwerte der Zielgröße über die Messeinrichtung zu erfassen und der Anlage zur Weiterverarbeitung zukommen zu lassen. Die Beseitigung der für eine Steuerung typischen Unabhängigkeit führt zu einer **Regelung**, die gem. DIN 19226 wie folgt definiert wird:

> *„Das Regeln, die Regelung, ist ein Vorgang, bei dem fortlaufend eine Größe, die Regelgröße (zu regelnde Größe), erfasst, mit einer anderen Größe, der Führungsgröße, verglichen und im Sinne einer Angleichung an die Führungsgröße beeinflusst wird. Kennzeichen für das Regeln ist der*

geschlossene Wirkungsablauf, bei dem die Regelgröße im Wirkungsweg des Regelkreises fortlaufend sich selbst beeinflusst“

Im Gegensatz zur Steuerung, bei der die Steuergröße nicht überwacht wird, existiert bei einer Regelung also ein Kreislauf, indem die zu regelnde Größe, die Regelgröße, auf die sie beeinflussende Einrichtung, die Regeleinrichtung, einwirkt. Eine solche Regelung wird als Regelkreis bezeichnet, in dem nun auf Störeinflüsse reagiert werden kann und die Regelung sich nicht auf eine bestimmte, nicht überprüfbare Reaktion der Regelgröße verlassen muss.

Im Falle des im Zusammenhang mit der Steuerung erwähnten Beispiels der Beeinflussung der Raumtemperatur, wird in einem Regelkreis nicht mehr die Außentemperatur als Referenzwert für die Einstellung des Dreiwege-Mischers erfasst. Vielmehr erfolgt eine Messung der durch den Regelkreis selbst beeinflussten Regelgröße, der Raumtemperatur, und eine Rückführung des Istwertes eben dieser in die Regeleinrichtung. Hierdurch hängt die Stellung des Dreiwege-Mischers und damit die Wärmezufuhr in den Heizkreis tatsächlich vom Istwert der Regelgröße ab. Wirkt nun, wie oben erwähnt, kalte Luft durch ein geöffnetes Fenster als Störgröße auf die Regelstrecke ein, erhält die Regeleinrichtung zeitnah eine Rückmeldung über die sinkende Raumtemperatur. Die Stellung des Dreiwege-Mischers wird entsprechend angepasst und ermöglicht eine erhöhte Zuführung des erhitzten Wärmeträgers in den Heizkreislauf.

Bis hierher dürften nun der grundlegende Aufbau eines Heizkreises samt seiner Komponenten und die Funktionsweise und Eigenschaften sowohl einer Steuerung als auch einer Regelung deutlich geworden sein.

2.1.4. Begriffe der Mess-, Steuerungs- und Regelungstechnik

Ergänzend dazu werden nun die im bisherigen Zusammenhang genutzten Begriffe wie *Regel-* und *Führungsgröße* anhand der schematischen Darstellung eines Regelkreises (Abbildung 4) näher beschrieben.

Ein solcher wie in Abbildung 4 dargestellter Regelkreis besteht im Wesentlichen aus zwei Komponenten:

a) der (Regel-) Strecke *S* und

b) der Regeleinrichtung *R*.

Die (Regel-) Strecke *S* repräsentiert hierbei die zwischen dem Stellort (an dem sich das Stellglied befindet) und dem Messort (an dem sich das Messglied befindet) liegende, zu regelnde Anlage.

Die vier Grundgrößen der Regelungstechnik, auf denen auch der in Abbildung 4 schematisch dargestellte Regelkreis basiert,

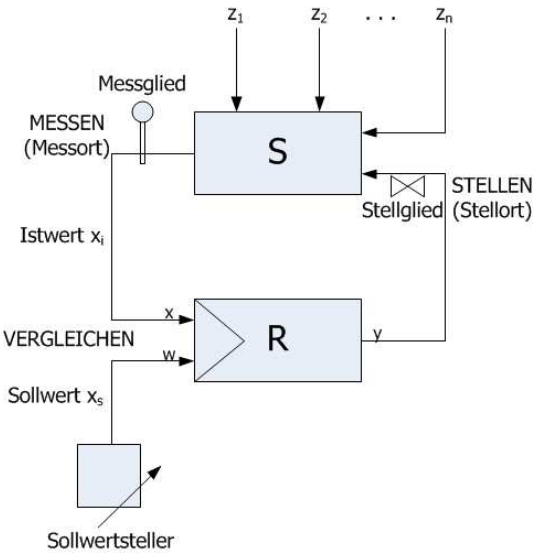

Abbildung 4: Schematische Darstellung eines Regelkreises als Blockschaltbild.
Hier zu erkennen sind der Regler *R* und die (Regel-) Strecke *S*. *x* symbolisiert die Regelgröße, x_i den Istwert, *w* die Führungsgröße, x_s den durch den Sollwertsteller vorgegebenen Sollwert, *y* die Stellgröße und z_1, $z_2 \ldots z_n$ stellen hierbei die Störgrößen dar.
(Quelle: in Anlehnung an [SCHRO], S. 14)

sind:

a) die Regelgröße *x*:

   In einer Steuerung wird diese Größe als Steuergröße bezeichnet. Es handelt sich hierbei um das eigentliche, zu beeinflussende bzw. konstant zu haltende, Wirkungsziel der Regelung. Im bereits erwähnten Beispiel des Heizkreises wäre dies die Raumtemperatur.

b) die Führungsgröße *w*:

   Dies ist die vorgegebene Größe, zu der sich die Regelgröße *x* innerhalb definierter Parameter äquivalent entwickeln soll. Wichtig in diesem Zusammenhang ist die Tatsache, dass es sich bei der Führungsgröße *w* um eine Messgröße handelt, die sich in ihrer Dimension von der der Regelgröße *x* unterscheidet. Im Beispiel des obigen Heizkreises würde die Führungsgröße *w* also nicht der vorgegebenen Raumtemperatur entsprechen, sondern der durch den entspr. installierten Temperaturfühler bei gewünschter Raumtemperatur gelieferten Messspannung. (vgl. [BOLLU], S. 7)

c) die Stellgröße *y*:

   Hierbei handelt es sich um die durch einen in der Regeleinrichtung erfolgten Vergleich des vorgegebenen Sollwertes x_s der Führungsgröße *w* mit dem gemessenen Istwert x_i der Regelgröße *x* ermittelte Größe. Sie stellt die Ausgangsgröße der Regeleinrichtung *R* und eine der Eingangsgrößen der (Regel-) Strecke *S* dar.

d) die Störgröße(n) z_1, z_2… z_n:

   Die Störgrößen z_1, z_2,… z_n sind die Größen, die unabhängig vom Regelkreis *R* und zumeist unvorhersehbar von außen auf die (Regel-) Strecke *S* einwirken, somit die Regelgröße *x* unerwünscht beeinflussen und vom Sollwert x_s entfernen.

Aufbauend auf diesen Erläuterungen und den schematischen Darstellungen in Abbildung 4 und Abbildung 5 sollen an dieser Stelle die einzelnen Komponenten und die Funktionsweise eines Regelkreises im Detail vorgestellt werden.

2.1.5. Aufbau und Funktion eines Regelkreises

Wie in Abbildung 4 zu erkennen, besteht in einem solchen Regelkreis die Aufgabe des am Messort befindlichen Messgliedes in der Erfassung (Messung) des Istwertes x_i der Regelgröße *x* (z. Bsp. Vorlauftemperatur ϑ_{VL} oder Raumtemperatur ϑ_R). In diesem Zusammenhang repräsentiert der Istwert x_i immer nur einen Momentanwert der Regelgröße *x* zu einem bestimmten Zeitpunkt.

Über den in Abbildung 4 dargestellten Sollwertsteller wird der Sollwert x_s der Führungsgröße *w* zur Weiterverarbeitung in der Regeleinrichtung *R* vorgegeben.

Diese Regeleinrichtung, die in Abbildung 4 noch als einzelner Block dargestellt ist, sollte zum tieferen Verständnis gem. Abbildung 5 etwas differenzierter betrachtet werden. Zu ihr gehören einerseits der Regler und andererseits ein Teil der Stelleinrichtung, die aus dem Stellantrieb und dem Stellglied besteht. Während der Stellantrieb (z. Bsp. der Stellantrieb eines Mischventils) der Regeleinrichtung

zugeordnet wird, gehört das Stellglied (z. Bsp. das Mischventil selbst) bereits zur (Regel-) Strecke *S*.

Die Stelleinrichtung mit ihren unterschiedlich zugehörigen Komponenten stellt also ein Bindeglied zwischen der Regeleinrichtung und der (Regel-) Strecke *S* dar.

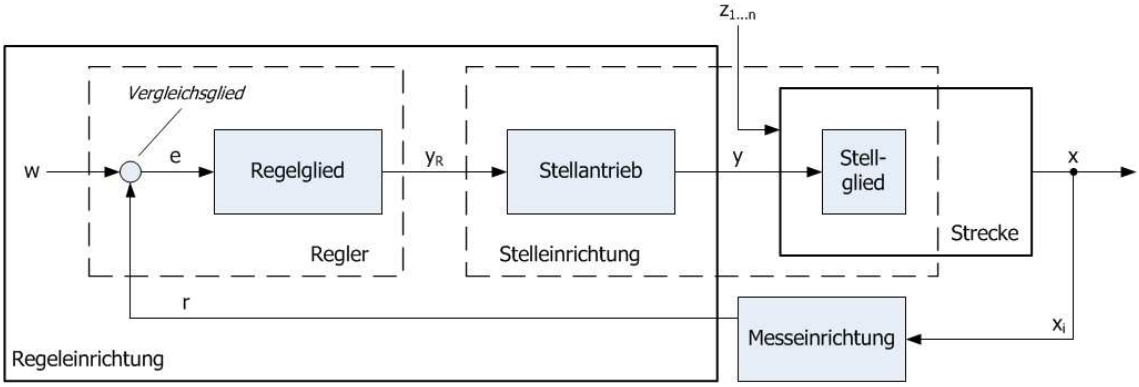

Abbildung 5: Blockschematische Darstellung des Wirkungsplans einer Regelung (Quelle: in Anlehnung an [FRAAS], S. 8 & [GEBHA], S. 15).

Wie bereits erwähnt, gehört neben dem Stellantrieb auch der Regler *R*, der sich aus dem Vergleichsglied und dem Regelglied zusammensetzt, zur Regeleinrichtung. Ebenso sind der Sollwertsteller und das Messglied als Bestandteil der Messeinrichtung der Regeleinrichtung zugehörig.

Als weitere Komponente der Messeinrichtung gehört der Fühler samt seiner Ummantelung allerdings zur (Regel-) Strecke *S*. Somit ist die Messeinrichtung das zweite Bindeglied zwischen der Regeleinrichtung und der (Regel-) Strecke *S*.

Die Messeinrichtung liefert also den als Istwert x_i erfassten Momentanwert der Regelgröße *x* als Rückführgröße *r* und der Sollwertsteller den als Führungsgröße *w* vorgegebenen Sollwert x_s an das Vergleichsglied des zur Regeleinrichtung gehörenden Reglers *R*. Die Rückführgröße *r* entspricht hierbei der der Regelgröße *x* proportionalen Messgröße, im Beispiel des Heizkreises also der Messspannung des Raumtemperaturfühlers. (vgl. [BOLLU], S. 7)

Das Vergleichsglied liefert mit der Differenz aus dem Sollwert x_s der Führungsgröße *w* und dem Messwert der Regelgröße *x*, also dem Istwert x_i, den Eingangswert für das Regelglied: die Regeldifferenz[7] *e*. Diese Regeldifferenz errechnet sich gem. folgender Formel:

Regeldifferenz = Sollwert – Istwert $e = x_s - x_i$ bzw. $e = w - x$	(2.1)

Die Regeldifferenz *e* wird schließlich im eigentlichen zentralen Bestandteil einer Regelung, dem Regelglied, als Eingangsgröße ausgewertet. Durch diese Auswertung der Regeldifferenz *e*, also der Information ob und in welcher Weise eine Abweichung der Regelgröße *x* von der Führungsgröße *w* vorliegt, kann das Regelglied die Reglerausgangsgröße Y_R ableiten (vgl. [BOLLU], S. 8). Im Beispiel des Heizkreises wäre dies dann die Spannung für den Stellmotor des Dreiwege-Mischers. Die dieser

[7] oftmals auch als Regelabweichung bezeichnet

Bestimmung der Reglerausgangsgröße Y_R zugrundeliegenden Algorithmen sind der zentrale Bestandteil einer jeden Regelung und werden im späteren Verlauf dieser Arbeit noch einen hohen Stellenwert einnehmen.

Als ausführende Komponente der Regelung transferiert der Stellantrieb die vom Regelglied in Form der Reglerausgangsgröße Y_R erhaltene Information über die Art und Weise der Beeinflussung der Regelgröße x in eine Änderung der Stellgröße y (vgl. [BOLLU], S. 9). In dem obigen Heizkreis wäre die Stellgröße y also die Stellung der Regelklappe des Mischventils.

Die von der Stellgröße y abhängige Komponente des Regelkreises wird Stellglied genannt und beeinflusst die Regelgröße x entweder auf indirektem oder direktem Wege. Aufgabe des Stellgliedes ist die Umsetzung der vom Stellantrieb gelieferten Stellgröße y in einen Masse- oder Energiestrom, welcher zur Beeinflussung der Regelgröße x der (Regel-) Strecke S zugeführt wird. In einem Heizkreis entsteht das Stellglied aus einer Kombination von Regelklappe des Mischventils, den Leitungen des Heizkreises und dem Wärmeverbraucher, also dem Heizkörper. Die Beeinflussung der Raumtemperatur ϑ_R erfolgt also über die Anpassung der Stellung der Regelklappe des Mischventils (Stellgrößenänderung) durch den Stellmotor (Stellantrieb) und die daraus resultierende Temperatur des Trägermediums des Heizkreises. An dieser Stelle wird der Momentanwert der Regelgröße x, der Istwert x_i, von der Messeinrichtung erfasst und, wie zu Beginn dieses Kreislaufes bereits beschrieben, als Rückführgröße r erneut an das Vergleichsglied des Reglers R geliefert. (vgl. [BOLLU], S. 9 & [SCHRO], S. 14)

2.1.6. Arten der Regelung

Nachdem nun verdeutlicht wurde, welches die Grundbegriffe und deren Funktionalitäten in einem Regelkreis sind, ist es noch notwendig, die Optionen zur Einflussnahme auf einen Regelkreis zu beschreiben. Es wird hierbei in Abhängigkeit der Vorgabe der Führungsgröße w zwischen drei Arten der Regelung unterschieden:

a) der Festwertregelung:

   Bei dieser Möglichkeit der Regelung wird am Sollwertsteller der Regeleinrichtung ein fester Sollwert x_s als Führungsgröße w vorgegeben. Im Falle des Heizkreises zur Raumtemperierung wäre dies hier die Raumtemperatur ϑ_R.

b) der Zeitplanregelung[8]:

   Der Sollwert x_s der Führungsgröße w wird bei diesem Vorgehen nach einem vorgegebenen Rhythmus, dem Zeitprogramm, geändert. Bei der Regelung der Raumtemperatur ϑ_R kann hier z. Bsp. zwischen Tag- und Nachtbetrieb der Heizungsanlage unterschieden werden. Im Rahmen einer Nachtabsenkung der Anlage wird der Sollwert für die Raumtemperatur x_{st} = 20°C am Tag ab 22.00 Uhr bis 6:00 Uhr des Folgetages auf den niedrigeren Sollwert x_{sn} = 16°C umgestellt.

[8] auch: Programmregelung

c) der Folgeregelung:

Bei einer Folgeregelung geschieht die Vorgabe des Sollwertes x_s der Führungsgröße w in Abhängigkeit einer nicht durch die Regeleinrichtung beeinflussbaren Messgröße. Der Sollwert x_s wird hierbei gemäß den Änderungen der unabhängigen Messgröße angepassten Vorgaben verschoben. In dem bisher benutzten Beispiel des einfachen Heizkreises wäre dies bei einer außentemperaturgeführten Regelung der Vorlauftemperatur der Fall. Die Führungsgröße w wäre hierbei die Außentemperatur ϑ_A während die Temperatur des Heizungsvorlaufs ϑ_V als Regelgröße anzusehen wäre, deren Sollwert x_s entspr. der Außentemperatur ϑ_A im Rahmen vorgegebener Gesetzmäßigkeiten angepasst wird.

2.1.7. Arten von Regelstrecken

Um die Regelung z. Bsp. der Vorlauftemperatur eines Heizkreises realisieren zu können, ist es unabdingbar, zumindest die grundlegenden Eigenschaften der durch die soeben beschriebenen Methoden zu beeinflussenden Regelstrecken näher darzulegen.

Grundlegend definiert wird eine Regelstrecke durch das ihr zugrunde liegende Übertragungsverhalten. Zur Bestimmung des Übertragungsverhaltens muss die entspr. Regelstrecke zunächst auf ihre Stell-Übergangsfunktion hin untersucht werden. Hierzu wird die Reaktion der Regelstrecke auf eine sprunghaften Änderung der Stellgröße y um einen Betrag Δy oder den gesamten Stellbereich Y_h hin untersucht. Diese sprunghafte Änderung am Streckeneingang wird als *Sprungfunktion der Stellgröße y* bezeichnet und ruft eine Wirkung am Streckenausgang, die *Sprungantwort der Regelgröße x* als Funktion der Zeit t, hervor. Diese aus dieser Beziehung von Ursache und Wirkung hervorgehende *Stell-Übergangsfunktion* definiert das Übertragungsverhalten einer Regelstrecke. Anhand dieses Übertragungsverhaltens erfolgt eine Unterscheidung zwischen den folgenden zwei Arten von Regelstrecken:

- Strecken ohne Ausgleich
- Strecken mit Ausgleich

Eine Strecke ohne Ausgleich strebt während des Regelungsvorgangs keinem endgültigen Wert für die Regelgröße x entgegen. Das heißt, es gibt keinen von der Stellgröße y abhängigen, gleichbleibenden Zustand der Regelgröße x, also keinen Ausgleichszustand. Vielmehr erfolgt bei einer Strecke ohne Ausgleich, die auch als I-Strecke bezeichnet wird, eine zur Stellgröße y und zur Zeitdauer Δt des Vorhandenseins dieser Stellgröße proportionale, anhaltende Regelgrößenänderung Δx. Die Regelgrößenänderung Δx ist also proportional zur Stellfläche $A_y = y * \Delta t$, wobei die Stellfläche A_y als Zeitintegral der Stellgröße y bezeichnet wird. (vgl. [SCHRO], S. 31 f.) Es gilt folgende Proportionalitätsbeziehung:

$$\Delta x \sim \int y \cdot dt \qquad (2.2)$$

Um aus dieser Proportionalitätsbeziehung eine Gleichung für die Änderung der Regelgröße Δx einer Strecke ohne Ausgleich herleiten zu können, wird die Verwendung eines Proportionalitätsfaktors, in diesem Fall als Integrierbeiwert K_I bezeichnet, erforderlich. Dieser Integrierbeiwert gibt die Geschwindigkeit der Änderung der

Regelgröße *x* bei einer Stellgröße *y* von einer Einheit (*y = 1*) an, ist also der Quotient aus der Regelgrößenänderung Δx und dem Zeitintegral der Stellgröße *y* gem. Gleichung (2.3). Aus dieser Gleichung kann schließlich die Gleichung für ein Strecke ohne Ausgleich hergeleitet werden (s. Gleichung (2.4)).

$$K_I = \frac{\Delta x}{\int y \cdot dt} \qquad (2.3)$$

$$\Delta x = K_I \cdot \int y \cdot dt \qquad (2.4)$$

Sind für eine Strecke ohne Ausgleich sowohl die Sprungfunktion der Stellgröße *y* über einen definierten Zeitraum Δt als auch die Sprungantwort der Regelgröße *x* in Form der Regelgrößenänderung Δx über den selben Zeitraum bekannt, lässt sich aus diesen Werten der Integrierbeiwert K_I berechnen.

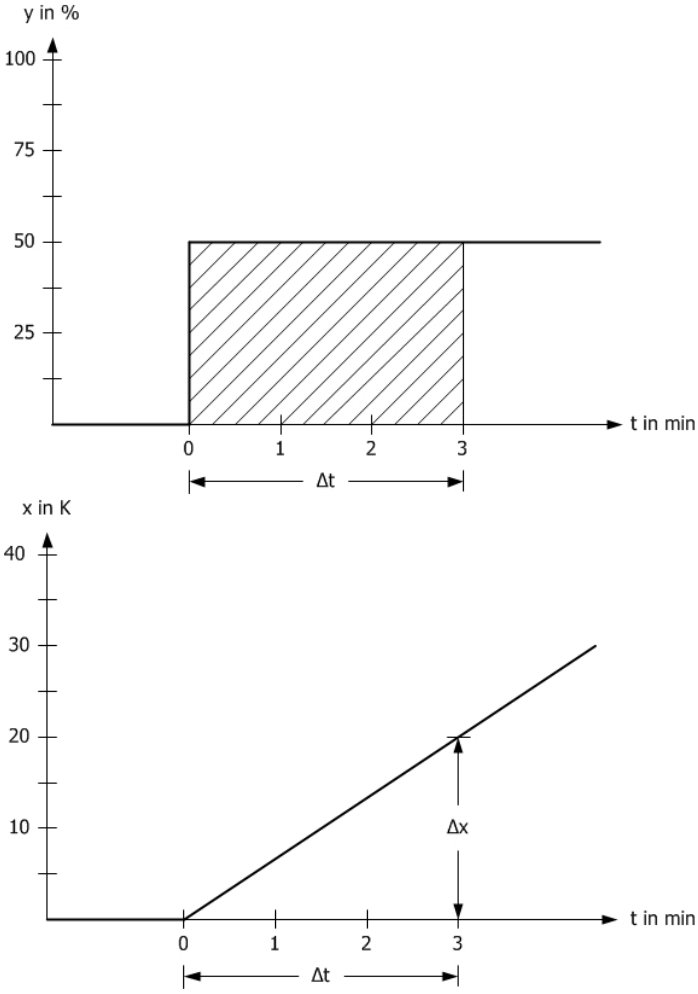

Abbildung 6: Stell-Übergangsfunktion einer I-Strecke (Regelstrecke ohne Ausgleich)

oben: Darstellung des Zeitintegrals der Stellgröße *y* mit $\Delta t = 3\ min$ und $y = 50\%$

unten: Darstellung der Regelgrößenänderung $\Delta x = 20K$ über den Zeitraum $\Delta t = 3\ min$

Das in dieser Darstellung verwendete Beispiel ist rein fiktiv, da in der Heizungstechnik keine reinen I-Strecken existieren. Für die hier dargestellte kontinuierliche Erhöhung der Regelgröße *x* müsste dem Heizkreis bei gleichbleibender Stellgröße *y* eine unendliche Menge an Wärmeenergie zugeführt werden. Andernfalls würde sich zwangsweise ein Ausgleichzustand einstellen.

(Quelle: in Anlehnung an [SCHRO], S. 32)

Für das Beispiel aus Abbildung 6 würde sich entspr. Gleichung (2.3) aus der Stellgrößenänderung $\Delta y = 50\%$, der Regelgrößenänderung $\Delta x = 20K$ und dem Zeitintervall $\Delta t = 3min$ der Integrierbeiwert $KI \approx 0.133^{K}/_{\%min}$ ergeben. Die Regelgröße *x* ändert sich bei einer Stellgröße von $y = 1\%$ (Öffnung des Dreiwege-Mischer z. Bsp.) in diesem Fall also mit einer Geschwindigkeit von $v_x \approx 0.133^{K}/_{min}$. Bei einer wie in Abbildung 6 angenommenen Stellgrößenänderung von $\Delta y = 50\%$ würde sich die

Regelgröße x auch mit der entsprechenden Geschwindigkeit von $vx = 0.133^{K}/_{\%min} * 50\% = 6.65^{K}/_{min}$ ändern. Bei einem Anliegen der Stellgröße über ein Zeitintervall von $\Delta t = 3min$ ergäbe sich wieder eine Regelgrößenänderung von $\Delta x = 6.65^{K}/_{min} * 3min \approx 20K$.

Im Gegensatz zu diesen I-Strecken, die in der Heiztechnik praktisch nicht vorkommen, sind für Strecken mit Ausgleich, sog. P-Strecken, mit der Temperatur-, Druck- oder Volumenstromregelung gleich diverse Anwendungsbeispiele vorhanden. Charakteristisch für P-Strecken ist die Tatsache, dass die Regelgröße x nach einer erfolgten Stellgrößenänderung Δy einen Ausgleichszustand anstrebt, dem keine weitere Regelgrößenänderung Δx mehr folgt.

Das Übertragungsverhalten einer P-Strecke wird demzufolge dadurch definiert, dass die Regelgrößenänderung Δx proportional zur Stellgrößenänderung Δy ist. Es gilt also folgende Proportionalitätsbeziehung:

$$\Delta x \sim \Delta y \qquad (2.5)$$

Um aus dieser Beziehung eine Gleichung zur Berechnung der Regelgrößenänderung Δx herleiten zu können, ist auch hier die Einführung eines Proportionalitätsfaktors erforderlich. In diesem Fall handelt es sich um den Übertragungsbeiwert K_P, welcher angibt, um wie viele Einheiten sich die Regelgröße x ändert, wenn bei der Stellgröße y eine Änderung von einer Einheit erfolgt. Der Übertragungsbeiwert K_P ist also der Quotient aus der Regelgrößenänderung Δx und der Stellgrößenänderung Δy gem. Gleichung (2.6) und führt zur Gleichung für eine Strecke mit Ausgleich gem. Gleichung (2.7).

$$K_P = \frac{\Delta x}{\Delta y} \qquad (2.6)$$

$$\Delta x = K_P \cdot \Delta y \qquad (2.7)$$

Da bei einer P-Strecke (Strecke mit Ausgleich) nun zusätzlich zur Stellgröße y auch die Regelgröße x in ihrem Wertebereich auf minimale und maximale Schranken begrenzt ist, folgt einer Änderung der Stellgröße y um den gesamten Stellbereich Y_h am Streckeneingang eine Änderung der Regelgröße x um den Regelbereich X_h der Strecke (vgl. [SCHRO], S.34). Für den Regelbereich einer P-Strecke gilt also folgende Beziehung:

$$X_h = K_P \cdot Y_h \qquad (2.8)$$

Mittels der Gleichungen (2.6) und (2.8) lässt sich bei bekanntem Regelbereich von z. Bsp. $X_h = 75K - 20K = 55K$ bei der Vorlauftemperatur eines Heizkreises und einem zugehörigen Stellbereich $Y_h = 100\%$ des Vorlaufmischers einfach der Übertragungsbeiwert der Strecke mit $K_P = 55^{K}/_{100\%} = 0.55^{K}/_{\%}$ bestimmen. Tritt am Streckeneingang also eine Stellgrößenänderung $\Delta y = 12\%$ ein, tritt gem. Gleichung (2.7) eine Regelgrößenänderung von $\Delta x = 0.55^{K}/_{\%} * 12\% = 6.6K$ auf.

2.1.8. Arten von Regeleinrichtungen

Bis zu diesem Zeitpunkt wurde das Proportionalitätsverhalten von Regelstrecken in Abhängigkeit nicht näher definierter Regeleinrichtungen (folgend als Regler bezeichnet) erläutert. Aufbauend auf diesen einführenden Erklärungen werden nun die verschiedenen Reglerarten und deren Funktionsweise beschrieben.

Im vorherigen Abschnitt wurden die Regelstrecken anhand ihres Übertragungsverhalten charakterisiert. Eben diese Charakterisierung wird genutzt, um die Reglertypen entspr. dem Übertragungsverhalten der Regelstrecke folgendermaßen zu unterteilen:

- unstetige Regler

  In diesen Bereich fallen Zweipunkt- und Mehrpunktregler, wobei im Rahmen dieser Arbeit nur die Zweipunktregler erläutert und genutzt werden.

- stetige Regler

  In den Bereich dieser Reglertypen fallen neben dem P- und dem I- auch deren Kombinationen und Erweiterungen in Form von PI- und PID-Reglern[9]. Diese Arbeit wird sich allerdings auf die Erläuterungen zu P-, I- und PI-Reglern beschränken, da PID-Regler in diesem Zusammenhang nicht genutzt werden.

2.1.8.1. Zweipunkt-Regler

Ein maßgebliches Element der unstetigen Regler stellt der Zweipunkt-Regler dar, da er im Normalfall für einfache Schaltvorgänge genutzt wird. Bei der mit diesem Regler realisierten Zweipunktregelung kann die Stellgröße *y* ausschließlich zwei Werte annehmen. Aus diesem Grund werden i. d. R. Stelleinrichtungen wie Pumpen, Brenner oder Schütze[10] mit den Stellzuständen *aus/ein* bzw. *zu/auf* von Zweipunktreglern angesteuert.

Die Übertragung des entsprechenden Stellsignals des Reglers erfolgt hierbei meist über einen Schalter wie z. Bsp. ein Relais. Diese Schalter erhalten von dem Zweipunkt-Regler dem programmierten Regelverhalten entsprechend das Signal zum Schließen oder Öffnen. Zumeist leitet ein solcher Zweipunkt-Regler von den aktuellen Messwerten anderer, oftmals stetiger und schwankender Eingangswerte ein Stellsignal ab. In diesem Zusammenhang unterliegt auch das abgeleitete

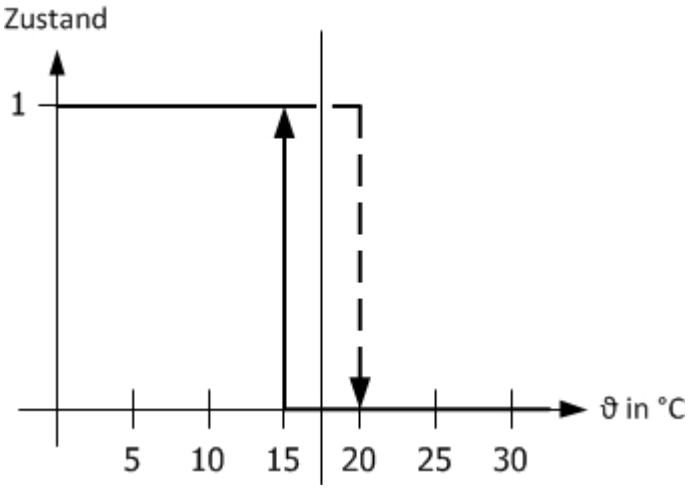

Abbildung 7: Darstellung einer beidseitigen Schalthysterese von 2.5°C um die Schaltgrenze von 17.5°C.

[9] P ≙ Proportional, I ≙ Integral, D ≙ Differential

[10] elektronischer oder elektrisch oder pneumatisch betätigter Schalter

Stellsignal entspr. Schwankungen um einen bestimmten Wert herum. Unterliegt das Stellsignal nun einem harten[11] Regelungsverhalten des Reglers, kann dies zu einer erhöhten Anzahl von Schaltvorgängen und damit zu erhöhter Belastung der Stelleinrichtungen führen. Aus diesem Grund erfolgt die Regelung über Zweipunkt-Regler grundsätzlich nicht ohne den Einsatz einer Funktion für die Schalthysterese.

Soll z. Bsp. wie in Abbildung 7 angedeutet unterhalb einer Außentemperatur von *17.5°C* der Brenner eines Heizkreises aktiviert und bei Überschreiten dieser Temperatur wieder deaktiviert werden, würde das punktgenaue Auslösen des entspr. Stellsignals bei Wertschwankungen der Außentemperatur um $^{+}/_{-}0.5°C$ zu einer erhöhten Belastung der Brennerkomponenten führen.

Bei der Umsetzung der Hysteresefunktion existieren grundsätzlich zwei verschiedene Möglichkeiten:

- Erweiterung des Schwellwertes in eine Richtung
- Beidseitige Erweiterung des Schwellwertes

Für den ersten Fall würde im obigen Beispiel der Hysteresewert von *2.5°C* entweder auf den Schwellwert addiert oder von diesem abgezogen werden. Bei einer Addition würde der Brenner erst bei Überschreiten bzw. Erreichen des erhöhten Schwellwertes von *17.5°C + 2.5°C = 20°C* deaktiviert werden, während eine Aktivierung nach wie vor bei Erreichen des ursprünglichen Schwellwertes von *17.5°C* erfolgen würde. Im Gegensatz dazu würde bei einer Subtraktion des Hysteresewerts der Brenner bei Erreichen oder Überschreiten des ursprünglichen Schwellwertes von *17.5°C* deaktiviert aber erst bei Unterscheitung des verringerten Schwellwertes von *17.5°C - 2.5°C = 15°C* wieder aktiviert werden. Mit diesen beiden Vorgehensweisen stehen sich zwei Methoden gegenüber:

1) eine punktgenaue Aktivierung des Brenners bei sinkender Außentemperatur und ein verzögertes Deaktivieren bei steigender Außentemperatur
2) eine punktgenaue Deaktivierung des Brenners bei steigender Außentemperatur und ein verzögertes Aktivieren bei sinkender Außentemperatur

Mit der ersten Methode wird auf jeden Fall ausreichend Wärme erzeugt, da der Brenner auch über den gewünschten Schwellwert hinaus betrieben wird. Der Nachteil dieser Methode liegt allerdings u. a. in einem erhöhten Ressourcenverbrauch, da ggf. auch bei Temperaturen Heizwärme erzeugt wird, bei denen dies nicht erforderlich wäre.

Die zweite Methode bietet u. a. den Vorteil eines geringeren Ressourceneinsatzes, da der Brenner punktgenau deaktiviert und erst bei unterschreiten einer geringeren Außentemperatur wieder aktiviert wird. Nachteilig wirkt sich dieses Vorgehen jedoch auf die Bereitstellung der Heizwärme aus, da der Heizbetrieb bereits erforderlich sein könnte, aber erst bei einer geringeren Außentemperatur erfolgt.

Diese Tatsachen bedingen im Normalfall den Einsatz der o. g. zweiten Möglichkeit, der beidseitigen Schwellwerterweiterung. In diesem Fall wird i. d. R. eine der Region entsprechende Heizgrenze als Schwellwert angenommen, die in beide Richtungen um den minimal möglichen Hysteresewert erweitert wird. Dieses Vorgehen vereint die

[11] der Regler aktiviert ein Stellsignal bei erreichen einer Schaltgrenze und deaktiviert dieses sofort wieder bei Unterschreitung der Schaltgrenze

Vorteile der beiden einseitigen Schwellwerterweiterungen. Zum einen wird hierbei nicht maßgeblich über die als Wohlfühltemperatur eingestufte Heizgrenze hinaus Heizwärme bereitgestellt. Zum anderen muss die Außentemperatur die Heizgrenze nicht zu deutlich unterschreiten, so dass die Heizwärme rechtzeitig zur Erzeugung der Wohlfühltemperatur bereitgestellt werden kann.

2.1.8.2. Stetige Regler

Im Gegensatz zu den unstetigen Reglertypen, bei denen die Stellgröße *y* ausschließlich eine feste Anzahl Zustände annehmen kann, ermöglichen stetige Regeleinrichtungen der Stellgröße y die Annahme jedes beliebigen Zustandes innerhalb des Stellbereiches Y_h.

Während bei den vorangegangenen Erläuterungen zu den Arten der Regelstrecken die Abhängigkeit der Regelgröße *x* von einer beliebigen, von einem Regler gelieferten Stellgröße *y* charakterisiert wurde, erfolgt die Systembetrachtung in diesem Fall von der Seite des Reglers. Es steht also die Betrachtung der gesetzmäßigen Abhängigkeit der Stellgröße *y* von der Regelgröße *x* im Vordergrund. (vgl. [SCHRO], S. 83) Hinsichtlich dieses Übertragungsverhaltens lassen sich standardmäßig folgende Arten von Regeleinrichtungen unterscheiden:

- P-Regler
- I-Regler
- PI-Regler
- PID-Regler

Wie eingangs bereits erwähnt, wird im Rahmen dieser Arbeit auf die ersten drei Arten von Reglern eingegangen, da letzterer hier keine Verwendung findet.

Grundsätzlich gilt es anzumerken, dass im Rahmen der HLK[12]-Technik vorwiegend P- und PI-Regler zum Einsatz kommen, während I-Regler aufgrund der typischerweise trägen Regelstrecken selten bis nie zum Einsatz kommen.

2.1.8.2.1. P-Regler

Bei einem P-Regler (proportional wirkenden Regler) ist die Stellgrößenänderung Δy proportional zur Regelgrößenänderung Δx und errechnet sich aus der Multiplikation mit einem Proportionalitätsfaktor, dem Übertragungsbeiwert K_P. Hierbei gibt der Übertragungsbeiwert an, um wie viele Einheiten sich die Stellgröße *y* ändert, wenn eine Änderung der Regelgröße *x* um eine Einheit erfolgt.

$$\Delta y \sim \Delta x \tag{2.9}$$

$$\Delta y = K_P \cdot \Delta x \tag{2.10}$$

Da der Stellbereich Y_h die Stellgröße *y* in definierten Grenzen einschränkt, ist auch die Proportionalitätsbeziehung zwischen der Stellgrößenänderung Δy und der

[12] Heizung-Lüftung-Klima

Regelgrößenänderung Δx nur für einen Teilbereich der Regelgröße x definiert. Dieser Teilbereich wird als P-Bereich X_P bezeichnet. Dieser P-Bereich definiert also die notwendige Regelgrößenänderung Δx für eine dem Stellbereich Y_h entsprechende Stellgrößenänderung Δy. Bei einem linearen Zusammenhang (der im gesamten Verlauf dieser Arbeit vorausgesetzt wird) zwischen der Regelgrößenänderung und der Stellgrößenänderung gelten also die in den Gleichungen (2.11) und (2.12) dargestellten Beziehungen.

$$X_P = \frac{Y_h}{K_P} \tag{2.11}$$

$$K_P = \frac{Y_h}{X_P} \tag{2.12}$$

In diesem Zusammenhang wird die Regelgrößenänderung Δx durch die Regelabweichung $e = w - x$ definiert. Damit die Regelabweichung e durch den P-Regler möglichst gering gehalten werden kann, ist es erforderlich, den Übertragungsbeiwert K_P so groß wie möglich festzulegen. Gerade im Rahmen des soeben erläuterten, begrenzten Stellbereiches Y_h birgt ein zu groß angesetzter Übertragungsbeiwert K_P allerdings die Gefahr des Überschwingens und im ungünstigsten Fall den Übergang des Reglers in eine dauerhafte Schwingung um den Sollwert herum.

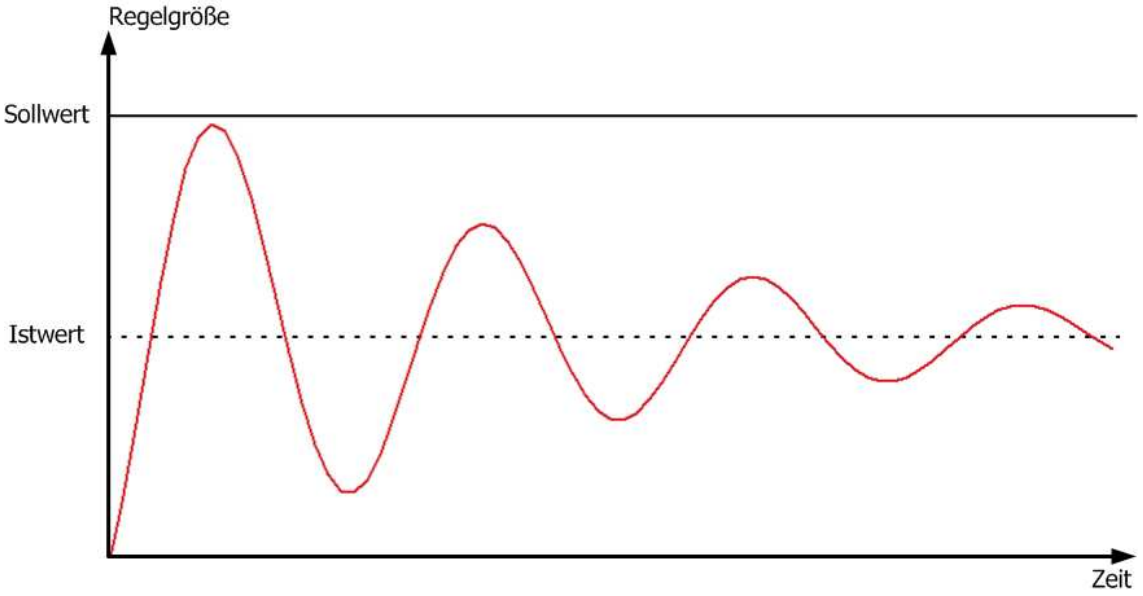

Abbildung 8: Verhalten eines P-Reglers. (Quelle: in Anlehnung an [BOLLU], S. 16)

In Abbildung 8 ist das typische Regelverhalten eines P-Reglers dargestellt, welches zugleich den ausschlaggebenden Nachteil dieses Reglers verdeutlicht. Da dieser Regler das Stellsignal y proportional zur vorhandenen Regelabweichung e bildet, ist dieses bei einer großen Regelabweichung e zu Beginn auch entspr. groß. Der Istwert x_i der Regelgröße x bewegt sich entspr. schnell auf den gewünschten Sollwert x_s zu. Die aus diesem Vorgang resultierende Abnahme der Regelabweichung e führt zu einer entspr. proportionalen Abnahme der Stellgröße y und dadurch zu einem erneuten, allerdings weniger starken Fallen des Istwertes x_i. Als Folge dieses Verhaltens pendelt sich der P-Regler, abhängig von der Größe des gewählten Übertragungsbeiwertes K_P, um einen

Istwert x_i knapp unterhalb des Sollwertes x_s ein. Es entsteht eine bleibende, durch einen P-Regler nicht zu beseitigende Regelabweichung e.

2.1.8.2.2. I-Regler

Die Stellgrößenänderung Δy ist bei einem I-Regler (integrierenden Regler) proportional zu der über einen Zeitraum Δt aufsummierten Regelabweichungen e. Diese gem. Gleichung (2.13) definierte Proportionalitätsbeziehung wird unter Einbeziehung eines Proportionalitätsfaktors, des Integrierbeiwertes K_I zu der diesen Reglertyp beschreibenden Gleichung (2.14).

$$\Delta y \sim \int e \cdot dt \qquad (2.13)$$

$$\Delta y = K_I \cdot \int e \cdot dt \qquad (2.14)$$

Hierbei bestimmt der Integrierbeiwert K_I die Verstärkung des Integralanteils der Reglergleichung. Veranschaulicht man die Gleichung für einen I-Regler, so entspricht der I-Anteil der Fläche unter der über die Zeit Δt durch die Regelabweichungen definierten Kurve. Diese Fläche kann bei digitalen Reglern durch die Summe der einzelnen Regelabweichungen *e(t)* innerhalb eines zu definierenden Abtastintervalls T_a angenähert werden. Hierbei wird als Annäherung an die evtl. tatsächlichen Regelabweichungen zum jeweiligen Zeitpunkt t eine Fläche der Höhe *e(t)* und der Breite T_a angenommen. Diese Teilflächen werden genauer, je mehr die Abtastrate erhöht, d. h. je kleiner das Abtastintervall T_a wird.

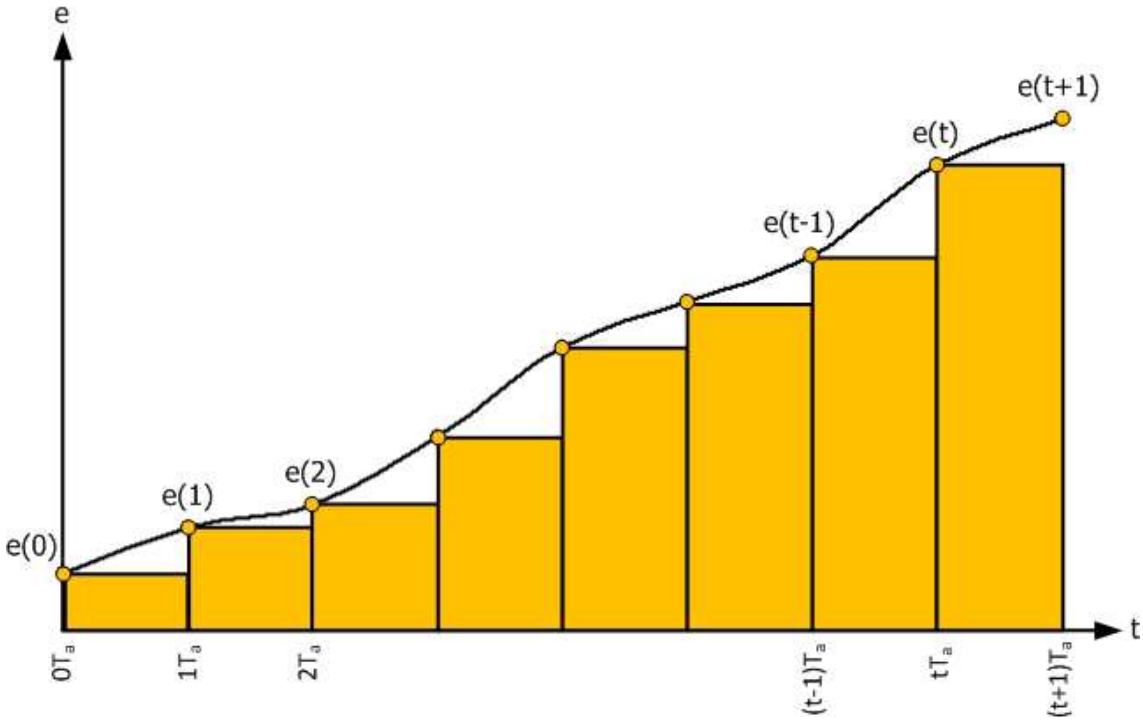

Abbildung 9: Annäherung an den Integral-Anteil eines I-Reglers mittels Summenbildung aus den einzelnen Rechteckflächen der Breite T_a und der Höhe e(t) zum jeweiligen Zeitpunkt t.

Legt man die in Abbildung 9 dargestellte Annäherung mittels Rechtecksummen zugrunde, so kann die Gleichung (2.14) eines I-Reglers zu folgender Summendarstellung auflösen:

$$\Delta y = K_I \cdot [e(0)T_a + e(1)T_a + ... + e(t-1)T_a + e(t)T_a]$$
$$\Delta y = K_I \cdot T_a \cdot [e(0) + e(1) + ... + e(t-1) + e(t)]$$
$$\Delta y = K_I \cdot T_a \cdot \sum_{i=0}^{t} e(i) \qquad (2.15)$$

Es ist deutlich erkennbar, dass mit einem I-Regler der Nachteil des P-Reglers der bleibenden Regelabweichung eliminiert wird. Dies erfolgt, sobald keine Regelabweichung mehr vorliegt und der I-Anteil in seinem Wert stagniert. Der Regler gibt ein gleich bleibendes Stellsignal y aus, bis erneut eine positive oder negative Regelabweichung vorliegt.

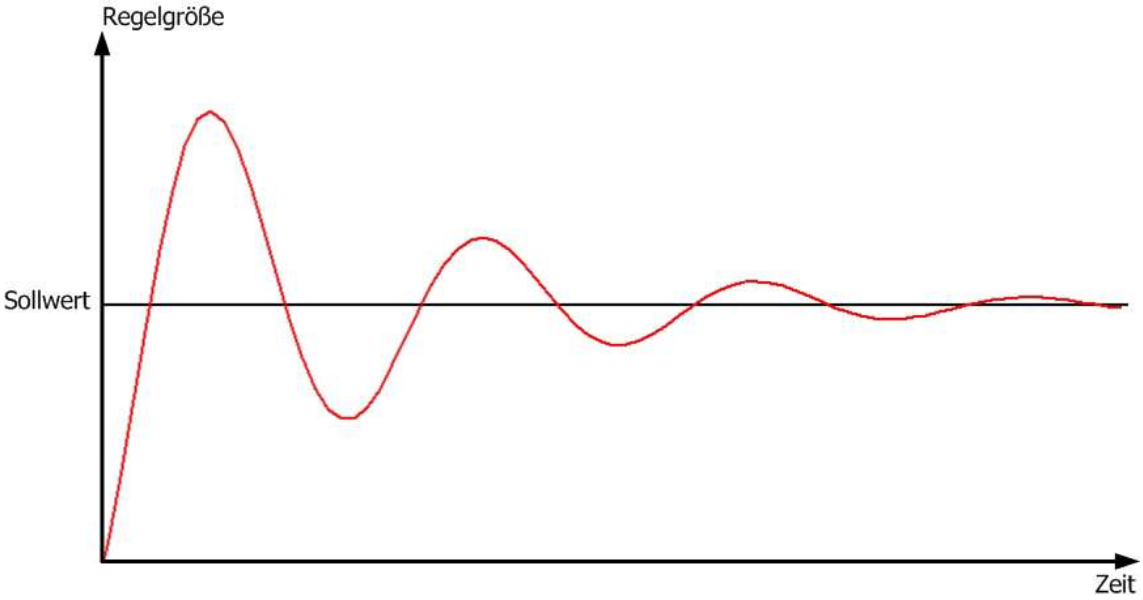

Abbildung 10: Verhalten eines I-Reglers. (Quelle: in Anlehnung an [BOLLU], S. 17)

Der Nachteil des I-Reglers liegt allerdings in seiner deutlichen Trägheit, da das Stellsignal maßgeblich von der Summe der einzelnen Regelabweichungen über einen bestimmten Zeitraum Δt abhängt. Der Wert einer einzelnen Regelabweichung zu einem bestimmten Zeitpunkt t fällt also weitaus weniger ins Gewicht als bei einem P-Regler, so dass der I-Regler den Istwert x_i wie in Abbildung 10 dargestellt in Form einer gedämpften Schwingung auf den Sollwert x_s regelt.

2.1.8.2.3. PI-Regler

Um eine typischerweise in der HLK-Technik vorkommende P-Strecke optimal regeln zu können, werden im Normalfall weder P- noch I-Regler getrennt eingesetzt. Diese würden entweder eine schnelle, jedoch nicht bleibende oder eine bleibende, aber äußerst träge Anpassung der Regelgröße an den Sollwert realisieren.

Aus diesem Grund werden die Vorteile der beiden eben erläuterten Regler zu einer Einheit, dem PI-Regler kombiniert. Dieser bietet mit der schnellen Reaktion als Vorteil

des P-Reglers und mit dem Vorteil des I-Regler günstiger stationärer Eigenschaften zwei wesentliche Aspekte, um eine Strecke zufriedenstellend regeln zu können.

Dabei setzt sich die Sprungantwort eines PI-Reglers aus zwei Teilen zusammen. Auf eine sprunghafte Änderung der Regeldifferenz erfolgt zuerst eine proportionale Verstellung des Stellgliedes, da der I-Anteil des Reglers zu Beginn noch keinen wesentlichen Einfluss auf das Stellsignal nimmt. Auf diese erste Proportionalverstellung folgt aufgrund der wachsenden Summe der Regeldifferenzen über die Zeit Δt eine zunehmende Verstellung des Stellgliedes mit einer durch die Nachstellzeit T_N beeinflussten Geschwindigkeit.

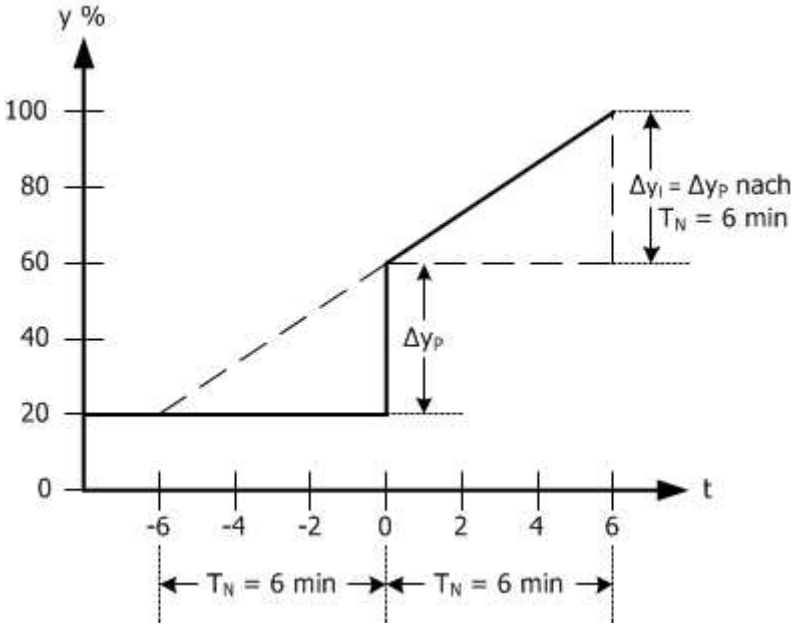

Abbildung 11: Darstellung der Sprung-Übergangsfunktion eines PI-Reglers und Verdeutlichung des Zusammenhangs zwischen der Stellgrößenänderung Δy_P und der Stellgrößenänderung Δy_I in Bezug auf die Nachstellzeit T_N.

(Quelle: in Anlehnung an [SCHRO], S. 130)

Mathematisch betrachtet stellt die Nachstellzeit T_N die Zeit dar, die die I-Verstellung für die gleiche Stellgrößenveränderung Δy benötigt, wie zuvor die P-Verstellung. Anschaulich dargestellt ist dieser Aspekt in Abbildung 11, auch wenn in diesem Beispiel ein idealer Regler mit einer senkrechten P-Verstellung angenommen wurde. Aus diesem Grund wird für die Bestimmung der Nachstellzeit T_N der Bereich des Zeitabschnitts nach der P-Verstellung angenommen. Diese genannten Punkte führen zu folgender Definition der Nachstellzeit T_N:

$$T_N = \frac{K_P}{K_I} \qquad (2.16)$$

Wie bereits erwähnt, ist die Nachstellzeit T_N also ein Maß für das Verhältnis des Übertragungsbeiwertes K_P der P-Komponente zum Integrierbeiwert K_I der I-Komponente. Wie aus den bisherigen Ausführungen und der Bezeichnung hervorgeht, handelt es sich bei einem PI-Regler um einen Zweikomponenten-Regler, dessen Funktion sich entspr. folgender Gleichung aus einer P-Verstellung Δy_P und einer I-Verstellung Δy_I zusammensetzt:

$$\Delta y_{PI} = \Delta y_P + \Delta y_I = (K_P \cdot e) + (K_I \cdot \int e \cdot dt) \qquad (2.17)$$

Benutzt man nun die oben aufgeführten Beziehungen für die Nachstellzeit, so lässt sich die Funktionsgleichung eines PI-Reglers folgendermaßen auflösen:

$$\Delta y_{PI} = (K_P \cdot e) + (K_I \cdot \int e \cdot dt)$$

$$\Delta y_{PI} = (K_P \cdot e) + (\frac{K_P}{T_N} \cdot \int e \cdot dt)$$

$$\Delta y_{PI} = K_P \cdot (e + \frac{1}{T_N} \cdot \int e \cdot dt) \qquad (2.18)$$

Nach Einsetzen der hergeleiteten Beziehung aus Gleichung (2.15) in die soeben aufgestellte Funktionsgleichung eines PI-Reglers ergibt sich folgende vereinfachte Darstellung dieser Funktion in Summenschreibweise:

$$\Delta y_{PI} = K_P \cdot (e + \frac{1}{T_N} \cdot \int e \cdot dt)$$

$$\Delta y_{PI} = K_P \cdot [e + \frac{T_a}{T_N} \cdot \sum_{i=0}^{t} e(i)] \qquad (2.19)$$

Diese Art zur Berechnung des Stellsignals für einen PI-Regler wurde schließlich im praktischen Teil dieser Arbeit genutzt, um einen solchen Regler auf Software-Basis für die Steuerung zu realisieren.

2.1.9. Die Heizkennlinie

Damit die im Rahmen dieses Projekts zu realisierende Steuerung überhaupt sinnvolle Regelprozesse ausführen kann, ist es erforderlich, einen Bezugspunkt zu definieren. In diesem Fall stellt der Messwert der aktuellen Außentemperatur diesen Bezugspunkt da, da hier eine außentemperaturgeführte Vorlaufregelung umgesetzt werden soll. Diese Regelung leitet den Sollwert für die aktuelle Vorlauftemperatur über einen zu definierenden Zusammenhang vom Messwert der aktuellen Außentemperatur ab. Diesen Zusammenhang repräsentiert die Heizkennlinie[13], da sie vorliegende Witterungszustände[14] auf einen regelbaren Wertebereich, die Vorlauftemperatur, der Heizungsregelung abbildet.

Um diese Abbildung algorithmisch realisieren zu können, müssen folgende Bestandteile zur Berechnung einer Heizkurve definiert werden:

- Auslegungstemperatur der Anlage
- Regelbereich der Anlage
- Steilheit der Heizkurve
- Parallelverschiebung[15] der Heizkurve

[13] Auch als Heizkurve bezeichnet, da es sich nicht zwangsweise um einen linearen Zusammenhang zwischen Außen- und Vorlauftemperatur handeln muss.
[14] Aus diesem Grund werden die entspr. Anlagen auch als witterungsgeführte Anlagen bezeichnet.
[15] auch: Niveau der Heizkurve

- durchschnittlicher Sollwert der Raumtemperatur
- Messwert der Außentemperatur

Die Auslegungstemperatur der Anlage gibt zum einen die maximal notwendige Temperatur für das Wärmeträgermedium bei einer minimal zulässigen Außentemperatur an. Zum anderen wird durch sie auch die minimal notwendige Temperatur des Wärmeträgermediums bei einer maximal zulässigen Außentemperatur definiert.

Das bedeutet, die Auslegungstemperatur definiert einen Temperaturbereich $\Delta\vartheta_{AT} = Max(\vartheta_{AT}) - Min(\vartheta_{AT})$, über dessen gesamten Wertebereich die Vorlauftemperatur innerhalb ihres Wertebereichs geregelt werden kann.

Dieser Wertebereich der Vorlauftemperatur stellt den Regelbereich der Anlage dar, der sich aus einer maximal und minimal zulässigen Temperatur zu $\Delta\vartheta_{VLT} = Max(\vartheta_{VLT}) - Min(\vartheta_{VLT})$.

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \qquad (2.20)$$

Aus diesen beiden Bereichen lässt sich die Steilheit der Heizkennlinie gemäß Formel (2.20) für die Steigung einer Geraden in einem kartesischen Koordinatensystem herleiten. Hierzu werden die beiden Extremwerte der Bereiche jeweils als ein Punktepaar betrachtet und gem. Formel (2.20) in Beziehung zueinander gesetzt.

Beim Aufgreifen des in Abbildung 12 dargestellten Beispiels beträgt gelten folgende Beziehungen:

- $Min(\vartheta_{AT}) = -20°C$
- $Max(\vartheta_{AT}) = 20°C$
- $Min(\vartheta_{VLT}) = 20°C$
- $Max(\vartheta_{VLT}) = 75°C$

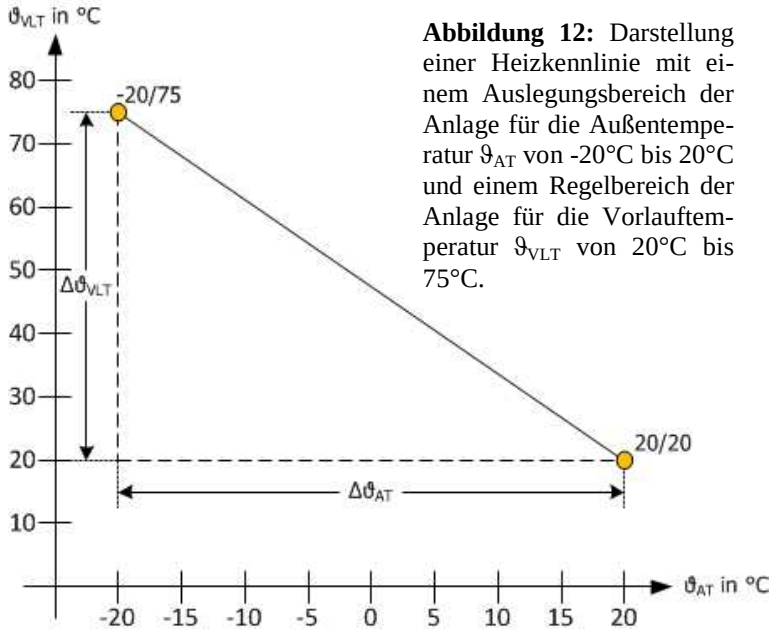

Abbildung 12: Darstellung einer Heizkennlinie mit einem Auslegungsbereich der Anlage für die Außentemperatur ϑ_{AT} von -20°C bis 20°C und einem Regelbereich der Anlage für die Vorlauftemperatur ϑ_{VLT} von 20°C bis 75°C.

In diesem Fall beträgt die maximale Vorlauftemperatur *75°C* bei einer minimalen Außentemperatur von *-20°C* und die minimale Vorlauftemperatur *20°C* bei einer maximalen Außentemperatur von *20°C*. Entsprechend den obigen Ausführungen können aus diesen Angaben die beiden, eine Gerade in einem kartesischen Koordinatensyste definierenden Punkte $P_1 = (x_1, y_1) = (-20, 75)$ und $P_2 = (x_2, y_2) = (20,20)$ abgeleitet werden. Aus diesen beiden Punkten kann schließlich die Steigung entspr. der Formel (2.20) wie folgt berechnet werden:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{20 - 75}{20 - (-20)} = \frac{-55}{40} = \underline{\underline{-1{,}375}}$$

Dieser Wert für die Steilheit der Heizkennlinie stellt ein Verhältnis zwischen der Vorlauf- und der Außentemperatur her und besagt in diesem Fall, dass die Vorlauftemperatur beim Anstieg der Außentemperatur um eine Einheit (*1°C*) um *-1,375* Einheiten steigt, also um *1,375°C* fällt. Von der anderen Seite betrachtet, muss ein Regler also dafür Sorge tragen, die Vorlauftemperatur um *1,375°C* anzuheben, wenn die Außentemperatur um *1°C* sinkt. Da diese Sichtweise der eigentlichen Regelrichtung entspricht wird für die Regelung der Vorlauftemperatur im Normalfall mit Vorzeichenumkehr gerechnet, welche auf die folgende Wiese bereits berücksichtigt wird:

$$m' = \frac{\Delta \vartheta_{VLT}}{\Delta \vartheta_{AT}} = \frac{Max(\vartheta_{VLT}) - Min(\vartheta_{VLT})}{Max(\vartheta_{AT}) - Min(\vartheta_{AT})} = \frac{75°C - 20°C}{20°C - (-20°C)} = \frac{55°C}{40°C} = \underline{\underline{1{,}375}}$$

Um mit Hilfe dieser Steilheit nun eine der Außentemperatur entsprechende Vorlauftemperatur berechnen zu können, sind noch drei weitere Informationen erforderlich. Zum einen ist es notwendig, die durch Heizleistung zu überbrückende Differenz zu berechnen und zum anderen müssen der aktuelle Messwert der Außentemperatur erfasst sowie der Sollwert für die durchschnittlich zu erreichende Raumtemperatur angegeben werden. Der aktuelle Messwert der Außentemperatur wird hierbei über einen entspr. analogen Eingang der Steuerung bereitgestellt während der durchschnittliche Sollwert manuell an der Steuerung vorgegeben wird. Aus diesen beiden Werten kann schließlich die durch die Anlage zu überbrückende Temperaturdifferenz gem. Formel (2.21) errechnet werden.

Temp.-Differenz = Sollwert Raumtemp. – Istwert Außentemp. $\Delta\vartheta = \vartheta_{swRT} - \vartheta_{iwAT}$	(2.21)

Da die Regelung und damit der Heizkreis mit einer minimalen Vorlauftemperatur betrieben wird, stellt diese Temperaturdifferenz den Wert für die Erhöhung der Vorlauftemperatur in Abhängigkeit der Außentemperatur dar. Diese zu überbrückende Temperaturdifferenz wirkt sich schließlich als Vielfaches der Steilheit auf die zu erreichende Vorlauftemperatur aus, so dass sich letztlich folgende Formel zur Erzeugung der Heizkennlinie und damit zur Berechnung der Vorlauftemperatur ergibt:

$\vartheta_{VLT} = Min(\vartheta_{VLT}) + (\vartheta_{swRT} - \vartheta_{iwAT}) * m' + PV$	(2.22)

Der letzte in dieser Formel, bisher nicht erläuterte Summand *PV* ist hierbei der Wert für die Parallelverschiebung der Heizkennlinie. Dieser ist notwendig, wenn eine Temperaturregelung mit unterschiedlichen Zeitprogrammen betrieben werden und beispielsweise während der Nacht eine Verringerung der bereitgestellten Heizwärme erfolgen soll. In einem solchen Fall würde die Heizkennlinie mit einem manuell einzustellenden Wert im Koordinatensystem noch oben oder nach unten verschoben. Eine negative Parallelverschiebung würde somit also eine Absenkung der durchschnittlichen Vorlauftemperatur und damit einhergehend das Erreichen niedrigerer Raumtemperaturen bei gleicher Außentemperatur bewirken.

Ein wichtiger Aspekt ist, dass die Heizkurve nur innerhalb der Grenzwerte der Außentemperatur $Max(\vartheta_{AT})$ und $Min(\vartheta_{AT})$ definiert ist und für Werte außerhalb dieses

Bereiches zwar korrekte Berechnungen liefert, welche allerdings außerhalb des Regelbereiches liegen. Aus diesem Grund muss bei einer softwaretechnischen Umsetzung darauf geachtet werden, diese Grenzfälle mit einer festen Einstellung für die Vorlauftemperatur abzufangen.

2.2. Theoretische Grundlagen von Rechnernetzen

Dieser Abschnitt wird eine überblickartige Einführung in die Grundlagen von Rechnernetzen geben und stellt die Basis für die Erläuterungen im Kapitel *Bussysteme und -protokolle in der Gebäudeautomation* dar.

2.2.1. Netzwerktopologien

Um einen gemeinsamen Austausch von Daten zwischen voneinander unabhängigen Einheiten zu ermöglichen, bedarf es einer grundlegenden Verbindung dieser Systeme untereinander. Hierzu ist der Einsatz einer den Anforderungen entsprechend geeigneten Struktur (Topologie) erforderlich. Im Folgenden werden die typischen Vernetzungsstrukturen kurz dargestellt und erläutert.

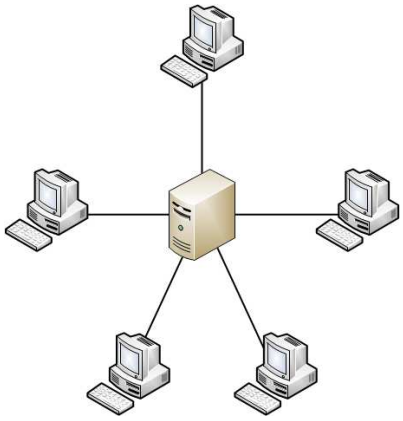

Abbildung 13: Sternstruktur eines Rechnernetzes mit zentralem (Vermittlungs-) Knoten

Bei einer Sternstruktur wie sie in Abbildung 13 dargestellt ist, werden alle in einem Netz befindlichen Teilnehmer an einen zentralen Knoten angeschlossen. In diesem Fall sind die einzelnen Stationen nicht miteinander verbunden und kommunizieren mittels Punkt-zu-Punkt-Verbindung (engl.: PTP = Point-to-Point) über die zentrale Vermittlungseinheit. Bei dieser zentralen Komponente muss es sich nicht zwangsläufig um eine spezialisierte Einheit mit eigener Steuerungsintelligenz handeln. In typischen lokalen Rechnernetzen übernimmt diese Funktion aber meist ein Switch. Fällt eine der Stationen aus, ist die übrige Kommunikation noch möglich, während bei einem Ausfall des zentralen Vermittlungsknotens das gesamte Netzwerk gestört und eine Kommunikation der Stationen untereinander nicht mehr möglich ist. Eine Option, die Wahrscheinlichkeit des Ausfalls des gesamten Netzes zumindest zu verringern, besteht darin, den zentralen Knoten redundant aufzubauen (zu doppeln).

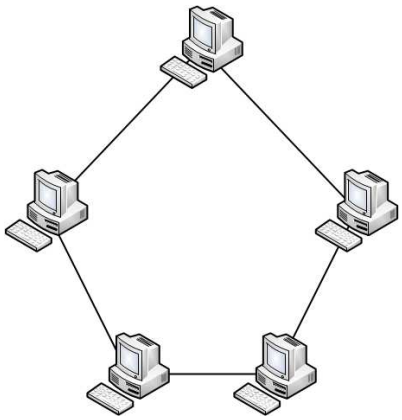

Abbildung 14: Ringstruktur eines Rechnernetzes mit den einzelnen aber ohne zentralen Knoten.

Eine weitere Möglichkeit, einzelne Rechnersysteme miteinander zu vernetzen, bietet die Ringstruktur. Im Gegensatz zur Sternstruktur existiert hier keine zentrale Vermittlungseinheit, welche den Datenaustausch der einzelnen Stationen untereinander ermöglicht. Vielmehr verfügt bei diesem Modell jede Station über einen eigenen Netzanschluss und ist über diesen jeweils zu zwei benachbarten Stationen verbunden. Jede Station übernimmt also die Funktionalität eines Netzwerkknotens. Die Datenübertragung erfolgt hierbei in eine festgelegte Richtung jeweils von einem Knoten zum nächsten, bis

der entspr. Zielknoten erreicht wurde. Während bei der Sternstruktur nur der Ausfall des zentralen Knotens, nicht aber einer Station, zu einem Erliegen der Netzkommunikation führt, genügt bei diesem Modell bereits der Ausfall eines Knotens, um die Kommunikation vollständig zu unterbrechen. Um die Wahrscheinlichkeit eines vollständigen Netzausfalls aber dennoch zu verringern, kommt bei diesem Modell inzwischen generell das sog. Protection-Switching zum Einsatz. Hierbei erkennt jeder Knoten dieser Struktur den Ausfall eines anderen Knotens und kann daraufhin von der ursprünglichen Richtung der Datenübertragung (dem Arbeitsweg) auf eine redundant vorgesehene Verbindung (den Ersatzweg) umschalten.

Die letzte Möglichkeit zur Vernetzung voneinander unabhängiger Rechner, die hier vorgestellt werden soll, ist die Busstruktur[16]. In dieser Struktur existieren weder ein zentraler Knoten noch einzelne als Knoten genutzte Stationen. Alle Stationen werden an ein gemeinsam genutztes Übertragungsmedium, den Bus, angeschlossen. Dies hat zur Folge, dass zu einem bestimmten Zeitpunkt immer nur eine Nachricht über den Bus transportiert werden kann. Um dies zu gewährleisten sind spezielle Verfahren notwendig, die gleichzeitige Zugriffe auf den Bus entweder verhindern oder erkennen und auflösen. Dies kann entweder mittels Arbitrierung[17] oder das CSMA/CD[17]-Verfahren erfolgen. Fällt in dieser Struktur eine Station aus, hat dies keinerlei Auswirkungen auf die übrigen Komponenten und das gesamte Netzwerk, da die einzelnen Station unabhängig voneinander und losgelöst von einem zentralen Knoten an den Bus angekoppelt sind. Da die Nachricht einer Buskomponente in diesem Netzwerk auf den gesamten Bus geleitet wird, ist die Terminierung der Busenden mittels geeignetem Widerstand unabdingbar. Ohne diese bei jedem Busteilnehmer vorhandene Möglichkeit zur Terminierung der Leitungsenden würden die elektrischen Signale am Busende nicht absorbiert werden und zu Echos führen. In diesem Fall käme es zu Kommunikationsfehlern innerhalb des gesamten Bussystems.

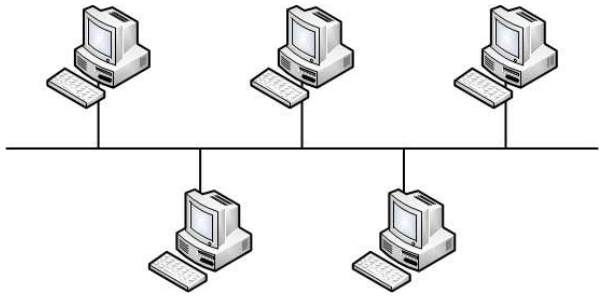

Abbildung 15: Busstruktur eines Rechnernetzes ohne einzelne und zentralen Knoten.

Nach dieser Einführung in die Netzwerktopologien sollen nun die Möglichkeiten zur Etablierung von Netzverbindungen sowie der Datenübermittlung in diesen Netzen kurz erläutert werden. Als Basis hierfür wird das ISO[18]/OSI-Referenzmodell[19] erläutert.

2.2.2. Das ISO/OSI-Referenzmodell

Das OSI-Referenzmodell stellt eine Grundlage für den Entwurf von Protokollen zur Kommunikation in Rechnernetzen dar. Wie der Name (offenes System für Kommunikationseinrichtungen) dabei andeutet, handelt es sich hierbei um ein Referenzmodell für herstellerunabhängige Kommunikationssysteme. Dazu definiert das Modell für die Kommunikationsaufgaben sieben aufeinander aufbauende Schichten

[16] auch Linienstruktur
[17] auf Arbitrierung und CSMA/CD-Verfahren wird in Kapitel 3.3: *Bussysteme und -protokolle in der Gebäudeautomation* näher eingegangen.
[18] ISO: International Organization for Standardization (Abk. abgeleitet vom griech. „*isos*“: *gleich*)
[19] OSI-Referenzmodell: Open Systems Interconnection-Referenzmodell

(engl.: *layer*), von der jede der über ihr liegenden Schicht Dienste bereitstellt und Dienste der unter ihr liegenden Schicht für sich nutzt. Für jede dieser Schichten ist dabei festgelegt, welche Aufgaben diese zu erfüllen hat. Die Umsetzung dieser Aufgaben übernimmt für jede Schicht ein entspr. (Kommunikations-) Protokoll, von denen je Schicht des Modells mehrere existieren. Ein solches Protokoll ermöglicht die Kommunikation zwischen einem Sender und einem Empfänger auf der entspr. Schicht; dies wird als logische (horizontale) Verbindung zwischen den Schichten bezeichnet. Eine vollständige Anordnung interagierender Protokolle von der Bitübertragungsschicht bis zur Anwendungsschicht wird als Protokollstapel (Protocol Stack) bezeichnet und ermöglicht erst den realen (vertikalen) Datenfluss.

Funktional werden die sieben Schichten dieses Modells in zwei Gruppen unterteilt, bei denen die Schichten eins bis vier die Gruppe der transportorientierten Schichten und die Schichten fünf bis sieben die Gruppe der anwendungsorientierten Schichten bilden.

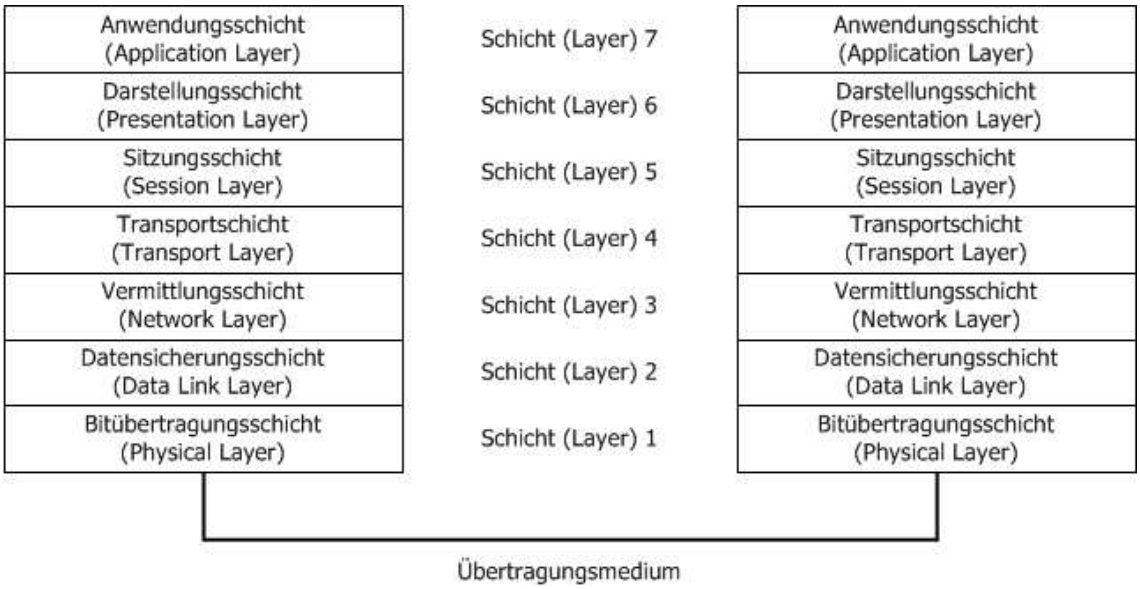

Abbildung 16: OSI-Referenzmodell für die Kommunikation in Rechnernetzen

Die Bitübertragungsschicht stellt in diesem Modell die unterste Ebene dar und definiert die notwendige Funktionalität zur physikalischen Übertragung digitaler Informationen. In diesen definitorischen Bereich gehört z. Bsp. Hardware wie die NICs[20], wobei das Übertragungsmedium selbst (also z. Bsp. das Netzwerkkabel) nicht Bestandteil der Definitionen des OSI-Modells ist.

Die zweite Schicht in diesem Modell ist die Datensicherungsschicht, welche die möglichst fehlerfreie Datenübertragung in einem Übermittlungsabschnitt gewährleisten und die Nutzung des Übertragungsmediums regeln soll. Hierzu werden Bitdatenströme in Blöcke (engl.: *Frames*) aufgeteilt und Verfahren zur Prüfsummenbildung im Rahmen der Kanalkodierung eingesetzt. Bei dieser Kanalkodierung werden den einzelnen Daten-Frames am Eingang des Übertragungskanals redundante Informationen hinzugefügt, die bei der Dekodierung am Ausgang des Übertragungskanals zumindest einen Rückschluss auf das Vorhandensein eines Übertragungsfehlers ermöglichen. Zudem ermöglicht die

[20] NIC: Network Interface Card (Netzwerkkarte)

Flusssteuerung[21] die dynamische Anpassung der Sendegeschwindigkeit von Daten-Frames an die Vorgaben des Empfängers.

In den Bereich der Vermittlungsschicht entfällt das Überwinden geographischer Entfernungen, indem Vermittlungssysteme (Router) in die Kommunikation zwischen zwei Endgeräten (Netzwerkknoten) einbezogen werden. Dies ermöglicht den Datentransfer zwischen zwei nicht direkt miteinander verbundenen Netzwerkknoten, ohne dass die auf dieser Ebene zwischengeschaltete Vermittlungseinheit den erhaltenen Daten-Frame auf die nächsthöhere Ebene weiterleiten muss.

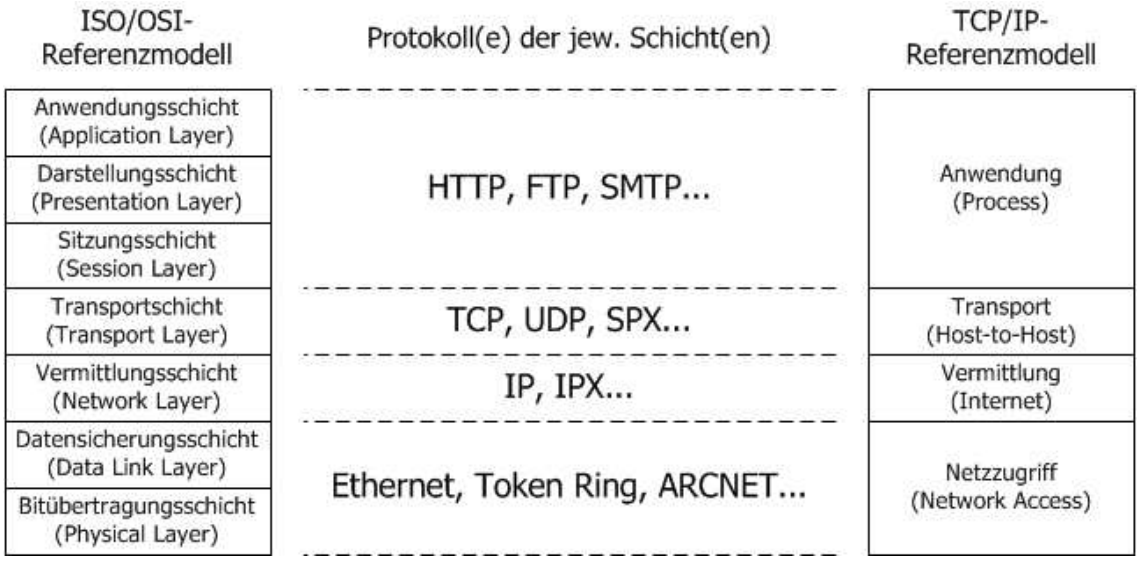

Abbildung 17: Die Ebenen des ISO/OSI-Referenzmodells und des TCP/IP-Referenzmodells[23] inkl. einer Auswahl zugehöriger Protokolle im Vergleich. (Quelle: in Anlehnung an [JACHS], S.2)

Die Transportschicht stellt mit ihrer Funktionalität die Schnittstelle zu den anwendungsorientierten Schichten dar, indem sie diesen einen klar definierten Zugriff auf die transportorientierten Schichten ermöglicht. Die höheren Schichten benötigen also keinerlei Kenntnis von den Eigenschaften des zugrundeliegenden Netzwerkes zu besitzen. Die Kernaufgaben dieser Schicht bestehen allerdings in der Segmentierung des Datenstroms sowie in der Ermöglichung des logischen Verbindungsaufbaus zwischen Anwendungsprozessen zweier unterschiedlicher Rechner.

Damit die geordnete Kommunikation zwischen zwei Netzwerkknoten sichergestellt werden kann, stellt die Sitzungsschicht Dienste für den notwendigen Datenaustausch zur Verfügung. Hierzu zählt u. a. die Bereitstellung und Verwaltung sog. Fixpunkte (Checkpoints) während einer Sitzung, um bei einem Ausfall der Transportverbindung diese Sitzung an diesem Checkpoint erneut zu etablieren und eine andernfalls erforderliche vollständige Neuübertragung der Daten zu vermeiden.

In der Darstellungsschicht erfolgt die Transformation lokaler Syntax und Zeichensätze eines Netzwerkknotens in für den Transport definierte Syntax und Zeichensätze. Da dieser Vorgang im Zielknoten des Netzwerkes auch umgekehrt erfolgt (gemeinsame Syntax/Zeichensätze zu lokalen Formaten), stellt die Darstellungsschicht den syntaktisch korrekten Datenaustausch auch zwischen zwei evtl. unterschiedlichen Systemen sicher.

[21] auch: Datenflusskontrolle

Die oberste Schicht des OSI-Referenzmodells, die Anwendungsschicht, definiert die Funktionen für den Zugriff auf das Netzwerk mittels Anwendungen. Sie steuert dabei die untergeordneten Schichten und passt die empfangenen Daten an den jeweiligen Anwendungsbereich an, um sie dem Anwendungsprogramm so zur Verfügung zu stellen.

2.2.3. Protokollstacks

Damit der vertikale Datenfluss innerhalb eines Systems sowohl von der Anwendungsschicht bis zur Bitübertragungsschicht als auch umgekehrt ermöglicht werden kann, erfolgt auf den verschiedenen Ebenen der Einsatz der bereits erwähnten (Kommunikations-) Protokolle. Allerdings existiert hier nicht für jede Schicht des Referenzmodells *das* Protokoll als Instanz, vielmehr haben sich für jede Ebene diverse verschiedene Protokolle mehr oder weniger etablieren können. Eine Sammlung von je einem Protokoll von der Bitübertragungsebene bis hin zur Anwendungsebene, aber auch für aufeinander folgende Teilebenen des Referenzmodells wird dabei als Protokollstack bezeichnet.

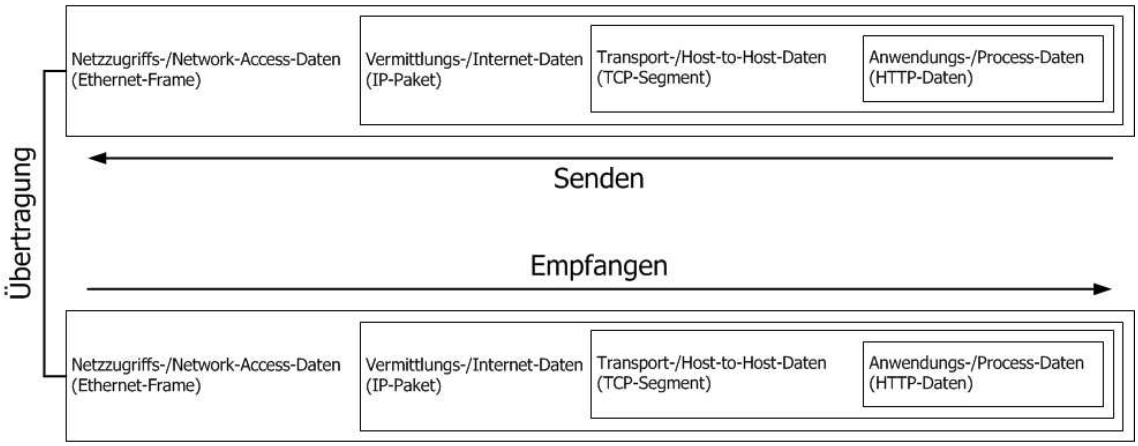

Abbildung 18: Schematische Darstellung des Aufbaus eines Ethernet-Frames unter Nutzung eines typischen Protocol Stacks der Internetprotokollfamilie, bestehend aus:

- http (Anwendungsebene)
- TCP (Transportebene)
- IP (Vermittlungsebene)
- Ethernet (Netzzugriffsebene).

Besonders die Internetprotokollfamilie[22] hat sich bei der Vernetzung von Rechnersystemen etablieren können und bietet die Grundlagen für viele der heute genutzten Protokollstacks. Noch vor der Einführung des ISO/OSI-Referenzmodells entstand hierfür das TCP/IP-Referenzmodell[23], welches für den Aufbau eines vollständigen Protokollstacks nur vier Ebenen definiert. Auch wenn es sich im allgemeinen Sprachgebrauch um das TCP/IP-Referenzmodell handelt, ist zu beachten, dass sowohl TCP (Transmission Control Protocol) als auch IP (Internet Protocol) Instanzen der Transport- und der Vermittlungsschicht dieses Modells darstellen. Die Umsetzung dieses Referenzmodells erfordert also weitaus mehr als diese beiden Instanzen, um einen vollständigen Protokollstack zu erstellen.

[22] auch als TCP/IP-Protokollfamilie bezeichnet
[23] ursprünglich DoD-Schichtenmodell (DoD = Department of Defence)

Ein typischer Anwendungsfall für einen solchen Protokollstack wäre z. Bsp. der Versand einer http-Nachricht über Ethernet. Hierbei werden die http-Daten in fast allen Fällen segmentiert in einen TCP-Frame eingebettet. Das so entstandene TCP-Segment besteht nun aus dem TCP-Header und den Nutzdaten (engl.: *Payload*) und wird wiederum als Datenpaket an die nächsttiefere, die Vermittlungs- bzw. Internetschicht weitergeleitet. Auf dieser Ebene erfolgt das Hinzufügen des IP-Headers und die Einbettung des TCP-Segments als Nutzdaten in ein IP-Paket. Auf der untersten Ebene, der Netzzugriffsebene, wird schließlich der Ethernet-Frame, bestehend aus den Header-Daten und den Nutzdaten in Form des IP-Pakets der darüber liegenden Schicht, generiert.

Diese Abarbeitung des Protokollstacks stellt die Umsetzung des bereits erwähnten vertikalen Datenflusses dar, bevor die so entstandenen Ethernet-Pakete über das Medium an den Empfänger übermittelt und da in umgekehrter Reihenfolge wieder aufgelöst werden. Hierbei werden auf jeder einzelnen Schicht gemäß den Vorgaben des eingesetzten Protokolls die zugehörigen Header-Informationen verarbeitet und der entspr. Frame entfernt, bevor eine Weiterleitung an die nächsthöhere Stack-Ebene erfolgt. Dies geschieht solange, bis die eigentlich zu übermittelnden Daten an die oberste Ebene des Empfängers weitergeleitet und somit der entspr. Anwendung zur Verfügung gestellt wurden.

3. Stand der Technik

Nachdem im vorangegangen Kapitel die anlagen- und regelungstechnischen Grundlagen der Gebäudeautomation beschrieben wurden, erfolgt hier nun ein Einblick in das System und den aktuellen technischen Stand der Gebäudeautomation.

3.1. Das Ebenenmodell der Gebäudeautomation

In unserer Gesellschaft existiert eine große Vielfalt verschiedenster Gebäude, alle für ihren speziellen Zweck konzipiert. Ganz gleich, ob es sich dabei um Wohn-, Industrie- oder öffentliche Gebäude wie z. Bsp. Schulen handelt, haben diese alle eine Gemeinsamkeit: Neben dem Vorhandensein einer entspr. Funktionalität ist der wichtigste Aspekt eines Gebäudes der Schutz der Gebäudenutzer vor äußeren Einflüssen und die Bereitstellung eines angenehmen Umgebungsklimas. Da Menschen moderner Industrienationen nahezu 95% ihres Lebens in Gebäuden verbringen (vgl. [SIEME], S. 7), ist es unabdingbar, mittels adäquater Raumluftqualität das soeben erwähnte angenehme Umgebungsklima zu realisieren. Zu den ausschlaggebenden Bestandteilen dieser Raumluftqualität gehören neben der Temperatur auch die in einem Gebäude oder Raum vorherrschenden Feuchte- und Lichtverhältnisse. Die optimale Ausgewogenheit dieser Aspekte führt erst zu einem annehmbaren Gebäude- bzw. Raumkomfort. Um diesen Gebäude- bzw. Raumkomfort optimal zu realisieren, kommen die Methoden und Hilfsmittel der Gebäudeautomation zum Einsatz. Wie der Begriff bereits andeutet, realisieren Automationssysteme die den Gebäuden und Räumen zugrundeliegenden Funktionen zum Einwirken auf das Umgebungsklima.

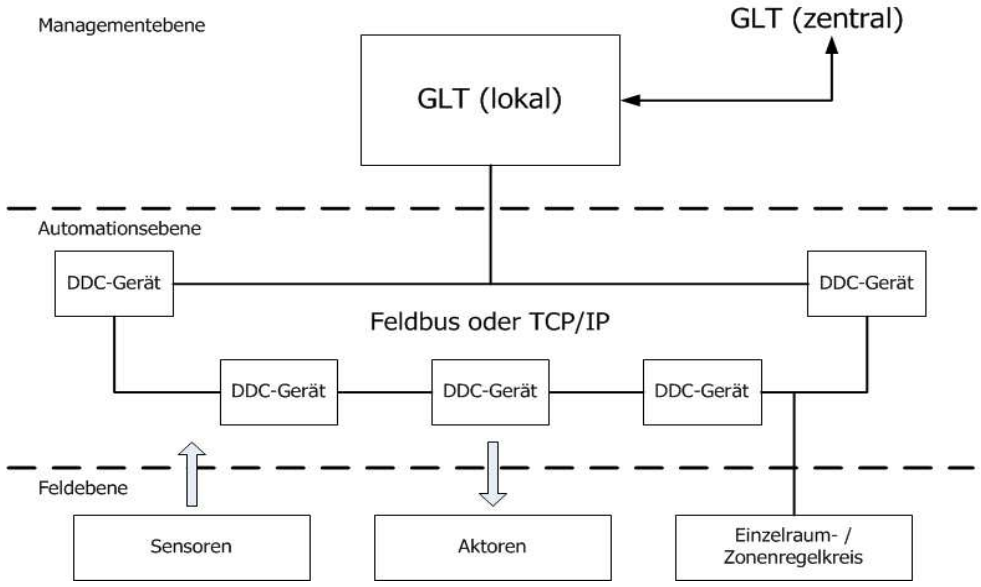

Abbildung 19: Schematische Darstellung der drei funktionalen Ebenen der Gebäude-automation (GA) inkl. der zugehörigen Komponenten.

Von den zu Beginn des 20. Jahrhunderts genutzten Analogreglern über die bis in die 1970er Jahre eingesetzten pneumatischen und rein elektrischen Regler wurden mit der Erfindung des Transistors die Aufgaben der Regelung auf elektronischem Wege ausgeführt. Ab dem Ende der 1970er Jahre erfolgte die Regelung mit dem zunehmenden Einfluss der Rechentechnik weitestgehend durch Digitalregler und ab Beginn der

1980er Jahre durch DDC-Systeme[24], deren Vernetzung über Bussysteme erfolgen konnte.

Aus diesen Grundlagen entwickelten sich die heute gängigen Standards und Konzepte in der Gebäudeautomation. Unter dem Begriff der Gebäudeautomation (GA) werden die zentrale Betriebsführung (Bedienung und Management) sowie die Überwachung, Optimierung, Steuerung und Regelung der Technischen Gebäudeausrüstung (TGA) zusammengefasst[25]. Die Unterteilung der Gebäudeautomation erfolgt dabei gem. Abbildung 19 in drei funktionale Ebenen:

a) die Feldebene,

b) die Automationsebene und

c) die Managementebene.

3.1.1. Feldebene der Gebäudeautomation

Die Feldebene stellt die Basis der Gebäudeautomation und den außerhalb lokal zentraler Vorrichtungen wie Schaltschränken befindlichen Bereich dar. Ihr zugeordnet sind die sog. Feldgeräte, die die Schnittstelle zu der zu regelnden bzw. steuernden Umgebung bilden. Neben den Sensoren wie Temperatur- oder Druckfühler gehören auch Aktoren wie Stellantriebe oder Pumpen zu diesen Feldgeräten.

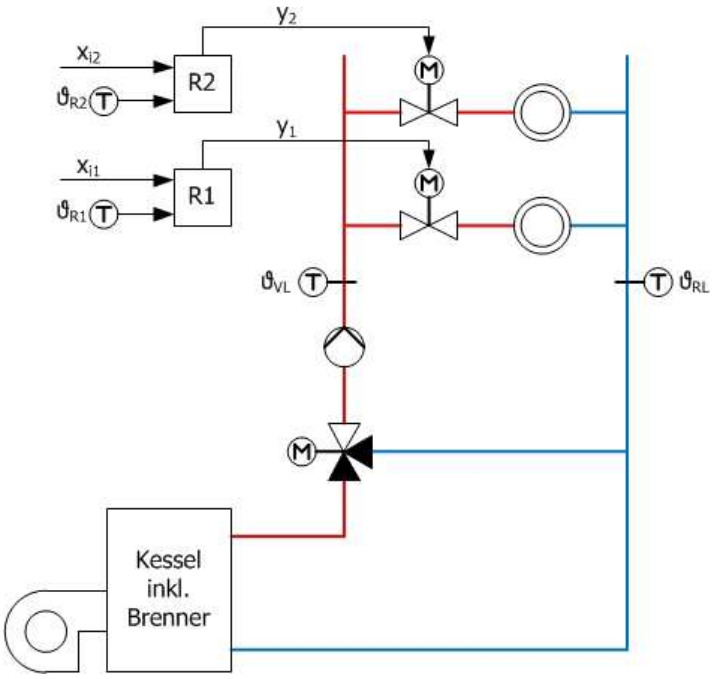

Abbildung 20: Anlage mit zwei Regelkreisen für unabhängige Räume (Einzelraumregelung).

x_{in}: Sollwert für Führungsgröße w des jew. Raumes n

ϑ_{Rn}: Messwert der Regelgröße x des jeweiligen Raumes n

R_n: Feldbusregler / Einzelraumregler des jew. Raumes n

y_n: Stellgröße für Durchgangsventil des jew. Raumes n

ϑ_{VL}: Vorlauftemperatur des ges. Heizkreises

ϑ_{RL}: Rücklauftemperatur des ges. Heizkreises
(Quelle: In Anlehnung an [SCHRO], S. 21)

Die Verbindung zur Kommunikation der Feldgeräte untereinander sowie zur darüberliegenden Automationsebene erfolgt über den Feldbus. Da Sensoren und Aktoren zumeist die einfachste Form der Feldgeräte darstellen, ist es üblicherweise nicht möglich, diese zwecks Interaktion mit der Automationsebene direkt in den Feldbus zu integrieren. Zu diesem Zweck werden sowohl komplexere Einzelgeräte als auch vollständige Regelkreise dem Bereich der Feldebene zugeordnet.

[24] DDC: Direct Digital Control
[25] vgl. Def. gem. VDI 3814 Blatt 1

Im Falle der komplexeren Einzelgeräte handelt es sich um Feldbusmodule, die mittels entspr. Ein- und Ausgänge den Anschluss einfacher Sensoren und Aktoren ermöglichen. Diese Module sind mit einer Schnittstelle ausgestattet, die die Anbindung an den Feldbus und somit an die Automationsebene ermöglichen.

Vollständige, von der Automationsebene oftmals nur indirekt abhängige Regelkreise sind u. a. bei Einzelraum- oder Zonenregelungen vorzufinden. Diese Regelkreise dienen der Ermittlung lokaler Umgebungszustände sowie der Einflussnahme auf diese. Hierzu zählt z. Bsp. die Erfassung und die Regelung einer Raumtemperatur unabhängig von den Bedingungen und Vorgaben eines anderen Raumes im selben Objekt. Die dafür eingesetzten Geräte werden Feldbusregler genannt und integrieren im Normalfall neben einem einfachen Sensor (in diesem Zusammenhang einen Temperaturfühler) sowie einem Ausgang z. Bsp. zur Ansteuerung von Stellantrieben von Misch- oder Durchgangsventilen noch einen Sollwertsteller.

Diese Einzelraumregler verfügen ebenfalls standardmäßig über eine Schnittstelle zur Anbindung an den Feldbus und verknüpfen somit die Feld- und die Automationsebene.

In der Feldebene werden also über die Sensoren Betriebszustände erfasst, während diese über die Aktoren verändert werden. Informationen werden in dieser Ebene also verarbeitet und für die nächsthöhere(n) Ebene(n) bereitgestellt. In den Bereich der Feldebene entfallen somit folgende Ein-/Ausgabefunktionen:

- Messen zur Erfassung von Momentanwerten wie Temperatur oder Druck (über Sensoren)
- Melden aktueller Stellungen von Schaltern (über Sensoren)
- Schalten z. Bsp. von Motoren (über Aktoren)
- Stellen von Ventil- oder Klappenantrieben (über Aktoren)

3.1.2. Automationsebene der Gebäudeautomation

Der eigentliche Kern einer regelungstechnischen Anlage in der Gebäudeautomation wird durch die Automationsebene repräsentiert. Zu ihr zählen die einzelnen für die Steuerung, Regelung und Überwachung der gebäudetechnischen Anlagen zuständigen DDC-Geräte und Kompaktregler. Im Normalfall erfolgt die Unterbringung dieser elektronischen Baugruppen in einem der zuständigen Anlage angehörigen Schaltschrank.

Sowohl bei den DDC-Geräten als auch bei den Kompaktreglern handelt es sich um zentrale Baugruppen der Gebäudeautomation, welche weitgehend autonom arbeiten. Während Kompaktregler kleine, für einen bestimmten Zweck spezialisierte Einheiten mit eingeschränktem Funktionsumfang und rudimentären Bedien- und Anzeigeoptionen darstellen, handelt es sich bei den komplexeren DDC-Geräten um aufwendige, mikroprozessorgesteuerte und oftmals mit einem komfortablen Bedienelement versehene computerähnliche Kompaktsysteme. Allen gemein ist jedoch die Integration zumindest einer Schnittstelle zum bereits erwähnten Feldbus, über den diese Baugruppen untereinander kommunizieren können und mittels der aufwendigeren DDC-Geräte an das Netzwerk der Managementebene angeschlossen werden. Die Verbindung zur Managementebene mittels der DDC-Geräte erfolgt dann allerdings nicht mehr über den Feldbus sondern entweder über gängige oder aber zumindest für

den Bereich der Gebäudeautomation standardisierte Kommunikationsmedien und -protokolle. Mehr zu diesem Thema folgt im Abschnitt *Bussysteme und -protokolle in der Gebäudeautomation* dieses Kapitels.

Diese zentralen Komponenten der Gebäudeautomation bieten mit ihren als Schnittstelle zwischen Mensch und Anlage fungierenden Bedienelementen bereits auf der Automationsebene Möglichkeiten zum Eingriff in die Regelung. Diese Möglichkeiten bestehen zum einen in der grundlegenden Konfiguration bestimmter Rahmenbedingungen der enthaltenen Regelkreise und zum anderen in der Überwachung der von den konfigurierten Regelkreisen erfassten Daten. Hierbei dienen einerseits die bereits erwähnten komplexeren Einzelgeräte wie Feldbusmodule aus der Feldebene und andererseits die in die DDC-Geräte und Kompaktregler integrierten Ein-/Ausgabeschnittstellen als Bindeglied zur Feldebene.

Besonders bei den komplexen DDC-Geräten verschwimmt die Abgrenzung zwischen der Feld- und der Automationsebene zusehends, da diese im Normalfall bereits mit einer geringen Anzahl sowohl analoger als auch digitaler Ein- und Ausgänge ausgestattet sind. Diese bieten in kleineren Anlagen mit nur einem oder wenigen Regelkreisen die Möglichkeit der direkten Ankopplung von Sensoren und Aktoren ohne den Umweg über einen Feldbus.

Eine weitere Option der indirekten Integration analoger und digitaler Ein- und Ausgänge besteht darin, DDC-Geräte mit sog. Erweiterungsmodulen auszustatten. In diesem Zusammenhang können einfache, mit entspr. Ein- und Ausgängen ausgestattete Feldgeräte direkt über am zentralen Regler vorgesehene Steckbrücken an das System gekoppelt werden. Im Regelfall erfolgt diese Anbindung dann jedoch wieder über den Feldbus, so dass eine eigentlich der Feldebene zugehörige Komponente in den Bereich der Automationsebene integriert wird.

Nicht zuletzt durch diese Lockerung der Grenzen zwischen der Feld- und der Automationsebene fallen folgende Funktionen (auch) in den Automationsbereich:

- Überwachen und Bedienen
- anlagen-/systemübergreifendes Optimieren
- Messen, Steuern, Regeln
- Schalten, Melden. Zählen

Die Möglichkeiten, die Grenzen zwischen der Feld- und der Automationsebene zu verschieben, haben zu einer erweiterten bzw. zusätzlichen Definition der Ebenen in der Gebäudeautomation geführt. Hierbei wurde die strikt hierarchische Ebenenunterteilung aufgelöst und durch ein flaches Modell für das Gebäudeautomationssystem ersetzt. Dieses System untergliedert sich danach in das Gebäudemanagementsystem, das (Anlagen-) Automationssystem inkl. Schaltschränken und das Raumautomationssystem[26]. Während im Ebenenmodell noch eine klare Trennung zwischen der Feld- und der Automationsebene erfolgte, sind die Funktionen dieser beiden Ebenen nun Bestandteil des Anlagen- und des Raumautomationssystems.

Während nach den Vorgaben des Ebenenmodells aus Abbildung 19 die Kommunikation zur Regelung noch maßgeblich zwischen den DDC-Geräten der

[26] vgl. *DIN 276 - Kosten im Bauwesen (Teil 1)*: Kostengruppe 480 (11/2006)

Automationsebene und den Feldgeräten bzw. den weitestgehend autarken Einzelraum- und Zonenreglern der Feldebene erfolgte, resultiert aus dem Modell gem. Abbildung 21 eine zunehmende Dezentralisierung der Regelung in der Gebäudeautomation. Werden im Bereich des Automationssystems DDC-Geräte vorwiegend zur Regelung von Energieerzeugern wie z. Bsp. Heizkesseln eingesetzt, übernehmen nicht minder komplexe Regler die Funktionalität zur Regelung der Umgebungsbedingungen in einzelnen Räumen oder Teilstrukturen eines Gebäudes (s. hierzu Kapitel 3.2: *Raumautomation als Bestandteil der Gebäudeautomation*). In beiden Teilsystemen sind die Regler dazu inzwischen mit den Möglichkeiten zur Anbindung an das entspr. Bussystem einerseits und an eine übergeordnete Netzwerkstruktur andererseits ausgerüstet (s. hierzu Kapitel 3.3: *Bussysteme und -protokolle in der Gebäudeautomation*).

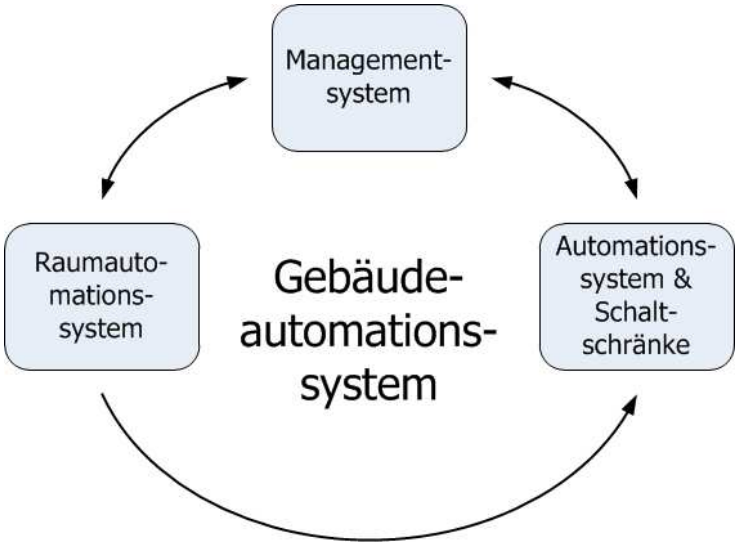

Abbildung 21: Schematische Darstellung der Gebäudeautomation in Anlehnung an die Definition der Kostengruppen im Bauwesen gem. DIN 276. Kostengruppe 480 „Gebäudeautomation“ unterteilt in:

- 481 „Automationssysteme“
- 482 „Schaltschränke“
- 483 „Management-Bedieneinrichtungen“
- 484 „Raumautomationssysteme“

Wie im Ebenenmodell gem. Abbildung 19 erfüllt die Managementebene bzw. das Managementsystem eine übergeordnete Aufgabe in der Gebäudeautomation. Die beiden Automationssysteme (Anlagenautomation und Raumautomation) kommunizieren jedoch unabhängig voneinander mit dem Managementsystem, während die Automationsebene vormals als Schnittstelle zwischen der Feld- und der Managementebene einzustufen war. Im Bereich des Raum- und Anlagenautomationssystems handelt es sich um eine einseitige Kommunikation vom Raum- zum Anlagenautomationssystem, um die bedarfsgeführte Regelung der bereits erwähnten Energieerzeuger anzufordern.

Beide Paradigmen, das streng hierarchische Ebenenmodell und das aufgelöste Modell, finden im Bereich der Gebäudeautomation vielfältig Anwendung und repräsentieren den aktuellen Stand der Technik. Um dies zu verdeutlichen werden an dieser Stelle exemplarisch zwei aktuelle Regler zweier Unternehmen vorgestellt. Es handelt sich hierbei um:

a) den BMR410 der Firma Kieback&Peter GmbH & Co. KG sowie

b) das Masterterminal PLM 717-1 inkl. Erweiterungsbaugruppe EWB 730.13 der Firma SABO Elektronik GmbH.

Bei dem Busmodulregler BMR410 handelt es sich um eine kompakte Automationsstation, die für die Regelung, Steuerung, Überwachung und Optimierung von Heizungs-, Lüftungs- und Klimaanlagen vorgesehen ist[27]. Da er einerseits mit der integrierten CAN-Schnittstelle[28] an den Feldbus angebunden und andererseits mit der vorhandenen Ethernetschnittstelle in gängige Netzwerktopologien integriert werden kann, entspricht er einem klassischen Regler der Automationsebene aus dem Ebenenmodell. Ebenfalls für diese Einordnung sprechen die zu Beginn dieses Absatzes erläuterten Aufgaben, welche als typischer Bestandteil der Automationsebene definiert sind. Die weiter oben angesprochene Auflösung der Grenzen zwischen der Feld- und der Automationsebene und damit der Übergang zu dem erweiterten Modell der Gebäudeautomation gem. DIN 276 ergibt sich allerdings aus den in diesem DDC-Gerät integrierten Ein- und Ausgängen. Neben acht universellen Ein- bzw. Ausgängen verfügt der BMR410 über zwei digitale Ein- und fünf digitale Ausgänge, womit Feldgeräte unabhängig weiterer Feldbuskomponenten direkt angeschlossen werden können. Der Regler ist somit Bestandteil des Automationssystems der Gebäudeautomation gem. der obigen definitorischen Einteilung.

Abbildung 23: Busmodulregler BMR410 der Firma Kieback&Peter GmbH & Co. KG

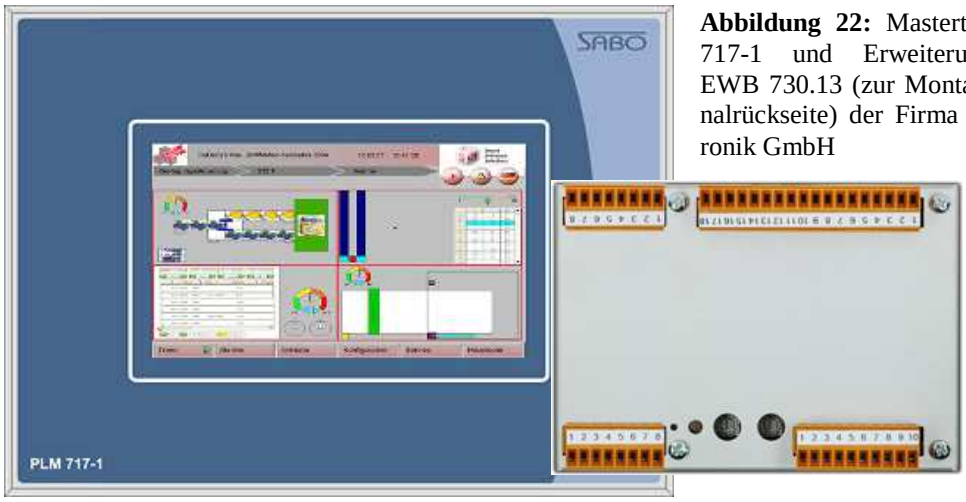

Abbildung 22: Masterterminal PLM 717-1 und Erweiterungsbaugruppe EWB 730.13 (zur Montage an Terminalrückseite) der Firma SABO Elektronik GmbH

Das Masterterminal PLM 717-1 allein für sich stellt bereits einen – wenn auch sehr rudimentären – Regler dar. Im Gegensatz zum o. g. Busmodulregler enthält dieser Regler einerseits eine komfortable Bedienoberfläche in Form eines 4,3-Zoll-Touchdisplays, andererseits jedoch nur je vier digitale Ein- und Ausgänge. Zusammen mit der vorhandenen CAN-Schnittstelle zur Feldbusanbindung und der optional integrierbaren Ethernet-Schnittstelle zur Einbindung in eine entspr. Netzwerkstruktur

[27] vgl. Datenblatt „Automationsstation BMR – Bedienfreundlichkeit mit Effizienz“ der Firma Kieback&Peter GmbH & Co. KG

[28] zu Bussystemen und -protokollen s. Kapitel 3.3: *Bussysteme und -protokolle in der Gebäudeautomation*

entspricht auch dieses Terminal dem klassischen DDC-Gerät der Automationsebene. Bereits die in diesem kompakten Terminal integrierten digitalen Ein- und Ausgänge bieten die Möglichkeit, reine Feldkomponenten direkt anzusteuern wodurch auch in diesem Fall die ehemals klare Trennung zwischen Feld- und Automationsebene zumindest aufgelockert wird. Wird das Masterterminal nun noch über die CAN-Schnittstelle mit der einleitend erwähnten Erweiterungsbaugruppe (EWB 730.13) ausgerüstet und so mit zusätzlichen analogen und digitalen Ein- und Ausgängen versehen, entfällt die klare Ebenentrennung. Der Regler samt seiner Schnittstellen ist in diesem Fall klar dem Automationssystem gem. DIN 276 zuzuordnen.

3.1.3. Managementebene der Gebäudeautomation

In der Gebäudeautomation stellt die Managementebene sowohl im streng hierarchischen als auch im dezentralen Modell die übergeordnete bzw. maßgebende Komponente dar. Hier erfolgen das Bedienen, das Überwachen und die Optimierung der gesamten Anlage.

Für diese Aufgaben wird die sog. Gebäudeleittechnik (GLT) eingesetzt, welche einerseits als die gesamte der Managementebene unterstellte technische Ausrüstung und andererseits ausschließlich als die genutzte Software zur Umsetzung obiger Aufgaben definiert werden kann. Im Regelfall handelt es sich bei der GLT um die Softwarekomponente, welche auf einem Standardserver läuft und vom Anbieter der Anlage zur Gebäudeautomation bereitgestellt wird.

Die Anbindung der untergeordneten Ebenen bzw. die Kommunikation mit Raum- und Automationssystem erfolgt über gängige Medien und einige standardisierte Protokolle. Die Funktionen der Managementebene können dabei sowohl lokal mittels bereitgestelltem Server inkl. der erforderlichen Software als auch zentral aus einer entfernt gelegenen Managementzentrale realisiert werden. Im Normalfall erfolgt die Kontrolle der zu regelnde Anlage eines Gebäudes durch eine lokal eingesetzte GLT. In diesem Fall ist das Personal des technischen Facilitymanagements für die Nutzung und Bedienung der GLT zuständig. Oftmals werden für das Management mehrerer Liegenschaften (vgl. Schalenmodell in Kapitel 3.2: *Raumautomation als Bestandteil der Gebäudeautomation*) zentrale Einrichtungen genutzt, die über gesicherte Remoteverbindungen auf die jeweils lokale GLT zugreifen können. Hierbei fallen sowohl bei der lokalen als auch bei der zentralen GLT folgende Aufgaben und Funktionen (diese werden als *Dataprocessing* bezeichnet) in den Bereich der Managementebene:

- Aufzeichnung und Archivierung von Mess- und Betriebsdaten
- Visualisierung und statistische Auswertung von Mess- und Betriebsdaten
- Ausgabe aktueller Betriebs- und Störmeldungen
- Kommunikation mit dem (Raum-) Automationssystem bzw. der Automationsebene
- Optimierung des Energieverbrauchs (auch systemübergreifend)
- Unterstützung für den Bediener bei notwendigen Eingriffen und Entscheidungen

3.2. Raumautomation als Bestandteil der Gebäudeautomation

Wie in den vorherigen Abschnitten bereits erwähnt, wird die Gebäudeautomation nicht mehr nur durch das immer noch aktuelle und genutzte Ebenenmodell beschrieben und definiert. Vielmehr wurde der Notwendigkeit nach einer umfassenderen Sichtweise entsprochen, indem das klassische Ebenenmodell durch die Anpassung der Kostengruppen gem. DIN 276 (vgl. Kapitel 3.1.2: *Automationsebene der Gebäudeautomation* und Abbildung 21) um eine definitorische Sichtweise erweitert wurde. Dies erfolgte, um der zunehmenden Bedeutung der Raumautomation als Bestandteil der gesamten Gebäudeautomation nachzukommen. Der Aufgabenbereich der Raumautomation erstreckt sich hierbei über sämtliche gewerkeübergreifende[29] Automationsaufgaben und -funktionen wie z. Bsp. Heizen, Kühlen oder Beleuchten. Im Gegensatz zur einleitend und weiter oben erwähnten Einzelraumregelung, die ausschließlich den Bereich der Raumlufttechnik[30] abdeckt, fallen in das Gebiet der Raumautomation u. a. Aufgaben wie der Blend- und Sonnenschutz als auch die Tageslichtnutzung. Die Raumautomation stellt somit also ein Gesamtkonzept zur optimierten Raumnutzung in weitestgehend allen Belangen dar, dem die Einzelraumregelung als Teildisziplin untergeordnet wird.

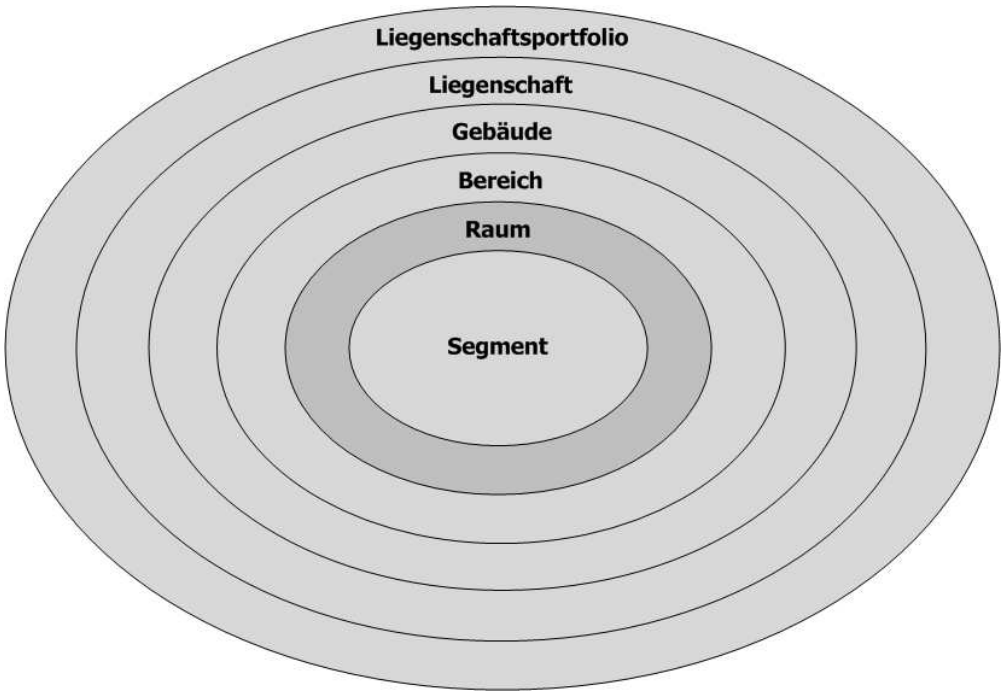

Abbildung 24: Schalenmodell zur Systembetrachtung in der Gebäudeautomation (Quelle: VDI 3813 Blatt 1, S. 11; Mai 2007)

Aufbauend auf dieser Definition und der in Abbildung 24 dargestellten Systembetrachtung erfolgt eine Einteilung in funktionale Untereinheiten (Subsysteme), die einerseits räumlich getrennt sein (wie z. Bsp. Einzelraum oder Flur) und andererseits sog. strukturbildende Subsysteme (wie z. Bsp. Zonen in Großraumbüros) darstellen können.

[29] Gewerk: in sich geschlossener Bauleistungsabschnitt
[30] kurz: RLT

Die kleinste funktionale Einheit in diesem Modell stellt dabei das Segment dar, welches bei Neubauten i. d. R. durch die Anordnung der Fenster (das Fensterraster) definiert und als Rastersegment bezeichnet wird. Große Flächen in Bürogebäuden, die flexibel als Großraumbüro oder als in variabel unterteilbare Einzelbüros genutzt werden sollen, werden hierbei in kleinstmögliche funktionale Einheiten, in gleichmäßige Rastersegmente, unterteilt.

Aus einem oder mehreren dieser Segmente setzt sich schließlich ein Raum zusammen. Dieser kann durch bauliche Umschließungsflächen wie Wände oder Decken oder organisatorisch als Zone in einem Großraumbüro definiert werden. Auf dieser Ebene nehmen die Funktionen der Raumautomation Einfluss, d. h. die Stellgrößen z. Bsp. einer Raumtemperaturregelung wirken auf sämtliche funktionsgleiche Aktoren identischer Segmente eines Raumes.

Ein Bereich setzt sich im Rahmen dieses Modells aus mehreren Räumen zusammen, wobei diese hierbei wie in einem Flur horizontal, über mehrere Etagen hinweg vertikal oder gemischt über mehrere Etagen und Flure hinweg wie in einem Atrium angeordnet sein können.

Ein Gebäude schließlich erstreckt sich über einen oder mehrer Bereiche und bildet in Kombination mit einem oder ebenfalls mehreren benachbarten Gebäuden eine Liegenschaft. Eine solche Liegenschaft kann letztlich als Teil einer Liegenschaftssumme Bestandteil eines Liegenschaftsportfolios sein.

Wie weiter oben bereits erwähnt, bezieht sich die Einzelraumregelung ausschließlich auf die RLT, so dass für die Funktionsdefinition der Raumautomation einen anderen Ansatz gem. [VDIB1] mit sich brachte. Bei der obigen Beschreibung der die Gebäudeautomation definierenden Ebenen wurde jeder Ebene ein eindeutiger Funktionsrahmen zugeordnet. Im Gegensatz zu diesen Beschreibungen erfolgt im Rahmen der Definition der Raumautomation eine davon losgelöste, den Anforderungen für die Anwendungen der Raumautomation entsprechende Gruppierung von Funktionen (vgl. [VDIB1], S. 14 ff.). Hierbei stellt eine Funktion der Raumautomation eine elementare Einheit dar, welche „eine spezifische Aufgabe oder eine typische Wirkung eines Systems, in der Regel innerhalb eines größeren Zusammenhangs“ ([VDIB1], S. 14) beschreibt. Als Ein- und Ausgabeschnittstelle sowie zur Kommunikation untereinander werden identische Raumfunktionen zusammengefasst und den folgenden festgelegten vier Funktionsgruppen zugeordnet:

- Anwendungsfunktionen

  Diese Funktionsgruppe umfasst sämtliche Funktionen zur Erfüllung der eigentlichen Anwendungs- und Applikationsaufgaben eines Segments. In diesen Bereich fallen z. Bsp. das Beleuchten, Verschatten oder Heizen des Segments *Raum* oder übergeordnete Funktionen der Gebäudeautomation wie die umfassende Bereitstellung von Heizwärme.

- Anzeige-/Bedienfunktionen

  Funktionen dieser Gruppe erfüllen die einem Segment zugeordneten Ansprüche zur Bedienung und Zustandsanzeige dieses Segments. Im Segment *Raum* fielen hierbei z. Bsp. das Anzeigen der aktuellen Raumtemperatur oder des Status’ der Jalousien sowie die Vorgabe bzw. Bedienung ebendieser.

- Service-/Diagnosefunktionen

  In dieser Gruppe werden hard- und softwareinterne Funktionen zusammengefasst, mittels derer die eigentliche Funktion der Raumautomation umgesetzt wird. Dies erfolgt u. a. mittels Schnittstellen zu den Anzeige- und Bedienfunktionen für die Übermittlung von Wartungs-, Betriebs- oder Störmeldungen.

- Managementfunktionen

  Bei dieser Funktionsgruppe handelt es sich um den Zusammenschluss sämtlicher Möglichkeiten der übergeordneten Datenverarbeitung zum effizienten Betrieb eines Automationssystems. Hierzu zählt vor allem die Einsparung von Energiekosten, deren Umsetzung jedoch mittels, meist automatischer, Ausführung von Funktionen der Gruppe der *Anwendungsfunktionen* vollzogen wird.

Aufbauend auf diesen Ausführungen wird es sich bei der Entwicklung der Steuerung im Rahmen dieser Arbeit also um eine klassische Steuerung der Gebäudeautomation handeln. Diese wird als DDC-Gerät in der Automationsebene als Schnittstelle zwischen der Feld- und der Managementebene eingebunden sein, auch wenn diese klassische Abgrenzung durch die mögliche Integration reiner Feldkomponenten auch hier verschwimmt. Im Kontext der Raumautomation wird diese Steuerung dem Bereich der Anwendungsfunktionen zuzuordnen sein, da sie die übergeordnete Aufgabe der globalen Wärmebereitstellung im Segment *Gebäude* realisieren wird.

3.3. Bussysteme und -protokolle in der Gebäudeautomation

In der Gebäudeautomation haben sich einige Bus- und Netzwerkstrukturen ebenso wie spezielle Protokolle im Verlauf der Zeit etablieren können. Wie bereits erwähnt wird bei der Netzwerkkommunikation in der Gebäudeautomation grundlegend zwischen horizontaler und vertikaler Kommunikation unterschieden. Hierbei erfolgt die horizontale Kommunikation zwischen den verschiedenen Komponenten innerhalb einer Ebene wie z. Bsp. der Feldebene oder der Automationsebene. Vertikale Kommunikation hingegen erfolgt Ebenenübergreifend und stellt erst die Funktionen der untersten Einheiten den übergeordneten Managementeinrichtungen der höheren Ebenen zur Verfügung.

Ergänzend zu den Ausführungen im Abschnitt *Theoretische Grundlagen von Rechnernetzen* werden hier nun allgemeine Zugriffsverfahren bei Bussystemen und, darauf aufbauend, typische Bussysteme der Gebäudeautomation erläutert.

3.3.1. Zugriffsverfahren bei Bussystemen

Damit die Kommunikation der einzelnen Komponenten untereinander geregelt über den zentrale Bus erfolgen kann, existieren Verfahren, um den Zugriff dieser Komponenten auf das gemeinsame Medium zu regeln. Folgende gebräuchliche Verfahren dienen der Sicherstellung des verlässlichen Datenaustauschs bei Bussystemen:

a) CSMA

Das Carrier Sense Multiple Access[31]-Verfahren stellt eine asynchrone[32] Methode dar, um die Berechtigung zur Nutzung einer gemeinsamen Busleitung zu erlangen. Im Rahmen des Carrier Sense, der Trägerprüfung, überwachen sämtliche an den Bus gekoppelten Teilnehmer die Busleitung. Beabsichtigt ein Teilnehmer eine Übertragung zu initiieren, so erfolgt dies nur, wenn die Busleitung über eine definierte Zeitspanne hinweg ungenutzt bleibt.

b) CSMA/CD

Wie die beiden folgenden Verfahren auch, stellt das Carrier Sense Multiple Access/Collision Detect-Verfahren eine Erweiterung zu CSMA dar. Hierbei wird ebenfalls die Busleitung auf bereits stattfindenden Datenverkehr überprüft und erst nach ausreichender Freizeit der Leitung mit dem Sendevorgang begonnen. Zusätzlich wird während der Übertragung überprüft, ob eine Kollision von Datenpaketen auf der Busleitung erfolgt, die zufällig zur gleichen Zeit oder, bedingt durch die Laufzeitverzögerungen von Busleitungen, zeitversetzt von einem weiteren Busteilnehmer gesendet wurden. In diesem Fall übermittelt jeder der Sender ein sog. JAM-Signal, bestehend aus einer 32 Bit langen Folge von 10101010…[33]. Daraufhin bricht jeder sendende Busteilnehmer den Übertragungsvorgang des aktuellen Datenpakets ab und wartet eine zufällige Zeitspanne, bis ein erneuter Versuch der Paketübertragung initiiert wird.

c) CSMA/CA

Ebenfalls basierend auf CSMA stellt das Carrier Sense Multiple Access/Collision Avoidance-Verfahren eine Erweiterung bzw. Ergänzung zu CSMA/CD dar. Sollte ein Busteilnehmer mit einem Netzwerkadapter ausgestattet sein, der ausschließlich eine Half-Duplex-Übertragung unterstützt, gäbe es für diesen keine Möglichkeit, die Busleitung während der Übertragung zu überwachen. Eine Collision Detection ist in diesem Fall also nicht umsetzbar. Vielmehr wird überprüft, ob das Übertragungsmedium eine bestimmte Zeitspanne[34] lang frei ist und nach Ablauf einer zufälligen Wartezeit[35] mit dem Sendevorgang begonnen. Ist der Sendevorgang nicht möglich, weil das Medium als belegt erkannt wird, stoppt der Busteilnehmer den Ablauf des Backoff, bis der NAV[36] den Wert null annimmt und ein erneuter DIFS[34] abgelaufen ist. Der Empfänger schließlich übermittelt dem Sender den erfolgreichen Empfang des Datenpaketes mittels ACK[37], nachdem ein SIFS[38] abgelaufen ist. Sollte der Backoff zweier Sender gleichzeitig ablaufen, wird kein ACK-Signal gesendet

[31] Trägerprüfung bei Mehrfachzugriff
[32] Übertragung erfolgt in freier Zeiteinteilung, ist also nicht an ein Taktsignal gekoppelt
[33] vgl. [PLATE], S. 7
[34] DIFS: Distributed Coordination Function Interframe Spacing ist die Wartezeit vor dem Senden eines Datenframes
[35] Backoffzeit
[36] Network Allocation Vector: Jeder Busteilnehmer übermittelt vor dem Senden die geschätzte Nutzungsdauer für den Bus an alle anderen Teilnehmer. Jeder Teilnehmer verwaltet diese Zeiten lokal in einem eigenen NAV, der konstant bis null abläuft.
[37] Acknowledgement: Bestätigung
[38] Short Interframe Spacing: Zeit, die vor dem Senden eines ACK-Signals vergehen muss.

werden und es kommt zu einem Timeout, nach welchem ein EIFS[39] Wartezeit vergeht, bevor jeder Sender mit der Übertragung seines Datenpaketes erneut von vorn (DIFS) beginnt.

d) CSMA/CR

Das Verfahren Carrier Sense Multiple Access/Collision Resolution stellt eine Kombination aus CSMA/CD und CSMA/CA dar. Hierbei werden Kollisionen von Datenpaketen durch Bitarbitrierung unterbunden, indem diesen Paketen i.d.R. eine Arbitrierungsphase vorangeht, die die zu versendenden Nachrichten untereinander priorisiert. Wie nebenstehend in Abbildung 25 am Beispiel des CAN-Bus zu erkennen, verfügt am Ende der Arbitrierungsphase nur noch ein Busteilnehmer über die Berechtigung, den Datenbus für seinen Sendevorgang zu nutzen. Der Vorteil bei diesem Verfahren liegt in der verzögerungsfreien Busnutzung und damit einhergehend der unmittelbaren Datenübertragung durch den Busteilnehmer mit der höchsten Nachrichtenpriorität.

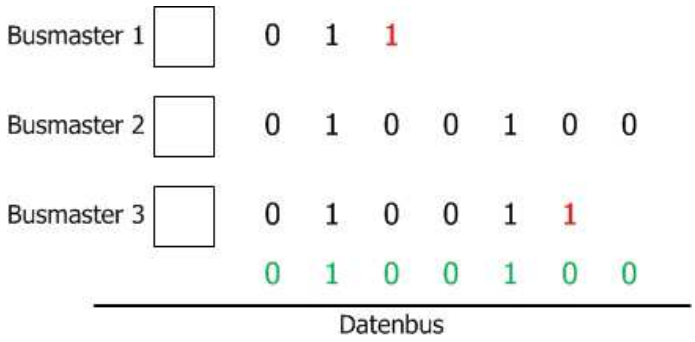

Abbildung 25: Bitarbitrierung beim CAN-Bus. 0 stellt hierbei den dominanten und 1 den rezessiven Wert dar. Zuerst stellt „Busmaster 1“ den Sendevorgang ein, da seine rezessive Null von den dominanten Nullen unterdrückt wird. Danach folgt „Busmaster 3“, so dass „Busmaster 2“ die höchste Priorität aufweist und den Datenbus zur Übermittlung nutzen kann.

3.3.2. Bussysteme und -Protokolle der Gebäudeautomation

Innerhalb der Feldebene wird zur Verbindung und Kommunikation der Einheiten untereinander der sog. Feldbus eingesetzt. In diesem Bereich haben sich besonders vier Bussysteme etabliert:

a) der CAN-Bus,

b) LON,

c) EtherCAT und

d) KNX/EIB.

Die Geräte der bekanntlich als Vermittlungsebene fungierenden Automationsschicht bieten auch im Hinblick auf die bereits beschriebenen Anforderungen immer eine Schnittstelle, um die Feldebene anzubinden. Die vertikale Kommunikation zwischen Feld- und Automationsebene wird somit also realisiert.

Zur Sicherstellung der Kommunikation zwischen der Automations- und der Managementebene wird in fast allen Fällen auf einen Standard zurückgegriffen: Ethernet. Dieser Standard ermöglich einerseits die horizontale Kommunikation der Geräte der Automationsebene untereinander und andererseits die Anbindung an so

[39] Extended Interframe Spacing: Wartezeit nach dem Erkennen einer Kollision durch das Fehlen des ACK-Signals

ziemlich alle Managementsysteme, sofern genormte Protokollstacks zum Einsatz kommen.

Wie im ebenen beschriebenen physikalischen Bereich haben sich auf der Softwareseite der Gebäudeautomation bestimmte Protokolle bzw. Protokollstacks durchgesetzt. Dies sind einerseits das zum CAN-Bus gehörige CAN- bzw. CANopen-Protokoll und andererseits das zu LON gehörige LonTalk-Protokoll. Zur Kommunikation über standardisierte Ethernet-Verbindungen haben sich besonders TCP/IP und BACnet bzw. BACnet/IP etabliert.

Da der Schwerpunkt dieser Arbeit nicht in der Verbindungsrealisierung einzelner Teilnehmer untereinander liegt, erfolgt an dieser Stelle nur ein kurzer Einblick in den CAN-Bus und BACnet. Dieses Bussystem und das Netzwerkprotokoll stellen in der Kommunikation der Gebäudeautomation weit verbreitete Standards dar. Während der CAN-Bus samt dem zugehörigen Protokoll CANopen im Rahmen dieser Arbeit zum Einsatz kommen, sind die Ausführungen zu BACnet als Verständnisgrundlage für die abschließende Bewertung im Rahmen dieser Arbeit zu betrachten.

3.3.2.1 CAN

Das Akronym CAN steht für „Controller Area Network“ und bezeichnet ein Bussystem, welches ursprünglich für die Automobilindustrie entwickelt wurde. Es handelt sich hierbei um ein asynchron[40] arbeitendes, serielles Bussystem, welches durch die ISO genormt ist und die beiden ersten Schichten, die Bitübertragungs- und die Datensicherungsschicht, des ISO/OSI-Referenzmodells definiert.

Im Rahmen der Datenübertragung arbeitet dieser Bus nach dem oben beschriebenen CSMA/CR-Verfahren auf NRZI[41]-kodierten Daten. Im Regelfall werden am CAN-Bus angeschlossene Teilnehmer gleichberechtigt, also nach dem Multi-Master-Prinzip betrieben. Hierbei erfolgt die Busfreigabe zur Kommunikation mittels des oben beschriebenen Arbitrierungsverfahren,

Die physikalische Verbindung der Busteilnehmer untereinander erfolgt i. d. R. über eine verdrillte Zweidrahtleitung, dessen Enden zur Vermeidung von Signal-Reflexionen typischerweise mit einem Abschlusswiderstand von 120Ω zu versehen sind. Abhängig von der auszuführenden Leitungslänge können über den CAN-Bus verschiedene Übertragungsgeschwindigkeiten gem. Tabelle 1 realisiert werden. Hierbei ist unbedingt zu beachten, dass jeder Teilnehmer am Bus auf die gleiche Übertragungsrate konfiguriert wurde. Die Festlegung dieser maximalen Werte für die Datenübertragung beruht auf der Zeit, die ein zu übermittelndes Signal benötigt, um vollständig von einem Ende der Leitung bis zum anderen zu gelangen. Dazu muss ein Signal auf Senderseite am Bus so lange anliegen, bis der entspr. Wert bis zu allen Bus-Teilnehmern über die gesamte Lei-

max. Leitungslänge	Übertragungsrate
25 m	1 Mbit/s
100 m	500 kBit/s
250 m	250 kBit/s
500 m	125 kBit/s
625 m	100 kBit/s
1000 m	50 kBit/s
2500 m	20 kBit/s
5000 m	10 kBit/s

Tabelle 1: CAN-Übertragungsraten in Abhängigkeit der Leitungslänge.

[40] jeder Teilnehmer darf zu jeder Zeit versuchen, Daten über den Bus zu übertragen
[41] Non Return to Zero Invert: einem der zwei Bit-Werte wird der bereits anliegende Leitungspegel zugeordnet, dem anderen ein Pegelwechsel.

tung propagiert wurde. Die Zeitspanne für das Anliegen eines Signals am Bus dar also unter keinen Umständen kürzer sein, als die Ausbreitungsdauer für dieses Signal über den gesamten Bus.

Die Kommunikation über den CAN-Bus ermöglicht i. d. R. ein an CANopen angelehntes Protokoll, welches die Anwendungsschicht (Schicht 7) des ISO/OSI-Referenzmodells abdeckt.

Im Gegensatz zur bisherigen Definition findet mit der im Rahmen dieser Arbeit genutzten Hardware ein anderer Ansatz Anwendung. Hier erfolgt der Betrieb des Master Terminals als alleiniger Bus-Master, während aller übrigen Module im Slave-Modus betrieben werden. Diese Tatsache zieht die folgende Priorisierung der Bus-Teilnehmer während des Systemstarts nach sich[42]:

- Start des CAN-Masters mit anschließendem Start der CAN-Slaves (diese befinden sich danach in einem voroperativen Zustand)
- Der CAN-Master sendet Initialisierungsnachrichten an die CAN-Slaves (diese befinden sich danach im STOP-Zustand)
- Nach vollständiger Initialisierung sendet der CAN-Master jedem CAN-Slave einen Startbefehl (erst jetzt befinden sich die CAN-Slaves im Betriebszustand).

Sofern die Initialisierung nur eines Slaves nicht erfolgreich ist, erfolgt auch kein Start der übrigen Slaves und das gesamte System kann nicht starten.

3.3.2.2 BACnet

Die Abkürzung BACnet steht für „Building Automation and Control network“ und repräsentiert ein aktuelles Netzwerkprotokoll für den Bereich der Gebäudeautomation. Dieses Protokoll erlaubt die Kommunikation zwischen Geräten und ermöglicht die Interoperabilität[43] zwischen Geräten verschiedener Hersteller. Die einleitend erwähnte Systemintegration findet in der Nutzung dieses Protokolls also ihre Umsetzung.

Hierbei handelt es sich um ein objektorientiertes Datenprotokoll, welches Anwendung auf allen Funktionsebenen der Gebäudeautomation findet. Dazu erfolgt in diesem Protokoll die Festlegung für Objektytpen (Object Types), Dienste (Services) und Netzwerke. Bei den Objekttypen handelt es sich um eine den Bereich der Gebäudeautomation weit umspannende Liste, die im Detail [AMEV] zu entnehmen ist, im Überblick u. a. folgende Komponenten beinhaltet:

- diverse Ein- und Ausgänge (z. Bsp. *Analog Input*, *Digital Output*,…)
- Gerät (*Device*)
- Zeitplan (*Schedule*)
- Zählwerteingabe (*Accumulator*)
- Regler (*Loop*)

[42] vgl. [SABO1], S. 37
[43] Hierbei handelt es sich um die Möglichkeit der Zusammenarbeit heterogener Systeme zur zielgerichteten, gemeinsamen Interaktion.

Insgesamt existierten bis zum Jahr 2007 38 Objekttypen, mit denen sich sämtliche physikalischen und kommunikativen Ein- und Ausgabefunktionen der Gebäudeautomation darstellen lassen. Jede Instanz eines solchen Objekttyps enthält dem Typ entsprechende Objekteigenschaften, die *Properties*, welche die Funktions- und Nutzungsmöglichkeiten definieren. Im Falle eines Temperaturfühlers würde dieser z. Bsp. als Instanz des Typs *Analog Input* definiert u. a. Eigenschaften wie die physikalische Einheit, den Namen und den Zahlenwert enthalten.

BACnet-Dienste (Services) sind Verfahrensbeschreibungen für die den Teilnehmern im Netzwerk zur Verfügung stehenden Möglichkeiten zur Kommunikation. Hierzu zählen u. a. Richtlinien für den Zugriff auf die *Properties* anderer Objekte mittels Lese- oder Schreibvorgang. So werden mittels BACnet z. Bsp. mehrer Methoden zur Erzeugung von Meldungen beschrieben, die neben der änderungsbezogenen Meldungserzeugung (COV[44]-Reporting) auch objektinterne und regelbasierte Meldungserzeugung umfassen. Bis zum Jahr 2007 wurden durch das BACnet-Protokoll insgesamt 42 Dienste definiert, die den folgenden fünf Kategorien zugeordnet wurden:

- Dienste für Objektzugriff (Object Access Services)
- Dienste für Geräte- und Netzwerkzugriff (Device and Network Management Services)
- Dienste zur Alarm- und Ereignisverarbeitung (Alarm and Event Services)
- Dienste für Dateizugriff (File Access Services)
- Dienste zur Gerätekonfiguration und -diagnose (Virtual Terminal Services)

Die Netzwerkkommunikation und damit einhergehend der Datenaustausch durch die BACnet-Dienste erfolgt nach dem Client-Server-Prinzip, wobei ein Client einen bestimmten Dienst bei einem Server anfordert. Die Antwort des Servers auf eine solche Anforderung erfolgt entweder durch die Bereitstellung des erforderlichen Dienstes oder die Ausführung eine notwendigen Aktion. Für die Kommunikation unterschiedlicher Geräte (auch unterschiedlicher Hersteller) untereinander sind fünf Interoperabilitätsbereiche definiert, die die betriebswichtigen Funktionsbereiche eines BACnet-Systems beschreiben. Dieses sind:

- Gemeinsame Datennutzung (Data sharing)
- Alarm- und Ereignisverarbeitung (Alarm and event management)
- Zeitplan (Schedule)
- Trendaufzeichnung (Trending)
- Geräte- und Netzwerkmanagement (Device and network management)

Jedem dieser Interoperabilitätsbereiche sind dabei die für die Erfüllung der dem Funktionsbereich entsprechenden Aufgaben definierten BACnet-Dienste, sog. BIBBs, zugeordnet. Diese BIBBs (BACnet Interoperability Building Blocks) definieren hierbei die Voraussetzungen zur Erfüllung interoperabler Kommunikation, die von BACnet-Geräten zu erfüllen sind. Entscheidend ist, dass korrespondierende BIBBs von Clients und Servern für die Realisierung der Interoperabilität erforderlich sind.

[44] Change Of Value

4. Entwicklung

Ziel dieses Kapitels ist die Darlegung der praktischen Umsetzung der in den vorangegangenen Kapiteln erläuterten Aspekte. Hierzu wird der Weg von den vorbereitenden Maßnahmen und den grundlegenden Überlegungen bis hin zum prototypischen Entwurf und der letztlichen Ergebniseinstufung aufgezeigt.

4.1. Grundlegende Planung

Zu Beginn der Entwicklung musste einerseits festgelegt werden, was letztlich in prototypischer Form entworfen werden und mit welchen Mitteln dieser Entwurf erfolgen sollte. Eine Bedingung stellte selbstverständlich die Entwicklung einer Steuerung bzw. Regelung wie sie in der Gebäudeautomation vorkommt dar, auch wenn zu diesem Zeitpunkt noch nicht feststand, aus welchem Bereich konkret. Eine Prämisse war allerdings, dass die Entwicklung losgelöst von bisherigen Vorgehensweisen, sowohl hardware- als auch softwareseitig, zu erfolgen hatte. Aus diesem Grund wurden mit der Firma *SABO Elektronik GmbH* ein unabhängiger Hersteller von Steuer- und Regelsystemen einerseits und mit der Firma *3S-Smart Software Solutions GmbH* ein neutraler Anbieter für ein Programmiersystem andererseits vorgegeben.

4.1.1. CoDeSys-Programmiersystem

Da im Rahmen dieses Projektes vorwiegend die Abschätzung der Realisierung regelungstechnischer Anforderungen mit neuen bzw. anderen Methoden im Vordergrund stand, wurde der Einsatz des CoDeSys[45]-Programmiersystems der Firma *3S-Smart Software Solutions GmbH* in der Version 2.3.9.32 unter *Microsoft Windows 7* festgelegt. Hierbei handelt es sich um ein Programmiersystem für eine Soft-SPS (Speicherprogrammierbare Steuerung in Software), welches sich besonders im Automatisierungsbereich des Maschinen- und Anlagenbaus als Standardwerkzeug etabliert hat.

4.1.1.1. Programmiersprachen in CoDeSys

Entwickelt wurde dieses Programmiersystem unter strikter Einhaltung der Vorgaben der internationalen Norm IEC 61131. Einen zentralen Punkt stellt hierin der dritte Teil dar, in dem die fünf grundlegenden Entwurfs- und Programmiersprachen definiert werden:

a) Anweisungsliste (AWL) ➔ Instruction List (IL)

   Bei der Anweisungsliste handelt es sich um eine mit Assembler vergleichbare, textuelle Form der Programmierung. Hierzu wird eine Folge von Anweisungen erzeugt, bei der jede Anweisung in einer neuen Zeile beginnt und einem vorgeschriebenen Format unterworfen ist. Einem Operator zu Beginn der Anweisungszeile folgt genau ein Operand. Hierbei können Anweisungen bzw. zusammengehörige Anweisungsblöcke durch ein als Einsprungadresse fungierendes vorangestelltes Label, abgeschlossen durch einen Doppelpunkt, von Sprungoperatoren aus dem Programmlauf erreicht werden. Ein einfaches Beispiel für ein Programm in AWL:

[45] **Co**ntroller **De**velopment **Sys**tem

```
MULT:   (* Einsprungmarke für Multiplikation *)
LD 2    (* 2 als Wert in das Arbeitsregister[46] laden *)
MUL 3   (* Wert in Arbeitsreg. mit 3 multiplizieren *)
ST erg  (* Ergebnis in der Variablen 'erg' speichern *)
```

Die Tatsache, dass je Operator nur ein Operand zulässig ist, schränkt die Möglichkeiten zur Strukturierung der mit AWL entwickelten Programme sehr stark ein. Umfangreichere Programme können somit nur mittels bereits erwähnter Sprunganweisungen strukturiert werden und sind bei wachsender Größe zunehmend unübersichtlicht.

b) Kontaktplan (KOP) ➔ Ladder Diagram (LD)

Der Kontaktplan stellt eine grafische Möglichkeit der Programmierung einer SPS dar und ist im Aufbau an das Design von Stromlaufplänen angelehnt. Im Funktionsumfang eigentlich auf die Realisierung von Verknüpfungssteuerungen eingeschränkt, wurde bereits in der VDI 2880 die Erweiterung des Kontaktplans um andere Sprachelemente vorgesehen. Somit bietet der KOP diverse

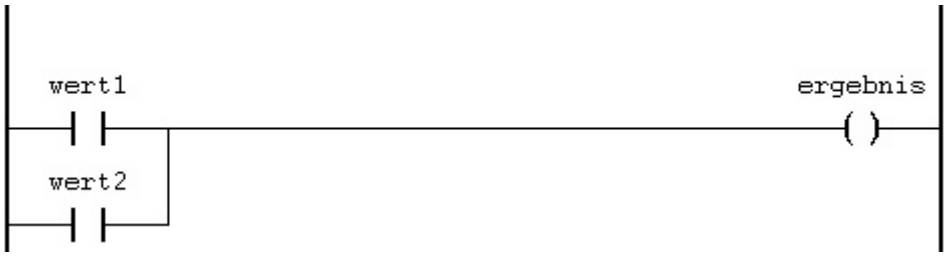

Abbildung 26: Ein simpler Kontaktplan (KOP) mit linker und rechter Stromschiene, den beiden Kontakten *wert1* und *wert2* sowie der Spule *ergebnis*.

Möglichkeiten, komplexe Konstrukte zu entwerfen, indem ebenfalls beliebig komplexe Funktionsblöcke integriert werden. Diese Funktionsblöcke können in einer beliebigen, durch die Norm IEC 61131-3 definierten, Sprache entwickelt worden sein, müssen allerdings unbedingt je einen binären Ein- und Ausgang aufweisen. Die grundlegenden Elemente, aus denen ein vollständiger Kontaktplan zusammengesetzt wird sind:

- linke und rechte Stromschiene
- horizontale und vertikale Linien
- Kontakte
- Spulen
- Funktionen und Funktionsbausteine

In Abbildung 26 ist ein einfacher Kontaktplan zu sehen, dessen linke und rechte Stromschiene durch die senkrechten Linien links und rechts außen repräsentiert werden. Dabei entspricht die linke Stromschiene der Phase und die rechte Stromschiene dem Nullleiter eines Strompfades, so dass nur die Objekte eines KOP bearbeitet werden, die mit der linken Stromschiene verbunden sind. Die erwähnte Abbildung kann zudem als einzelnes Netzwerk Bestandteil eines

[46] auch als Akkumulator bezeichnet

komplexen Kontaktplans sein. Diese Netzwerke werden im Programmiersystem senkrecht aufeinander folgend dargestellt, so dass der Inhalt der Variablen *ergebnis* in einem Folgenetzwerk erneut verwendet werden kann. Variablen stellen im KOP sog. Signale dar, die einen Zustand in Form eines Kontaktes anlegen oder in Form einer Spule annehmen. Werden wie in Abbildung 26 in einem Netzwerk einzelne Objekte parallel angeordnet, entspricht dies einer ODER-Verknüpfung, eine Reihenanordnung entspricht einer UND-Verknüpfung. In der o. g. Abbildung werden die Werte der booleschen Variablen *wert1* und *wert2* in Form der jeweiligen Kontakte angelegt und logisch miteinander ODER-verknüpft. An der Spule auf der rechten Seite wird der entsprechende Zustand schließlich als logischer Wert in der Variablen *ergebnis* gespeichert.

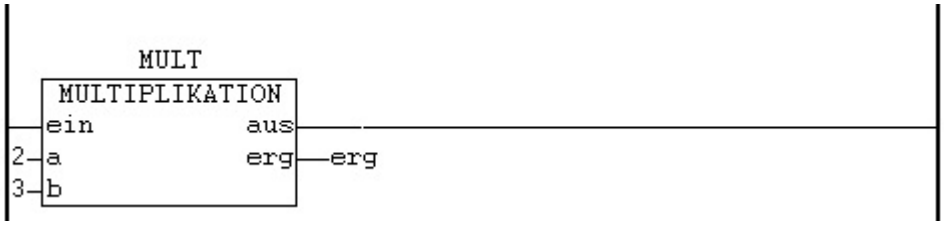

Abbildung 27: Nutzung der Instanz *MULT* des einfachen Funktionsbausteins *MULTIPLIKATION* in einem KOP-Netzwerk.

In Abbildung 27 ist das einfache Beispiel aus der Beschreibung zur AWL unter Zuhilfenahme der Instanz *MULT* eines komplexeren Funktionsbausteins *MULTIPLIKATION* dargestellt. Während sowohl die beiden Eingänge *a* und *b* als auch der Ausgang *erg* die Nutzung ganzzahliger Werte gestatten, verlangen der Eingang *ein* und der Ausgang *aus* die Nutzung boolescher Werte bzw. Variablen, damit eine Anbindung des Funktionsbausteins an das Netzwerk erfolgen kann.

Die Verarbeitungsreihenfolge von Kontakten, Spulen und Funktionsbausteinen wird durch den Datenfluss bestimmt, so dass zuerst Objekte mit bereits erfolgter Wertzuweisung und schließlich die übrigen Objekte mit den Initialwerten verarbeitet werden.

In einem KOP lassen sich Steuerungen somit relativ einfach direkt aus den Vorlagen von Stromlaufplänen umsetzen und trotzdem komplexe Konstrukte realisieren. Nicht ohne erhöhten Aufwand und den Umweg über andere Sprachen des Programmiersystems umsetzbar sind jedoch die Lösung schwieriger algorithmischer Probleme.

c) Funktionsbausteinsprache (FBS) ➔ Function Block Diagram (FBD)

Wie der Kontaktplan stellt auch die Funktionsbausteinsprache ein Sprache zur grafischen Entwicklung einer SPS dar. Oftmals wird diese Sprache auch als Funktionsplan (FUP) bezeichnet. Wie die elementaren Operationen in der AWL bietet diese Sprache elementare Funktionen, die beliebig mit eigenen Funktionen und Instanzen von Funktionsbausteinen gemischt werden können. Auch hier erfolgt die Anordnung in Netzwerken innerhalb derer die einzelnen Objekte systematisch aneinander gereiht werden. Die Programmierumgebung erzeugt die Verbindungen zwischen Ein- und Ausgängen beim Ein- oder Hinzufügen neuer

Objekte automatisch, so dass komplexe Steuerungen einfach mittels grafischer Elemente entworfen werden können. Hierbei unterstützt die Funktionsbausteinsprache u. a. die folgenden Sprachelemente:

- sämtliche in der IEC 61131-3 definierte Operatoren
- mehrfache Ein- und Ausgänge
- Negationen und Kommentare
- Sprünge

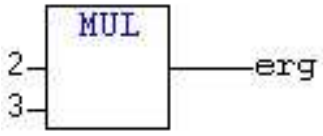

Abbildung 28: Ein einfaches Beispiel für die Nutzung einer Funktion in der FBS

In Abbildung 28 ist das bereits erwähnte Beispiel der Multiplikation nochmals mit einer Standardfunktion der Funktionsbausteinsprache dargestellt. Die beiden festen Werte werden an die Funktion übergeben, multipliziert und das Ergebnis schließlich in der Variablen *erg* gespeichert. Der Ausgang könnte wiederum auf den Eingang einer anderen Funktion oder der Instanz eines anderen Funktionsbausteins gelegt und darin weiterverarbeitet werden. Neben dieser Möglichkeit zur Strukturierung stellen auch die Sprünge eine Möglichkeit dar, auf den Programmablauf einzuwirken. Es lassen sich also auch mit dieser Sprache komplexe Konstrukte entwickeln, deren algorithmische Möglichkeiten aber ähnlich wie bei dem Entwurf als KOP auf die Einbindung anderer Funktionen und Funktionsbausteine beschränkt sind.

d) Ablaufsprache (AS) ➔ Sequential Function Chart (SFC)

Mit der Ablaufsprache werden Steuerungen mit streng vorgegebener Zustandsabfolge beschrieben und entwickelt. Um dies zu bewerkstelligen muss vor Beginn der Implementierung genau untersucht und festgelegt werden, welche Zustände eine Anlage annehmen kann und in welcher Reihenfolge dies zu erfolgen hat. Die Ablaufsprache basiert daher auf den Grundlagen der Petri-Netze und es ist zwingend notwendig zu überprüfen, ob die erforderliche Anlagenfunktion überhaupt mittels AS umsetzbar ist.

Die Ablaufsprache nimmt gegenüber den übrigen Sprachen eine übergeordnete Position ein und ist somit zur Strukturierung des Ablaufs des eigentlichen Hauptprogramms gedacht. Diese Ablaufsteuerung besteht aus einer Kette sequenzieller Schritte, die jeweils einen unterschiedlichen Zustand der Anlage repräsentieren. Durch diese sequenzielle Anreihung einzelner Schritte werden Schrittketten erzeugt, die eine klare Ablaufsequenz vorgeben. Ergänzend zu den sequentiellen Abläufen ist es auch möglich, innerhalb der Schrittketten zu verzweigen und Sprünge auszuführen. Die einzelnen aufeinander folgenden Schritte einer Schrittkette sind über Transitionen miteinander verbunden, die für das Weiterschalten von einem Schritt zum nächsten verantwortlich sind.

Im Gegensatz zu den übrigen Sprachen lässt sich mit der Ablaufsprache allein keine autonome Steuerung implementieren, da es sich hier wie bereits erwähnt um die Möglichkeit der strukturellen Anordnung einzelner Bestandteile handelt. Diese Bestandteile können bzw. müssen in einer beliebigen anderen Sprache aus der IEC 61131-3 entwickelt worden sein.

e) Strukturierter Text (ST) ➔ Structured Text (ST)

Der Strukturierte Text stellt eine mit Pascal vergleichbare Programmiersprache dar, bei der, wie bei den übrigen durch die IEC 61131-3 definierten Sprachen, nicht zwischen Groß- und Kleinschreibung unterschieden wird. Wie bei den meisten Hochsprachen besteht auch hier die Möglichkeit, mehr oder minder komplexe Ausdrücke zu formulieren, welche als elementarer Teil von (Teil-) Anweisungen ausgewertet werden. Im Gegensatz zur AWL kann ein Ausdruck, und somit auch eine Anweisung, mehrere Operatoren und Operanden beinhalten. Jeder Operator unterliegt hierbei einer klar definierten Rangordnung und bestimmt somit die Reihenfolge, in der er auf die jeweiligen Operanden angewendet und der Ausdruck letztlich ausgewertet wird. Bei gleicher Rangordnung werden die betreffenden Operatoren von links nach rechts auf ihre Operanden angewendet, wobei diese Reihenfolge durch entsprechende Klammerung angepasst werden kann. Jede durch ein Semikolon abzuschließende Anweisung kann sich über mehrer Zeilen im Quellcode erstrecken und kann bzw. muss einem der folgenden Anweisungstypen zuzuordnen sein:

- Zuweisung (stellt die elementare Anweisung dar)
- Verzweigung (IF, CASE)
- Schleife (FOR, WHILE,…)
- Funktionsaufruf
- Funktionsbausteinaufruf mit Ergebniszuweisung

Die Ablaufsteuerung erfolgt beim Strukturierten Text also mit den für die imperative Programmierung typischen Kontrollstrukturen:

- Sequenz als lineare Abfolge von Anweisungen,
- Selektion als (un-) vollständige Alternative oder Fallauswahl,
- Iteration als verschiedenartige Schleifenkonstrukte.

Das eingangs bei den Erläuterungen zur AWL erwähnte Beispiel würde in ST folgendermaßen implementiert werden:

```
erg:= 2 * 3;
```

Gerade bei dem Entwurf umfangreicherer Programme zur Lösung komplexer algorithmischer Probleme mit einer SPS bietet der Strukturierte Text die Möglichkeit, verständliche und überschaubare Programmierlösungen zu erstellen. Im Gegensatz zur assemblernahen Programmierung mit AWL, bei der eine Strukturierung ohne explizite Sprunganweisungen nicht möglich ist, kommt ST dank der Kontrollstrukturen und der Möglichkeiten der Funktionsaufrufe ohne die Verwendung expliziter Label und Sprunganweisungen aus. Einzig der zumeist erhöhte Speicherbedarf kompilierter mit ST entwickelter Programme kann in Einzelfällen bei Überschreitung der Speichergrenzen gerade auf kleineren SPS zu Schwierigkeiten im Ablauf führen.

f) Freigrafischer Funktionsplaneditor (CFC) ➔ Continuous Function Chart (CFC)

Bei dem Freigrafischen Funktionsplaneditor handelt es sich ebenfalls um eine grafische Sprache zur Programmierung einer SPS, die allerdings nicht Bestandteil der IEC 61131-3 ist. Im Gegensatz zur Funktionsbausteinsprache werden die einzelnen Elemente während des Entwurfs vollkommen frei auf der Oberfläche eingefügt, können beliebig mit Ein- und Ausgängen anderer Objekte verknüpft werden und können beliebig komplex sein. Die Abarbeitungsreihenfolge der einzelnen Elemente wird hierbei durch eine Ordnungszahl an dem entspr. Objekt vorgegeben. Diese Reihenfolge ist durch manuelle Anpassung der Ordnungszahl jederzeit änderbar.

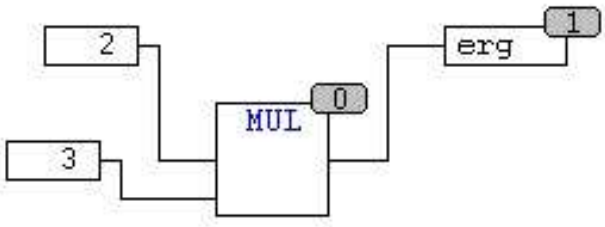

Abbildung 29: Eine einfache Multiplikation, realisiert in CFC, wobei die einzelnen Komponenten frei positioniert sind.

In Abbildung 29 ist die Umsetzung des bisher verwendeten Beispiels einer einfachen Multiplikation dargestellt. Gut zu erkennen ist dabei die freie Anordnung sowohl der beiden Werte für den Eingang der Multiplikationsfunktion als auch der übrigen Bestandteile. Die Zuweisung an die Variable *erg* würde auch dann korrekt erfolgen, wenn der entspr. Ausgang symbolisch vor dem Symbol für die Multiplikation angeordnet wäre. Grund hierfür ist, dass die Reihenfolge für die Abarbeitung nicht wie bei dem Kontaktplan oder der Funktionsbausteinsprache durch die horizontale und sequenzielle Anordnung der Elemente vorgegeben wird. Vielmehr wird diese Reihenfolge durch die graue, den Bestandteilen zugeordnete Ziffer aufsteigend vorgegeben.

Neben dem Strukturierten Text stellt diese Methode eine der umfangreicheren Möglichkeiten zum Programmentwurf dar, auch wenn mit dieser Sprache allein nicht alle algorithmischen Anforderungen zu realisieren sind.

4.1.1.2. Projektorganisation in CoDeSys

Umfassender Rahmen für die Programmierung einer Steuerung stellt im CoDeSys-Programmiersystem das Projekt dar. Dieses beinhaltet sämtliche Bestandteile, die zur Realisierung der erwünschten Funktion der Steuerung notwendig sind. Im Einzelnen sind dies:

- Bausteine
- Datentypen
- Visualisierung(en)
- Systemressourcen und Bibliotheken

Die Verwaltung und Bearbeitung des Projektes erfolgt innerhalb des CoDeSys-Programmiersystems über den in Abbildung 30 dargestellten, funktional strukturierten Hauptbildschirm. Neben der Menü- und der Funktionsleiste im oberen Fensterbereich

befinden sich auf der rechten Seite der Arbeitsbereich und darunter das Meldungsfenster. Während im Arbeitsbereich sämtliche, in der Anzahl unbegrenzte, Editorfenster angezeigt werden, erscheinen im Meldungsfenster alle Hinweise aus dem letzten Übersetzungs- und Überprüfungsvorgang. Die linke Seite beinhaltet den Object Organizer mit den entspr. Registerkarten für die o. g. Bestandteile zur Projektverwaltung. Je nachdem, welche Registerkarte angewählt ist, ändert sich der kontextbezogene Inhalt des Object Organizers.

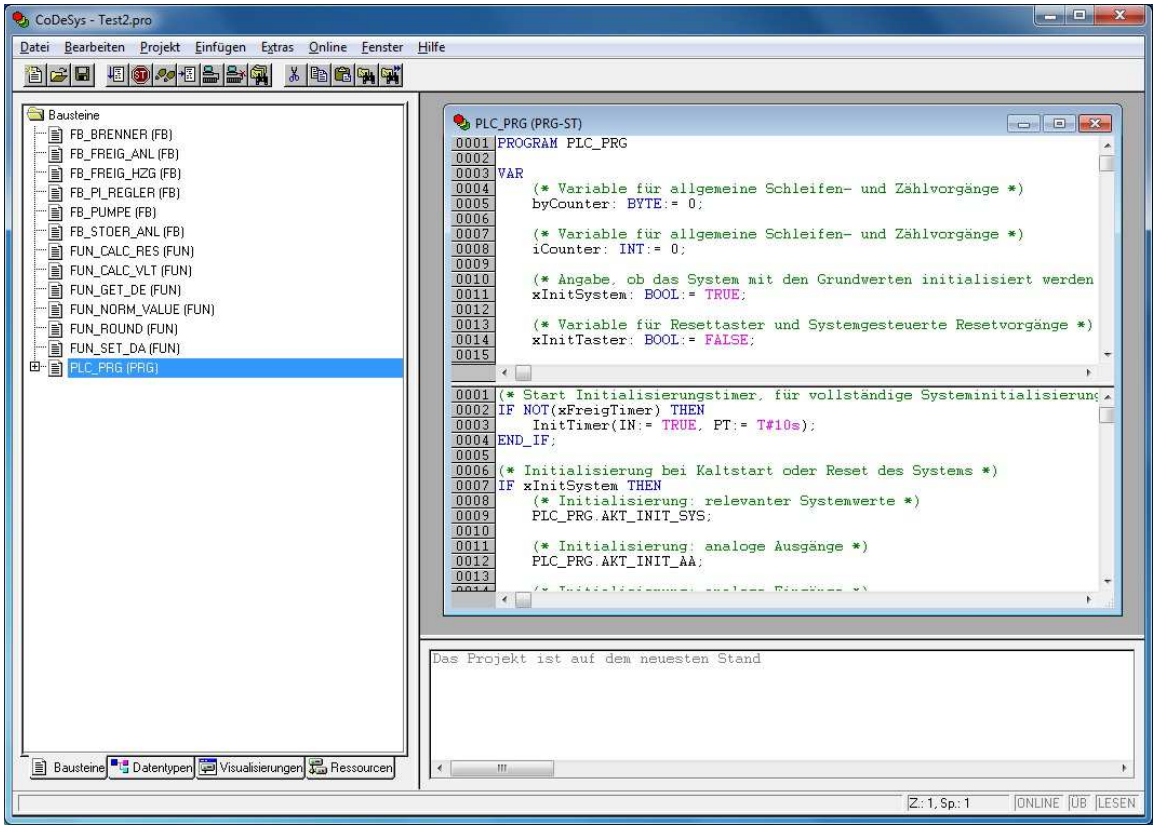

Abbildung 30: Der Hauptbildschirm des CoDeSys-Programmiersystems

Im Bereich der Bausteine erfolgt die Erstellung und Verwaltung der *Programm-Organisationseinheiten* (POEs), welche die softwareseitige Realisierung der algorithmischen Anforderungen einer Steuerung darstellen. Diese POEs können innerhalb eines Projektes gemischt in jeder der bereits erläuterten Sprachen erstellt werden, wobei hierbei folgende Einheiten existieren:

a) Funktion

Diese Komponente stellt die kleinste POE dar, welche über beliebig viele Eingänge Eingangswerte an die Eingangsvariablen übergeben bekommt. Intern erfolgt in einer Funktion eine algorithmische Verarbeitung der Werte der Eingangsvariablen zu genau einem Ergebnis. Dieses Ergebnis wird als Funktionswert zur weiteren Verarbeitung an den aufrufenden Ausdruck zurückgegeben. Zur internen Verarbeitung können beliebig viele lokale Variablen deklariert werden, die außerhalb der Funktion nicht zugreifbar sind. Ein Funktion verfügt über keinerlei internen Speicher, so dass bei gleichen Eingangswerten immer der gleiche Funktionswert zurückgegeben wird.

b) Funktionsbaustein

Ein Funktionsbaustein, im CoDeSys-Programmiersystem auch als Funktionsblock bezeichnet, kann neben beliebig vielen internen und Eingangsvariablen auch über beliebig viele Ausgangsvariablen verfügen. Ein Funktionsbaustein muss zur Nutzung instanziiert werden und kann somit mehrfach innerhalb eines Programms oder eines anderen Funktionsbausteins genutzt werden. Zudem verfügt ein jedes individuelle, instanziierte Exemplar eines Funktionsbausteins über ein Gedächtnis, indem die Werte der internen Variablen zwischen zwei Aufrufen gespeichert werden.

c) Programm

Das Programm stellt die POE zur Definition eines steuerungstechnisch zusammengehörigen Funktionsablaufs dar. Es beinhaltet Anweisungen und Ausdrücke, welche auch die erwähnten Konstrukte (Funktionen und Funktionsbausteine) beinhaltet und nutzt. Ebenso wie ein Funktionsbaustein kann ein Programm über beliebig viele Eingangs-, Ausgangs- und interne Variablen verfügen. Ein Programm verfügt ebenfalls über einen internen Speicher und stellt in einer SPS die oberste Ebene dar, die von einem Task direkt aufgerufen werden kann.

Die Bausteine vom Typ Funktionsblock und Programm können durch Aktionen ergänzt werden. Strukturell und funktional stellt eine Aktion einen Teil des übergeordneten Bausteins dar und kann somit auch auf die Daten des übergeordneten Bausteins zugreifen. Aktionen werden nur ausgeführt, sofern diese explizit aufgerufen werden.

Das CoDeSys-Programmiersystem bietet die von den meisten Programmiersprachen bekannten Datentypen zur Verwendung:

- der boolesche Datentyp

  Variablen diesen Datentyps können die beiden Wahrheitswerte TRUE und FALSE annehmen, benötigen allerdings trotzdem 8 Bit Speicherplatz im System.

- ganzzahlige Datentypen

  Zu diesen Datentypen gehören BYTE (8 Bit), WORD (16 Bit), DWORD (32 Bit), SINT (8 Bit), USINT (8 Bit), INT (16 Bit), UINT (16 Bit), DINT (32 Bit) und UDINT (32 Bit).

- Gleitpunkttypen

  Zu den Gleitpunkttypen gehören die beiden Datentypen REAL (32 Bit) und LREAL (64 Bit) und sind erforderlich bei der Nutzung rationaler Zahlen.

- String

  Variablen dieses Datentyps ermöglichen die Speicherung beliebig langer Zeichenketten, wobei sämtliche Stringfunktionen jedoch ausschließlich Zeichenketten mit einer Länge von 1 bis 255 verarbeiten können. Bei der Deklaration kann eine STRING-Variable auf eine Zeichenanzahl begrenzt

werden. Erfolgt dies nicht, kann eine solche Variable standardmäßig 80 Zeichen aufnehmen.

- Zeitdatentypen

  Zusätzlich zu diesen üblichen Datentypen, bietet das Programmiersystem die Möglichkeit zur Verwendung sog. Zeitdatentypen. Zu diesen gehören TIME, TIME_OF_DAY (TOD), DATE und DATE_AND_TIME (DT). Während die Zeitangabe bei den ersten beiden Typen in Millisekunden erfolgt, wird bei den Typen DATE und DT die Zeit in Sekunden angegeben.

Zusätzlich zu diesen bereits definierten Datentypen können im CoDeSys-Programmiersystem im Object Organizer unter der Registerkarte *Datentypen* eigene Typen definiert werden. Hierbei handelt es sich um Strukturen (STRUCT), die ein oder mehrere Elemente gleicher und/oder unterschiedlicher Datentypen enthalten.

In der Registerkarte *Visualisierungen* können grafische Oberflächen erstellt werden, die zur Target-Visualisierung im Rahmen des Projekts auf das Zielsystem übertragen werden können. Diese Visualisierungen dienen der Darstellung und ggf. Beeinflussung der Projektvariablen um dem Nutzer einer Steuerung mit grafischem Display u. a. die Konfiguration der entspr. Steuerung zu ermöglichen.

Über die Registerkarte *Ressourcen* werden einerseits die globalen Variablen der Steuerung angelegt und verwaltet und andererseits Bibliotheken über den Bibliotheksverwalter eingebunden, die für die Implementierung der Bausteine zusätzliche, bereits erstellte Funktionen und Funktionsbausteine enthalten können. Über den Punkt *Zielsystemeinstellungen* unter dieser Registerkarte erfolgt die grundlegende Konfiguration des Zielsystems, auf dem die Steuerung betrieben werden soll. Hier werden u. a. Informationen zur Zielplattform, zur zugehörigen Speicheraufteilung, den Netzfunktionen und den gewünschten Visualisierungsoptionen hinterlegt. Zudem bietet die *Steuerungskonfiguration* unter der Registerkarte *Ressourcen* die Möglichkeit, die entsprechenden Ein- und Ausgänge der einzelnen z. Bsp. über den CAN-Bus angeschlossenen Module zu verwalten sowie der Steuerung weitere Module hinzuzufügen. Module, die in diesem Bereich hinzugefügt werden, sind im Rahmen der Projekterstellung über die entspr. symbolischen Hardwareadressen für den Zugriff verfügbar.

4.1.2. Steuer- und Regelsystem

Die Hardware, für die die Steuerung entworfen werden soll, besteht aus zwei Komponenten der Firma SABO Elektronik GmbH. Diese Komponenten sind:

a) Master Terminal PLM 727-2

   Dieses Terminal stellt die zentrale Hardwarekomponente dar, auf dem die mit dem CoDeSys-Programmiersystem entwickelte Steuerung betrieben werden soll. Es handelt sich hierbei um die Ausführung mit der Artikelnummer MTB.727.24. Zum Einsatz kommt hier ein Mikrocontroller mit der von der Motorola 68000er-Familie abgeleiteten ColdFire RISC-Architektur. Neben einem 7" TFT-Touch-Display mit einer Auflösung von 800 x 400 Bildpunkten verfügt dieses Terminal über zwei RS232, zwei RS485 und zwei CAN-Anschlüsse. Zusätzlich bei dieser Ausführung sind noch ein USB-, ein Ethernet (RJ45) und ein Steckplatz für eine

SD-Memory Card vorhanden. Für einen minimalen Betrieb bietet dieses Terminal zusätzlich noch jeweils vier digitale Ein- und Ausgänge.

b) Erweiterungsbaugruppe EWB 730.20

Bei diesem Erweiterungsmodul handelt es sich um eine Baugruppe, die direkt für die Montage auf der Rückseite des o. g. Master Terminals vorgesehen ist. Dieses Modul wird über den CAN-Bus mit dem Master Terminal verbunden und erweitert dieses neben jeweils 16 digitalen Ein- und Ausgängen um zehn analoge Ein-, vier analoge Spannungsausgänge (0..10 VDC[47]) und zwei analoge Stromausgänge (0..20mA). Der im Mikrocontroller des Master Terminals integrierte A/D-Wandler ermöglicht eine digitale Auflösung analoger Ein- und Ausgangswerte von 12 Bit, d. h. analoge Mess- und Stellwerte können im diskreten Bereich in insg. 4096 (0..4095) Stufen dargestellt werden.

Abbildung 31: Simulationskompaktgerät der Firma SABO Elektronik GmbH, bestehend aus dem Master Terminal PLM 727-2 und der Erweiterungsbaugruppe EWB 730.20.

Damit die mit dem *CoDeSys-Programmiersystem* entwickelte Regelung später auf dem in Abbildung 31 dargestellten Simulationskompaktgerät lauffähig sein würde, wurde das SoftSPS-Laufzeitsystem *CoDeSys Control* der Firma *3S-Smart Software Solutions GmbH* benötigt. Das beschriebene Master Terminal ist bereits für die Nutzung des Laufzeitsystems lizenziert und mit diesem in der Version *21204055* ausgestattet.

4.2. Prototypische Entwicklung

Das für den Verlauf der prototypischen Entwicklung gesetzte Ziel lag darin, eine Folgeregelung in Form einer außentemperaturgeführten Vorlaufaufregelung eines Heizkreises umzusetzen. Diese Umsetzung sollte der Einstufung sowohl der Soft- als auch der Hardware dienen, die in diesem Rahmen genutzt wurde. Daraus resultierend sollte am Schluss eine Aussage darüber abgeleitet werden, ob einerseits die Hardware und andererseits die Software den Entwicklungs- und Nutzungsansprüchen genügt.

[47] Volts of Direct Current: Gleichspannung

Der Fokus für die Einstufung der Hardware lag während des Entwicklungsvorgangs auf folgenden Punkten:

- Modularität

  Hier stand besonders die Frage im Vordergrund, wie einfach sich Ergänzungsmodule zu einem Master Terminal hinzufügen bzw. entfernen lassen.

- Funktionalität

  Dieser Punkt diente der Einschätzung, welcher Funktionsumfang mit einem Master Terminal und durch die Erweiterung dieses Terminals mit verfügbaren Ergänzungsmodule zu realisieren ist.

- Bedienbarkeit

  Dieser Punkt betrifft das Master Terminal und sollte der Einschätzung dienen, inwiefern sich entwickelte Software in die Steuerung integrieren lässt.

Wie bereits für die Hardware sollten während der Entwicklung folgende Aspekte der Software betrachtet werden:

- Bedienbarkeit

  Unter diesem Aspekt wurde besonders die allgemeine Arbeit mit dem CoDeSys-Programmiersystem betrachtet. Dazu zählten Punkte wie die erste Installation, das Anlegen eines Projektes und die gewöhnliche Entwicklungsarbeit bis ein Projekt auf der Zielplattform lauffähig ist.

- Modularität

  Dieser Gesichtspunkt stellte einen schwerwiegenden Teilbereich der Softwareentwicklung dar. Hierbei sollte abgeschätzt werden, ob entwickelte Softwarekomponenten einfach wiederverwendet und komplette Steuerungen einfach portierbar gestaltet werden können.

4.2.1. Anlegen und Konfigurieren des Projekts

Um mit der Implementierung der Steuerung beginnen zu können, musste zu Beginn ein leeres Projekt im CoDeSys-Programmiersystem erstellt werden. Hierzu ist es erforderlich zu wissen, dass nach erfolgter Installation der Programmierumgebung noch zusätzliche Targets installiert werden müssen. Bei den Targets handelt es sich um die vom Hersteller der Hardware zu liefernden Konfigurationsdateien, die die Schnittstelle zwischen dem CoDeSys-Programmiersystem und der zu programmierenden Hardware bilden. In diesen Targets werden neben den I/O-Modulen und dem Funktionsumfang der Hardware auch der Codegenerator und das Speicherlayout konfiguriert.

Im Fall der Hardware der *SABO Elektronik GmbH* waren diese Targets im Downloadbereich der Homepage als ZIP-Archiv erhältlich. Einmal entpackt konnten die Targets mit Hilfe des zur CoDeSys-Programmierumgebung gehörigen Programms *InstallTarget* (zu finden in der über das Startmenü erreichbaren Programmgruppe *3S Software*) in die CoDeSys-Umgebung eingebunden werden.

Um die Implementierung zu ermöglichen, musste zuerst ein zur gewählten Hardware kompatibles Projekt erstellt werden. Nach dem Start des Programmiersystems und der Auswahl *Neu* aus dem CoDeSys-Menüpunkt *Datei* erschien der Dialog zur

Auswahl des Zielsystems. Wurden die Targets der Firma *SABO Elektronik GmbH* korrekt installiert, konnte an dieser Stelle zwischen mehreren Systemen gewählt werden. In diesem Fall wurde der Punkt *SABO PLM-700* (in Abbildung 32 blau unterlegt) aus dem Pulldown-Menü *Konfiguration* gewählt, da hier das Master Terminal PLM 727-2 genutzt wurde.

Abbildung 32: Anfängliche Einstellung zum Zielsystem beim Anlegen eines neuen Projektes in der CoDeSys-Programmierumgebung.

Wurde die entsprechende Wahl getroffen, erweiterte sich das aktive Fenster für die Einstellungen zum Zielsystem wie in Abbildung 33 zu erkennen. Entsprechend den Angaben der Hardwarehersteller waren bei den Einstellungen unter der Registerkarte *Zielplattform* keinerlei ergänzende Angaben zu oder Änderungen bei der Konfiguration notwendig.

In der Registerkarte *Speicheraufteilung* erfolgten die vom genutzten PLM-System abhängigen Angaben zu den verfügbaren Speicherbereichen. Diese konnten bei dem hier genutzten Master Terminal in Erfahrung gebracht werden, indem das System im STOP-Modus (Schalter S201 auf der Rückseite in Mittelstellung) gestartet wurde. Im Display erschienen nun neben den Angaben zur Laufzeitversion und zur Netzwerk-Konfiguration auch Informationen zum verfügbaren Speicher. Für das Master Terminal PLM 727-2 wurden hierbei folgende Angaben angezeigt:

```
Datamemory: 524288
Codememory: 3145728
Bausteine: 2048
```

Diese Angaben erfolgten in Byte bzw. in einer absoluten Anzahl der von der Steuerung in einem Projekt verwaltbaren Bausteine. Der Wert für den Codememory musste in dieser Registerkarte im Bereich *Code* und der Wert für den Datamemory in die Bereiche *Retain* und *Größe des gesamten Datenspeichers* eingetragen werden. Die Anzahl der Bausteine war hier bei dem Bereich *Maximale Anzahl von Bausteinen* einzugeben.

Bei den Einstellungen unter der Registerkarte *Allgemein* war nur eine erweiterte Angabe erforderlich. Die Option zur Aktualisierung nicht verwendeter I/O's musste aktiviert werden, damit CoDeSys einen Task erzeugt, der auch momentan nicht verwendete Ein- und Ausgänge der Steuerung und der Module aktualisiert.

Damit die Steuerung später automatisch mit den über den CAN-Bus angeschlossenen Modulen kommunizieren konnte, musste unter der Registerkarte

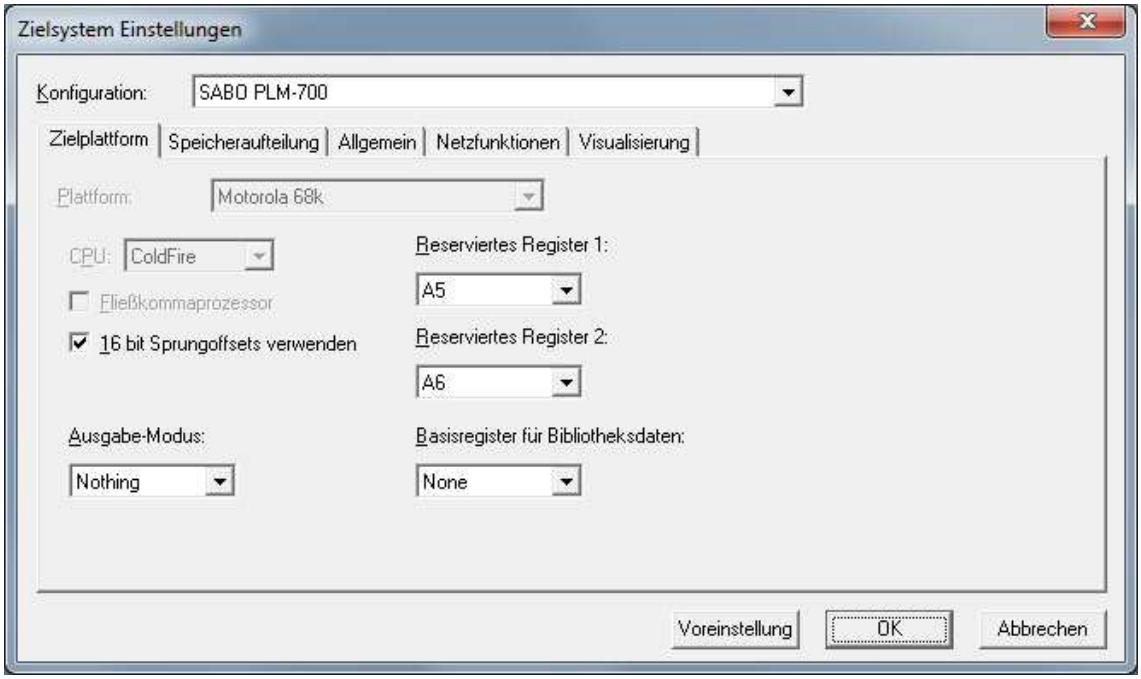

Abbildung 33: Erweiterter Dialog zu den Einstellungen des Zielsystems beim Anlegen eines neuen Projekts im CoDeSys-Programmiersystem.

Netzfunktionen die Option *Netzvariablen unterstützen* aktiviert und als Name für das Netzwerkinterface *CAN* eingetragen werden.

Da das hier verwendete Master Terminal ein Display besitzt und somit eine Target-Visualisierung unterstützt, musste der Punkt *Target-Visualisierung* aktiviert werden bzw. bleiben. Dieses Terminal bietet auch die Option einer Web-Visualisierung, die hier allerdings nur unter einer zweistündigen Testlizenz integriert war. Da diese ohnehin nicht genutzt werden sollte und somit nur unnötig Daten hätten auf die Steuerung übertragen werden müssen, wurde der Punkt *Web-Visualisierung* deaktiviert. Zuletzt mussten noch die Werte für die Displaymaße eingetragen werden, die für die Anzeigenbreite 800 Pixel und für die Anzeigenhöhe 480 Pixel betragen.

Waren diese Angaben für das zu programmierende Zielsystem vollständig erfolgt, konnte der Dialog *Zielsystem Einstellungen* mit *OK* bestätigt und verlassen werden.

Im nun automatisch erscheinenden Dialog *Neuer Baustein* wurde der erste Baustein des Projekts angelegt. Hierbei konnte der Name frei gewählt werden, wurde für dieses Projekt allerdings auf dem Standard *PLC_PRG* belassen. Da dies der zentrale Baustein war, der später von einem Task aufgerufen werden sollte, wurde als Bausteintyp *Programm*

Abbildung 34: Anlegen des ersten Bausteins für die Steuerung im CoDeSys-Programmiersystem.

gewählt. Zur Realisierung der hier vorliegenden Aufgabe wurden algorithmisch anspruchsvolle Lösungen erforderlich, so dass für diesen Baustein und für alle folgenden dieses Projekt *Strukturierter Text* (ST) als zur Norm IEC 61131-3 konforme Sprache gewählt wurde.

Die grundlegenden Einstellungen waren hiermit erfolgt und im Arbeitsbereich erschien ein geöffnetes Editor-Fenster für den ersten Baustein des Projekts mit dem Namen *PLC_PRG*. Im Object Organizer auf der linken Seite wurde der Baustein ebenfalls als Bestandteil des Projekts aufgelistet und bot die Möglichkeit, ein geschlossenes Editor-Fenster per Doppelklick erneut zur Bearbeitung zu öffnen.

Für die abschließende Konfiguration des Projekts war es erforderlich, im Object Organizer auf die Registerkarte *Ressourcen* zu wechseln und die *Taskkonfiguration* mittels Doppelklick zu öffnen (s. Abbildung 35). Hier wurden die verschiedenen Tasks für die Abarbeitung der angelegten Programme erzeugt und verwaltet. Jedem im Bereich von 0 bis 31 priorisierten Task konnten hierbei mehrere Programme zugeordnet werden, die bei Ausführung des Tasks abgearbeitet wurden. Die Taskausführung konnte entweder freilaufend oder zyklisch, also zeitlich definiert, oder intern oder extern ereignisgesteuert erfolgen.

Der erste Task dieses Projekts wurde automatisch mit *PLC_PRG_TASK* benannt und mit der Priorität 1 versehen. Damit dieser Task auf den ersten Blick verständlicher wurde, erfolgte für dieses Projekt eine Umbenennung in *Main* und eine Einstellung auf einen freilaufenden Ablauf. Bei einer freilaufenden Abarbeitung eines Tasks werden sämtliche diesem Task angehängte Programme vollständig abgearbeitet, bevor die Bearbeitung erneut von vorn beginnt. Ein zyklisches Intervall z. Bsp. war weder erwünscht noch vorteilhaft, da für den Abarbeitungszyklus eine feste Intervallzeit angegeben werden muss. Sollten aufwendige algorithmische Berechnungen nicht innerhalb dieser Zykluszeit erfolgen, wird die Abarbeitung der an diesen Task angehängten Programme abgebrochen und erneut von vorn gestartet.

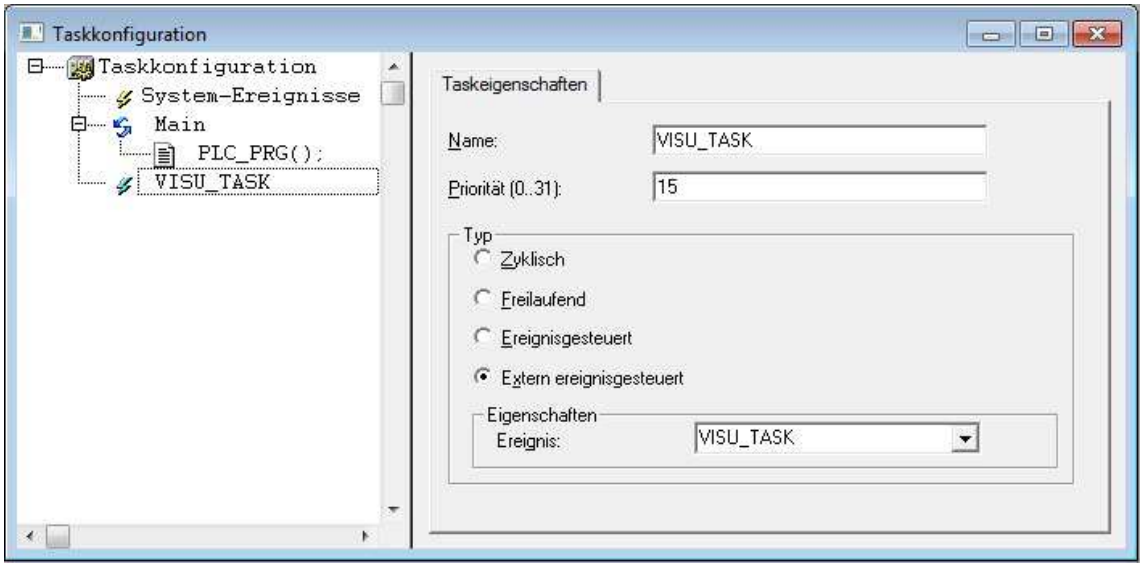

Abbildung 35: Die Taskkonfiguration des CoDeSys-Programmiersystems für den Ablauf des Programms für die Steuerung.

An diesen Main-Task wurde bereits das zuvor angelegte Programm *PLC_PRG* angehängt, so dass dieses den Hauptablauf der Steuerung repräsentierte. Da jedes Projekt mehrere Programme enthalten kann, können diese auch an diesen *Main*- oder einen beliebigen anderen Task angehängt werden. Für dieses Projekt genügte die alleinige Zuordnung des Hauptprogramms *PLC_PRG* zu dem freilaufenden Task *Main*.

Der zweite Task dieses Projekts war der *VISU_TASK*. Hierbei handelt es sich um einen vom System bereitgestellten Task, der für das Zeichnen der Elemente für die Target-Visualisierung und die Verarbeitung der Nutzereingaben über das an die Steuerung angeschlossenen Touch-Display zuständig ist. Für dieses Projekt blieben der vorgegebene Name *VISU_TASK* und die festgelegte Priorität von 15 erhalten. Da dieser Task extern über den Systemtask *VISU_TASK* gesteuert wird, musste zum einen das automatisch angehängte Programm *MAINTARGETVISU_PAINT_CODE* entfernt und die Ablaufsteuerung auf *Extern ereignisgesteuert* umgestellt werden. Im Pulldown-Menü *Ereignis* wurde nun der Eintrag *VISU_TASK* ausgewählt.

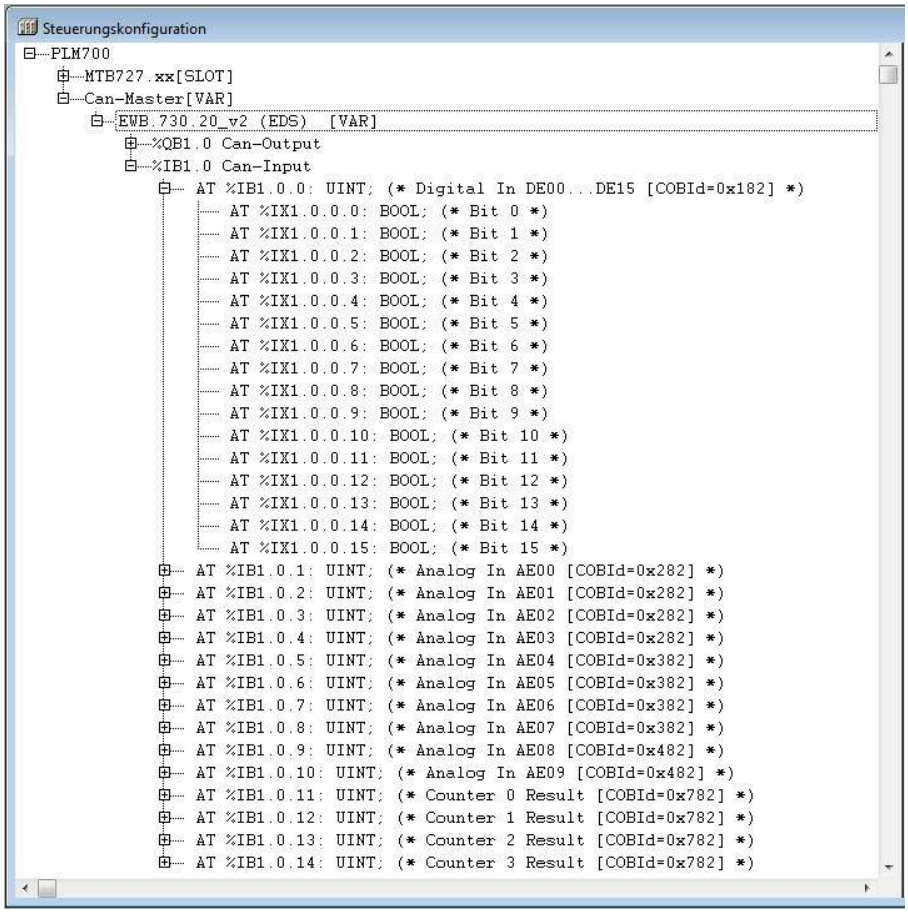

Abbildung 36: Gliederungsebene der Steuerungskonfiguration im CoDeSys-Programmiersystem

Nach Abschluss der Taskkonfiguration musste nun die Steuerungskonfiguration mittels Doppelklick auf den entspr. Eintrag der Registerkarte *Ressourcen* im *Object Organizer* geöffnet werden. Hier erfolgte die weiterführende Konfiguration der Steuerung und der an sie angeschlossenen Module. Bis zu diesem Zeitpunkt wurde dem CoDeSys-Programmiersystem zu Beginn nur die Gerätefamilie (*SABO PLM-700*) mitgeteilt, für die die Programmierung erfolgen sollte. Wie bei der Taskkonfiguration war auch in diesem Konfigurationsfenster ein übergeordnetes Modul (diesmal mit dem Namen *PLM700*) vorhanden. An dieses Modul wurde bereits automatisch ein nicht

löschbares Konfigurationsmodul für ein Master Terminal (in diesem Fall das erste verfügbare mit dem Namen *MPM730.xx*) angefügt. Wie im Abschnitt *Steuer- und Regelsystem* erwähnt, wurde in diesem Projekt das Master Terminal mit der Artikelbezeichnung *MTB 727.24* verwendet, so dass mittels Rechtsklick auf den falschen Eintrag *MPM700.xx* das Kontextmenü aufgerufen werden musste. Über den Menüpunkt *Element ersetzen* wurde das korrekte Konfigurationsmodul mit dem Namen *MTB727.xx* ausgewählt und eingesetzt. Dieses beinhaltete nun bereits die absoluten Adressvariablen für den Zugriff auf die jeweils vier digitalen Ein- und Ausgänge.

Für die Anbindung weiterer Module über den CAN-Bus war es nun erforderlich, mittels Rechtsklick auf den Namen *PLM700* des Hauptmoduls das Kontextmenü zu öffnen. Über den Menüpunkt *Unterelement anhängen* wurde nun ein Modul für den *CAN-Master* hinzugefügt. Dieses Modul gab dem CoDeSys-Programmiersystem Zugriff auf die CAN-Schnittstelle des Masters und ermöglichte die Anbindung weiterer CAN-Teilnehmer als Unterelemente dieses Master-Moduls. Wichtig war hierbei, dass jedes Master Terminal immer die für die Identifizierung am CAN-Bus notwendige *Node-ID 1* automatisch zugewiesen bekommt. Einsehbar ist diese Einstellung, wenn der Eintrag *Can-Master* ausgewählt und auf der rechten Seite des Fensters zur *Steuerungskonfiguration* die Registerkarte *CAN Parameter* selektiert wird. Für dieses Projekt wurde die *Node-ID* auf 1 und die *Baudrate* für den CAN-Bus auf 125kBit/s belassen. In der Registerkarte *Modulparameter* musste unter dem Eintrag *UpdateTask* als Wert der Name des Tasks eingetragen werden, aus dem der Aufruf des Can-Device' erfolgen sollte. In diesem Projekt handelte es sich um den Task mit dem Namen *Main*, in dem auch der Ablauf des Hauptprogramms *PLC_PRG* gesteuert wurde.

Um im CoDeSys-Programmiersystem Zugriff auf die Ein- und Ausgänge des Erweiterungsmoduls zu bekommen, musste mittels Rechtsklick auf den Eintrag *Can-Master* erneut das Kontextmenü aufgerufen und aus dem Eintrag *Unterelement anhängen* das Modul für die im Abschnitt *Steuer- und Regelsystem* vorgestellte Erweiterungsbaugruppe ausgewählt werden. In diesem Fall handelte es sich um den Eintrag mit dem Namen *EWB.730.20_v2*. Auch hier war die Angabe der *Node-ID* in der Registerkarte *CAN Parameter* erforderlich. Diese *Node-ID* wurde mittels Drehschalter auf der Rückseite des Erweiterungsmoduls eingestellt und für dieses Projekt auf 2 festgelegt.

Sowohl bei den Konfigurationsmodulen für das Master Terminal und den Can-Master als auch bei dem an den Can-Master angehängten Modul für die Erweiterungsbaugruppe gibt es die Möglichkeit, unter der Registerkarte *Basisparameter* eine *Knotennummer* zu vergeben. Diese ist für das Master Terminal immer unabänderbar auf 0 festgelegt und kann für den Can-Master beliebig gewählt werden. Hierbei gilt es zu beachten, dass auf jeder Gliederungsebene jede Knotennummer einzigartig sein muss. Würde beispielsweise ein zweites Modul für einen Can-Master an die Konfiguration der Steuerung (PLM700) angehängt werden, müsste dieses eine andere Knotennummer zugewiesen bekommen. Ebenso verhält es sich mit den Konfigurationsmodulen für die an den Can-Master angehängten Erweiterungsbaugruppen. Diese stellen die nächsttiefere Gliederungsebene der Steuerungskonfiguration dar, so dass mit der Nummerierung der einzelnen Knoten wieder bei 0 begonnen werden kann. Im Normalfall werden die Knotennummern automatisch vom Programmiersystem in der Reihenfolge vergeben, in der das jeweilige Modul an die entspr. Gliederungsebene angehängt wird. Für dieses Projekt erhielt das

Modul für das Master Terminal die Knotennummer 0, der Can-Master die Knotennummer 1 und das Erweiterungsmodul wieder die Knotennummer 0.

Wichtig ist, dass die Node-ID und die Knotennummern unterschiedliche Identifikationseinheiten darstellen und nicht miteinander gleichbedeutend sind. Die Node-ID dient ausschließlich der Identifikation des jeweiligen Gerätes am CAN-Bus während die Knotennummer eine strukturelle Ordnung in der Gliederung und Zuordnung der Geräte zueinander herstellt. Dies ist erforderlich, um die einzelnen Komponenten im CoDeSys-Programmiersystem gezielt adressieren zu können. Möchte man in diesem Fall beispielsweise das erste, dem Can-Master untergeordnete Gerät in der Steuerungskonfiguration ansprechen, so wählt man zuerst die Knotennummer des Can-Masters aus der obersten Gliederungsebene. Getrennt durch einen Punkt ergänzt man diese Angabe nun durch die Knotennummer des entspr. an den Can-Master angehängten Gerätes. Im Fall dieses Projektes ergäbe sich die Adresse 1.0 aus der Knotennummer 1 für den Can-Master und der Knotennummer 0 für die dem Can-Master untergeordnete Erweiterungsbaugruppe.

4.2.1.1. Adressierung im CoDeSys-Programmiersystem

Für den weiteren Verlauf ist es unabdingbar, die Adressierungsmethode des CoDeSys-Programmiersystems an dieser Stelle näher zu erläutern. Hierfür erfolgt eine Adressierung der entspr. Ein- oder Ausgänge eines Moduls mittels direkter Darstellung einzelner Speicherzeilen. Dazu werden innerhalb der Programmierumgebung spezielle Zeichenketten eingesetzt, die sich aus dem Prozentzeichen (%), einem Bereichspräfix, einem Größenpräfix und einer Folge natürlicher Zahlen, getrennt durch einen Punkt zusammensetzen. Das Bereichspräfix gibt dabei an, um welche Art von Zugriffsort es sich bei der folgenden Adresse handelt, das Größenpräfix ergänzt diese Information um die Angabe, um welchen Typ es sich bei der Adressierung handelt. Folgende Bereichspräfixe sind in der CoDeSys-Programmierumgebung verfügbar:

- I verweist auf einen physikalischen Eingang
- Q verweist auf einen physikalischen Ausgang
- M verweist auf eine physikalische Adresse im SPS-Speicher

Während die beiden erstgenannten Präfixe eine immanente Stellung bei der Steuerungsprogrammierung einnehmen, sind die Verweise mittels M-Präfix (Merker) fast vollständig erlässlich geworden.

Für die Angabe, welcher Datentyp unter dem zuvor genannten Bereich adressiert werden soll, stehen folgende Präfixe zur Verfügung:

- X Einzelbitgröße (ausschließlich für Datentyp BOOL, 8 Bit)
- B Byte (8 Bit)
- W Word (16 Bit)
- D Double Word (32 Bit)
- L Long Word (64 Bit)

Allein mit diesen Möglichkeiten wäre z. Bsp. der erste digitale Eingang des Erweiterungsmoduls nicht adressierbar. Hier spielen nun die zuvor genannten

Knotennummern eine bedeutende Rolle, da sie letztlich den Pfad zu dem gewünschten Gerät identifizieren. In diesem Projekt entspricht also die 1 der Knotennummer für den Can-Master, an den das Erweiterungsmodul angebunden ist. Das Erweiterungsmodul selbst wird schließlich durch die Knotennummer 0 identifiziert. Bis hierher würde die symbolische Adresse also *%IB1.0* lauten, ohne dass klar ist, welcher Eingangsbereich genau angesprochen werden soll. Die digitalen Eingänge des Moduls *EWB 730.20* werden nebeneinander liegend als *UINT* bzw. *WORD* (beide erstrecken sich mit 16 Bit über den jeweils gleichen Wertebereich) verwaltet, so dass in der Steuerungskonfiguration an den Input-Bereich des Erweiterungsmoduls ein weiteres Untermodul angehängt ist. Dieses hat die Knotennummer 0 erhalten und ist in keiner Weise editierbar, da es einen integralen Bestandteil der Steuerungskonfiguration des Erweiterungsmoduls darstellt. Bis hierher ließe sich also das *WORD* für den digitalen Eingangsbereich des ersten an den Can-Master angehängten Erweiterungsmoduls symbolisch mit *%IB1.0.0* adressieren. Da sämtliche Ein- und Ausgänge letztlich aus einer zusammengehörigen Abfolge einzelner Bits bestehen, die allerdings byteweise alloziert werden, wird für die Bereichsgröße bis zur vorletzten Gliederungsebene immer das Präfix *B* angegeben. Um nun explizit einen einzelnen digitalen Eingang des Erweiterungsmoduls ansprechen zu können, erfolgt eine weitere Vertiefung der Gliederungsebene, in der die jeweiligen Einzelbits wieder mittels Knotennummern durchnummeriert angehängt sind. Hier kann nun eine symbolische Adressierung der Einzelbits und damit der einzelnen Eingänge mittels der Adressangabe *%IX1.0.0.0* für den ersten, *%IX1.0.0.1* für den zweiten usw. digitalen Eingang erfolgen. Es lassen sich in der Programmierumgebung also alle 16 digitalen Eingänge des Erweiterungsmoduls einzeln durch boolesche Variablen mit einem Speicherbedarf von je 8 Bit (insg. also 16 x 8 Bit = 128 Bit) oder gesammelt in einer Variablen vom Typ *WORD* mit einem Speicherbedarf von 16 Bit darstellen.

4.2.2. Abschließende Projektkonfiguration

Um eine korrekte Nutzung der analogen Eingänge zu ermöglichen, mussten über die Registerkarte *Service Data Objects* der *Steuerungskonfiguration* noch die entspr. Werte für den jeweiligen analogen Eingang gem. Tabelle 2 angegeben werden. Die hier getroffene Auswahl hat den Effekt, dass die Steuerung die bitweise Wertedarstellung entsprechend anpasst und sämtliche Spannungs- und Strommesswerte als 16-Bit-Integer vom Typ *UINT* (entspricht dem Wertebereich eines *WORD* von *0* bis $2^{16} - 1 = 65535$) mit einer durch den integrierten A/D-Wandler bestimmten Genauigkeit von 12 Bit (tatsächlicher Wertebereich *0* bis $2^{12} - 1 = 4095$) darstellt (s. [SABO3], S. 3).

Wert	Bedeutung
0	0..10V
1	0..20mA
2	Pt1000 / 0°C..650°C
3	Pt1000 / -50°C..150°C
4	Ni1000 / -50°C..150°C
5	Pt100 / -50°..150°C

Tabelle 2: Werte für die Konfiguration der analogen Eingänge am Erweiterungsmodul in der Steuerungskonfiguration

Sämtliche Temperaturmesswerte entsprechen einer durch den Widerstandswert des angeschlossenen Typs von Temperaturfühler hervorgerufenen Spannung. Diese wird durch den A/D-Wandler der Steuerung in einen entsprechenden Zahlenwert aus dem (durch die Konfiguration des Eingangs gem. Tabelle 2) definierten Wertebereich transformiert. Hierbei ist zu beachten, dass sämtliche Temperaturwerte durch die Steuerung als mit 10 multiplizierte 16-Bit-Integer-Zahl vom Typ *INT* (Wertebereich von *-32768* bis *32767*) dargestellt werden (s. [SABO3], S. 3). Im Hinblick auf die Genauigkeit des

A/D-Wandlers von 12 Bit ergäbe sich in diesem Fall ein steuerungsseitiger Wertebereich von -2048 bis 2047 (*$-2^{12} / 2 = -2048$* bis *$2^{12} / 2 - 1 = 2047$*). Dies würde den Anschluss eines beliebigen Pt- oder Ni-Temperaturfühlers[48] mit einem Norm-Temperaturbereich von -204.8°C bis 204.7°C ermöglichen. Die hier genutzte Steuerung unterstützt gem. Tabelle 2 einen maximalen Temperaturbereich von -50°C bis 150°C bei Verwendung eines Pt1000- oder Ni1000-Temperaturfühlers. Für die im Rahmen dieses Projektes vorliegenden Anforderungen war dieser Messbereich vollkommen ausreichend, da im Normalfall mit minimalen Außentemperaturen von -20°C und maximalen Vorlauftemperaturen von 100°C gearbeitet wurde. Wichtig in diesem Zusammenhang ist allerdings, dass die Steuerung bei entspr. Konfiguration tatsächlich nur Werte im Bereich von -500 (für -50°C) bis 1500 (für 150°C) korrekt darstellt. Bei einer manuellen Regulierung des am entspr. Eingang anliegenden Widerstandswertes über den zugehörigen Drehregler am Simulationskompaktgerät wurden nicht definierte Widerstandswerte nicht mehr korrekt und zumeist mit dem Wert 9999 dargestellt. Während des Simulationsbetriebs (s. Beschreibung *Demonstrationsmodus* auf S. 79) der Steuerung werden sämtliche analogen Eingänge daher als Spannungseingänge (0..10V) eingerichtet und als *WORD*-Wert interpretiert, um eine vereinfachte Bedienung der Steuerung und Darstellung der Werte zu realisieren. Im realen Anlagenbetrieb erfolgt die Interpretation und Verarbeitung der Werte der analogen Eingänge als *INT*-Typ, sofern diese korrekt gem. Tabelle 2 konfiguriert wurden.

Damit das Projekt vollständig für die Arbeit mit und die Programmierung der Steuerung vorbereitet war, mussten nun noch erforderliche Bibliotheken über den *Bibliotheksverwalter* unter der Registerkarte *Ressourcen* des *Object Organizers* eingebunden werden. Dieser ließ sich wieder mittels Doppelklick im Arbeitsbereich öffnen und durch den Aufruf des Kontextmenüs im linken oberen Bereich (der Übersicht über bereits in das Projekt eingebundener Bibliotheken) konnten durch die Auswahl *Weitere Bibliotheken…* dem Projekt zusätzliche Bibliotheken hinzugefügt werden. Für die fehlerfreie Anbindung von Erweiterungsmodulen an die Steuerung über den CAN-Bus mussten folgende Bibliotheken eingebunden werden:

- 3S_CanDrv.lib
- 3S_CANopenManager.lib
- 3S_CANopenMaster.lib
- 3S_CANOPENNETVAR.lib

Damit sämtliche Funktionalitäten der Target-Visualisierung verfügbar waren, auf diverse Systemkomponenten zugegriffen werden konnte und eine bitgenaue Variablenmanipulation ermöglicht wurde, mussten zusätzlich noch die drei folgenden Bibliotheken in das Projekt eingebunden werden:

- SysLibTargetVisu.lib
- Util.lib
- UPD_E_007.lib

Als letzte Einstellung war über das Menü *Online* und den Menüpunkt *Kommunikationsparameter* noch festzulegen, auf welche Art die Kommunikation

[48] auf Platin (Pt) oder Nickel (Ni) basierender Widerstands-Temperaturfühler

zwischen dem Rechner mit dem CoDeSys-Programmiersystem und der zu programmierenden Steuerung erfolgen sollte. Für dieses Projekt wurde der Weg über die Ethernet-Verbindung mittels TCP/IP gewählt, die in der Einstellung der Kommunikationsparameter mittels Klick auf den Button *Gateway* konfiguriert wurde. Hierzu wurde die Verbindung zum Gateway auf *Lokal* gestellt und anschließend mit einem Klick auf den Button *Neu* eine neue zu diesem Gateway gehörige Ethernet-Verbindung eingerichtet. Diese Verbindung erhielt in diesem Projekt den Namen *tcp 242 route* und für die Art der Verbindung wurde *Tcp/Ip (Level 2 Route)* gewählt. Die Kommunikation zur Steuerung erfolgte über die IP-Adresse 192.168.10.10 und den zugehörigen Port 1202. Gemäß den Herstellerangaben musste hierbei die Motorola-Byte-Order aktiviert werden.

Um mit der eigentlichen Implementierung beginnen zu können musste mit dem Anlegen des ersten Visualisierungsbildschirms im Projekt noch ein letzter vorbereitender Schritt erfolgen. Hierzu wurde unter der Registerkarte *Visualisierungen* mittels Rechtsklick in den Bereich des *Object Organizers* eine neues *Visu-Objekt* mit dem Namen *PLC_VISU* hinzugefügt. Sofern dieser Name benutzt wird, ist dies immer der erste Bildschirm, der bei einem Start des erstellten Projekt auf der Zielsteuerung dargestellt wird. In allen anderen Fällen wird das erste Visualisierungselement in der Liste der Visualisierungen automatisch als Startbildschirm festgelegt.

Nach Abschluss dieser Projekterstellung lag eine für die Programmierung der Steuerung vollständig konfigurierte Umgebung vor. Sowohl der Baustein für das Hauptprogramm *PLC_PRG* als auch alle folgend angelegten Bausteine wurden dabei automatisch in einen Deklarations- und einen Implementierungsteil gegliedert.

Hierbei sollte die Benennung der Variablen möglichst an die ungarische Notation angelehnt erfolgen, bei der jede Variable zunächst aus einem kurzen, aussagekräftigen Basisnamen besteht. Bei diesem Basisnamen sollte jedes Wort mit einem Großbuchstaben beginnen, während die übrigen Buchstaben klein geschrieben werden (z. Bsp. *InitCounter*). Diesem Basisnamen sollten schließlich dem jeweiligen Datentyp entsprechende Präfixe gem. Tabelle 3 vorangestellt werden. Für eine Zählvariable des Typs *BYTE* würde der Bezeichner also z. Bsp. *byInitCounter* lauten. Für eine verschachtelte Variablendeklaration sollten die Präfixe schließlich in der Reihenfolge ihres Auftretens dem Basisnamen vorangestellt werden, so dass ein *POINTER* auf ein *BYTE* z. Bsp. mit *pbyInitCounter* benannt werden würde. Handelt es sich bei der Deklaration um Konstanten, so sollte diesen das Präfix *c* gefolgt von einem Unterstrich dem Typ-Präfix vorangestellt werden, so dass eine Konstante z. Bsp. den Bezeichner *c_byAnzahlEing* tragen würde. Globale Variablen sollten dem Typ-Präfix noch ein *g* gefolgt von einem Unterstrich und globale Konstanten das Kürzel *gc* gefolgt von einem Unterstrich vorangestellt bekommen. Der Bezeichner einer globalen Variable sollte dann z. Bsp. *g_byAnzahlEing* und der Bezeichner einer entspr. globalen Konstante *gc_byAnzahlEing* lauten. Für die benutzerdefinierten Datentypen sollten die zugehörigen Präfixe entspr. im Implementationsteil als Kommentar angefügt werden.

Datentyp	Präfix
BOOL	x
BYTE	by
WORD	w
DWORD	dw
SINT	si
INT	i
REAL	r
STRING	s
POINTER	p
ARRAY	a

Tabelle 3: Datentypabhängige Präfixe zur Variablendeklaration.

4.2.3. Definition des Programmablaufs

Bevor mit der Implementierung begonnen werden konnte, war es notwendig, sowohl die letztlich gewünschte Funktionalität der Steuerung als auch die hierfür benötigten Komponenten zu beschreiben.

Wie im Abschnitt *Anlegen und Konfigurieren des Projekts* erwähnt, sollte das Steuerungsprogramm in einem freilaufenden Modus abgearbeitet werden. Es war also erforderlich, vor Beginn der Implementierung zu klären, in welche Teilschritte der Programmablauf zerlegt werden konnte.

Wie in Abbildung 37 dargestellt ist, wurde der gesamte Programmablauf während eines Operationszyklus' der Steuerung in fünf Operationen aufgeteilt. Es handelt sich hierbei um:

- Initialisierung
- Initialisierungsabschluss abwarten
- Prüfung Freigabe
- Freigabe erneut prüfen
- Betriebszyklus

Nach dem *Start* der Steuerung sollte diese mit der *Initialisierung* beginnen. Hierbei sollten sämtliche relevanten Systemwerte sowie alle analogen und digitalen Ein- und Ausgänge eingerichtet werden. Nach Abschluss des Initialisierungsvorgangs sollte ein entspr. Flag gesetzt werden, welches den vollständigen Durchlauf der Setup-Routinen signalisiert. Um während des Startvorgangs der Steuerung Messwert-Peaks besonders an den analogen Eingängen zu vermeiden, sollte parallel zum Initialisierungsvorgang ein entspr. Initialisierungstimer laufen. Dieser musste auf jeden Fall den Zeitbedarf des Initialisierungsvorgangs überschreiten, da er letztlich den Abschluss des Initialisierungsvorgangs signalisieren sollte. Dieser Timer wurde notwendig, da besonders die analogen Eingänge während der Setup-Routinen relativ träge reagierten und bei zu schnellem Durchlauf der Initialisierung sonst falsche bzw. nicht existente Werte in

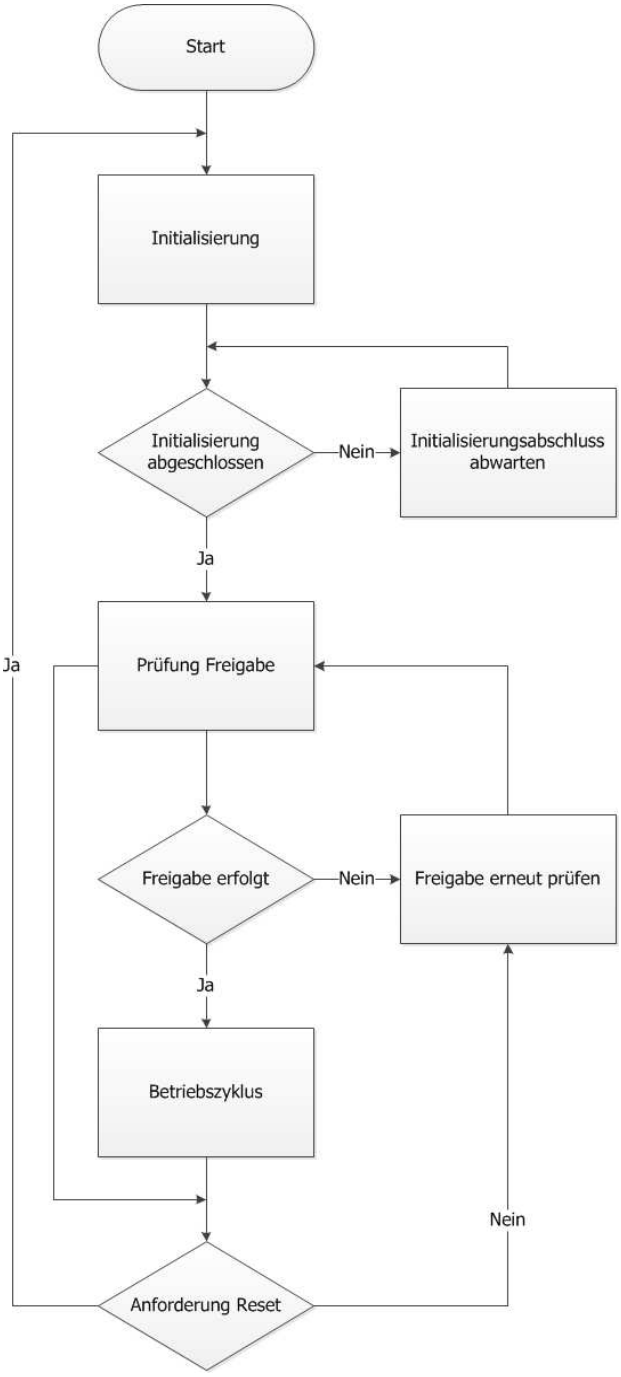

Abbildung 37: Ablaufplan für einen Zyklus der freilaufenden Steuerung.

den später folgenden Betriebszyklus propagierten. Auch das vom Timer getrennte Flag für die Setup-Routinen war erforderlich, da diese Routinen sonst immer wieder aufgerufen wurden, so langer der Setup-Timer noch aktiv war.

Solange der Initialisierungstimer aktiv ist, sollte sich die Steuerung also im Zustand *Initialisierungsabschluss abwarten* befinden. Nach dessen Ablauf sollte die Steuerung in den Zustand *Prüfung Freigabe* versetzt werden. In diesem Zustand erfolgte zum einen die Überprüfung der Anlagenbestandteile (Brenner und Pumpe) auf Störungen und zum anderen eine Überprüfung der Heizfreigabe. Diese Heizfreigabe sollte nur dann erfolgen, wenn ein bestimmter Wert für die Außentemperatur nicht überschritten wird. Ebenso sollte die Freigabemeldung für die einzelnen Anlagenkomponenten nur dann erfolgen, wenn keine Funktionsstörungen anliegen. Erst dann sollte die Freigabe der Anlage und somit der Regelung für den Heizbetrieb, einen *Betriebszyklus*, erfolgen. Bereits während dieser Freigabeprüfung sollten sämtliche Ein- und Ausgänge der Steuerung aktualisiert werden, da diese Werte, wie z. Bsp. Außen- und Vorlauftemperatur, auch ohne den Heizbetrieb von Interesse sind. Zudem sollte auch der Reset der Steuerung während der Freigabeprüfung ermöglicht werden, so dass ein Neustart der Steuerung auch dann möglich war, wenn der Heizbetrieb nicht freigegeben wurde.

Ein Betriebszyklus sollte in diesem Zusammenhang einmal ausgeführt werden, während die Freigabe der Anlage erfolgt ist und der Main-Task einmal das Hauptprogramm *PLC_PRG* durchlief. Danach sollte das System erneut in den Status *Prüfung Freigabe* geschaltet werden und mit dem folgenden taskgesteuerten Programmablauf wieder bei der Freigabeprüfung einsetzen.

4.2.4. Implementierung der Steuerungssoftware

Im Ergebnis sollte die Steuerung die außentemperaturgeführte Regelung der Vorlauftemperatur eines wie in Abbildung 1 dargestellten Heizkreises übernehmen. Um dies zu gewährleisten waren folgende grundlegende Aufgaben von der Steuerung zu übernehmen:

- Erfassen/Messen der Außentemperatur
- Erfassen/Messen der Vorlauftemperatur
- Erfassen des Betriebszustandes der Vorlaufpumpe
- Erfassen des Betriebszustandes des Brenners
- Ansteuern der Vorlaufpumpe
- Ansteuern des Brenners
- Ansteuern des Stellmotors für den Drei-Wege-Mischer im Vorlauf

Um diese Funktionalität zu realisieren, musste die Steuerung auf globaler Ebene definiert werden. Hierzu wurden im Bereich *Ressourcen* des *Object Managers* in der Variablenliste mit dem Namen *Globale_Variablen* einige globale Konstanten und Variablen deklariert. Zu diesen zählen vor allem die globalen Konstanten für die Anzahl der an das Master Terminal angeschlossenen Erweiterungsmodule, für die Anzahl digitaler Ein- und Ausgänge am Master Terminal sowie für die Anzahl digitaler und analoger Ein- und Ausgänge am Erweiterungsmodul.

Zudem wurden in diesem Bereich globale Variablen zur Verwaltung der physikalischen Adressen am Master Terminal und am Erweiterungsmodul deklariert. Diese globale Deklaration sollte einen späteren Eingriff für fortführende Projektierungen erleichtern und einen zentralen Anlaufpunkt für evtl. anfallende Adressänderungen oder Erweiterungen durch Modulergänzungen bieten.

Aus dem Programm heraus sollte schließlich mittels Pointer auf die Adressen dieser Variablen und somit indirekt auch auf die physikalischen Adressen verwiesen werden. Dies bietet den Vorteil, dass aus dem Programm heraus immer auf die korrekte physikalische Adresse zugegriffen wird, sofern diese während der Projektierungsphase korrekt in der globalen Variablenliste zugeordnet wurde.

4.2.4.1. Adressierung digitaler Ein- und Ausgänge

An dieser Stelle soll exemplarisch die Umsetzung der Verwaltung der digitalen und analogen Eingänge für das Erweiterungsmodul dargelegt werden. Hierfür erfolgten in der globalen Variablenliste folgende Deklarationen:

```
(* Anzahl der mit dem Masterterminal verbundenen
   Erweiterungsmodule *)
gc_AnzEM: BYTE:= 1;

(* Anzahl digitaler Eingänge am Erweiterungsmodul *)
gc_byAnzDEE: BYTE:= 16;

(* Anzahl analoger Eingänge am Erweiterungsmodul *)
gc_byAnzAEE: BYTE:= 10;
```

Gemäß der bereits erläuterten Konvention zur Namensvergabe für Variablenbezeichner handelt es sich hier um globale Konstanten (*gc_*) vom Typ *BYTE* (*by*). Die Abkürzung *Anz* steht hierbei für *Anzahl* und die Akronyme *DEE*, *AEE* und *EM* für *DigitalEingängeErweiterungsmodul*, *AnalogEingängeErweiterungsmodul* bzw. *ErweiterungsModule*. Diese Form der Abkürzungen wurde konsequent durch den gesamten Programmaufbau beibehalten und dient somit der schnellen Identifikation von Ein- und Ausgabeoperationen auf bestimmten Variablen.

Aus den obigen Angaben ist nun zu erkennen, dass ein Erweiterungsmodul an den Can-Master angeschlossen und gem. den Beschreibungen in Abschnitt *Anlegen und Konfigurieren des Projekts* der Steuerung hinzugefügt wurde. Zudem wurde mit den Angaben festgelegt, dass das Erweiterungsmodul insg. über 16 digitale Ein- und 10 analoge Ausgänge verfügt. Hier muss unbedingt eine exakte dem entspr. Datenblatt zu entnehmende Angabe erfolgen, da es während der Zugriffsoperationen im Programmablauf sonst zu Adressüberschneidungen und Fehlfunktionen der Steuerung kommen kann.

Für den Zugriff auf die physikalischen Adressen der beiden o. g. Eingänge wurden in der globalen Variablenliste zwei weitere globale Variablen angelegt:

```
(* physikalische Adresse für digitale Eingänge am ERSTEN
   Erweiterungsmodul *)
g_byAdrDEE_1 AT %IB1.0.0: WORD;

(* physikalische Adresse für analoge Eingänge am ERSTEN
   Erweiterungsmodul *)
```

```
g_byAdrAEE_1 AT %IB1.0.1: WORD;
```

Die Adressierung erfolgt hierbei gem. den Beschreibungen aus dem Abschnitt *Adressierung im CoDeSys-Programmiersystem*. Neu ist in diesem Zusammenhang das Schlüsselwort *AT*, mit dem eine symbolische Adressierung realisiert wird, indem die Bindung eines Variablenbezeichners an eine physikalische Adresse der Steuerung erfolgt. Im Programm kann nun an jeder Stelle mit dem entspr. Variablenbezeichner auf die zugeordnete physikalische Adresse zugegriffen werden und eine evtl. notwendige Änderung dieser Adresse kann bequem an dieser zentralen Stelle erfolgen.

4.2.4.2. Verwaltung digitaler Ein- und Ausgänge

Damit diese digitalen und analogen Eingänge nun im Programm und während des Betriebs der Steuerung zugreifbar sind, wurden im Deklarationsteil des Hauptprogramms *PLC_PRG* entspr. Variablen deklariert. Für den Zugriff auf die digitalen Eingänge sämtlicher Erweiterungsmodule wurde dies folgendermaßen umgesetzt:

```
(* Verwaltung der digitalen Eingänge an den
   Erweiterungsmodulen *)
apwDEE: ARRAY[1..gc_byAnzEM] OF POINTER TO WORD;
```

Hierbei handelt es sich um ein Array, welches der Aufnahme von Pointern dient, die jeweils auf einen Adressbereich des Datentyps *WORD* referenzieren. Die Größe dieses Arrays ist hierbei abhängig von dem Wert, der in der globalen Variablenliste der Konstanten *gc_byAnzEM* zugeordnet wurde. Es wird also der Speicherplatz für maximal 16 digitale Eingänge - dies entspricht dem durch ein *WORD* repräsentierten Speicherplatz in Bit - für jedes angeschlossene Erweiterungsmodul reserviert. Jedes Bit der referenzierten *WORD*-Adresse stellt hierbei also den Zustand eines digitalen Eingangs des Erweiterungsmoduls dar. Die digitalen Eingänge brauchen somit also nicht jeweils einzeln über ihre entsprechende physikalische Adresse angesprochen zu werden, sondern können mittels einer Adresszuweisung in der globalen Variablenliste im Programm über einen auf diese Adresse gerichteten *WORD*-Pointer dereferenziert werden.

Während des im Abschnitt *Definition des Programmablaufs* beschriebenen Initialisierungsvorgangs werden diesem soeben erwähnten Array die Startadressen der digitalen Eingänge des jeweiligen Erweiterungsmoduls zugewiesen. Dies wurde in der zum Hauptprogramm *PLC_PRG* gehörigen Aktion *AKT_INIT_DE* (im Klartext: *Aktion zur Initialisierung der digitalen Eingänge*) auf folgende Art umgesetzt:

```
(* Zuweisung physikalische Adresse für digitale EINGÄNGE am
   ERSTEN Erweiterungsmodul*)
apwDEE[1]:= ADR(g_wAdrDEE_1);
```

Da in diesem Projekt lediglich ein Erweiterungsmodul an das Master Terminal angeschlossen wurde, erfolgt hier die Zuweisung für genau dieses erste und einzige Element des Pointer-Arrays. In diesem Fall zeigt dieses erste Element des Pointer-Arrays für die digitalen Eingänge an den Erweiterungsmodulen auf einen *WORD*-Datentyp an der Adresse der globalen Variable mit dem Namen *g_wAdrDEE_1*. Wie zuvor erläutert, war dies in diesem Projekt die physikalische Adresse *%IB1.0.0*. Der Zugriff auf das die Zustände sämtlicher digitaler Eingänge enthaltende *WORD*-Datum des ersten Erweiterungsmoduls erfolgt aus dem Programm heraus nun über eine simple

Pointer-Dereferenzierung. Im Falle einer einfachen Wertzuweisung an eine andere Variable des Typs *WORD* mit dem Namen *wBeispiel* sähe der Ausdruck wie folgt aus:

```
wBeispiel:= apwDEE[1]^;
```

Der Dereferenzierungsoperator (^) dient dabei dem Zugriff auf den Inhalt der an der ersten Position des Arrays gespeicherten Adresse. Es wird also der als *WORD* interpretierte Wert an der Adresse *%IB1.0.0* in der Variablen *wBeispiel* gespeichert.

Um nun tatsächlich auf die einzelnen Bit-Werte des *WORD* zugreifen zu können, wurde in diesem Projekt die Funktion mit dem Namen *FUN_GET_DE* implementiert. Diese verlangt die Übergabe von zwei Eingangswerten in Form eines *BYTE* und eines *DWORD*. Der erste Funktionsparameter mit dem Namen *byEingang* stellt dabei die Nummer des digitalen Eingangs am Erweiterungsmodul dar, dessen Wert ausgelesen werden soll. Der zweite Funktionsparameter mit dem Namen *dwSource* muss dabei den aus der entspr. Adresse des Moduls dereferenzierten Wert in Form eines *DWORD* übergeben bekommen. Der Datentyp *DWORD* wurde hier bewusst gewählt, da dieser einen Speicherplatz von 32 Bit bietet und somit aufwärtskompatibel für evtl. Weiterentwicklungen ist. Dies erfordert vor der Übergabe des dereferenzierten Wertes an die Funktion eine Typ-Konversion, die jedoch recht einfach durch die im CoDeSys-Programmiersystem integrierten Konvertierungs-Funktionen zu realisieren ist. Das Funktionsergebnis wird hierbei als *BOOL* zurückgegeben und enthält den Status (*TRUE* oder *FALSE* bzw. *1* oder *0*) des gewünschten digitalen Eingangs. Funktionsintern wird das Auslesen des Bit-Wertes an der Stelle *byEingang* von *dwSource* mit der durch die Bibliothek *Util.lib* bereitgestellte Funktion *EXTRACT* realisiert. Die Zuweisung des Funktionswertes wird dazu mit folgendem Funktionsaufruf realisiert:

```
FUN_GET_DE:= EXTRACT(dwSource, byEingang);
```

Ebenso wie das Abfragen der digitalen Eingänge erfolgt das Setzen der digitalen Ausgänge (z. Bsp. für Schaltbefehle für die Vorlaufpumpe) über eine entspr. Funktion mit dem Namen *FUN_SET_DA*. Diese Funktion erwartet die Werteübergabe an ihre drei Funktionsparameter mit den Namen *byAusgang*, *xStatus* und *dwSource*. Hierbei muss an den *BYTE*-Parameter *byAusgang* der zu setzende Ausgang, an den *BOOL*-Parameter *xStatus* der Wert, auf den *byAusgang* gesetzt werden soll und an *dwSource* der aktuelle Status aller digitalen Ausgänge übergeben werden. Als Funktionswert gibt diese Funktion ein *DWORD* zurück, welches den aus an die Funktionsparameter übergebenen Werten neu generierten Status aller digitalen Ausgänge enthält. Das setzen der digitalen Ausgänge des Erweiterungsmoduls wurde also folgendermaßen realisiert:

```
apwDAE[1]^:=
   DWORD_TO_WORD(FUN_SET_DA(0,1,WORD_TO_DWORD(apwDAE[1]^)))
```

Da auch die digitalen Ausgänge in einem Pointer-Array mit dem Verweis auf die Adressen des jeweils ersten digitalen Ausgangs der Erweiterungsmodule verwaltet werden, muss die Zuweisung des Funktionswerts an die entspr. dereferenzierte Adresse (hier: die Adresse des ersten Erweiterungsmoduls) erfolgen. In diesem Beispiel wird an Stelle 0 des aktuellen *WORD*-Wertes des ersten Erweiterungsmoduls (*apwDAE[1]^*) der Bit-Wert auf 1 gesetzt. Dieser von der Funktion *FUN_SET_DA* zurückgegebene neue *WORD*-Wert wird schließlich in den Bereich der entspr. dereferenzierten Adresse der Steuerung geschrieben. Der erste digitale Ausgang am Erweiterungsmodul würde also aktiviert werden bzw. bleiben.

Neben diesem Zugriff auf die digitalen Eingänge des Erweiterungsmoduls wurde auch der Zugriff auf die analogen Eingänge realisiert. Diese Umsetzung gestaltete sich etwas aufwendiger und wurde im Deklarationsteil des Hauptprogramms *PLC_PRG* folgendermaßen verwirklicht:

```
(* Verwaltung der analogen Eingänge an den
   Erweiterungsmodulen *)
apawAEE: ARRAY[1..gc_byAnzEM] OF POINTER TO
   ARRAY[1..gc_byAnzAEE] OF WORD;
```

Vorweg soll an dieser Stelle für das weitere Verständnis die Anordnung der analogen Eingänge am Erweiterungsmodul verdeutlicht werden. Während mit einem *WORD* bis zu 16 (und damit alle an diesem Erweiterungsmodul vorhandenen) digitale Eingänge verknüpft werden können, beansprucht bereits ein analoger Eingang die gesamte Datenbreite eines *WORD* von 16 Bit.

Wie im Abschnitt *Steuer- und Regelsystem* erläutert, setzt die Steuerung die Ein- und Ausgabe analoger Werte mittels eines 12 Bit A/D-D/A-Wandlers um. Das bedeutet, jeder analoge Wertebereich (entweder 0..10VDC oder 0..20mA) wird auf insgesamt $2^{12}=4096$ Werte (von 0..4095) diskretisiert. Da kein Datentyp mit 12 Bit Datenbreite existiert, wird hierfür ein *UINT* bzw. *WORD* mit 16 Bit Datenbreite genutzt.

Die zehn analogen Eingänge des Erweiterungsmoduls werden also durch je ein *WORD* mit 16 Bit Datenbreite repräsentiert. Damit für die Adressierung in der globalen Variablenliste nur eine Startadresse angegeben werden muss, wurde die Adresse des ersten analogen Eingangs mittels Pointer als Startadresse eines *WORD*-Arrays verwendet. Dabei wurde die Größe dieses Arrays ebenfalls in der globalen Variablenliste als globale Konstante mit dem Namen *gc_byAnzAEE* deklariert und bereits weiter oben in diesem Abschnitt beschrieben.

Die Verwaltung der analogen Eingänge erfolgt also, indem ein Pointer auf ein *WORD*-Array mit *gc_byAnzAEE* Elementen die Adresse des ersten analogen Eingangs des Erweiterungsmoduls enthält.

Wie bereits für die digitalen Eingänge erfolgt auch hier die Speicherung sämtlicher Pointer der einzelnen Erweiterungsmodule in einem Array der Größe *gc_byAnzEM*, so dass die Verwaltung der analogen Eingänge der Erweiterungsmodule letztlich in einem Array von Pointern (Speicheradressen), die wiederum auf ein Array von *WORD*-Variablen zeigen, erfolgt.

Um nun den Wert eines bestimmten analogen Eingangs z. Bsp. in einer lokalen Variable mit dem Namen *wBeispiel* vom Typ *WORD* speichern zu können, muss auf folgende Weise dereferenzierend auf das entspr. Array zugegriffen werden:

```
wBeispiel:= apawAEE[1]^[2];
```

Hier wird die an der ersten Position des Pointer-Arrays hinterlegte Adresse zu *%IB1.0.1* dereferenziert und ab dieser Adresse auf das zweite Element des hinterlegten *WORD*-Arrays zugegriffen. Es erfolgt in diesem Beispiel also ein Zugriff auf den zweiten analogen Eingang des Erweiterungsmoduls, indem auf die Startadresse einfach der entspr. Offset eines Array-Elements von 16 Bit angerechnet und somit ab der Adresse *%IB1.0.2* als *WORD* gelesen wird.

Wie zu erkennen ist, gestaltet sich die Datenhaltung für die analogen Ein- und Ausgänge komplizierter als für die digitalen Ein- und Ausgänge. Das Abfragen und

Setzen der Werte für die analogen Ein- und Ausgänge ist jedoch wesentlich einfacher, da hier direkt auf *WORD*-Werte zugegriffen werden kann.

Nachdem nun die grundlegende Handhabung der analogen und digitalen Ein- und Ausgänge der Steuerung beispielhaft erläutert wurde, soll an dieser Stelle die Verknüpfung dieser mit den internen Repräsentationen der entspr. Elemente der Feldebene dargelegt werden.

4.2.4.3. Interne Repräsentation der Feldebene

Wie einleitend erwähnt, sollte diese Steuerung Mess- und Stellvorgänge ausführen können. Hierzu war es erforderlich, für die einzelnen Komponenten der Feldebene entspr. Variablen für den Zugriff anzulegen. Folgende Elemente der Feldebene sollten dabei mit der Steuerung verknüpft werden:

- Temperaturfühler für Außentemperatur

  Hierbei handelt es sich um ein an einen analogen Eingang angeschlossenes Element wie z. Bsp. ein Pt1000-Temperaturfühler.

- Temperaturfühler für Vorlauftemperatur

  Auch bei diesem Element handelt es sich um eine an einen analogen Eingang anzuschließende Komponente der Feldebene.

- Temperaturfühler für Rücklauftemperatur

  Wie die beiden vorherigen Temperaturfühler stellt auch dieser eine an einen analogen Eingang anzuschließende Komponente dar.

- Brenner des Heizkessels

  Für dieses Projekt vereinfachend angenommen handelt es sich hier um einen einstufigen Brenner, der insg. nur drei Zustände annehmen kann:

  1) Aus/Standby
  2) Betrieb
  3) Störung

  Es werden insg. also zwei digitale Eingänge für die Betriebsmeldung und die Störmeldung benötigt sowie ein digitaler Ausgang, um dem Brenner den Schaltbefehl für den Betrieb zukommen zu lassen.

- Vorlaufpumpe

  Auch bei der Vorlaufpumpe handelt es sich um ein Gerät, welches wie der Brenner drei Zustände annehmen kann:

  1) Aus/Standby
  2) Betrieb
  3) Störung

  Es werden also auch hier insg. zwei digitale Eingänge für die Betriebsmeldung und die Störmeldung sowie ein digitaler Ausgang für den Schaltbefehl benötigt.

- 3-Wege-Vorlaufmischer

  Der Stellmotor des Vorlaufmischers soll über ein stetiges Signal, also über einen analogen Ausgang betrieben werden. Das Stellsignal soll durch die Steuerung im diskreten Bereich zwischen 0..4095 berechnet und mittels analogem Ausgang als stetiges Signal zwischen 0..10VDC an den Stellmotor weitergeleitet werden.

4.2.4.4. Datentypen für die Elemente der Feldebene

Damit die programminterne Verwaltung der einzelnen an die entspr. Ein- und Ausgänge der Steuerung angeschlossenen Komponenten der Feldebene möglich war, wurden zunächst eigene Datentypen definiert. Dies erfolgte sowohl für analoge als auch für digitale Ein- und Ausgänge. Grundlegende Überlegung hierfür war, dass jeder dieser Datentypen einen Bezeichner verwalten sollte sowie Informationen darüber, welchem Modul der Steuerung und an welchem Ein- bzw. Ausgang des entspr. Moduls die jeweilige Variable dieses Typs betrieben wird.

Für die analogen und digitalen Eingänge ergab sich also folgende grundlegende Struktur:

```
TYPE EINGANG : (* Präfix: ein *)
STRUCT
   sBezeichnung: STRING[20];
   byModul: BYTE;
   byEingang: BYTE;
END_STRUCT
END_TYPE
```

Entsprechend diesem Beispiel wurde die Grundstruktur für die analogen und digitalen Ausgänge folgendermaßen definiert:

```
TYPE AUSGANG : (* Präfix: aus *)
STRUCT
   sBezeichnung: STRING[20];
   byModul: BYTE;
   byAusgang: BYTE;
END_STRUCT
END_TYPE
```

Bereits zu diesem Zeitpunkt ist zu erkennen, wie die spätere Abbildung einer Variable eines solchen Typs auf einen realen Ein- bzw. Ausgang erfolgen sollte. Die Strukturvariable *byModul* gibt in beiden Fällen die Nummer des Moduls an, an dem die entspr. Feldkomponente betrieben wird. Hierbei wird für die an den Can-Master angehängten Erweiterungsmodule mit 1 zu zählen begonnen, während der Wert 0 immer auf das Master Terminal verweist. Die Strukturvariablen *byEingang* bzw. *byAusgang* schließlich geben an, an welchem Anschluss des Moduls die entspr. Feldkomponente betrieben wird. Für eine frei wählbare Beschreibung der jeweiligen Komponente in der Visualisierung wurde die Strukturvariable *sBezeichnung* mit einer maximalen Länge von 20 Zeichen gewählt. Der Inhalt dieser *STRING*-Variablen sollte durch den Bediener der Steuerung während des Betriebs editierbar sein.

Der lesende Zugriff auf einen analogen Eingang der Steuerung könnte unter Einbeziehung aller bisher vorgestellten Möglichkeiten also auf folgende Weise realisiert werden:

```
(* Beispiel analoger Eingang *)
einBspAE: EINGANG;

(* Speicherplatz für Wert von einBspAE *)
wBspWert: WORD;

(* Initialisierung des Eingangs *)
einBspAE.sBezeichnung:= 'einBspAE';
einBspAE.byModul:= 1;
einBspAE.byEingang:= 1;

(* Auslesen des Wertes des ersten analogen Eingangs am
   ersten Erweiterungsmodul der Steuerung *)
wBspWert:= apawAEE[einBspAE.byModul]^[einBspAE.byEingang];
```

In diesem Beispiel wird, wie weiter oben bereits beschrieben, das erste Element des Arrays *apawAEE* als Adresse interpretiert, durch den Operator ^ dereferenziert und schließlich auf das erste Element des referenzierten *WORD*-Arrays zugegriffen. Der an dieser Stelle (Adresse *%IB1.0.1* + (*einBspAE.byEingang* - *1)* * 16 Bit = Adresse *%IB1.0.1*) hinterlegte Wert wird als *WORD* interpretiert und der Variablen *wBspWert* zugewiesen. Der aktuelle Stand des ersten analogen Eingangs des ersten Erweiterungsmodul kann so also ausgelesen werden.

Damit im weiteren Verlauf der Programmierung und vor allem auch beim späteren Entwurf der Visualisierung nicht jedes mal auf die oben dargelegte Weise auf die Werte der entspr. Eingänge zugegriffen werden musste, wurde die Erweiterung und Unterteilung der Strukturen erforderlich. So sollte während des Betriebs der Steuerung mit jedem Taskzyklus des Hauptprogramms eine Aktualisierung der analogen und digitalen Ein- und Ausgänge stattfinden. Die Speicherung dieser aktuellen Werte sollte schließlich in einer Variablen als Bestandteil der entspr. Struktur erfolgen, so dass eine weitere Verwendung dieser Werte z. Bsp. über einen einfachen Zugriff wie *wBspWert.wWert* erfolgen konnte. Spätestens ab diesem Zeitpunkt wurde es erforderlich, die Strukturen der Typdefinitionen in analoge und digitale Eingänge zu unterteilen. Einem digitalen Eingang genügte das Hinzufügen einer weiteren Variable zur Speicherung des entspr. aktuellen Zustands, so dass die Typdefinition für einen digitalen Eingang letztlich wie folgt aussah:

```
TYPE DIGITAL_EIN : (* Präfix: de *)
STRUCT
   sBezeichnung: STRING(20);
   byModul: BYTE;
   byEingang: BYTE;
   xZustand: BOOL;
END_STRUCT
END_TYPE
```

Die hinzugefügte *BOOL*-Variable mit dem Namen *xZustand* sollte somit der Speicherung des Wertes des Anschlusses *byEingang* am Modul *byModul* dienen und während des Betriebs regelmäßig aktualisiert werden.

Für die analogen Eingänge wurde eine Erweiterung der Struktur um zwei Variablen notwendig. Einerseits sollte der entspr. Wert während des Aktualisierungsvorgangs der Steuerung in einer dafür ausreichend genauern Variablen des Typs *WORD* gespeichert werden, andererseits wurde für die später notwendigen Berechnungen die Konvertierung und Speicherung dieses Wertes in einer Variablen des Typs *REAL* erforderlich. Die Typdefinition für analoge Eingänge sah also wie folgt aus:

```
TYPE ANALOG_EIN : (* Präfix: ae *)
STRUCT
   sBezeichnung: STRING[20];
   byModul: BYTE;
   byEingang: BYTE;
   wWert: WORD;
   rWert: REAL;
END_STRUCT
END_TYPE
```

Die Speicherung und Abfrage des aktuellen analogen Wertes am Eingang *byEingang* des Moduls *byModul* sollte analog zur Vorgehensweise bei den digitalen Eingängen erfolgen. Die einzige Ergänzung stellte hier die Variable *rWert* des Typs *REAL* dar, deren Wert ebenfalls während des Aktualisierungsvorgangs aus dem zuvor abgefragten Wert *wWert* konvertiert werden sollte.

Aufbauend auf diesen Implementierungen wurden schließlich die Strukturen für analoge und digitale Ausgänge entwickelt. Hierbei unterscheidet sich die Typdefinition für den digitalen Ausgang nicht wesentlich von der des digitalen Eingangs und wurde folgendermaßen umgesetzt:

```
TYPE DIGITAL_AUS : (* Präfix: da *)
STRUCT
   sBezeichnung: STRING[20];
   byModul: BYTE;
   byAusgang: BYTE;
   xZustand: BOOL;
END_STRUCT
END_TYPE
```

Diese Struktur wurde implementiert, um eine Differenzierung zwischen digitalen Ein- und Ausgängen zu erlangen, indem bei den entspr. Variablen dieser Typen das entspr. Präfix vorangestellt wird. Die Bezeichner der Variablen *dePumpe* und *daPumpe* z. Bsp. lassen die unterschiedliche Funktionalität bereits auf den ersten Blick im Quellcode erkennen. Zudem wurde damit erreicht, dass die Positionen im entspr. Array für einen Ausgang mit *daPumpe.byAusgang* und einen Eingang mit *dePumpe.byEingang* angegeben werden.

Im Gegensatz zur visuellen Differenzierung, welche bei den Strukturen für die digitalen Ein- und Ausgänge im Vordergrund stand, wurde für die Strukturen der analogen Ein- und Ausgänge auch eine funktionale Differenzierung erforderlich. Genügte es, für einen analogen Eingang die Speicherung eines erfassten Wertes in einer Strukturvariablen zu ermöglichen, musste die Struktur für analoge Ausgänge um zwei Variablen zur Erfassung der unteren und der oberen Grenze des möglichen Stellwertes ergänzt werden. Wie zuvor erwähnt, arbeitet die in diesem Projekt verwendete Steuerung mit einem 12 Bit-D/A-Wandler, der auf digitaler Seite einen diskreten

Wertebereich zwischen 0 und 4095 ermöglicht. Die Strukturvariable zur Speicherung des Sollwertes für den entspr. analogen Ausgang ist allerdings vom Typ *WORD* und könnte einen Wertebereich von *0* bis $2^{16} - 1 = 65535$ repräsentieren. Bei späteren Berechnungen für das Stellsignal an einem bestimmten analogen Ausgang würde demnach ein zu großer Wertebereich genutzt. Dies hätte zur Folge, dass am entspr. analogen Ausgang der Steuerung bereits nach ca. 6% des diskreten Stellbereichs 100% des stetigen Stellbereichs (z. Bsp. 10VDC) anliegen. Ein Ventil, welches gemäß interner Berechnung zu diesem Zeitpunkt erst zu rund 6% geöffnet sein sollte wäre tatsächlich also vollständig geöffnet (vgl. Formel 4.1).

*max. diskr. Stellber. / max. Wert * 100% = berechn. Stellsignal* *4095 / 65535 * 100% ≈ 6,25%*	(4.1)

Wie die Struktur für analoge Eingänge sollte auch die Struktur für analoge Ausgänge über eine Strukturvariable verfügen, die den Wert einer *REAL*-Berechnung zur Konvertierung in einen Wert des Typs *WORD* speichern kann. Im Rahmen des Aktualisierungsvorgangs sollte dieser ganzzahlige Wert des Typs *WORD* an den entspr. analogen Ausgang übermittelt werden. Zusätzlich zu den beiden o. g. wurde eine weitere Strukturvariable eingeführt, die das aktuelle Stellsignal als prozentualen Anteil des maximalen Wertes für ein Stellsignal speichert. Dies war erforderlich, da für den Bediener der Steuerung innerhalb der Visualisierung nicht der Wert für ein Stellsignal sondern vielmehr ein repräsentativer Wert für den aktuellen Stellstatus einer entspr. Komponente dargestellt werden sollte. Im Normalfall wird ein Bediener mit der Anzeige *3276* neben einem Vorlaufmischer nicht viel anzufangen wissen, während die Angabe *80%* schnelle Rückschlüsse auf die Klappenstellung zulässt. Die Struktur für einen analogen Ausgang wurde nach diesen Überlegungen also folgendermaßen implementiert:

```
TYPE ANALOG_AUS : (* Präfix: aa *)
STRUCT
   sBezeichnung: STRING[20];
   byModul: BYTE;
   byAusgang: BYTE;
   wMinY: WORD;
   wMaxY: WORD;
   wWert: WORD;
   rWert: REAL;
   rPWert: REAL;
END_STRUCT
END_TYPE
```

Wie soeben beschrieben, stellen die Strukturvariablen *wMinY* und *wMaxY* die untere und obere Grenze des verfügbaren diskreten Stellbereichs dar und die Strukturvariable *rPWert* den erläuterten prozentualen Anteil des momentanen Stellwertes *wWert* vom verfügbaren Stellbereich *wMaxY - wMinY*.

4.2.4.5. Anwendung der Datentypen für die Feldebene

Um die praktische Nutzung der hier vorgestellten Entwicklungen zu verdeutlichen, werden an dieser Stelle die Implementierungen für drei durch die Steuerung zu regelnde bzw. überwachende Komponenten exemplarisch aufgeführt.

4.2.4.5.1. Der Außentemperaturfühler

Im Deklarationsteil des Hauptprogramms *PLC_PRG* wurde eine Variable für den Außentemperaturfühler wie folgt deklariert:

```
(* analoger Eingang: Messwert Außentemperatur *)
aeAT: ANALOG_EIN;
```

Nach dem Initialisierungsvorgang stehen mit dieser Variable sämtliche Möglichkeiten zur Werterfassung und -verarbeitung eines von einem Temperaturfühler an die Steuerung gelieferten elektrischen Pendants[49] des Messwertes zur Verfügung. Während des ersten Startvorgangs der Steuerung erfolgt die Initialisierung dieser Variable mittels der zum Hauptprogramm gehörigen Aktion *AKT_INIT_AE* (im Klartext: *Aktion zur Initialisierung der analogen Eingänge*). Neben dieser Variablen werden mit dieser Aktion sämtliche übrige, analoge Eingänge repräsentierende Variablen sowie das für den Zugriff auf die analogen Eingänge zuständige Array *apawAEE[]* initialisiert. Folgende Initialisierung wurde im Rahmen dieses Projektes für den analogen Eingang zur Erfassung der Außentemperatur implementiert:

```
(* Initialisierung analoger Eingang: Messwert
   Außentemperatur *)
aeAT.sBezeichnung:= 'Außentemperatur';
aeAT.byModul:= 1;
aeAT.byEingang:= 1;
aeAT.rWert:= 0;
```

Standardmäßig wurde also der erste Eingang am ersten Erweiterungsmodul (*apawAEE[1]^[1]* ≙ *%IB1.0.1*) als Quelle für den Messwert der Außentemperatur festgelegt. Im Gegensatz zu der für spätere Berechnungen genutzten Variable *rWert* braucht die Variable *wWert* nicht mit einem Initialwert belegt zu werden, da diese im Verlauf der Aktualisierungsvorgänge zyklisch einen neuen, aktuellen Wert zugewiesen bekommt.

Die Aktualisierung der Komponenten der Variablen *aeAT* für die Außentemperatur sowie für alle übrigen Variablen für analoge Eingänge erfolgt mittels gesonderter, ebenfalls zum Hauptprogramm *PLC_PRG* gehöriger Aktion mit dem Namen *AKT_REF_AE* (im Klartext: *Aktion zur Aktualisierung*[50] *der analogen Eingänge*). Um die Aktualisierung während des Programmlaufs zu realisieren, wurde die Aktion *AKT_REF_AE* auf folgende Weise implementiert:

```
(* Aktualisierung analoger Eingang: Messwert
   Außentemperatur *)
aeAT.wWert:= apawAEE[aeAT.byModul]^[aeAT.byEingang];
```

[49] Wie in *Begriffe der Mess-, Steuerungs- und Regelungstechnik* und *Abschließende Projektkonfiguration* beschrieben, liefert auch ein Temperaturfühler keine Werte in °C sondern, bedingt durch sein temperaturabhängiges Widerstandsverhalten, einen der momentanen Temperatur entspr. Spannungswert.

[50] Refresh: Daher die Abkürzung *REF*, da die Abkürzung *AKT* bereits für den Hinweis auf eine Aktion benutzt wird.

```
IF gc_xTestMode THEN
   aeAT.rWert:= FUN_ROUND(
      FUN_NORM_VALUE(
         WORD_TO_REAL(aeAT.wWert),
         SINT_TO_INT(AnlagenDaten.siMinAT - 5),
         BYTE_TO_INT(AnlagenDaten.byMaxAT + 5),
         12),
      2);
ELSE
   aeAT.rWert:= INT_TO_REAL(WORD_TO_INT(aeAT.wWert)) / 10;
END_IF;
```

Die erste Anweisungszeile wurde bisher ausreichend erläutert, so dass es genügt zu erwähnen, dass hier die Zuweisung des aktuellen Referenzwertes für die aktuelle Außentemperatur an die Variable *wWert* erfolgt.

Neu hingegen sind die meisten Bestandteile der folgenden *IF*-Klausel. Die Eintrittsbedingung *gc_xTestMode* stellt eine globale Konstante des Typs *BOOL* dar, die der Angabe dient, ob die Steuerung im Präsentations- bzw. Demonstrationsmodus betrieben wird. Diese Implementierung war notwendig, da sowohl für die lokale Programmierung als auch für die weitere von kompletten Heizanlagen losgelöste Nutzung des Simulationskompaktgerätes diverse analoge und digitale Eingänge über Drehregler und Kippschalter bedient werden sollten. Deklariert und initialisiert wurde die globale Konstante *gc_xTestMode* in der globalen Variablenliste, um schnell zwischen den Betriebsmodi (1 = Testmodus, 0 = Anlagenmodus) umschalten zu können.

Zusätzlich zu dieser neuen Konstante wurden in diesem Zusammenhang zwei neue Funktionen implementiert, die der Wertemanipulation dienen. Zum einen handelt es sich hierbei um die Funktion *FUN_ROUND*, welche der Rundung einer Zahl des Typs *REAL* auf eine bestimmte Anzahl Stellen hinter dem Komma dient. Hierzu muss diese Funktion eine Zahl des Typs *REAL* (*rInitValue*) und die Anzahl signifikanter Stellen hinter dem Komma (*bySigPos*), auf die gerundet werden soll, übergeben bekommen. Der Wert der übergebenen Zahl wird mit der *bySigPos*-Zehnerpotenz multipliziert, in einen Wert des Typs *INT* umgewandelt (wobei auf jeden Fall die erwünschten Nachkommastellen erhalten bleiben) und nach der Rückkonvertierung in eine Zahl des Typs *REAL* wieder durch die *bySigPos*-Zehnerpotenz dividiert. Dieser Wert wird schließlich als Funktionsergebnis zurück gegeben. Als Beispiel sei hier die Rundung der Zahl *17.48732* auf zwei Nachkommastellen aufgeführt:

$17.48732 * 10^{bySigPos} = 17.48732 * 10^{2} = 1748.732$
REAL_TO_INT(1748.732) = 1749
INT_TO_REAL(1749) = 1749.000…
$1749.000\ldots / 10^{bySigPos} = 1749 / 10^{2} = \underline{\underline{17.49}}$

Von Bedeutung ist in diesem Zusammenhang die Tatsache, das bei der Typkonvertierung von *REAL* zu *INT* korrekt gerundet wird und die Stellen nach dem Komma somit ebenfalls korrekt gerundet ausgegeben werden.

Die zweite neue Funktion mit dem Namen *FUN_NORM_VALUE* innerhalb der hier behandelten Aktualisierungsfunktion dient der Anpassung eines Ursprungswertes an einen bestimmten neuen Wertebereich. Diese Funktion wurde nur für den Testmodus

implementiert, um den durch den manuellen Drehregler für den analogen Eingang darstellbaren Wertebereich auf einen geeigneten Temperaturwertebereich abzubilden, also eine Wertenormierung durchzuführen.

Wie im Abschnitt *Abschließende Projektkonfiguration* erläutert, wurden die analogen Eingänge für den Demonstrationsmodus auf den Spannungsbetrieb konfiguriert. Für den Betrieb eines Temperaturfühlers müsste diese Einstellung in der Steuerungskonfiguration entspr. Tabelle 2 angepasst werden. Die durch die Drehregler an den analogen Eingängen erzeugbaren Werte lagen demnach also in einem Bereich zwischen 0 und 4095. Da über diese Eingänge Temperaturwerte sowohl für die hier erläuterte Außentemperatur als auch für die Vorlauf- und Rücklauftemperatur simuliert werden sollten, wurde es erforderlich, den o. g. Wertebereich auf einen angemessenen Zielbereich abzubilden. Für die hier implementierte Regelung sollte die Auslegungstemperatur der Anlage auf den Bereich zwischen -20°C und 20°C festgelegt werden. Die Abbildung des ursprünglichen Wertebereichs in den erwünschten Zielbereich wurde daher wie folgt umgesetzt:

- Berechnung der neuen Darstellungsauflösung
- Division des aktuellen Wertes in Ursprungsdarstellung durch die neue Auflösung
- Addition des Mindestwertes für die neue Darstellung

Für die Berechnung der neuen Darstellungsauflösung wurde die Funktion *FUN_CALC_RES* implementiert. Diese erwartet als Funktionsparameter die Übergabe eines Wertes für eine obere (*iUpperLimit*) und eine untere Schranke (*iLowerLimit*) des neuen Wertebereichs sowie die Datenbreite des ursprünglichen Wertebereichs in Bit (*byBitRes*). Die Auflösung des neuen Wertebereichs wird anhand dieser Vorgaben folgendermaßen errechnet:

$$\frac{2^{byBitRes}-1}{(iUpperLimit-iLowerLimit)} \qquad (4.2)$$

Zur Verdeutlichung soll an dieser Stelle die Darstellungsauflösung des bereits erwähnten Temperaturbereichs von -20°C bis 20°C unter Nutzung der Datenbreite in Bit des ursprünglichen Darstellungsbereichs berechnet werden. Für den analogen Eingang steht ein Wertebereich mit einer Datenbreite von 12 Bit mit einem maximal darstellbaren Wert von $2^{12} - 1 = 4095$ zur Verfügung. Die Berechnung der neuen Darstellungsauflösung sieht demnach wie folgt aus:

$byBitRes = 12$
$2^{byBitRes} - 1 = 2^{12} - 1 = 4095$
$(iUpperLimit - iLowerLimit = (20°C - (-20°C)) = 40°C$
$4095 / 40°C = \underline{\underline{102.375/°C}}$

Das Ergebnis zeigt, dass jede im Zielwertebereich darzustellende Einheit (in diesem Fall °C) 102.375 Einheiten im ursprünglichen Wertebereich entsprechen.

Um aus dem aktuellen Messwert den korrekten Zielwert zu berechnen, muss zuerst errechnet werden, wie viele Einheiten dieser aktuelle Messwert im Zielsystem darstellt. Dies geschieht mittels Division des Ursprungswertes (folgend als Beispiel 952 angenommen) durch die neue Darstellungsauflösung:

$952 / 102.375/°C \approx \underline{\underline{9.3°C}}$

Dieses Ergebnis wäre bereits der korrekte Wert, sofern der neue Wertebereich zwischen 0 und 40 liegen würde, da mit der obigen Bestimmung des neuen Wertebereichs die Wertebereichsspanne von 40 Einheiten errechnet wurde. Die Aussage darüber, in welchem Zahlenbereich sich diese Einheiten befinden bleibt während dieser Berechnung jedoch nicht erhalten. Aus diesem Grund muss für die korrekte Wertdarstellung eine Korrekturaddition um die untere Schranke des Zielwertebereichs erfolgen. In diesem Beispiel würde die Berechnung des korrekten Zielwertes folgendermaßen durchgeführt werden:

```
9.3°C + (-20) = -10.7°C
```

Der ursprüngliche durch den analogen Eingang präsentierte Wert von 952 entspräche in der Zieldarstellung einem Wert von -10.7°C.

Die zu Beginn dieser Ausführungen erwähnte Funktion *FUN_NORM_VALUE* liefert also einen von einem analogen Eingangswert abhängigen, auf eine nutzerdefinierte Skala normierten Zielwert.

Entsprechend dieser Erläuterungen erfolgt die Aktualisierung des zuvor als *WORD* interpretierten Wertes des analogen Eingangs als in den Zielbereich abgebildete Zahl des Typs *REAL*. Wichtig ist, dass dies nur während des Betriebs der Steuerung zu Demonstrationszwecken erfolgt.

Befindet sich die Steuerung im Anlagenbetrieb, ist also die globale Konstante *gc_xTestMode* auf *False* bzw. *0* gesetzt, wird die Berechnung des aktuellen Temperaturwertes auf einfachere Weise wie folgt ausgeführt:

```
aeAT.rWert:= INT_TO_REAL(WORD_TO_INT(aeAT.wWert)) / 10;
```

Wie im Abschnitt *Abschließende Projektkonfiguration* erläutert, wird der Wert des erfassten Messwerts in diesem Fall als Zahl des Typs *INT* interpretiert, in eine *REAL*-Zahl konvertiert und gem. der Herstellervorgaben durch zehn dividiert.

4.2.4.5.2. Der Brenner

Um die Funktionen des Brenners zu steuern und zu überwachen war es erforderlich, insgesamt drei Variablen im Deklarationsteil des Hauptprogramms *PLC_PRG* zu deklarieren:

```
(* digitaler Eingang: Betriebsmeldung Brenner *)
deBMB: DIGITAL_EIN;

(* digitaler Eingang: Störmeldung Brenner *)
deSMB: DIGITAL_EIN;

(* digitaler Ausgang: Schaltbefehl Brenner *)
daSBB: DIGITAL_AUS;
```

Mit diesen zwei digitalen Eingängen und dem digitalen Ausgang konnte die Steuerung des Brenners gemäß den Anforderungen realisiert werden. Wie bei dem bereits erläuterten analogen Eingang erfolgt auch für die digitalen Eingänge die Initialisierung während des Startvorgangs durch die dem Hauptprogramm zugehörige Aktion *AKT_INIT_DE*. Die Implementierung hierfür wurde folgendermaßen vorgenommen:

```
(* Zuweisung physikalische Adresse für digiale
   Eingänge am Masterterminal *)
pbyDEM:= ADR(g_byAdrDEM);

(* Zuweisung physikalische Adresse für digitale
   EINGÄNGE am ERSTEN Erweiterungsmodul*)
apwDEE[1]:= ADR(g_wAdrDEE_1);

(* Initialisierung digitaler Eingang: Betriebsmeldung
   Brenner *)
deBMB.sBezeichnung:= 'Betrieb Brenner';
deBMB.byModul:= 0;
deBMB.byEingang:= 0;

(* Initialisierung digitaler Eingang: Störmeldung
   Brenner *)
deSMB.sBezeichnung:= 'Störung Brenner';
deSMB.byModul:= 0;
deSMB.byEingang:= 1;
```

Entscheidend hierbei ist, dass die Datenhaltung für die digitalen Eingänge des Master Terminals und die digitalen Eingänge der Erweiterungsmodule getrennt erfolgt. Erforderlich wurde diese Maßnahme, da das hier in der Version 2.3.9.32 genutzte CoDeSys-Programmiersystem die Verwendung dynamischer Arrays nicht ermöglicht. Das hier genutzte Erweiterungsmodul verfügt mit 16 über mehr digitale Eingänge als das Master Terminal mit vier digitalen Eingängen. Wie bereits erläutert, erfolgt die Datenhaltung für die 16 digitalen Eingänge des Erweiterungsmoduls mittels einer Variablen des Typs *WORD*. Dies würde bei einer Adressierung der vier digitalen Eingänge am Master Terminal jedoch fehlerhafte Werte erzeugen. Da die digitalen Eingänge des Master Terminals steuerungsintern als *BYTE* (8 Bit) verwaltet werden, würde eine 16-Bit-Leseoperation ab der Startadresse des ersten digitalen Eingangs undefinierte Werte in den übrigen 8 Bit der *WORD*-Variablen liefern.

Die Startadresse für die digitalen Eingänge am Master Terminal wird somit in einer Pointer-Variable des Typs *BYTE* mit dem Namen *pbyDEM* hinterlegt, während die Adressen der digitalen Eingänge an den Erweiterungsmodulen auf die bereits bekannte Art mittels Array verwaltet werden.

Im obigen Beispiel wurde mit der Variablen *deBMB* die Verwaltung der Betriebsmeldung des Brenners und mit der Variablen *deSMB* die Verwaltung der Störmeldung des Brenners ermöglicht. Die Aktualisierung der Komponentenwerte dieser beiden Variablen erfolgt während des Aktualisierungsprozesses durch die dem Hauptprogramm zugeordnete Aktion *AKT_REF_DE*. Die Implementierung soll an dieser Stelle am Beispiel der Variablen *deBMB* verdeutlicht werden:

```
(* Aktualisierung digitaler Eingang: Betriebsmeldung
   Brenner *)
IF gc_xTestMode THEN
   deBMB.xZustand:= daSBB.xZustand;
ELSE
   IF deBMB.byModul = 0 THEN
      deBMB.xZustand:= FUN_GET_DE(deBMB.byEingang,
         WORD_TO_DWORD(pbyDEM^));
```

```
    ELSE
       deBMB.xZustand:= FUN_GET_DE(deBMB.byEingang,
          WORD_TO_DWORD(apwDEE[deBMB.byModul]^));
    END_IF;
END_IF;
```

Grundlegend erfolgt hierbei eine Überprüfung, ob sich die Steuerung im Demonstrations- oder normalen Betriebsmodus befindet. Im Falle des Betriebes im Demonstrationsmodus' wird der aktuelle Status des Schaltbefehls des Brenners (Variable *daSBB*, wird folgend erläutert) direkt auf den Status der Variablen für die Betriebsmeldung des Brenners (*deBMB*) abgebildet. Dies war erforderlich, da in diesem Betriebsmodus keine tatsächliche Rückmeldung an dem entspr. digitalen Eingang durch den Brenner anlag. Im Gegensatz dazu erfolgte bei der Störmeldung (Variable *deSMB*) keine automatische Rückkopplung dieser Art, da diese manuell mittels der vorhandenen Kippschalter des Simulationskompaktgerät auslösbar sein sollte.

Sofern sich die Steuerung im realen Betriebsmodus befindet, musste für die Aktualisierung eine Unterscheidung zwischen dem Master Terminal und den Erweiterungsmodulen erfolgen. Sofern sich der Eingang für die entspr. Betriebsmeldung am Master Terminal (Modul 0) befindet, sollte der aktuelle Status aus dem zuständigen *BYTE*-Speicherbereich (in diesem Fall referenziert durch den Pointer *pbyDEM*) ausgelesen werden. Andernfalls sollte die Zustandsaktualisierung durch Auslesen des Zustandes des Eingangs am entspr. konfigurierten Erweiterungsmodul erfolgen.

Mittels dieser beiden Variablen wurde das erfassen des Brennerstatus' (Betriebs- oder Störmeldung) ermöglicht, so dass noch die Implementierung des Schaltbefehls, um den Brenner ein- oder auszuschalten erforderlich war. Dies erfolgte, indem die Variable *daSBB* als digitaler Ausgang mit der dem Hauptprogramm *PLC_PRG* angehörigen Aktion *AKT_REF_DA* wie folgt aktualisiert wurde:

```
(* Aktualisierung digitaler Ausgang: Schaltbefehl
   Brenner *)
IF daSBB.byModul = 0 THEN
   pbyDAM^:= DWORD_TO_BYTE(FUN_SET_DA(daSBB.byAusgang,
      daSBB.xZustand, BYTE_TO_DWORD(pbyDAM^)));
ELSE
   apwDAE[daSBB.byModul]^:= DWORD_TO_WORD(FUN_SET_DA(
      daSBB.byAusgang, daSBB.xZustand,
      WORD_TO_DWORD(apwDAE[daSBB.byModul]^)));
END_IF;
```

Auch in diesem Fall ist wieder die Unterscheidung zwischen dem Zugriff auf die digitalen Ausgänge des Master Terminals und der Erweiterungsmodule zu erkennen. Ein entspr. digitaler Ausgang kann somit einfach aus dem Programm heraus aktiviert oder deaktiviert werden, indem der Zustand der entspr. Variable des Typs *DIGITAL_AUS* geändert wird. Im Falle des Schaltbefehls für den Brenner würde durch den Aufruf *daSBB.xZustand:= TRUE* der in dieser Variablen hinterlegte Ausgang am zugehörigen Modul im Rahmen der Aktualisierungsphase aktiviert werden.

4.2.4.5.3. Der Vorlaufmischer

Um den Vorlaufmischer mit einem stetigen Signal ansteuern zu können wurde die Variable *aaSWVLM* des Typs *ANALOG_AUS* im Deklarationsteil des Hauptprogramms *PLC_PRG* angelegt. Der Zugriff auf den auszugebenden Stellwert sollte aus dem Programm heraus über die Komponentenvariable *aaSWVLM.rWert* erfolgen, indem berechnete Sollwerte dieser Variablen des Typs *REAL* zugewiesen werden. Nachdem der entspr. prozentuale Wert für den momentan an diese Komponentenvariable zugewiesenen Wert errechnet und der Komponentenvariable aaSWVLM.rPWert zugewiesen wurde, sollte der aktuelle Sollwert dem entspr. dereferenzierten Element des Pointer-Arrays für die analogen Ausgänge zugewiesen werden. Für die Realisierung dieser Anforderung wurde folgende Implementierung umgesetzt:

```
(* Aktualisierung analoger Ausgang: Sollwert
   Vorlaufmischer *)
aaSWVLM.wWert:= REAL_TO_WORD(aaSWVLM.rWert);
aaSWVLM.rPWert:= FUN_ROUND(aaSWVLM.rWert /
   WORD_TO_REAL(aaSWVLM.wMaxY - aaSWVLM.wMinY) * 100, 2);

apawAAE[aaSWVLM.byModul]^[aaSWVLM.byAusgang]:= aaSWVLM.wWert;
```

Hier erfolgt zuerst die Konvertierung des in der Komponentenvariable gespeicherten *REAL*-Wertes des Stellsignals in einen Wert des Typs *WORD* und schließlich die Zuweisung an die Komponentenvariable *aaSWVLM.wWert*. Schließlich wird der prozentuale Anteil des in der Komponentenvariable *aaSWVLM.rWert* anliegenden Stellwerts vom maximalen Stellwert errechnet und in der Komponentenvariable *aaSWVLM.rPWert* gespeichert. Der maximale Stellwert errechnet sich hierbei aus der Differenz der oberen und der unteren Stellwertgrenze. Dies ist z. Bsp. dann erforderlich, wenn ein Stellbereich mittels Minimalwert auf ein unteres Niveau begrenzt wurde. Im Rahmen der letzten Zuweisung dieses Aktualisierungsvorgangs wird der dem Stellwert entsprechende *WORD*-Wert an die der Komponentenvariable *aaSWVLM.byAusgang* entsprechende Array-Position an der dereferenzierten Adresse der Steuerung geschrieben.

4.2.4.6. Funktionsbausteine für die Regelungsrealisierung

Damit mittels all dieser internen Repräsentationen analoger und digitaler Ein- und Ausgänge die gewünschte Regelung realisiert werden konnte, war noch die Implementierung folgender Datentypen und Funktionsblöcke erforderlich:

- Datentyp zur Speicherung globaler Anlagendaten
- Funktionsbaustein für Heizfreigabe
- Funktionsbaustein für Störungsüberprüfung
- Funktionsbaustein für Anlagenfreigabe
- Funktionsbaustein für einen PI-Regler
- Funktionsbaustein für Brennersteuerung
- Funktionsbaustein für Pumpensteuerung

4.2.4.6.1. Zentrale Datenhaltung

Zur Speicherung globaler Anlagendaten wurde für dieses Projekt ein gesonderter Datentyp mit dem Namen *ANL_DAT* angelegt. Dies hatte den Vorteil, dass sämtliche für die Konfiguration und den Betrieb der Steuerung ausschlaggebenden Daten zentral über eine im Hauptprogramm *PLC_PRG* mit dem Namen *AnlagenDaten* deklarierte Variable dieses Typs zugreifbar waren. Folgende Anlagendaten werden dabei zentral über die Variable dieses Typs verwaltet:

- `byHG: BYTE;`

  Hierbei handelt es sich um eine Komponentenvariable, in der der Wert für die Heizgrenze der Anlage hinterlegt wird. Oberhalb dieses Wertes schaltet die Steuerung den Heizbetrieb aus und erst bei Absinken der Außentemperatur unter diesen Wert wieder an. Diese Variable nimmt Werte zwischen *0…255* der Einheit *°C* auf und wird während der Systeminitialisierung mit *20* initialisiert.

- `byTolHG: BYTE;`

  Diese Variable enthält die Toleranz für die beidseitige Hysterese der Heizgrenze. Dieser Wert wird sowohl auf den Wert der Variablen *byHG* addiert als auch von ihm subtrahiert, um die obere und die untere Schranke für die Schalthysterese zu bestimmen. Diese Variable kann Werte im Bereich *0…255* der Einheit *°C* annehmen und wird mit dem Wert *2* initialisiert.

- `siMinAT: SINT;`

  Diese Variable gibt die minimale Auslegungstemperatur der Anlage an und kann Werte zwischen *-128…127* der Einheit *°C* annehmen. Die Initialisierung erfolgt mit *-20*.

- `byMaxAT: BYTE;`

  Mit dieser Komponentenvariable wird der maximale Wert der Auslegungstemperatur der Anlage für die Verwendung gespeichert. Im Gegensatz zur Variable für die minimale Außentemperatur können hier nur Werte im Bereich *0…255* mit der Einheit *°C* erfasst werden. Initialisiert wird diese Variable mit *20*;

- `byMinVLT, byMaxVLT: BYTE;`

  Zur Speicherung der minimalen und maximalen Werte des Regelbereiches wurden diese beiden Variablen angelegt. Sie können jeweils Werte im Bereich *0…255* der Einheit *°C* annehmen. Die Initialisierung erfolgt mit *byMinVLT:= 20* und *byMaxVLT:= 75*.

- `byMinRLT, byMaxRLT: BYTE;`

  Wie auch für den Vorlauf wurden für den Rücklauf zwei Variablen angelegt. Diese kommen im Rahmen dieser Entwicklung außer zu Visualisierungszwecken nicht zum Einsatz, werden aber für den evtl. Fall der nicht linearen Heizkennlinienberechnung benötigt. Auch hier können Werte im Bereich *0…255* der Einheit *°C* gespeichert werden. Während des Initialisierungsvorgangs werden den Variablen die Werte *byMinRLT:= 20* und *byMaxRLT:= 65* zugewiesen.

- `siPV: BYTE;`

  In dieser Variablen erfolgt die Speicherung des Wertes für die Parallelverschiebung der Heizkurve. Dieser Wert beeinflusst die Sollwertberechnung der Vorlauftemperatur unmittelbar und dauerhaft, da eine Programmregelung im Rahmen dieses Projektes nicht umgesetzt wurde. Der Wertebereich dieser Variablen ist *-128…127* der Einheit *°C*. Die Initialisierung erfolgt mit *0*.

- `rZeitNLB: REAL;`

  Der Wert dieser Variablen gibt die Nachlaufzeit des Brenners in Sekunden an und wird mit *10* initialisiert. Die Nachlaufzeit ist definiert als die Zeit, die der Brenner noch im Zustand *Betrieb* bleibt, nachdem die Berechnungen der Regelung die Deaktivierung des Heizbetriebs ergeben haben. Diese Variable kann sämtliche Werte aus dem durch 32 Bit darstellbaren reellen Zahlenbereich annehmen.

- `rZeitNLP: REAL;`

  Diese mit dem Wert *10* initialisierte Variable repräsentiert die Nachlaufzeit der Vorlaufpumpe in Sekunden. Der Wertebereich ist mit dem der Variable *rZeitNLB* identisch.

- `bySWRT: BYTE;`

  Als Bestandteil zur Berechnung der Heizkennlinie dient diese Komponentenvariable zur Speicherung des Sollwertes der durchschnittlichen Raumtemperatur. Initialisiert wird diese Variable mit dem Wert *20* und kann Werte im Bereich *0…255* der Einheit *°C* annehmen.

- `rKp: REAL;`

  Diese Variable enthält den Wert für den Übertragungsbeiwert *KP* des P-Anteils des PI-Reglers, der aus dem errechneten Sollwert der Vorlauftemperatur einen Wert für das Stellsignal der Vorlaufpumpe errechnet. Die Angabe diese Proportionalitätswertes erfolgt dimensionslos im Bereich der durch 32 Bit darstellbaren reellen Zahlen. Die Initialisierung erfolgt mit dem Wert *5*.

- `rTa: REAL;`

  In dieser Variable wird der Wert für das Abtastintervall des I-Anteils des PI-Reglers in Sekunden gespeichert. Der zulässige Wertebereich erstreckt sich auch hier über die durch 32 Bit darstellbaren reellen Zahlen. Initialisiert wird diese Variable mit *0.02*, was 20 Millisekunden entspricht.

- `rTn: REAL;`

  Zur Speicherung der Nachstellzeit des I-Anteils des PI-Reglers wurde diese Variable angelegt und während der Initialisierung mit dem Wert *3* initialisiert. Auch hier sind Werte im Bereich der durch 32 Bit darstellbare reellen Zahlen der Einheit Sekunden zulässig.

- `bySWVLT: REAL;`

  Diese letzte Komponentenvariable dient der Speicherung des mittels der Heizkennlinie ermittelten Wertes für die Vorlauftemperatur in °C.

4.2.4.6.2. Regelung der Heizfreigabe

Damit die Regelung überhaupt den Betrieb der Anlage zulässt, wurde ein gesonderter Funktionsblock zur Überprüfung der Heizfreigabe implementiert. Dieser überprüft anhand dreier übergebener Werte mittels beidseitiger Hysterese, ob der Anlage grundsätzlich der Heizbetrieb ermöglicht wird.

Hierzu wurde im Hauptprogramm *PLC_PRG* mit dem Namen *FreigHzg* eine Instanz dieses Funktionsbausteins angelegt. Dessen Eingabeparameter *rAT*, *bHG* und *bTolHG* die aktuellen Werte der Außentemperatur, der Heizgrenze und der Toleranz für die Hysterese der Heizgrenze übergeben bekommen. Hierbei erfolgt der Aufruf aus dem Hauptprogramm wie folgt:

```
(* Überprüfen, ob aktuelle Außentemperatur Heizbetrieb erfordert *)
FreigHzg(
   rAT:= aeAT.rWert,
   bHG:= AnlagenDaten.byHG,
   bTolHG:= AnlagenDaten.byTolHG);
```

Spätestens an dieser Stelle ist der Vorteil der oben erläuterten zentralen Datenhaltung zu erkennen. Zum einen werden hier der aktuelle Messwert der Außentemperatur aus der Komponentenvariable *rWert* der analogen Eingangsvariable *rAT* und zum anderen die Werte für die Heizgrenze und die zugehörige Toleranz aus der zentralen Variable *AnlagenDaten* übergeben.

Die Berechnung der Heizfreigabe unter Einbeziehung einer Hysteresefunktion wurde dabei folgendermaßen umgesetzt:

```
(* Überprüfung, Heizgrenze unterschritten *)
IF rAT < (bHG - bTolHG)
   THEN xFreigHzg := TRUE;
   (* Überprüfung, ob Heizgrenze überschritten *)
   ELSIF rAT > (bHG + bTolHG)
      THEN xFreigHzg := FALSE;
END_IF;
```

Hierbei wird zunächst überprüft, ob die Heizgrenze abzüglich der Toleranz unterschritten wurde und eine Freigabe für den Heizbetrieb erfolgen kann. Ist dies nicht zutreffend, erfolgt eine Überprüfung, ob die Heizgrenze zuzüglich der Toleranz überschritten wurde und der Heizbetrieb somit eingestellt werden soll. Es gibt also einen sogenannten *Totbereich* zwischen *bHG - bTolHG* und *bHG + bTolHG*, in dem keine Änderung des aktuellen Status des Heizbetriebes erfolgt.

Dem Ausgabeparameter *xFreigHzg*, bei der Instanziierung mit *FALSE* initialisiert, wird schließlich entweder ein boolescher Wert dem Freigabestatus entsprechend zugewiesen oder er behält den der Berechnung des vorausgegangenen Zyklus' entsprechenden Wert bei.

4.2.4.6.3. Überprüfung auf Störung der Anlage

Damit die Regelung im Falle der Störung einer Anlagenkomponente den Betrieb zur Sicherheit unterbrechen kann, wurde der Funktionsbaustein *FB_STOER_ANL* angelegt und im Hauptprogramm mit dem Namen *StoerAnl* instanziiert.

Dieser Funktionsbaustein stellt fünf schließende und fünf öffnende boolesche Eingabeparameter zur Verfügung und speichert eine Aussage darüber, ob eine

Funktionsstörung der Anlage vorliegt als booleschen Wert im Ausgabeparameter mit dem Namen *xStoerAnl*.

Schließende Eingabeparameter werden mit *FALSE* initialisiert und lösen dann eine Anlagenstörung aus, wenn sie den Wert *TRUE* annehmen, das meldende Element also einen Fehler meldet. Ein Beispiel ist die Störmeldung des Brenners, deren Wert in der Komponentenvariable *xZustand* der digitalen Eingangsvariablen *deSMB* gespeichert wird. Liegt eine Brennerstörung an, erfolgt ein Kontaktschluss, der den Wert auf *TRUE* setzt und diesen als aktive Störung propagiert.

Ein öffnender Eingabeparameter wird mit *TRUE* initialisiert und löst eine Anlagenstörung aus, sobald er den Wert *FALSE* annimmt. Die Betriebsmeldung des Brenners z. Bsp. könnte als öffnender Eingabeparameter angesehen werden, sofern der Brenner im Dauerbetrieb laufen würde. Sobald der Dauerbetrieb, symbolisiert durch den Wert *TRUE* der Komponentenvariable xZustand der digitalen Eingangsvariable *deBMB*, unterbrochen wird, wird der Wert *FALSE* entspr. propagiert.

Der Wert für den Ausgabeparameter *xStoerAnl* wird dabei aus einer logischen ODER-Verknüpfung der schließenden und der negierten Werte der öffnenden Eingabeparameter gebildet.

4.2.4.6.4. Freigabe des Anlagenbetriebs

Für die letztliche Freigabe der Anlage und damit verbunden die Freigabe der Regelung wurde der Funktionsblock *FB_FREIG_ANL* implementiert. Dieser verknüpft mit seinen Eingabeparametern *xFreigHzg* und *xStoerAnl* die Zustandsmeldungen der beiden vorherigen Bausteine (*FreigHzg.xZustand* und den negierten Wert von *StoerAnl.xZustand*) zu einem booleschen Wert, der im Ausgabeparameter *xFreigAnl* gespeichert wird. Instanziiert wird dieser Funktionsbaustein im Hauptprogramm *PLC_PRG* unter dem Namen *FreigAnl*.

Nur, wenn der Wert von *FreigAnl.xFreigAnl TRUE* ist, kann die Steuerung arbeiten und die Anlage mittels der folgend erläuterten Funktionen und Funktionsbausteine regeln.

4.2.4.6.5. Berechnung der Vorlauftemperatur

Zur Berechnung der Vorlauftemperatur und der Speicherung des Wertes in *AnlagenDaten.bySWVLT* wurde die Funktion *FUN_CALC_CLT* implementiert.

Diese Funktion erwartet Werte für ihre Eingabevariablen *byMinVLT*, *byMaxVLT*, *siMinAT*, *byMaxAT*, *siPV*, *bySWRT*, *rIWAT*, die der Berechnung der Heizkennlinie entspr. den Erläuterungen des Abschnitts *Die Heizkennlinie* des Kapitels *Grundlagen der Mess-, Steuerungs- und Regelungstechnik* dienen. Es handelt sich hierbei um die Werte für die minimale und die maximale Vorlauftemperatur (Regelbereich), die minimale und die maximale Außentemperatur (Auslegungsbereich), die Parallelverschiebung, den Sollwert für die durchschnittlich zu erreichende Raumtemperatur und den aktuellen Messwert der Außentemperatur.

Auch in diesem Zusammenhang verdeutlicht sich erneut der Vorteil der zentralen Datenhaltung, da bis auf den Wert der aktuellen Außentemperatur sämtliche Werte über die Variable *AnlagenDaten* im Hauptprogramm abrufbar sind.

Die Berechnung des Sollwertes für die Vorlauftemperatur wurde im Rahmen dieses Projekts folgendermaßen implementiert:

```
(* Berechnung der Steigung der Heizkurve *)
rSteilheit:= (BYTE_TO_REAL(byMaxVLT) - BYTE_TO_REAL(byMinVLT)) /
   (BYTE_TO_REAL(byMaxAT) - SINT_TO_REAL(siMinAT));

(* Berechnung des Sollwertes für die VLT *)
rVLT:= BYTE_TO_REAL(byMinVLT) + (BYTE_TO_REAL(bySWRT) - rIWAT) *
   rSteilheit + siPV;

(* Überprüfung, ob berechnete VLT über Regelbereich liegt *)
IF rVLT > byMaxVLT THEN
   FUN_CALC_VLT:= byMaxVLT;
ELSE
   (* Überprüfung, ob berechnete VLT unter Regelbereich liegt *)
   IF rVLT < byMinVLT THEN
      FUN_CALC_VLT:= byMinVLT;
   ELSE
   (* Zuweisung des berechneten VLT-Wertes, wenn innerhalb des
      Regelbereichs *)
      FUN_CALC_VLT:= rVLT;
   END_IF;
END_IF;
```

Aufbauend auf den Erläuterung aus dem Abschnitt *Die Heizkennlinie* erfolgt hier zunächst die Berechnung der Steilheit der Heizkennlinie und im Anschluss daran die Berechnung des Sollwertes für die Vorlauftemperatur. Die folgenden Überprüfungen stellen sicher, dass bei einer berechneten Vorlauftemperatur außerhalb des Regelbereiches (unterhalb $Min(\vartheta_{VLT})$ oder oberhalb $Max(\vartheta_{VLT})$) der Funktionswert auf den entspr. Extremwert gesetzt wird. Bewegt sich das Ergebnis im Rahmen des Regelbereichs, entspricht der Funktionswert dem berechneten Wert.

Im Hautprptogramm wird der Funktionswert dieser Berechnung schließlich der Komponenetenvariablen *bySWVLT* der Variablen *AnlagenDaten* zugewiesen.

4.2.4.6.6. Berechnung des Stellsignals für den Vorlaufmischer

Um aus dem errechneten Sollwert der Vorlauftemperatur nun ein brauchbares Stellsignal für den Stellmotor des Vorlaufmischers zu erzeugen, wurde in PI-Regler als Funktionsbaustein implementiert und im Hauptprogramm mit dem Namen *piRegler* instanziiert.

Die Umsetzung des PI-Regler basiert hierbei auf den detaillierten Ausführungen des Abschnitts *Stetige Regler* und wurde wie folgt realisiert:

```
IF xInit THEN
   rY:= 0;
   rE:= 0;
   rKp:= 0;
   rESum:= 0;
   xInit:= FALSE;
END_IF;

(* Berechnung des Übertragungsbeiwertes für den P-Anteil *)
IF rKp = 0 THEN
   rKp:= (rMaxY - rMinY) / (rMaxX - rMinX);
END_IF;
```

```
(* Starten des Intervalltimers für die Fehlerwertabtastung *)
tpTa(IN:= TRUE, PT:= REAL_TO_TIME(rTa * 1000));

(* Aktualisierung des Stellsignals nach Ablauf der Abtastzeit *)
IF NOT(tpTa.Q) THEN
    (* Berechnung der Regeldifferenz *)
   rE:= rSW - rIW;

   (* Aufsummierung der Regeldifferenzen für den I-Anteil *)
   IF ((rY > rMinY) AND (rY < rMaxY)) THEN
      rESum:= rESum + rE;
   END_IF;

   (* Reglergleichung zur Berechnung des Stellsignals *)
   rY:= rKp * (rE + rTa * rESum / rTn);

   IF rY < rMinY THEN
      rY:= rMinY;
   END_IF;

   IF rY > rMaxY THEN
      rY:= rMaxY;
   END_IF;

   tpTa(IN:= FALSE);
END_IF;
```

Hierzu wurden vier interne lokale Variablen angelegt, um einerseits die aktuelle Regelabweichung *rE* und andererseits die Summer der bisherigen Regelabweichungen *rESum* berechnen zu können. Zudem wurden die Variablen *xInit* für die Initialisierung des Reglers bei der ersten Benutzung und *tpTa* als Timer für das Abtastintervall angelegt.

Während des ersten Starts des Reglers werden sämtliche für die Berechnung des Stellsignals entscheidenden Komponentenvariablen mit *0* initialisiert und die Initialisierungsvariable *xInit* auf *FALSE* gesetzt. Dies verhindert die erneute Reinitialisierung des PI-Reglers während des normalen Betriebs, ermöglicht aber einen Reset zur Laufzeit.

Der Timer *tpTa* des Typs *TP* (aus der Bibliothek *standard.lib*) erhält als Eingabewerte einen booleschen Wert *IN*, der den Timer startet, sobald er den Wert *TRUE* annimmt und einen *TIME*-Wert *PT*, der das zu messende Zeitintervall in Millisekunden angibt. Sobald *IN* auf *TRUE* gesetzt wird, nimmt die Ausgabevariable *Q* ebenfalls den Wert *TRUE* an und die Zeit wird in der Ausgabevariablen *ET* in Millisekunden hochgezählt. Sobald *ET* den Wert von *PT* erreicht hat, behält *ET* den Wert und die Ausgabevariable *Q* nimmt den Wert *FALSE* an.

Bevor der Timer für das Abtastintervall und die Berechnung des Stellsignals gestartet werden, erfolgt eine Überprüfung des Übertragungsbeiwertes *rKp* für den Übertragungsbeiwert des Reglers. Wurde diesem der Wert *0* zugewiesen, wird er automatisch gem. den Ausführungen des Abschnitts *P-Regler* berechnet.

Nach dem Start des Timers für das Abtastintervall wird überprüft, ob das Abtastintervall abgelaufen und somit die Erfassung der Regeldifferenz und die Berechnung des Stellsignals erforderlich ist.

Ist der Timer für das Abtastintervall abgelaufen, wird die aktuelle Regeldifferenz *rE* aus der Differenz des Sollwertes *rSW* und des aktuellen Istwertes *rIW* der Vorlauftemperatur errechnet.

Wurde die Regeldifferenz berechnet erfolgt eine Überprüfung, ob das aktuelle Stellsignal noch innerhalb des Regelbereiches liegt[51]. Ist dies zutreffend, wird die aktuelle Regeldifferenz auf die bisherige Summe aller Regeldifferenzen (die integrierte Regelabweichung über die Zeit) aufaddiert.

Erst jetzt erfolgt die Berechnung des neuen Stellsignals entspr. den Erläuterungen des Abschnitts *PI-Regler*.

Zum Abschluss dieser Stellsignalberechnung erfolgt noch eine Überprüfung, ob sich der berechnete Wert noch innerhalb der Grenzen des Stellbereichs befindet und wird ggf. auf die jeweiligen Extremwerte gesetzt.

Wurde der Wert für das Stellsignal *rY* schließlich gültig berechnet, erfolgt ein Reset des Timers um den Ablauf des nächsten Abtastintervalls zu ermöglichen. Im Hauptprogramm wird dieser Wert nach der Berechnung der Komponentenvariablen *rWert* des analogen Ausgangs *aaSWVLM* zugewiesen.

4.2.4.6.7. Betrieb des Brenners und der Vorlaufpumpe

Für den Betrieb des Brenners und der Vorlaufpumpe sollten im Rahmen dieses Projekts Nachlaufzeiten implementiert werden. Diese bieten die Möglichkeit einer einseitigen Hysterese, indem die Komponenten nach der Deaktivierung durch die Regelung noch eine vorgegebene Zeit weiter in Betrieb bleiben. Von besonderer Bedeutung ist dies vor allem dann, wenn Schwankungen vorliegen, die eine schnelle Schaltfolge nach sich ziehen und die Komponenten somit unnötig (mechanisch) beanspruchen würden.

Die Funktionsbausteine *FB_BRENNER* und *FB_PUMPE* wurden im Hauptprogramm als *Brenner* und *PumpeVL* instanziiert und ermöglichen die Aktivierung und Deaktivierung gem. den eben genannten Bedingungen. Hierzu werden Eingangswerte für die Eingangsvariablen *xSB* vom Typ *BOOL* und *rZeitNL* vom Type *REAL* erwartet. Der Wert von *xSB* gibt dabei an, ob die entspr. Komponente aktiviert (*TRUE*) oder deaktiviert (*FALSE*) werden soll. In der Variablen *rZeitNL* erfolgt die Übergabe des Wertes für die Nachlaufzeit der entspr. Komponente in Millisekunden. Die Ausgabevariable *xFreigabe* schließlich nimmt entspr. den internen Bedingungen den Wert des Schaltbefehls *xSB* an.

Im Rahmen dieser Bedingungen wird zunächst überprüft, ob die Komponente einen Schaltbefehl *xSB* (Aktivierung) im deaktivierten Zustand erhält. Trifft dies zu, werden der interne Status *xEin* und der Wert der Ausgabevariablen *xFreigabe* auf *TRUE* (aktiviert) gesetzt und ein Reset des Timers für die Nachlaufzeit *TimerNL* ausgeführt. Die Komponente befindet sich in diesem Fall im Betriebszustand.

Im Rahmen einer weiteren Überprüfung erfolgt eine Kontrolle, ob sich die Komponente im Betriebszustand (*xEin* = *TRUE*) befindet und als Schaltbefehl ein

[51] Diese Überprüfung wird als *anti Windup* bezeichnet, da im Falle einer anhaltenden Regeldifferenz und voller Ausschöpfung des Stellbereiches sonst eine undefiniert große Aufsummierung der Regeldifferenzen erfolgen würde.

Signal zur Deaktivierung (*xSB = FALSE*) erhalten hat. Trifft dies zu, erfolgt keine Änderung des aktuellen Zustands und der Nachlauftimer wird gestartet. Erst nach Ablauf dieses Timers werden der interne Status (*xEin*) und der Wert der Ausgabevariablen (*xFreigabe*) auf *FALSE* gesetzt. Dies erfolgt allerdings nur, wenn der Schaltbefehl zur Deaktivierung auch nach Ablauf des Timers noch anliegt, der Regler den Heizvorgang also nicht erneut aktiviert hat.

Der Überprüfung, ob sich die Komponente bereits im Betriebszustand befindet aber dennoch einen Schaltbefehl zur Aktivierung erhält, dient die letzte Abfrage des Funktionsbausteins. Trifft dies zu, besteht die Möglichkeit, dass zuvor der Ablauftimer für die Nachlaufzeit gestartet wurde, so dass in diesem Fall ein Reset und eine Deaktivierung des Timers erfolgen.

4.2.4.7. Programmablauf

Entsprechend des Ablaufplans aus dem Abschnitt *Definition des Programmablaufs* wurde die bis zu diesem Punkt beschriebene Implementierung der Steuerungssoftware zur außentemperaturgeführten Vorlaufregelung umgesetzt. Dieser Abschnitt dient nochmals der Übersicht des Ablaufs des Hauptprogramms *PLC_PRG* der Steuerung.

Nach dem Start der Steuerung erfolgt die Initialisierung aller relevanten Systemvariablen und –bestandteile mittels folgender Anweisungen:

```
(* Start Initialisierungstimer, für vollständige
   Systeminitialisierung *)
IF NOT(xFreigTimer) THEN
   InitTimer(IN:= TRUE, PT:= T#10s);
END_IF;

(* Initialisierung bei Kaltstart oder Reset des Systems *)
IF xInitSystem THEN
   (* Initialisierung: relevanter Systemwerte *)
   PLC_PRG.AKT_INIT_SYS;

    (* Initialisierung: analoge Ausgänge *)
   PLC_PRG.AKT_INIT_AA;

    (* Initialisierung: analoge Eingänge *)
   PLC_PRG.AKT_INIT_AE;

    (* Initialisierung: digitale Ausgänge *)
   PLC_PRG.AKT_INIT_DA;

    (* Initialisierung: digitale Eingänge *)
   PLC_PRG.AKT_INIT_DE;

   (* Initialisierung ist erfolgt, eine weitere ist nicht
      erforderlich *)
   xInitSystem:= FALSE;
END_IF;
```

Dieser Vorgang entspricht der in Abbildung 37 dargestellten Initialisierungsphase und führt nach erfolgreichem Abschluss, in diesem Fall nach Ablauf des Timers *InitTimer* zur Phase der Freigabeprüfung (*Prüfung Freigabe*).

Diese Prüfung wurde im Hauptprogramm auf die folgende Art implementiert:

```
(* Abfrage, ob manueller Systemreset erfolgen soll *)
IF xInitTaster THEN
   xFreigTimer:= FALSE;
   xInitSystem:= TRUE;
END_IF;

(* Aktualisierung: analoge Eingänge *)
PLC_PRG.AKT_REF_AE;

(* Aktualisierung: analoge Ausgänge *)
PLC_PRG.AKT_REF_AA;

(* Aktualisierung: digitale Eingänge *)
PLC_PRG.AKT_REF_DE;

(* Aktualisierung: digitale Ausgänge *)
PLC_PRG.AKT_REF_DA;

(* Überprüfen, ob aktuelle Außentemperatur Heizbetrieb erfordert *)
FreigHzg(
   rAT:= aeAT.rWert,
   bHG:= AnlagenDaten.byHG,
   bTolHG:= AnlagenDaten.byTolHG);

(* Überprüfung auf Störungen von Anlagenkomponenten *)
StoerAnl(
   xSt_1s:= deSMB.xZustand,
   xSt_2s:= deSMP.xZustand);

(* Überprüfung auf Freigabe der Anlage *)
FreigAnl(
   xFreigHzg:= FreigHzg.xFreigHzg,
   xStoerAnl:= StoerAnl.xStoerAnl);
```

Während dieser Phase wird überprüft, ob ein manueller Reset der Steuerung durch das Betätigen des entspr. Tasters in der Visualisierung angefordert wurde. Zudem erfolgt während dieser Phase die Aktualisierung sämtlicher Ein- und Ausgänge, da deren Werte (besonders der analoge Messwert der aktuellen Außentemperatur *aeAT.rWert*) als Prüfungsbedingungen der einzelnen Freigabebausteine genutzt werden. Wie weiter oben in der Beschreibung der einzelnen Funktionsbausteine erläutert, erfolgen in einem ersten Schritt der Freigabeprüfung die Abfrage der Heizfreigabe und der Störungsfreigabe. Sollten die Heizfreigabe erteilt sein und keine Störungen in der Anlage vorliegen, gibt die Freigabeüberprüfung der gesamten Anlage den Betriebszustand (*Betriebszyklus*) frei.

Nach erfolgreicher Prüfung der Freigabe wird ein Zyklus des Hauptprogramms zur Regelung der Anlage durchlaufen, bevor ein erneuter Wechsel in den Zustand der Freigabeprüfung erfolgt.

Der Betriebszyklus im Hauptprogramm wurde dazu wie folgt implementiert:

```
(* Regelaufgaben nur bei Freigabe der Anlage ausführen *)
IF FreigAnl.xFreigAnl THEN

   (* Berechnung Sollwert: Vorlauftemperatur *)
   AnlagenDaten.bySWVLT:= FUN_CALC_VLT(
```

```
            AnlagenDaten.byMinVLT,
            AnlagenDaten.byMaxVLT,
            AnlagenDaten.siMinAT,
            AnlagenDaten.byMaxAT,
            AnlagenDaten.siPV,
            AnlagenDaten.bySWRT,
            aeAT.rWert);

         (* Berechnung Sollwert: Stellsignal Vorlaufmischer *)
         piRegler(
            rIW:= aeVLT.rWert,
            rSW:= AnlagenDaten.bySWVLT,
            rKp:= AnlagenDaten.rKp,
            rTa:= AnlagenDaten.rTa,
            rTn:= AnlagenDaten.rTn,
            rMinX:= AnlagenDaten.byMinVLT,
            rMaxX:= AnlagenDaten.byMaxVLT,
            rMinY:= WORD_TO_REAL(aaSWVLM.wMinY),
            rMaxY:= WORD_TO_REAL(aaSWVLM.wMaxY),
            rY=>aaSWVLM.rWert);

         (* Betrieb Brenner starten, wenn Vorlaufmischer offen ist *)
         Brenner(
            xSB:= (aaSWVLM.wWert > 0),
            rZeitNL:= AnlagenDaten.rZeitNLB,
            xFreigabe=>daSBB.xZustand);

         (* Betrieb Vorlaufpumpe starten, wenn Vorlaufmischer offen *)
         PumpeVL(
            xSB:= (aaSWVLM.wWert > 0),
            rZeitNL:= AnlagenDaten.rZeitNLP,
            xFreigabe=>daSBP.xZustand);

      (* Brenner und Vorlaufpumpe abstellen, wenn keine Anlagenfreig. *)
      ELSE
         Brenner(
            xSB:= 0,
            rZeitNL:= 0,
            xFreigabe=>daSBB.xZustand);

         PumpeVL(
            xSB:= 0,
            rZeitNL:= 0,
            xFreigabe=>daSBP.xZustand);
      END_IF;
```

Während eines Betriebszyklus' wird zunächst der Sollwert der Vorlauftemperatur berechnet, bevor aus diesem Wert und den Vorgaben der Konfiguration (Werte in *AnlagenDaten*) das Stellsignal für den Stellmotor des Vorlaufmischers errechnet wird. Nach diesen Berechnungen erfolgen die Ansteuerung des Brenners und der Vorlaufpumpe, wobei diese aktiviert sind, solange der Vorlaufmischer geöffnet ist.

Ist zu Beginn keine Anlagenfreigabe erfolgt, werden sowohl der Brenner als auch die Pumpe umgehend deaktiviert (bzw. bleiben deaktiviert), da die fehlende Freigabe entweder durch Überschreitung der Heizgrenze oder im kritischen Fall durch die Fehlfunktion einer Anlagenkomponente hervorgerufen worden sein kann.

4.2.5. Entwicklung der Visualisierung

Die Visualisierung der bis hierher beschriebenen Regelung wurde auf mehrere Anzeigen (Elemente in der Registerkarte *Visualisierungen* des *Object Organizers*) aufgeteilt. Dieses sind im einzelnen:

- INIT_VISU

  Diese Visualisierung enthält lediglich ein Element mit den typischen Nutzfarben und der Farbpalette, die von den übrigen Visualisierungen zur Darstellung der einzelnen Objekte genutzt wird. Die Initialisierung der Farbpalette erfolg beim Start der Steuerung während des Initialisierungsvorgangs in der dem Hauptprogramm *PLC_PRG* zugeordneten Aktion *AKT_INIT_SYS*. Hierzu wird folgende Anweisung verwendet:

  ```
  (* Initialisierung der Farbtabelle *)
  SystemSetParameter(
     ParameterID:= 3012, Value:= ADR(g_sMasterBMP));
  ```

  Hierfür wird entspr. der Beschreibung des Hersteller der Steuerungshardware (*SABO Elektronik GmbH*) auf den Systemparameter *3012* zugegriffen und diesem das in dieser Visualisierung enthaltene Farb-Bitmap als Grundlage zur Initialisierung der Farbpalette übergeben. Wichtig ist, dass die globale Variable des Typs *STRING* den korrekten Namen (max. 16 Zeichen und klein geschrieben) der Bitmap-Datei enthalten muss. Zudem muss sich diese Datei zur korrekten Kompilierung des Projekts im selben Ordner wie Projektdatei(en) befinden.

- PLC_VISU

  Dies ist der Hauptbildschirm, der sämtliche Komponenten zur Darstellung des Heizkreises und der aktuellen Regelung sowie die Buttons für den Reset und den Zugriff auf die Systemeinstellungen enthält.

  Sämtliche grafischen Darstellung sind gezeichnete Bitmaps und als feste Bestandteile in diese Visualisierung integriert. Für jedes dieser Objekte können während des Entwurfs mittels Doppelklick auf das entspr. Objekt dessen Eigenschaften konfiguriert werden. Entscheidend sind hier die Kategorien *Text*, *Variablen* und *Eingabe*.

  In der Kategorie *Text* wird festgelegt, welche feste Bezeichnung das dargestellte Objekt erhalten und/oder welche Art von formatiertem Text eingeblendet werden soll. Hierbei können Formatierungskürzel der Form *%s* (für einen String) o. ä. genutzt werden. Diese Möglichkeit wurde im Rahmen dieses Projekts eingesetzt, um Variablenwerte wie z. Bsp. den aktuellen Messwert der Außentemperatur *aeAT.rWert* darzustellen.

  Mit der Kategorie *Variablen* wird u. a. die Möglichkeit geboten, den symbolischen Platzhaltern (wie z. Bsp. *%s*) aus der Kategorie *Text* eine Variable für den darzustellenden Wert zuzuweisen.

  Um bestimmte Werte von Variablen zur Laufzeit ändern zu können, müssen unter der Kategorie *Eingabe* entspr. Aktionen festgelegt werden. Diese Aktionen ermöglichen z. Bsp. die Einblendung einer vordefinierten Tastatur während des

Betriebs. Bearbeitet werden können hiermit die Werte der Variablen, die zuvor zur Darstellung in der Visualisierung angegeben wurden.

Im linken oberen Bereich des Hauptbildschirms der Visualisierung befindet sich die Anzeige für die Außentemperatur, bestehend aus dem Namen und dem aktuellen Messwert. Diese beiden Werte werden während des Betriebs der Steuerung aus den Variablen *aeAT.sBezeichnung* und *aeAT.rWert* ausgelesen.

Im rechten oberen Bereich werden sowohl der Status der Heizfreigabe als auch der Status der gesamten Anlage eingeblendet. Diese basieren auf den Werten der Variablen FreigHzg.xFreigHzg, StoerAnl.xStoerAnl und InitTimer.Q. Hierbei wird die letzte Variable genutzt, um den aktiven Initialisierungsvorgang der Steuerung anzuzeigen.

Die untere rechte Ecke der Visualisierung befinden sich die beiden Buttons, um einerseits einen Reset der Steuerung durchzuführen und andererseits Zugriff auf den Hauptbildschirm der Steuerungskonfiguration zu erlangen. Diese Buttons wurden über die Kategorie *Eingabe* mit einer entspr. Aktivierung der zugehörigen Visualisierung (*SETUP_SYS*) bzw. dem Tasten der Variablen *xInitTaster* konfiguriert.

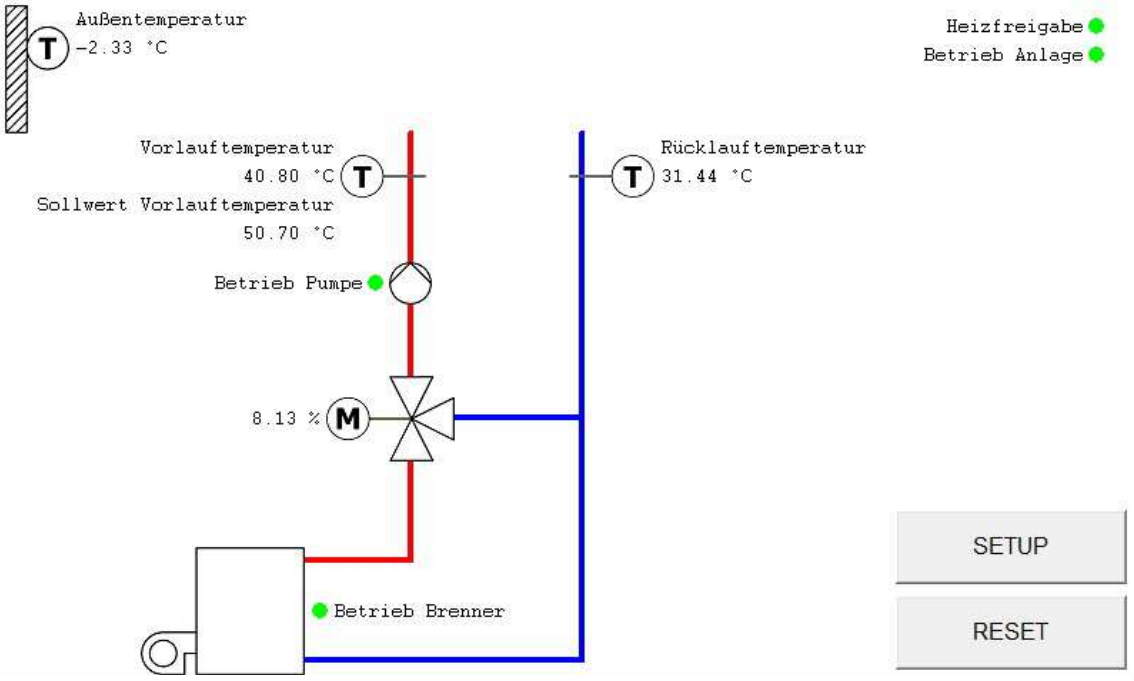

Abbildung 38: Hauptbildschirm der Steuerungs-Visualisierung .

Die übrige Darstellung erfolgt hierbei gem. Abbildung 38 in Anlehnung an Abbildung 1, allerdings ohne die Einbindung des Wärmeverbrauchers. Sämtliche übrige Elemente enthalten die entspr. zugehörigen Einblendungen der aktuellen Werte.

- SETUP_SYS

  Diese Visualisierung dient der Konfiguration der Steuerung. Hierin kann auf sämtliche Werte der Anlagenvariablen *AnlagenDaten* zugegriffen werden. Außerdem ist es hier möglich auf die Konfigurationsmenüs der analogen und

digitalen Ein- und Ausgänge zuzugreifen. Dieser Bildschirm kann über den unten rechts angeordneten Button *Hauptbildschirm* wieder verlassen werden.

- SETUP_F_AT / SETUP_F_VLT / SETUP_F_RLT / SETUP_BR / SETUP_VLP / SETUP_VLM

  Diese Konfigurationsbildschirme sind aus der *SETUP*-Visualisierung erreichbar und ermöglichen den Zugriff auf die einzelnen Komponenten der digitalen und analogen Ein- und Ausgänge. Entspr. deren Möglichkeiten können hier sowohl die Fühler für die Außen-, die Vorlauf- und die Rücklauftemperatur als auch der Brenner, die Vorlaufpumpe und der Vorlaufmischer konfiguriert werden. Innerhalb dieser Konfigurationsbildschirme erfolgt die Zuweisung der Bezeichnungen, der Module und der Ausgänge, die für die jeweilige Funktionalität wie Betriebs- oder Störmeldung sowie Schaltbefehle oder Klappenstellung genutzt werden sollen.

5. Ergebnis- und Realisierbarkeitseinstufung

Nachdem sowohl die theoretischen Grundlagen als auch die praktische Umsetzung der Steuerung erläutert wurden, wird nun an dieser Stelle eine Ergebnisabschätzung der Entwicklung erfolgen. Hierbei soll der Schwerpunkt vor allem auf den eingangs im Abschnitt *Prototypische Entwicklung* erwähnten Gesichtspunkten wie Modularität, Funktionalität und Bedienbarkeit liegen.

5.1. Ergebnis aus Hardware-Sicht

Die eingesetzte Hardware der *Firma SABO Elektronik GmbH* vermittelte dabei während der gesamten Entwicklungsarbeit einen recht ausgewogenen Eindruck. Zu den Stärken zählt hier insbesondere die gute Grundausstattung sowohl des Master Terminals als auch des eingesetzten Erweiterungsmoduls. Für die hier anfallenden Aufgaben der außentemperaturgeführten Vorlaufsteuerung standen genügend Ein- und Ausgänge, sowohl analog als auch digital, zur Verfügung. Es wäre also durchaus möglich, komplexere Regelkreise mit der hier vorliegenden Hardware zu realisieren.

Besonders zu Beginn dieser Arbeit war allerdings eine recht zeitintensive Einarbeitung besonders in die Systemspezifikationen erforderlich. Der Vorteil des genutzten Programmiersystems (Erläuterungen hierzu folgen), herstellerübergreifend kompatibel zu sein, war im Nachhinein der Hardware als Nachteil anzulasten. Wie aus den beiden Abschnitten *Anlegen und Konfigurieren des Projekts* und *Abschließende Projektkonfiguration* ersichtlich wurde, war für die erste Nutzung der Hardware eine umfassende Grundkonfiguration notwendig. Ohne weitere Einsicht in die jeweiligen Datenblätter wäre mittels der verfügbaren Step-by-step-Anleitung der Firma *SABO Elektronik GmbH* zwar eine Grundkonfiguration möglich, jedoch nicht sehr sinnvoll gewesen. Insbesondere wurde es erforderlich, sich mit der Speicher- und Modulorganisation der Hardware vertraut zu machen, da hier das größte Fehlerpotential in der grundlegenden Projektkonfiguration lag. Die projektbezogene Organisation der Erweiterungsmodule stellte hierbei eine besondere Herausforderung dar, da nicht gleich zu Beginn der Unterschied zwischen einer Adressierungs-Knotennummer und einer CAN-Node-Id ersichtlich war.

Den schwerwiegendsten Nachteil der Hardware stellt wohl die nur während der Projektkonfiguration variable Nutzung der analogen Eingänge dar. So müssen diese vor der Nutzung im Realbetrieb für jedes einzelne Modul über den Punkt *Service Data Objects* der *Steuerungskonfiguration* definiert werden. Bei der Implementierung der Steuerungssoftware lag der Fokus besonders auf dem Aspekt der Modularität, der durch diesen Punkt allerdings zunichte gemacht wurde. So konnte keine Option gefunden werden, einen während der Projektierung als Spannungseingang definierten analogen Eingang auf einen Stromeingang umzustellen. Sogar die Konfiguration auf einen bestimmten analogen Messfühlertypen (Pt100, Ni1000 etc.) musste feststehend erfolgen, da hier entspr. den Ausführungen des Abschnitts *Abschließende Projektkonfiguration* auch der Wertebereich für die digitale Darstellung der Messwerte festgelegt wurde. So bietet die im Rahmen dieses Projekts implementierte Steuerungssoftware die Möglichkeit, sämtliche dargestellten Fühlertypen (Außentemperatur, Vorlauftemperatur) sowohl an beliebige Module als auch an beliebige Eingänge zu binden, was die Systemmodularität immens steigern würde. Das System selber jedoch unterbindet eine derart frei konfigurierbare

Steuerungsumgebung. Sollte sich ein Anlagenbetreiber z. Bsp. (aus welchen Gründen auch immer) dazu entscheiden, einen ursprünglichen Temperaturfühler des Typs Pt1000 (Messbereich -50°C bis 150°C, digitaler Wertebereich -500 bis 1500) gegen einen Fühler des Typs Pt1000 (Messbereich 0°C bis 650°C, digitaler Wertebereich 0 bis 6500) auszutauschen, müsste eine Rekonfiguration des entspr. analogen Eingangs erfolgen. Selbst bei einer Änderung des Anschlusses eines analogen Messfühlers müsste eine erneute Konfiguration des entspr. Eingangs erfolgen, so dass die hier entwickelte Steuerungssoftware zwar die Möglichkeit zur freien Parametrierung bietet, welche systembedingt jedoch ausgehebelt wird. Dieser Aspekt konnte demnach leider nicht im Sinne einer nicht bzw. nur marginal begrenzten Modularität umgesetzt werden.

Einen ähnlich nachteiligen Eindruck vermittelten auch die Displays (getestet wurden hier das 4,3″-Touch-Display des Master Terminals *MTB717.14*, das 7″-Touch-Display des Master Terminals *MTB727.24* sowie das 4,3″-Touch-Display des Basismoduls *BSM242.14*). Neben der modularen Konfiguration der genutzten Ein- und Ausgänge sollte im Rahmen dieser Arbeit auch die grafische Oberfläche zur Status-Anzeige und Bedienung der entspr. Steuerung auf andere Hardware (des selben Herstellers) mit Display übertragbar sein. Dies ist leider nicht ohne erheblichen Aufwand realisierbar, da keines der Displays die Skalierung von Grafiken unterstützt. Nun bieten die unterschiedlichen Displays allerdings auch unterschiedliche Auflösungen (320 x 240, 480 x 272 und 800 x 480), so dass für jede entwickelte Steuerungsoberfläche die jeweiligen Symbole in ihrer Darstellungsgröße angepasst werden müssten. Mit erheblichem Aufwand ließe sich sicherlich eine allumfassende Symbol- und Grafikbibliothek anlegen, die sämtliche steuerungsabhängig skalierten Bildobjekte beinhaltet, dies würde jedoch den Verwaltungs- und Projektierungsaufwand deutlich erhöhen. Zudem sind der Hardware auch weitere Grenzen gesetzt, indem jede Bilddatei auf das Bitmap-Format mit max. 256 Farben beschränkt ist und der Speicherplatzbedarf aller genutzten Bildobjekte zusammen die 2MB-Grenze nicht überschreiten darf. Besonders bei der Realisierung der Regelung größerer Anlagen mit mehreren Heiz-, Lüftungs- und sonstigen Regelkreisen könnte dies unter Umständen zu unerwünschten Einschränkungen führen. Im Rahmen der hier vorliegenden Umsetzung der Visualisierung wurden insgesamt 30 zweifarbige Bitmap-Symbole entworfen, die zusammen nur 122kB beanspruchen, die Darstellungsmöglichkeiten des Displays aber auch nicht annähernd nutzen. Sofern alle Symbole auf diese Art entwickelt werden, lassen sich speicherplatzbedingt bei einem durchschnittlichen Bedarf von $122\text{kB} / 30 \approx 4{,}1\text{kB}$ je Symbol maximal $2048\text{kB} / 4{,}1\text{kB} \approx 500$ Symbole in die Steuerungshardware integrieren. Nicht berücksichtigt sind hierbei allerdings eventuelle Kundenwünsche nach grafischen Hintergründen für jede Visualisierung sowie die Darstellung kundenspezifischer Symbole. Zuzüglich zu diesen Nachteilen ist der verfügbare Zeichensatz erheblich eingeschränkt und nur als Systemzeichensatz der Größen 8, 12, 17 und 22 verfügbar. Es ist also durchaus möglich, mit der Programmierumgebung gestalterisch aufwendige Visualisierungen zu erzeugen, allerdings besteht dann das Risiko, dass Texte nicht korrekt dargestellt werden. Laut Herstellerangaben unterstützt die Steuerungshardware den Standard-Zeichensatz nach ISO 8859-1, womit allerdings keine Zeichen des russischen oder asiatischen Raums darstellbar sind. Entspr. den Angaben „können spezielle Font-Dateien auf der Steuerung installiert werden“ (s. [SABO2], S. 136), wie genau dies erfolgt (eigenständig oder nur bei der Auslieferung der Hardware) und ggf. zu welchem Preis wird allerdings nicht erwähnt.

Ein weiterer technischer Schwachpunkt liegt in der Anbindung der Hardware an das aktuelle Programmiersystem. Dieses ist bereits in einer neuen Version (v3.5) verfügbar, wurde allerdings noch nicht vom Hersteller der Hardware freigegeben. Neuerungen wie die Möglichkeit der objektorientierten Programmierung bleiben somit ungenutzt und lassen die Hardware bereits jetzt antiquiert wirken. Eine Entwicklung neuer Herangehensweisen ist somit trotz der Nutzung von SoftSPS im Bereich der Gebäudeautomation nur in sehr eingeschränktem Maße möglich und immer hinter dem aktuellen Stand des Marktes zurück.

Insgesamt bietet die Firma *SABO Elektronik GmbH* mit ihren Steuerungen und Modulen eine breite und solide Hardwarepalette, deren Stärken in der Hardwaremodularität allerdings nicht über die gravierenden Mängel in der Konfigurations- und Nutzungsmodularität hinwegtäuschen konnten. Die Realisierung einer herstellerspezifisch plattformunabhängigen Steuerungssoftware wurde bereits dadurch eingeschränkt bzw. verhindert. Die vorliegende Software kann leider nicht ohne weitere Anpassungen auf anders spezifizierten Komponenten des Herstellers eingesetzt werden. Im Hinblick auf den eingangs erwähnten Schwerpunkt der Funktionalität (skalierbarer Funktionsumfang der Steuerung) kann sicherlich ein positives Resümee gezogen werden, die mangelnde Realisierbarkeit der Modularität (Konfiguration der Hardware) und der Bedienbarkeit (Integration der Software in die Steuerung) ermöglichen jedoch keinen praxistauglichen Einsatz der Hardware über den Rahmen der aktuellen, ohnehin vorliegenden technischen Möglichkeiten hinaus.

5.2. Ergebnis aus Software-Sicht

Bereits mit den Erläuterungen im Abschnitt *CoDeSys-Programmiersystem* wurde auf den Funktionsumfang der hier genutzten Programmierumgebung eingegangen. Durch die Möglichkeit, mit ST eine hochsprachenähnliche Programmiersprache zum Entwurf einer SPS auf softwarebasis nutzen zu können, sind der Entwicklung neuer Bausteine oder gar Bibliotheken für den Bereich der Gebäudeautomation kaum Grenzen gesetzt. So existieren mit den Bibliotheken oscat.lib der OSCAT und HLK 2 der SABO Elektronik GmbH bereits Bibliotheken, die entspr. Funktionalitäten für den Bereich der Gebäudeautomation bereitstellen. Allerdings wurden diese im Rahmen dieser Arbeit bewusst nicht eingesetzt, da deren Implementierung in keiner Form nachvollziehbar ist und somit bereits beim Entwurf eine Abhängigkeit vom Know-how und dem Willen zur Weiterentwicklung von den jew. Anbietern entstanden wäre.

Nach der im vorangegangenen Abschnitt erwähnten Projektkonfiguration konnte bereits auf die Steuerung zugegriffen und ein erstes kompiliertes Programm (was natürlich noch keinerlei Funktionalität besaß) übertragen werden. Während des gesamten Verlaufs der Entwicklungsarbeit gestaltetet sich das Kompilieren und Übertragen weitestgehend problemfrei. Ausführliche Meldungen im Ausgabefenster bzw. ein Live-Modus zur Überwachung sämtlicher Programmdaten ermöglichten und vereinfachten die Fehlersuche sowie die Ablauf- und Code-Analyse.

Ein schwerwiegendes, nachhaltig wirkendes Manko war und ist die vollkommen ungenügende Stabilität des Programmiersystems. Immer wieder kam es zu unerwarteten Abstürzen und damit einhergehend zu Datenverlust, sofern nicht kurz zuvor eine Sicherung erfolgt ist. Besonders häufig trat dieses Problem während des Kompiliervorgangs auf, aber oftmals auch während des Verbindungsaufbaus bzw. einer aktiven Verbindung zur Steuerung. In diesem Fall verweigerte meist auch noch die

Steuerung den Dienst, da z. Bsp. die Übermittlung der aktuellen Steuerungssoftware nicht vollständig erfolgen konnte. Ein Reset sowohl der Hardware als auch der Software waren in diesem Fall unabdingbar. Dieser Fehler war in keiner Weise reproduzierbar und trat unabhängig von Netzwerkstrukturen oder genutzter Hardware auf.

So umfangreich die gestalterischen Möglichkeiten mit den verschiedenen IEC 61131-3-konformen Programmiersprachen auch sein mögen, so existieren doch einige nicht unerhebliche Mängel.

Als vollkommen unmöglich erwies sich z. Bsp. die indizierte Adressinitialisierung für die Steuerung. So musste an einer Stelle der Software unbedingt die Bindung einer Variablen an eine Hardwareadresse mittels AT-Deklaration erfolgen. Bei dieser Deklaration war es nicht möglich einzelne Bestandteile der Adresse (z. Bsp. *%IX1.0.0*) indiziert anzusprechen (z. Bsp. *%IX1.0.i*). Folgender Beispiel soll dies verdeutlichen:

```
FOR i:= 1 TO gc_byAnzEM DO
   FOR j:= 1 TO gc_byAnzDEE DO
      (* Speicherung der Adressen in einem Array *)
      aAdrDEE[i][j]:= %IX1.i.j;
   END_FOR;
END_FOR;
```

Dieser Code funktioniert nicht, da die indizierte Adressbildung wie in der vierten Zeile nicht zulässig ist. Somit ist die explizite Adressbindung wie im Rahmen dieses Projekts in den Initialisierungsbausteinen *AKT_INIT_xx* und der globalen Variablenliste *Globale_Variablen* erfolgt unabdingbar. Dies sollte im Sinne der Modularität allerdings vermieden werden, um hierbei auch eine Trennung zwischen Entwicklung der Steuerungssoftware und der Nutzung eben dieser zur Projektierung zu erreichen. Der Grundgedanke bestand darin, eine vollkommen modulare Steuerungssoftware zu entwickeln, die bei der späteren Nutzung zur Projektierung nur an einer einzigen Stelle konfiguriert werden sollte. Somit hätte die Steuerungssoftware nur als Bibliothek vorzuliegen brauchen, in der einzig eine globale Variablenliste zur Konfiguration der genutzten Modulanzahl und der entspr. Ein- und Ausgänge zugreifbar hätte sein müssen. In dem hier vorliegenden Entwicklungsstatus muss ein Projektingenieur allerdings an mindestens zwei Stellen konfigurierend in die Steuerungssoftware eingreifen: in der globalen Variablenliste und in jeder einzelnen Routine zur Initialisierung der Ein- und Ausgänge. Damit ist zum einen die strikte Trennung zwischen Programmierung und Projektierung und zum anderen die modulare Systemgestaltung nicht mehr möglich.

Erschwerend hinzu kommt die Tatsache, dass die Programmierumgebung die Nutzung dynamischer Arrays nicht gestattet. Als Beispiel sei hier die Erweiterung des Master Terminals mit zwei Modulen, einmal mit 16 analogen und einmal mit 8 analogen Eingängen genannt. Um die Adressen der jeweiligen analogen Eingänge getrennt zugreifbar zu speichern, würde nun ein Array der Größe 2 angelegt, um zwischen den beiden Modulen differenzieren zu können. Dieses Array müsste nun erneut zwei Arrays enthalten können, allerdings eines der Größe 16 und eines der Größe 8. Im ersten Array würde die Speicherung der 16 Adressen der analogen Eingänge des ersten Moduls und im zweiten Array die Speicherung der Adressen der analogen Eingänge des zweiten Moduls erfolgen. Mit diesem Verfahren könnten die Erweiterungsmodule zentral über die globale Variablenliste konfiguriert und ohne ein Zutun des entspr. Projektingenieurs programmintern verwaltet werden. Da dies nicht

umgesetzt werden konnte, wurde die Steuerungssoftware auch hier in ihrer Modularität eingeschränkt.

5.3. Ergebniszusammenfassung

Die einschränkenden Nachteile sowohl der Hard- als auch der Software haben es nicht erlaubt, eine wie zu Beginn geplante, vollkommen modulare Steuerungssoftware zu implementieren.

Neben diesen Aspekten spielt auch die Zykluszeit der SPS noch eine entscheidende Rolle. Diese gibt die Periodendauer an, die dem Laufzeitsystem zur vollständigen Abarbeitung des Hauptprogramms *PLC_PRG* zur Verfügung steht. Da es sich bei der Steuerung, gerade unter dem Aspekt des nicht vorhersehbaren manuellen Nutzereingriffs zur Laufzeit, um ein nichtdeterministisches System handelt, kann die tatsächlich benötigte Abarbeitungszeit nicht bereits während der Programmierung festgelegt werden. Die einzige Möglichkeit zur korrekten Abarbeitung besteht also in der freilaufenden Abarbeitung des Programms, was aber keine Aussagen mehr über das zeitliche Verhalten einer Steuerung zulässt.

Zudem bietet das CoDeSys-Programmiersystem zwar die Möglichkeit der Anbindung der Steuerung an ein TCP/IP-basiertes Netzwerk, stellt aber keinerlei BACnet-Funktionalität zur Verfügung. Diese stellt aber den einzig genormten Standard zur interoperablen Anlagenregelung dar und ist im Bereich der Gebäudeautomation unverzichtbar.

Im Ergebnis wurde auf der Steuerung eine prototypische Regelung entspr. aktueller technischer Möglichkeiten entwickelt, welche als eigentliche Neuheit SoftSPS-basiert abläuft. Mit den derzeitig verfügbaren Mitteln der genutzten Hardware und der Programmierumgebung lassen sich die Aspekte der Entwicklung, Anlagenautomation und Anlagenprojektierung mit einem einheitlichen Standard bequem unter einer Oberfläche vereinen. Allerdings wird dieser Vorteil durch die oben erläuterten Mängel der Modularität und damit einhergehend der Bedienbarkeit und Funktionalität vollständig beseitigt. Ein Einsatz der hier genutzten Hard- und Software stellt einerseits eine Neuerung im Bereich der Gebäudeautomation dar, zieht derzeit aber keine nennenswerten technischen Innovationen nach sich, die den Einsatz begründen würden.

6. Zusammenfassung

Insgesamt konnte im Rahmen dieser Arbeit gezeigt werden, dass es durchaus möglich ist, in der Entwicklung der Gebäudeautomation alternative Wege zu finden und zu beschreiten.

Es wurde eine funktionsfähige Steuerung entwickelt, die aktuellen Anforderungen entspr. die außentemperaturgeführte Vorlaufregelung ermöglicht. Neu in diesem Zusammenhang war die Tatsache, dass mit Nutzung des CoDeSys-Programmiersystems der Firma 3S-Smart Software Solutions GmbH eine Entwicklungsumgebung genutzt wurde, die die Entwicklung im Rahmen einer internationalen Norm zur Steuerungsentwicklung ermöglicht.

Leider war es trotz dieser Norm nicht möglich, die gewünscht modulare Steuerungssoftware zu implementieren. So existiert derzeit keine Lösung, um das Problem der mangelnden Trennung zwischen Entwicklung und Projektierung zu lösen. Zudem gab es nach Abschluss der Entwicklungsphase dieser Arbeit auch keine Möglichkeit, die Mängel in der Darstellung der Visualisierung zu beseitigen und damit den Entwicklungsaufwand für die Steuerungssoftware und den Verwaltungsaufwand für eine umfangreiche Projektierung zu verringern.

Neben diesen nicht zu beseitigenden Problemen existieren trotz allem noch diverse Möglichkeiten, die vorhandene Implementierung zu verbessern bzw. zu erweitern.

So wäre im Rahmen der aktuellen Entwicklung eine bessere Anpassung an des eigentliche Konzept der SPS denkbar. Derzeit noch als Funktion oder Aktion implementierte Programmbestandteile sollten in einem nächsten Schritt als reine Funktionsbausteine umgesetzt werden. Dies böte den Vorteil, eine Steuerung auch vollständig mittels der grafischen Entwicklungsumgebungen zu konfigurieren und den eigentlichen Programmablauf mittels zustandsorientierter Ablaufsprache (AS) umzusetzen.

Zudem sollten die Funktionen der Pumpe und des Brenners erweitert werden. Hierbei könnte die Steuerung der Pumpe um einen Blockierschutz erweitert werden, der die Pumpe in einem frei zu definierenden Zeitintervall für einen kurzen Zeitraum laufen lässt. Bei längeren Stillstandzeiten der Heizungsanlage würde dies die Wahrscheinlichkeit für das mechanische Blockieren der Pumpe verringern. Im Falle des Brenners könnte dieser um die Funktionen für mehrstufige bzw. modulierende Brenner erweitert werden, um so eine größeren Freiheitsgrad für die Projektierung zu bieten.

Ist die Visualisierung für den Hauptbildschirm bereits relativ annehmbar, sollte in einem weiteren Schritt die Anpassung der verschiedenen Visualisierungen für die Konfiguration erfolgen. Diese sind derzeit rein funktional mit verschiedenen Buttons ausgestattet und wenig strukturiert angeordnet. Dies hätte den Rahmen des ohnehin aufwendigen Projekts allerdings bei weitem gesprengt und stellt daher einen Aspekt zur Verbesserung des aktuellen Entwurfs dar.

Darüber hinaus wäre die Implementierung einer Pointer-orientierten Listen-Struktur denkbar, um damit doch noch eine Art dynamischer Datenhaltung zu ermöglichen. Allerdings wäre dieses Vorhaben mit der objektorientierten Version 3.5 der Programmierumgebung wesentlich komfortabler umsetzbar.

Im Rahmen weiterführender Entwicklungen müsste in einem ersten Ansatz unbedingt die BACnet-Anbindung realisiert werden. Dieser Aspekt ist für die vollständige Einbindung und Systemintegration unabdingbar, da die Steuerung ohne diese Option nach heutigem Stand der Technik nicht effektiv einsetzbar ist.

Zusätzlich wäre die Ausweitung der Funktionalität der Steuerung auf die Lüftungs- und Klimabereiche der Gebäudeautomation ein notwendiger Schritt der Weiterentwicklung. Ohne diese weitere Entwicklung wäre die Steuerung für die alleinige Heizkreisregelung bei weitem überdimensioniert und würde einen effizienten Einsatz nicht ermöglichen.

Als Ergebnis dieser Arbeit kann festgehalten werden, dass es sinnvoll ist, auch im Bereich der Gebäudeautomation neue Optionen zur Realisierung der Marktanforderungen zu erforschen. Dabei hat sich herausgestellt, dass es durchaus möglich ist, etablierte Verfahren aus anderen Automationsbereichen zu adaptieren. Selbst wenn die anfänglichen Anforderungen an die zu entwickelnde Steuerung nicht vollends zufriedenstellend umgesetzt werden konnten, hat sich herausgestellt, dass mit dem hier beschriebenen Vorgehen und entspr. innovativem Entwicklungsaufwand durchaus eine Alternative zu den bisherigen Entwicklungswegen gefunden werden konnte.

Literatur- und Quellenverzeichnis

[AMEV] Arbeitskreis Maschinen- und Elektrotechnik staatlicher und kommunaler Verwaltungen (AMEV): *BACnet in öffentlichen Gebäuden*; 2007

[BACIG] BACnet Interest Group Europe: *BACnet Grundlagen*; Website: http://www.big-eu.org/bacnet/basics.php; letzter Aufruf: 08.10.2012

[BALOW] Balow, Jörg: *Systeme der Gebäudeautomation*; 1.Auflage; cci Dialog GmbH; 2012

[BAUNE] BauNetz Media GmbH: *Gebäudeautomation*; Website: http://www.baunetzwissen.de/index/Elektro-Gebaeudeautomation_33381.html; letzter Aufruf: 08.10.2012

[BECKE] Becker, Dr. Ulrich: *Automatisierungstechnik nach internationaler Norm programmieren (13)*; Fachzentrum Automatisierungstechnik und vernetzte Systeme im BTZ Rohr-Kloster

[BOLLU] Bollue, Kai: *Seminar Grundlagen der Regelungstechnik*; 21.12.2001

[CANIA] CAN in Automation e.V.: *Controller Area Network (CAN)*; Website: http://www.can-cia.org/index.php?id=systemdesign-can; letzter Aufruf: 08.10.2012

[CHIPK] Chipkin Automation Systems: BACnet Architecture; Website: http://www.chipkin.com/bacnet-architecture/; Website: http://www.chipkin.com/bacnet-architecture/; letzter Aufruf: 08.10.2012

[FALLK] Fall, Kevin R. & Stevens, W. Richard: *TCP/IP Illustrated-Volume 1*; Second Edition; Addison-Wesley; 2012

[FRAAS] Fraaß, Prof. Dr. Mathias: *Meß- und Regelungstechnik II*; Technische Fachhochschule Berlin; 2003

[GEBHA] Gebhardt, Prof. Dr.-Ing. Andreas: *Regelungstechnik (Skript zur Vorlesung)*; Fachhochschule Aachen

[JACHS] Jachs, Roman: *Der TCP/IP-Protokollstapel*; 1998/1999

[JOTIE] John, Karl-Heinz & Tiegelkamp, Michael: *IEC 61131-3: Programming Industrial Automation Systems*; Springer-Verlag Berlin Heidelberg; 2001

[K&PBMR] Kieback&Peter GmbH & Co. KG: *Automationsstation BMR-Bedienfreundlichkeit mit Effizienz*; Kieback&Peter GmbH & Co. KG

[K&P2009] Kieback&Peter GmbH & Co. KG: Broschüre *Setzen Sie auf Energieeffizienz*; Kieback&Peter GmbH & Co. KG; 2009

[K&PDDC] Kieback&Peter GmbH & Co. KG: *Automationssystem DDC4000-Mit grenzenlosen Möglichkeiten*; Kieback&Peter GmbH & Co. KG

[K&PRA] Kieback&Peter GmbH & Co. KG: Raumautomation. Website: http://www.kieback-peter.de/de-de/loesungen/raumautomation/ Aufruf: 03.08.2012

[KLEEM] Kleemann, Prof. Dr.-Ing. Gerhard: *Grundlagen der Regelungstechnik (Lehrmaterial)*; Fachhochschule Oldenburg, Ostfriesland, Wilhelmshaven; 02.06.2005

[KOPPE] Koppe, Uwe: *Einführung in CANopen*; MicroControl Systemhaus für Automatisierung; 01.03.2000

[KRANZ] Kranz, Hans R.: *BACnet Gebäude-Automation 1.4*; 1.Auflage; SDV Saarländische Druckerei und Verlag GmbH; 2005

[OMPLS] Original Marken Partner: Lernsysteme: CAN-Bus; Website: http://lernsystem.original-marken-partner.de/3-CAN-Bus.404.0.html; letzter Aufruf: 08.10.2012

[PARET] Paret, Dominique: *Multiplexed Networks for Enbedded Systems*; John Wiley & Sons, Ltd; 2007

[PETRY] Petry, Jochen: *IEC 61131-3 mit CoDeSys V3: Ein Praxisbuch für SPS-Programmierer*; 3S-Smart Software Solutions GmbH

[PLATE] Plate, Prof. Jürgen: *Grundlagen Computernetze*; Hochschule München, FB04; 07.03.2012

[PARET] Paret, Dominique: *Multiplexed Networks for Embedded Systems*; John Wiley & Sons, Ltd; 2007

[SABO1] SABO Elektronik GmbH: *Systemfamilie PLM 700: Systemhandbuch Teil 1*; SABO Elektronik GmbH; 14.09.2011

[SABO2] SABO Elektronik GmbH: *Systemfamilie PLM 700: Systemhandbuch Teil 2*; SABO Elektronik GmbH; 10.05.2012

[SABO3] SABO Elektronik GmbH: *Datenblatt zum Erweiterungsmodul EWB.730.20 D2*; SABO Elektronik GmbH; September 2011

[SABO4] SABO Elektronik GmbH: *Datenblatt zum Master Terminal EWB.730.13 D2*; SABO Elektronik GmbH; September 2011

[SABO5] SABO Elektronik GmbH: *Datenblatt zum Master Terminal MTB.717.13/14 D1*; SABO Elektronik GmbH; September 2011

[SABO6] SABO Elektronik GmbH: *Datenblatt zum Master Terminal MTB.727.23/24 D1*; SABO Elektronik GmbH; September 2011

[SAMSO] SAMSON AG: *Begriffe und Symbole der Regelungstechnik*; SAMSON AG; 2000

[SAMSO2] SAMSON AG: *BACnet*; SAMSON AG; 2009

[SCHNE] Schneider, Wolfgang: *Praktische Regelungstechnik*; 3. Auflage; Vieweg+Teubner; 2008

[SCHRA] Schramek, Prof. Dr.-Ing. Ernst-Rudolf: *Taschenbuch für Heizung und Klimatechnik*; Oldenbourg Industrieverlag München; 2009

[SCHRO] Schrowang, Horst: *Regelungstechnik für Heizungs- und Lüftungsbauer*; Krammer-Verlag; Düsseldorf; 1976

[SIEME] Siemens Schweiz AG – Building Technologies Group: *Einführung in die HLK- und Gebäudetechnik*; Siemens Schweiz AG; 2007

[SMART] 3S-Smart Software Solutions GmbH: *Handbuch für die SPS Programmierung mit CoDeSys 2.3*; 3S-Smart Software Solutions GmbH; Dokument Version 3.2; 27.09.2004

[UPHAU] Uphaus, Josef: *Regelungstechnik - Projekte für den Lernfeldunterricht*; 1.Auflage; Bildungsverlag EINS; 2005

[VDIB1] Verein Deutscher Ingenieure e.V.: *VDI 3813 Blatt 1: Raumautomation – Grundlagen*; Beuth Verlag GmbH; Mai 2007

[VIESS] Viessmann Werke GmbH & Co. KG: Heizkennlinien; Website: http://www.heizung.de/de/lexikon/f_bis_j/Heizkennlinie.html; letzter Aufruf: 08.10.2012

[WEDEM] Wedemeyer, Thomas: Grundlegende Informationen zum CAN-Bus; Website: http://www.thomas-wedemeyer.de/elektronik/can-bus.html; letzter Aufruf: 08.10.2012

Selbständigkeitserklärung

Hiermit versichere ich, Ralph Freudrich, die vorliegende Diplomarbeit selbständig angefertigt zu haben. Für die Anfertigung habe ich keine anderen als die angegebenen Quellen genutzt. Alle Stellen, die wort- oder sinngemäß aus anderen Veröffentlichungen stammen, wurden als solche gekennzeichnet.

Berlin/Bremen, den 09.Oktober.2012

(Ralph Freudrich)