Masterarbeit

KONZEPT ZUR OBJEKTORIENTIERTEN PROGRAMMIERUNG EINES SHUTTLE-LIFTS

ausgeführt am

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Ing. Roland Robosch, BSc
1610322008

betreut und begutachtet von
FH-Prof. Dipl.-Ing. Dieter Lutzmayr

Graz, im Jänner 2018

.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

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

.....

Unterschrift

DANKSAGUNG

Meiner ganzen Familie, besonders meiner Verlobten Gudrun, meinem Sohn Vinzent und meiner Tochter Rosalie, möchte ich besonders für die moralische Unterstützung, Motivation und Verständnis während des Studiums und der Erstellung dieser Arbeit danken.

Ein besonderer Dank gilt FH-Prof. Dipl.-Ing. Dieter Lutzmayr für die Betreuung und Unterstützung zur Entstehung dieser Masterarbeit.

Weiteres möchte ich Dipl.-Ing. Franz Michael Fasch für die Motivation und Unterstützung jeglicher Art danken.

KURZFASSUNG

Bereits Ende der Sechzigerjahre begann die Entwicklung der strukturierten Programmierung, welche sich lange Zeit als Programmierstandard darstellte. Mit Beginn der Windows-Ära etablierte sich Ende der Achtzigerjahre zusätzlich die objektorientierte Programmierung. Dieser Schritt führt nun zu einem Paradigmenwechsel im Bereich der Programmierung von Speicher-Programmierbaren-Steuerungen. Einige Systeme wie zum Beispiel CoDeSys unterstützen bereits verschiedene Bereiche der objektorientierten Programmierung. Siemens-Steuerungen hingegen bieten derzeit ohne Zusatzsoft- und Hardware noch keinen direkten Zugang zur objektorientierten Programmierung an.

Ziel dieser Arbeit ist es die derzeitigen Möglichkeiten aufzuzeigen, um mit Siemens Steuerungssystemen, vorzugsweise Siemens 1500er Steuerungen, objektorientierte Programmierung abzubilden. Die Grundlage dafür wird durch die Definition der Norm IEC 61131-3:2003 bezüglich der Wiederverwendbarkeit von Funktionsbausteinen sowie auch der Erweiterung der Objektorientierung der IEC 61131-3:2013 geschaffen. Zur Untersuchung werden die Varianten objektorientierte Programmierung mit Funktionsbausteinen sowie die objektorientierte Programmierung mit SCL und Siemens SIMOTION Systemen herangezogen. Anhand einer Evaluierung werden die verschiedenen Methoden gegenübergestellt. Resultierend erscheint die herkömmliche Variante objektorientierte Programmierung mit Funktionsbausteinen als am Besten zur Realisierung eines Steuerungskonzeptes eines Shuttle-Liftes geeignet.

Aufbauend auf der gewählten Methode werden Steuerungs- wie auch Visualisierungs-Programmierrichtlinien definiert, somit kann eine bestmögliche Umsetzung des Konzeptes erfolgen. Abschließend werden diese Richtlinien in einer Beispielprogrammierung angewandt um daraus Schlüsse für weitere Entwicklungsschritte ziehen zu können.

ABSTRACT

At the end of the sixties the development of structured programming started, which was the standard in programming for a long time. With the beginning of the Windows era at the end of the eighties, another programming standard, which is called object-oriented programming, has also been established. This development has led to a paradigm shift in the programming of programmable logic controllers.

Different systems, such as CoDeSys, already support various areas of object-oriented programming. On the other hand, Siemens controllers do not support an object-oriented programming without additional soft- or hardware.

The aim of this master thesis is to demonstrate the currently existing possibilities for the usage of object-oriented programming with Siemens programmable controllers. The basis is created by the definition of the standard IEC 61131-3:2003 regarding the reusability of function blocks as well as the extension of object-oriented programming of the standard IEC 61131-3:2013. For this reason, the variants of object-oriented programming with function blocks as well as the object-oriented programming with Siemens SCL and Siemens SIMOTION are investigated. Therefore, different programming styles are compared with an

evaluation. The results point out that the conventional variant object-oriented programming with function blocks appears to be the most suitable method for the implementation of a control concept of a shuttle lift.

Based on the selected method, a control as well as visualization programming guidelines is defined, so that the best possible implementation of the concept can proceed. Finally, these guidelines are used in a sample programming to draw conclusions for further steps in the development.

INHALTSVERZEICHNIS

1 EINLEITUNG

Der Automatisierungsgrad verschiedenster Produktionsanlagen und Maschinen nimmt von Jahr zu Jahr zu. Dieser Anstieg hat zur Folge, dass auch Prozesse, vor allem Logistikprozesse, welche zur Produktion benötigt werden, ebenfalls zunehmend automatisiert abgebildet werden.

Als Ursprung der Logistik wird das Militär angesehen, da schon die erste Definition der Logistik im Zusammenhang mit der Kriegskunst getroffen wurde. Der byzantinische Kaiser Leontos bezeichnete in seinem Werk „Summarische Auseinandersetzung der Kriegskunst“ die Logistik als die dritte Kriegskunst, mit der Aufgabe, das Heer mit ausreichend Nachschub zu versorgen.[1]

Die heutige Logistik bezieht sich nicht mehr alleine auf Kriege, sondern stellt einen unvorstellbar großen weltweiten Wirtschaftszweig dar. Logistik kann global, wie zum Beispiel in Bezug auf die weltweite Versendung von Gütern aber auch lokal, wie beispielsweise in Bezug auf den Güterfluss in einem Unternehmen, betrachtet werden. Für Industrieunternehmen bildet die lokale Logistik einen immer wichtigeren Teil der Produktion. Die Aufgabe der Logistik lässt sich damit beschreiben, die richtige Ware bzw. die richtigen Bauteile zum richtigen Zeitpunkt am richtigen Ort bereit zu stellen. Zusätzlich soll dies, wenn es irgendwie möglich ist, auch noch kostengünstig erfolgen und rund um die Uhr garantiert werden, um effizient produzieren zu können. Das Unternehmen Knapp Industry Solutions bietet basierend auf den angeführten Anforderungen der Logistik das sogenannte YLOG-Shuttle an. Hierbei handelt es sich um ein komplettes System, welches aus einem Regal, einem Shuttle-Lift und einem oder mehreren Shuttles besteht.

Um die bereits angeführten Eigenschaften der Logistik weiterhin effektiv realisieren zu können, ist es unabdinglich, das YLOG-Shuttle-System weiterzuentwickeln und damit auf dem aktuellen Stand der Technik zu halten. Ebensolch ein Entwicklungsschritt wird durch die Einführung der neuen Siemens Steuerungsgenerationen notwendig.

Mit dem Umstieg auf die neuen 1500er Speicher-Programmierbaren Steuerungen (SPS) von Siemens wurde die Entscheidung getroffen, die Programmierung neu aufzubauen, da die bestehende Software nicht mehr als State of the Art angesehen werden kann und zudem auch noch zu unübersichtlich ausgeführt ist. Mit dieser Arbeit soll eine Grundlage zur Erstellung einer neuen Steuerungssoftware des *YLOG-Shuttle-Systems* geschaffen werden. Der Schwerpunkt bezieht sich auf die Möglichkeiten der objektorientierten Programmierung (OOP) mit Siemens-Steuerungen. Es soll eine Programmiermethode erarbeitet werden, mit welcher die Programmierung eines *YLOG-Shuttle-Systems* mit einem objektorientierten Programmierungskonzept erfolgen kann. Hierfür werden drei verschiedene Methoden der OOP mit Siemens-TIA-Portal-Steuerungen betrachtet, die durch ein Evaluierungsverfahren bewertet werden, um somit eine Entscheidung treffen zu können, welche der Methoden am besten zur Erstellung der Steuerungssoftware des YLOG-Shuttle-Systems geeignet ist. Zur Umsetzung der aus der Evaluierung resultierenden Methode müssen Programmierrichtlinien definiert werden, um dadurch sicherzustellen, dass

[1] Vgl. www.dasWirtschaftslexikon.com (2016), Online-Quelle [4.12.2017].

diverse Ansätze der OOP abgebildet werden können. Aufbauend auf diesen Richtlinien wird eine Beispielprojektierung erstellt, welche abschließend zu einer Bewertung dieser Programmiermethode herangezogen wird.

1.1 Knapp Industry Solutions

Das Unternehmen *KNAPP Industry Solutions* (KIN) ist eine eigenständige Tochter der *KNAPP AG* mit Firmensitz in Dobl. Der Fokus des Unternehmens liegt auf Industrie- und Produktions- sowie auf kleinen bis mittelgroßen Distributionslösungen. Die zurzeit verfügbaren Produkte werden durch das *YLOG-Shuttle* und verschiedene *Open-Shuttle Systeme* abgebildet. Der Grundstein des Unternehmens wurde im Jahr 2005 gelegt, mit dem Entwicklungsstart der YLOG-Shuttle-Technologie. Bereits 2008 wurde die Erstinstallation des Systems durchgeführt und in den darauffolgenden Jahren konnten mehr als zehn weitere Projekte realisiert werden. Im Jahr 2013 wurde die Gründung der *YLOG Industry Solutions GmbH – A Member of KNAPP Group* vollzogen und somit auch die Übernahme des *YLOG-Shuttle*-Produktes eingeleitet. Das Unternehmen wurde 2015 umfirmiert in *KNAPP Industry Solutions*, womit auch die Gründung der *Business Unit Industry Solutions* mit den Kerngebieten Industrie und Produktion erfolgte.

Derzeit werden am Standort Dobl 56 Mitarbeiter beschäftigt, welche die Entwicklung aber auch die Montage und Inbetriebnahme der Systeme durchführen. Zugleich wird an weiteren neuen Projekten geforscht, um somit den Stand des Unternehmens im Bereich der fahrerlosen Transportsysteme zu festigen und weiter auszubauen.

1.2 Ausgangssituation

Die Einführung von TIA Portal mit der neuen Steuerungsgeneration von Siemens im Unternehmen KNAPP Industry Solutions führte zu einer Überarbeitung der aktuellen Steuerungsprogrammierung des YLOG-Shuttle-Systems. Die bisher bestehenden Anlagen basieren auf Siemens 300er-Steuerungen und der Programmierung mit Simatic STEP7. Die bei diesen Anlagen angewandte Programmierung wurde aus verschiedenen Systemen zusammengesetzt und ist nicht mehr überschaubar. Die Programmierung wurde ohne eine Definition von Richtlinien umgesetzt, wodurch es dem Programmierer möglich war, das Programm nach seinen Vorstellungen zu gestallten. Daraus resultiert der Nachteil, dass die Programmierung in den meisten Fällen nur vom Ersteller selbst nachvollzogen werden kann. Dieser Aspekt erschwert zusätzlich die Fehlerbehebung sowie auch die Wartung und Erweiterung von bestehenden Anlagen und stellt zudem ein erhöhtes Risiko der Anlagen-Nachbetreuung dar.

Der erste Entwicklungsschritt wurde mit dem Umstieg auf die neue Steuerungsgeneration von Siemens getätigt. Da es nicht mehr einfach möglich war, die aktuelle Programmierung auf den neuen Steuerungen umzusetzen, musste der Programmaufbau neu erarbeitet werden. Gegenüber der bestehenden Programmierung konnte somit ein beachtlicher Fortschritt erzielt werden, führte aber noch nicht zum gewünschten Ergebnis. Die Programmierung an sich wurde bereits weitaus strukturierter und vor allem modular aufgebaut, erwies sich dennoch als zu komplex. Aus diesem Grund wurde die Entscheidung getroffen, dass das im nachfolgenden Kapitel beschriebene System softwaretechnisch erneut zu überarbeiten ist und damit ein Schritt in Richtung objektorientierte Programmierung getätigt werden soll.

2 SYSTEMBESCHREIBUNG

Die Logistik in Industrieunternehmen, aber auch in großen Verteilzentren, wird ein immer wichtiger Aspekt um die Produktion wie auch die Verteilung von Waren schnell und kosteneffizient durchführen zu können. Aus diesem Anlass stellt KNAPP Industry Solutions ein System zur Verfügung, welches voll automatisiert das Ein- und Auslagern von Gütern in Boxen ermöglicht. Dieses System wird als YLOG-Shuttle-System bezeichnet, welches beispielhaft in Abbildung 1 abgebildet ist.

Dieses System verwaltet ein Regal, das mit verschiedenen Gütern bestückt ist, voll automatisiert. Dabei werden über Import- und Export-Transportbänder, die nicht in Abbildung 1 enthalten sind, Transportboxen dem Lift zu- bzw. abgeführt. Befindet sich der Lift mit einem nicht beladenen Shuttle in der Ebene des Imports, auf welchen eine Box bereitsteht, dann wird diese, sofern eine Beladefreigabe vorhanden ist, auf das Shuttle aufgeschoben. Das übergeordnete System gibt dem Lift vor, auf welche Ebene der Behälter zu befördern ist. Der Lift verfährt mit der Ladung auf die vorgegebene Ebene und gibt in dieser das Shuttle an das Regal ab. Im Regal positioniert das Shuttle die Box an der durch das übergeordnete System befohlenen Stelle. Umgekehrt besteht die Möglichkeit der Auslagerung. Beim Auslagerungsvorgang wird vom Shuttle eine Kiste von einer definierten Position aufgenommen und zum Lift transportiert. Der Lift verfährt auf die Ebene des Shuttles, nimmt dieses auf und bewegt es auf die Ebene des Exportbandes, wo das Transportgut vom Shuttle an den Export abgegeben wird. Zusätzlich dient der Lift auch zum einfachen Ebenenwechsel der Shuttles, um Güter aus anderen Bereichen des Regales aufzunehmen oder umzulagern.

Jedes Shuttle verfügt über eine eigene Steuerung, mit der Fahrbefehle vom übergeordneten System empfangen und ausgeführt werden. Die Ansteuerung des Liftes erfolgt mit einer Sicherheits-SPS von Siemens, mit welcher die Hardwarekomponenten des Liftes über die Peripherie der Steuerung angesteuert und überwacht werden.

Abbildung 1: YLOG-Shuttle System, Quelle: KIN.

Grundsätzlich setzt sich das in Abbildung 1 dargestellte System aus den folgenden Komponenten zusammen.

- *Lift mit Antrieb*

Der Lift dient zur Beförderung der Liftkabine zwischen den Ebenen. Dabei können Systeme bis 22 m Höhe realisiert werden.

- *Liftkabine*

Mit der Liftkabine wird das Shuttle aufgenommen, um dieses mit dem Lift auf eine andere Ebene zu transportieren.

- *Shuttle*

Das Shuttle stellt ein fahrerloses Transportsystem dar und dient zum Transport der Boxen innerhalb des Regales sowie auch aus dem Lager in die Produktion. Da ein Querfahren ebenfalls möglich ist, können die Shuttles zwischen den einzelnen Gassen wechseln. Somit kann durch jedes Shuttle jeder Stellplatz erreicht werden, wodurch die Redundanz des Systems enorm steigt.

- *Regal*

Die Boxen werden in dem Hochregal gelagert, welches durch die schmale Gassenbreite eine extrem hohe Lagerdichte ermöglicht.

- *Steuerung*

Zur Steuerung des Liftes werden bei Neu-Anlagen ausschließlich Siemens 1500er-Sicherheitssteuerungen eingesetzt. Diese dienen zur Steuerung des Liftes und der Liftkabine sowie auch des Imports und Exports, sofern vorhanden. Die Shuttle-Systeme besitzen ein eigenes, von der Liftsteuerung unabhängiges System.

- *Visualisierung*

Um den Betrieb der Anlage so einfach wie möglich gestalten zu können werden Visualisierungssysteme eingesetzt. Vorrangig werden Siemens-Panels verwendet, die dem Bediener die Möglichkeiten bieten das Liftsystem zu steuern und den aktuellen Status der Anlage zu erheben.

- *Sicherheitstechnik*

Das System agiert vorwiegend komplett automatisiert, dennoch müssen Inbetriebnahme- und Wartungsarbeiten durchgeführt werden. Um den Bediener bei solchen Arbeiten zu schützen, ist eine ausführliche Sicherheitstechnik notwendig, die durch die Sicherheitssteuerung und die dazu erforderlichen Hardwarekomponenten umgesetzt wird.

- *Übergeordnetes System*

Der Transport der Shuttles erfolgt im Automatikbetrieb nicht willkürlich. Ein übergeordnetes System teilt dem Lift mit, auf welcher Ebene ein Shuttle aufgenommen bzw. abgegeben werden muss.

2.1 Mechanik

Zum Transport der Shuttles zwischen den verschiedenen Ebenen wird eine entsprechende Mechanik benötigt, welche in Abbildung 2 detailliert abgebildet ist. Wie bereits zuvor beschrieben baut das gesamte System auf mehreren Komponenten auf. Die Mechanik setzt sich aus den im Nachfolgendem beschriebenen Teilen Lift-Grundplatte und Säulen, Liftantrieb, Riemen, sowie der Liftkabine mit Fahrbahnklappe zusammen. Darüber hinaus besteht die Möglichkeit, dass in der Liftkabine Ladekontakte zum Laden der Shuttles während des Transportes verbaut werden.

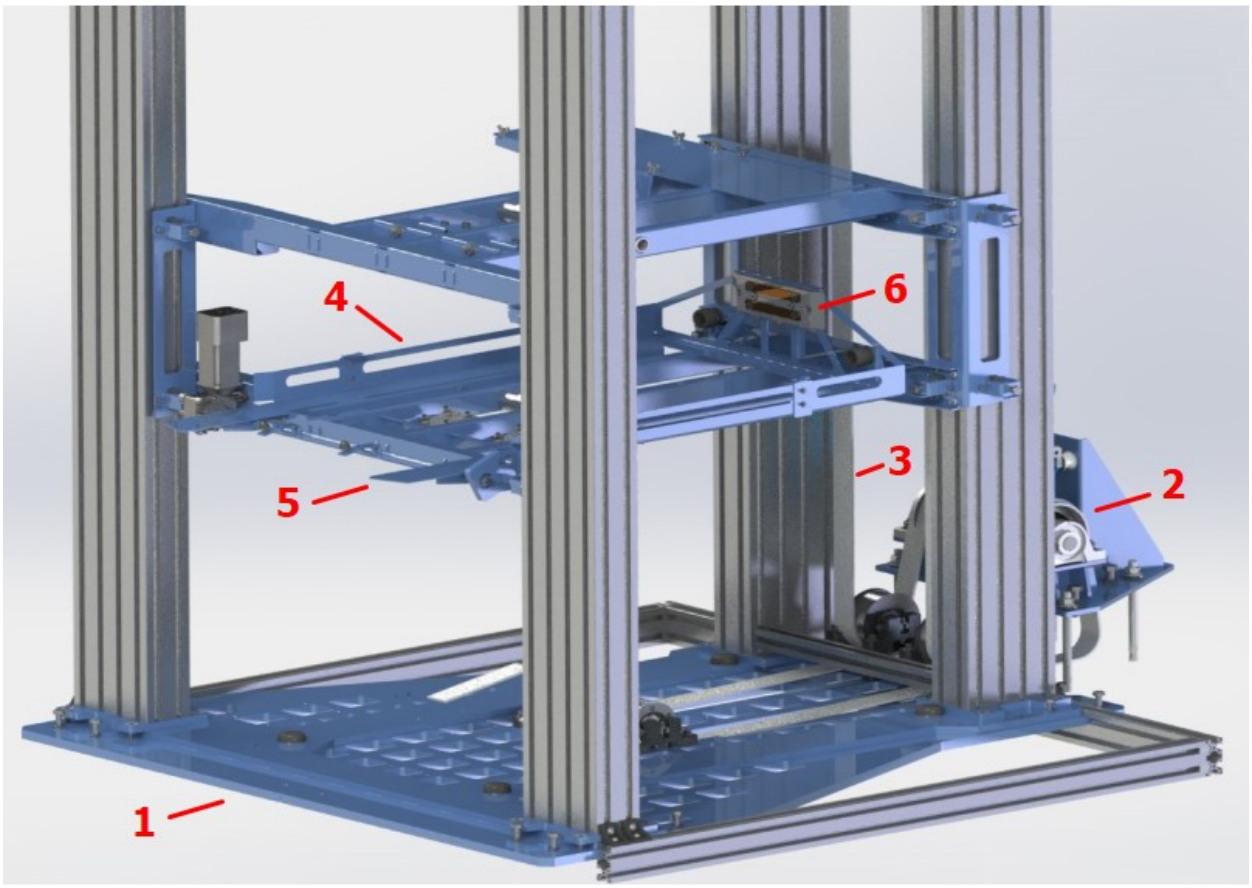


Abbildung 2: Aufbau der Lift Mechanik, Quelle: Eigene Darstellung.

1. *Lift-Grundplatte und Säulen*

Der gesamte Liftaufbau setzt auf der Grundplatte, die als massive Stahlplatte ausgeführt ist, auf. Sie dient als Träger für die Liftsäulen, aber auch zur Aufnahme des Liftantriebes sowie der Umlenkung der Riemen. Die Säulen müssen beim Aufbau genau eingerichtet werden, da diese von entscheidender Bedeutung für das Verfahren der Liftkabine sind, da über die Liftsäulen die Kabine geführt wird. Werden diese nicht exakt eingerichtet, kommt es dazu, dass die Kabine zu klemmen beginnt und im schlimmsten Fall zerstört wird.

2. *Liftantrieb*

Die Mechanik wird modular aufgebaut, das bedeutet es können unterschiedliche Liftantriebe unterschiedlicher Hersteller verbaut werden. Dennoch erfüllt der Antrieb jeweils die gleiche Aufgabe, den Transport der Liftkabine über Riemenscheiben und Riemen. Der Antrieb selbst wird über ProfiNet mittels Steuersignalen der Sicherheitssteuerung angesteuert und meldet aktuelle Statusinformationen an diese

zurück. Zusätzlich werden über diese Kommunikation Sicherheitssignale ausgetauscht, um einen sicherheitsgerechten Betrieb des Antriebes zu gewährleisten.

3. *Riemen*

Der Liftantrieb ist mit den Riemenscheiben verbunden und treibt diese an. Das Antreiben der Riemenscheiben führt dazu, dass die Riemen bewegt werden, was wiederum ein Auf- bzw. Abwärtsfahren der Liftkabine zur Folge hat. Für ein gleichmäßiges Verfahren des Liftes werden zwei Antriebsriemen verwendet.

4. *Liftkabine*

Wie schon beschrieben, ist die Liftkabine mit den Antriebsriemen verbunden und ermöglicht die Aufnahme des Shuttles zum Transport. Zudem beinhaltet die Liftkabine optional die beiden weiteren Komponenten Fahrbahnklappe und Ladekontakt.

5. *Fahrbahnklappe*

Zur Absturzsicherung des Shuttles werden auf den Fahrbahnschienen des Regales Absturzsicherungen, welche von einfachen Klappen dargestellt werden, montiert. Um diese durch die Fahrbahnklappe betätigen zu können, besteht ein Abstand zwischen Lift und Regal, welcher bei der Überfahrt des Shuttles überwunden werden muss. Damit das Shuttle diesen Spalt ohne Probleme überwinden und auf die Kabine auffahren kann, wird die Fahrbahnklappe geöffnet und somit auch die Absturzsicherung des Regales betätigt. Abbildung 2 stellt den geöffneten Zustand der Klappe dar. Wenn sich das Shuttle in der Kabine befindet, schließt die Klappe wieder und sichert es zusätzlich, sodass dieses während des Transportes nicht aus der Kabine stürzen kann. Durch das Schließen der Fahrbahnklappe wird auch die Absturzsicherung wieder geschlossen und dadurch ein Abstürzen des Shuttles aus dem Regal verhindert.

6. *Ladekontakte*

Damit das System so effizient wie möglich umgesetzt werden kann und es möglich ist, alle Bereiche des Regales unabhängig von einer Energieversorgung zu erreichen, werden die Shuttles mit verbauten Akkumulatoren betrieben. Die Ladekontakte in der Liftkabine bieten die Option, das Shuttle auch während es sich in der Kabine befindet zu laden. Die Ansteuerung der Ladekontakte erfolgt ebenfalls durch die Sicherheitssteuerung.

2.2 Steuerung und Visualisierung

Die Steuerung des Liftes erfolgt durch eine Siemens Sicherheits-SPS. Neu-Anlagen werden mit Siemens 1500er-Sicherheitssteuerungen, welche im Folgenden aus Gründen des Leseflusses als TIA Portal-Steuerungen bezeichnet werden, ausgeführt. Bestehende Anlagen wurden mit Siemens 300er Sicherheitssteuerungen, nachfolgend als STEP7-Steuerungen bezeichnet, realisiert. Die Aufgabe der SPS ist es, die für den Betrieb des Liftes benötigten Hardwarekomponenten zu steuern. Die Ansteuerung erfolgt über die Peripherie-Schnittstellen der SPS. Da die meisten Feldgeräte bereits über eine ProfiNet-Schnittstelle verfügen, werden auch diese vorrangig verwendet. Zur Ansteuerung dieser Komponenten ist eine Programmierung des Anwenderprogrammes notwendig. Dieses baut auf verschiedenen Organisations-, Funktions- und Daten-Bausteinen sowie auch Funktionen auf. Die Programmierung wird in

den in Siemens-Systemen verfügbaren Programmiersprachen Anweisungsliste (AWL), Funktionsplan (FUP), Kontaktplan (KOP) und Structured Control Language (SCL) umgesetzt. Da es nicht die Aufgabe der Steuerung ist, die Lagerplätze des Regales zu verwalten, ist eine Kommunikation zu einem übergeordneten Lagerverwaltungs-System über einen Materialflussrechner (MFR) notwendig. Über dieses System werden der SPS verschiedene Befehle mitgeteilt, die durch das Anwenderprogramm interpretiert und ausgeführt werden müssen. Die Realisierung des Datenaustausches zwischen der SPS und dem MFR erfolgt über eine definierte Schnittstelle mittels TCP-Verbindung.

Zusätzlich müssen in der Steuerung verschiedene Parameter verwaltet werden, wie die der Ebenen-Positionen in Millimeter oder auch Maschinen-Parameter der Hardwarekomponenten. Da eine nachträgliche Änderung der Parameter möglich sein muss, werden diese auf einer Visualisierung abgebildet. Eine weitere Aufgabe der Visualisierung ist es, aktuelle Zustände der Anlage dem Bediener zur Verfügung zu stellen und auch einen Betrieb des Liftes ohne übergeordnetes System zu ermöglichen. Derartige Betriebsarten werden vorwiegend für Inbetriebnahme- bzw. Wartungsarbeiten benötigt und können nur mit einer entsprechenden Sicherheitstechnik gewährleistet werden. Da die Sicherheitstechnik mittels Sicherheitssteuerung realisiert wird, müssen Sicherheitssignale über eine geeignete Peripherie erfasst werden. Mit diesen Signalen kann die Steuerung, mittels Sicherheitsprogrammes, alle Aktoren gezielt abschalten bzw. einen sicheren Betrieb gewährleisten. Dieses Sicherheitsprogramm wird mittels standardisierten Bausteinen, welche von Siemens bereitgestellt werden und getestet sind, realisiert.

2.2.1 Hardware und Peripherie

TIA Portal-Steuerungen bieten wie bereits auch die Vorgängersteuerungen die Möglichkeit einen Datenaustausch mit der Umwelt über verschiedene Methoden zu realisieren. Kommunikation über ProfiNet-Schnittstellen oder auch mittels ProfiBus DP-Schnittstellen sind möglich. Mit ProfiNet ist ein simultaner Austausch von Daten mit anderen ProfiNet-Geräten sowie Human Maschine Interfaces (HMI), aber auch zu anderen Steuerungen und verschiedenen weiteren Systemen realisierbar. Die ProfiBus DP-Schnittstelle ermöglicht gleich wie ProfiNet eine Kommunikation zu weiteren Geräten, wobei hier die Steuerung als DP-Master agiert.

Eine Schnittstelle zur Umwelt wird mittels Ein- und Ausgängen geschaffen. Über digitale Eingänge lassen sich verschiedenste Zustände wie zum Beispiel Endlagen eines Zylinders detektieren. Analoge Eingänge bieten die Möglichkeit physikalische Größen, wie Temperatur oder Druck, über einen Analogwert mit der entsprechenden Skalierung, zu erfassen. Das Gegenstück der digitalen Eingänge wird durch digitale Ausgänge gebildet. Mit diesen können Betriebsmittel wie zum Beispiel ein Zylinder in bestimmte Endlagen gebracht werden, indem durch einen Ausgang ein Ventil in eine definierte Stellung geschaltet wird. Wie auch bei den Eingängen können alternativ ebenfalls Analogwerte ausgegeben werden. Um einen physikalischen Wert richtig an das Stellglied, welches 0 – 10 V oder 4 – 20 mA zu Ansteuerung benötigt, zu senden, ist es notwendig den errechneten Steuerwert richtig zu skalieren. Je nach Anwendung kann somit eine Drehzahl oder eine Ventilstellung vorgegeben werden.

Wie bereits angemerkt, kann die Umwelt über Eingänge wahrgenommen werden, wodurch definierte bzw. erwartete Zustände überwacht werden können. Hierbei handelt es sich nicht nur um eine Endlagenkontrolle

eines Zylinders oder dergleichen, auch Sicherungen und Schaltzustände lassen sich so überwachen, sofern eine dementsprechende Logik in den Bausteinen des Anwenderprogrammes ausgeführt wird.

2.2.2 Bausteine

Grundlegend definiert die IEC 61131-3, wie später in Unterkapitel 3.2 genauer ausgeführt, durch die Programmorganisationseinheit (POE) die drei Bausteintypen Funktion, Funktionsbaustein und Programm. Siemens TIA Portal unterscheidet sich in dieser Hinsicht etwas von der Norm. Es werden zwar auch unter TIA Portal die Bausteintypen Funktion und Funktionsbaustein zur Verfügung gestellt, jedoch der Typ Programm nicht direkt. Für Programmaufrufe werden Organisationsbausteine (OB) zur Verfügung gestellt. Zudem werden Instanz-Datenbausteine zur Instanziierung und herkömmliche Datenbausteine zur Sicherung von Daten in der Steuerung abgebildet. Letztere können durch die Anwendung von PLC-Datentypen sehr strukturiert aufgebaut werden. Zur Schaffung eines Überblickes über die in TIA Portal verfügbaren Bausteintypen wird nachfolgend auf die jeweiligen Typen genauer eingegangen.

2.2.2.1 Organisationsbausteine

Die IEC 61131-3 sieht vor, dass die Ausführung von Programmen durch einen ein Task periodisch erfolgt.[2] Dabei ist es nicht von Bedeutung, ob in diesem Task ein oder mehrere Programme ausgeführt werden. Siemens-Steuerungen, unabhängig ob STEP7 oder TIA Portal, realisieren Tasks mittels Organisationsbausteinen, die eine Schnittstelle, wie in Abbildung 3 dargestellt, zwischen Anwenderprogramm und dem Betriebssystem der Steuerung darstellen. Anderen Bausteinen ist es nicht möglich Organisationsbausteine aufzurufen, da diese beim Eintreten eines Ereignisses nur durch das Betriebssystem der Steuerung aufgerufen werden können.[3]

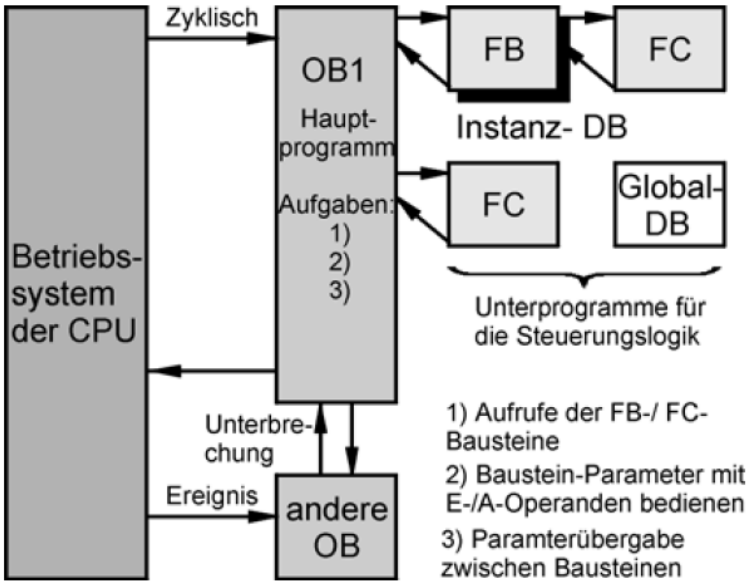


Abbildung 3: Aufgabenverteilung von Bausteinen, Quelle: Wellenreuther/Zastrow (2008), S. 38.

[2] Vgl. EN 61131-3:2013 (2014), S. 18.

[3] Vgl. Wellenreuther/Zastrow (2008), S. 36.

Durch den Organisationsbaustein 1 (OB1) wird die zentrale Organisation des Anwenderprogrammes dargestellt, über welchen die jeweiligen Aufrufe der Funktionsbausteine oder auch Funktionen organisiert werden, siehe hierzu Abbildung 3. Ergänzend zum OB1 werden weitere Organisationsbausteine zur Verfügung gestellt, welche anhand der Priorität abgearbeitet werden. Den niedrigsten Rang nimmt der OB1 ein, wodurch jeder andere Organisationsbaustein die Abarbeitung des Hauptprogrammes unterbrechen kann.[4]

Bausteine, welche die zyklische Abarbeitung des OB1 unterbrechen, werden als Weckalarmbausteine bezeichnet. Der Aufruf dieser OBs erfolgt durch das Betriebssystem in definierten Zeitintervallen, welche durch den Anwender parametriert werden können und ein Vielfaches von 1 Millisekunde sein müssen. Die Möglichkeiten von Organisationsbausteinen beschränkt sich nicht nur auf Programmaufrufe und Weckalarme. Mit der Tabelle 1 soll ein Überblick über weitere wichtige Organisationsbausteine von TIA Portal-Steuerungen und deren Aufgaben geschaffen werden.[5]

OB-Nr.	Ereignisklasse	Funktion
OB 20	Time delay interrupt	Die Programmabarbeitung des OB 1 wird nach Ablaufen einer parametrierten Zeit unterbrochen.
OB 30	Cyclic interrupt	Weck-Alarm Organisationsbausteine unterbrechen in definierten Zeitabschnitten die Abarbeitung des OB 1.
OB 82	Diagnostic error interrupt	Werden bei diagnosefähigen Baugruppen Diagnosemeldungen eingestellt, werden diese durch den OB 82 bei Auftreten eines Fehlers abgearbeitet und in dieser Zeit die zyklische Programmabarbeitung unterbrochen.
OB 83	Pull or plug of modules	Wird ein Modul gezogen bzw. gesteckt, welches projektiert ist, aber nicht deaktiviert wurde, unterbricht dieser Organisationsbaustein die zyklische Abarbeitung des OB 1.
OB 86	Rack or station failure	Bei einem Ausfall eines DP-Master, Slave oder Submodules wird der OB 86 durch das Betriebssystem aufgerufen.
OB 100	Startup	Wird ausgeführt, wenn die Betriebsart der Steuerung von STOP auf RUN wechselt.
OB 121	Programming error	Der OB 121 wird ausgeführt, wenn während der Programmabarbeitung ein Fehler auftritt.
OB 122	IO access error	Der Aufruf des IO access error erfolgt, wenn ein Zugriffsfehler auf die Peripherie auftritt.

Tabelle 1: Auswahl an Organisationsbausteinen der S7-1500, Quelle: Siemens (2013), Online-Quelle [4.12.2017].

[4] Vgl. Wellenreuther/Zastrow (2008), S. 36.

[5] Vgl. Wellenreuther/Zastrow (2008), S. 166 f.

Jeder dieser Organisationsbausteine kann durch ein Anwenderprogramm erweitert werden, um somit unter anderem Fehlermeldungen einfach auswerten zu können. Die Programmierung eines Organisationsbausteines kann prinzipiell in den Programmiersprachen AWL, FUP oder KOP erfolgen, auch eine Programmierung in SCL ist möglich. Ebenso besteht die Möglichkeit, Funktionen und Funktionsbausteine in den Organisationsbausteinen aufzurufen.

2.2.2.2 Funktionsbaustein und Funktion

Zur Umsetzung des Anwenderprogrammes werden zwei grundlegende Bausteine zur Verfügung gestellt, um somit eine strukturierte Programmierung zu ermöglichen. Hierbei wird zwischen dem Funktionsbaustein und der Funktion unterschieden, welche nachfolgend detaillierter ausgeführt werden, um somit die Unterschiede dieser beinen POE-Typen nachvollziehen zu können.

- *Funktionsbaustein*

Ein Funktionsbaustein (FB) ermittelt aus den Zuständen der Ein- und Ausgangsvariable und auch der internen Zustandsvariablen neue Zustände für die Ausgangsvariablen. Dabei bleiben die Zustände der Ausgangsvariablen sowie auch der der Zustandsvariablen (statische Variablen) über die Bearbeitung eines SPS-Zyklus hinaus erhalten. Daher spricht man hierbei von einem Baustein mit Gedächtnis. Um diese Eigenschaft realisieren zu können, muss der Funktionsbaustein instanziiert werden. Bei der Instanziierung wird eine Art Kopie (Instanz) des Bausteines in einem Datenbaustein erstellt. Die Instanz muss zudem einen eigenen Namen erhalten, um somit die Daten der Kopie im Speicher verwalten zu können. Das Prinzip der Instanziierung von Funktionsbausteinen bietet zudem die Möglichkeit, dass Funktionsbausteine im Anwenderprogramm mehrfach verwendet werden können. Durch das Speichern der Zustandsvariablen wird auch der grundlegende Unterschied zur Funktion definiert. So sind die Ausgangsparameter eines Funktionsbausteines nicht nur von dessen Eingangsparametern abhängig, sondern auch von den Zuständen der internen Zustandsvariablen. Funktionen hingegen liefern bei gleichem Zustand an den Eingängen das gleiche Ergebnis am Ausgang.[6]

- *Funktion*

Wie bereits angemerkt liefert eine Funktion (FC) Rückgabewerte, welche nur von den Zuständen der Eingangsvariablen abhängig sind. Das bedeutet, eine Funktion liefert bei gleichen Zuständen an den Eingängen im nächsten SPS-Zyklus die gleichen Ergebnisse an den Ausgängen. Daher ist eine Funktion für Anwendungen geeignet, bei welchen das Funktionsergebnis nur aus den Zuständen der Eingangsvariablen zu ermitteln ist, und keine Zustände über mehrere Zyklen hinaus gespeichert werden müssen. Funktionen können, da sie über keinen Speicher verfügen, auch nicht instanziiert werden, dennoch ist es möglich FCs mehrfach innerhalb eines Programmzyklus aufzurufen und somit aufgrund von unterschiedlichen Eingangsvariablen die zugehörigen Funktionsergebnisse zu erhalten.[7]

[6] Vgl. Wellenreuther/Zastrow (2008), S. 20.

[7] Vgl. Wellenreuther/Zastrow (2008), S. 20.

2.2.2.3 Daten

Wie bereits erwähnt, benötigen Funktionsbausteine einen Speicherbereich, einen sogenannten Instanz-Datenbaustein, um die Daten der Instanziierung zu erstellen. Zudem werden noch weitere Datenbausteine zur Verfügung gestellt, um einen erweiterten Speicher für Anwenderdaten bereitzustellen, welcher mit PLC-Datentypen strukturiert aufgebaut werden kann. Im Nachfolgenden werden die PLC-Datentypen und Datenbausteine näher erläutert.

- *Datenbaustein*

Siemens STEP7- und auch TIA Portal-Steuerungen bieten zwei Arten von Datenbausteinen (DB), den Instanz-Datenbaustein zur Instanziierung von Funktionsbausteinen und den Globaldatenbaustein zum Speichern von beliebigen Informationen. Mehrfach wurde bereits auf den Instanz-Datenbaustein (Instanz-DB) verwiesen, welcher als Speichercontainer für Funktionsbausteine dient. Der Datenbaustein muss mit dem Aufruf des Funktionsbausteines angegeben werden, wodurch der Instanz-DB erstellt wird und die zugehörigen Daten generiert werden. Global-Datenbausteine dienen als Speicher, auf welche aus allen Bereichen des Programmes zugegriffen werden kann. Zugriffe können symbolisch, aber auch absolut erfolgen, sofern keine Bausteinoptimierung aktiviert ist.[8]

Optimierte Bausteine verfügen über keine definierte Bausteinstruktur. Den Elementen werden symbolische Namen zugeordnet aber keine Adressen. Die Daten werden im Baustein automatisch angeordnet um den verfügbaren Speicher optimal auszunützen und somit keine Lücken im Speicher entstehen können. Dieses Verfahren bietet unter anderem den Vorteil, dass ein Zugriff immer auf schnellstem Weg erfolgt, da die Daten durch das System selbst optimiert und verwaltet werden. Zudem werden Zugriffsfehler, welche durch indirekte Adressierung vorkommen können, vermieden. Einzelne Elemente des Bausteines können als remanent definiert werden, bisher konnte nur ein ganzer Baustein als remanent parametriert werden. Ein weiterer großer Vorteil wird durch die standardmäßig zur Verfügung gestellte Speicherreserve erzielt, wodurch Schnittstellen im laufenden Betrieb der Steuerung erweitert werden können.[9]

- *PLC- Datentypen*

Als PLC-Datentyp wird ein durch den Anwender definierter Datentyp bezeichnet, welcher sich aus mehreren Datentypen zusammensetzen kann. Ein PLC-Datentyp kann somit aus den verschiedensten Komponenten unterschiedlicher Datentypen aufgebaut werden, welche auch die Typen ARRAY oder STRUCT (Struktur) beinhalten. [10]

Mittels PLC Datentypen werden Strukturen geschaffen, die dem Anwender die Programmierung erleichtern. Durch die Gruppierung der Daten können Datenbausteine übersichtlich aufgebaut werden. Die Wiederverwendbarkeit der Datentypen ermöglicht zudem eine einheitliche Struktur von gleichen Daten. Darüber hinaus wird durch die Wiederverwendbarkeit und die zentrale Verwaltung der Daten der große

[8] Vgl. Wellenreuther/Zastrow (2008), S. 37.

[9] Vgl. Division Digital Factory Siemens (Hrsg.) (2017a), S. 51 ff.

[10] Vgl. Division Digital Factory Siemens (Hrsg.) (2017a), S. 274.

Vorteil geschaffen, dass bei Änderungen oder Erweiterungen des Datentyps eine Aktualisierung automatisch bei allen Zugriffen des Datentyps erfolgen kann.

Datentypen eignen sich außerdem hervorragend zur Realisierung von Schnittstellenparametern. Dateneinheiten können als eine gesamte Einheit übergeben werden, was unter anderem einen Datenaustausch zwischen Steuerung und Visualisierung vereinfacht. Darüber hinaus ist ebenso eine Verwaltung der PLC-Datentypen wie auch der von Funktionen und Funktionsbausteinen durch Bibliotheken möglich, wodurch sich Datentypen und Bausteine zentral organisieren lassen.

2.2.3 Bibliotheken

TIA Portal bietet die Möglichkeit Programmelemente wie FCs, FBs oder PLC-Datentypen mit Bibliotheken zu verwalten. Jedes TIA Portal-Projekt verfügt über eine eigene Projektbibliothek, zu der noch weitere globale Bibliotheken zur Verfügung stehen. Projektbibliotheken sind an ein Projekt gebunden und können nur im Projekt verwendet werden. Globale Bibliotheken hingegen können projektübergreifend angewendet werden. Die Bibliotheken sind untereinander kompatibel, wodurch ein Kopieren oder Verschieben der Elemente ermöglicht wird.[11] Grundsätzlich wird zwischen der im Folgenden beschriebenen Projektbibliothek und den drei verschiedenen globalen Bibliotheken unterschieden:[12]

- *Projektbibliothek*

Diese Bibliothek ist in das Projekt eingebunden und wird gemeinsam mit dem Projekt geöffnet, geschlossen und auch gespeichert. Die Projektbibliothek wird zur Verwaltung der benötigten bzw. der innerhalb des Projektes eingesetzten Programmelemente angewandt.

- *Systembibliothek*

Die Systembibliothek stellt eine globale Bibliothek dar, welche durch Siemens für eigene Software-Produkte zur Verfügung gestellt wird. Diese enthält Funktionsbausteine und Funktionen für bestimmte Anwendungen, welche innerhalb eines Projektes angewandt, jedoch nicht verändert bzw. bearbeitet werden können.

- *Unternehmensbibliothek*

Unternehmensbibliotheken müssen an einem zentralen Ort, wie einem Netzlaufwerk, zur Verfügung gestellt werden. Die Verwaltung der Bibliothek wird durch TIA Portal automatisiert vorgenommen. Wird eine neue Version der Unternehmensbibliothek freigegeben, wird der Anwender aufgefordert, die Bibliothek auf die letzte Version upzudaten.

- *Anwenderbibliothek*

Anwenderbibliotheken zählen ebenfalls zu den globalen Bibliotheken und sind im Unterschied zur Projektbibliothek an kein Projekt gebunden. Aus diesem Grund können diese auch zwischen Anwendern weitergegeben werden bzw. ist es mittels Netzlaufwerk möglich, einen gemeinsamen Zugriff einzurichten.

[11] Vgl. Division Digital Factory Siemens (Hrsg.) (2017b), S. 7.

[12] Vgl. Division Digital Factory Siemens (Hrsg.) (2017b), S. 7 f.

2.2.3.1 Bibliothekselemente

Grundlegend bestehen die Projekt- wie auch die globalen Bibliotheken aus den Bibliothekselementen Typen und Kopiervorlagen. Als Typen werden Objekte bezeichnet, welche für die Abarbeitung des Programmes benötigt werden. Darunter versteht man Bausteine wie Funktionsbausteine aber auch Funktionen, PLC-Datentypen und zusätzlich Visualisierungselemente wie HMI-Anwenderdatentypen und Bildbausteine. All diese Typen können durch die Verwaltung der Bibliotheken versioniert werden, was eine Weiterentwicklung vereinfacht und darüber hinaus eine projektweite Aktualisierung von neuen Versionen ermöglicht.[13]

Im Gegensatz zu Typen können Kopiervorlagen nicht versioniert werden. Dennoch ist es möglich, beinahe jedes Objekt in einer Kopiervorlage abzuspeichern und diese wieder in ein Projekt einzufügen. Kopiervorlagen bieten unter anderem die Möglichkeit, häufig eingesetzte Elemente wie Textlisten, Meldeklassen, Variablen oder ganze Variablentabellen und weitere Elemente standardisiert zu erzeugen. Dabei können die Objekte der Vorlage aus mehreren Elementen zusammengesetzt werden. Im Zusammenhang mit der Liftsteuerung könnte so zum Beispiel der aufrufende Baustein, welcher aus mehreren Bausteinen gebildet wird, als Kopiervorlage hinterlegt werden und somit einen weiteren Einsatz des Bausteines schnell ermöglichen.

Damit ein komfortables Arbeiten sowie auch eine professionelle Weiterentwicklung des Anwenderprogrammes ermöglicht wird, ist es zielführend, eine umfangreiche Bibliothek aufzubauen, welche von mehreren Anwendern eingesetzt werden kann. Aus diesen Gründen empfiehlt sich die Einrichtung einer Unternehmensbibliothek. Eine Unternehmensbibliothek stellt eine durch einen Administrator verwaltete globale Bibliothek dar. Unter der Verwaltung wird das Zuweisen von neuen bzw. das Ändern von bestehenden Bibliotheken verstanden, welche durch die Freigabe automatisch auf die neue Version aktualisiert bzw. neue Bibliotheken in das TIA Portal hochgeladen werden.

2.2.3.2 Einbindung der Unternehmensbibliothek

Die Unternehmensbibliothek kann in einem beliebigen Verzeichnis auf einem Netzlaufwerk aber auch auf der Festplatte eines Computers abgelegt werden. Die Einbindung der Unternehmensbibliothek ins TIA Portal erfolgt durch eine XML-Datei, in welcher die Verzeichnisse und Namen der Bibliothek eingetragen sind. Zusätzlich muss diese Datei mit der Bezeichnung *CorporateSettings.xml* unter dem Pfad *C:\ProgramData\Siemens\Automation\Portal V14\CorporateSettings* abgelegt werden. Die Konfigurationsdatei wird während des TIA Portal-Starts überwacht. Werden Änderungen in dieser Datei festgestellt, wird der Anwender zu einer Aktualisierung der Bibliothek aufgefordert.[14]

Nachdem eine neue Unternehmensbibliothek erstellt wurde, muss die Konfigurationsdatei dahingehend erweitertet werden. Abbildung 4 zeigt hierzu das XML-Konfigurationsfile. Die Zeilen neun und zehn verweisen auf den Ablagepfad der aktuell in der Konfigurationsdatei eingetragenen globalen Bibliotheken.

[13] Vgl. Division Digital Factory Siemens (Hrsg.) (2017b), S. 7.

[14] Vgl. Division Digital Factory Siemens (Hrsg.) (2017b), S. 44 ff.

```
<?xml version="1.0" encoding="utf-8"?>
<Document>
<Settings.Settings ID="0">
 <ObjectList>
  <Settings.General ID="1" AggregationName="General">
   <AttributeList>
    <CorporateLibraryPaths>
     <!-- Beispiel für zwei Einträge -->
     <Item>E:\#Masterarbeit\Lib\LibMain\Main.a114</Item>
     <Item>Z:\MasterarbeitLib\MasterLib.a114</Item>
     <!-- Tragen Sie hier gegebenenfalls alle weiteren globalen Bibliotheken ein. -->
    </CorporateLibraryPaths>
   </AttributeList>
  </Settings.General>
 </ObjectList>
</Settings.Settings>
</Document>

```

Abbildung 4: Unternehmensbibliothek Konfigurationsdatei, Quelle: Division Digital Factory Siemens (Hrsg.) (2017b), S. 46. (leicht modifiziert).

Werden Änderungen in den Bibliotheken vorgenommen, so wird beim Öffnen eines TIA Portal Projektes die in der Abbildung 5 dargestellte Meldung ausgegeben und der Anwender damit aufgefordert die Bibliothek zu aktualisieren.

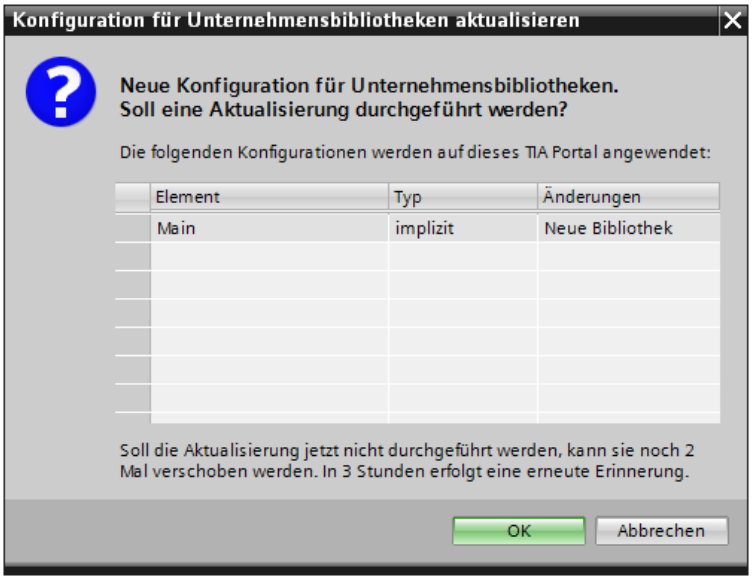


Abbildung 5: Aktualisierung der Unternehmensbibliothek, Quelle: Eigene Darstellung.

Im Gegensatz zu Anwenderbausteinen müssen Bausteine der Sicherheitstechnik nicht in eigenen Bibliotheken verwaltet werden. Zur Abbildung der Sicherheitstechnik in der SPS werden spezielle fehlersichere Bausteine durch das Optionspaket Simatic Safety bereitgestellt. Durch die Anwendung dieser Bausteine können Systeme so ausgeführt werden, dass das Risiko für Mensch und Maschine, wie im nachfolgenden Unterkapitel dargestellt, minimiert wird.

2.3 Sicherheitstechnik

Für den Personenschutz ist es unumgänglich Maschinen mit einer entsprechenden Sicherheitstechnik auszustatten. Gerade die Anwendung eines automatisierten Liftes, welcher mit hoher Geschwindigkeit

Fahrbefehle ausführt, und die Kombination mit einem unübersichtlichen Regal stellen ein hohes Risiko dar. Deswegen muss sichergestellt werden, dass ein Betrieb des Liftes nur bei geschlossenen Sicherheitsbereichen möglich ist. Des Weiteren muss dafür gesorgt werden, unerlaubten Zutritt zum Lift zu unterbinden. Sollte es dennoch dazu kommen, muss ein sofortiges Anhalten des Liftes gewährleistet werden. Um Anforderungen dieser Art erfüllen zu können, wird die gesamte Sicherheitstechnik durch ein eigenes Sicherheitsprogramm in der SPS in Verbindung mit speziellen fehlersicheren Komponenten umgesetzt.

Die Programmierung der Sicherheitstechnik erfolgt durch eigene fehlersichere Bausteine in der Steuerung. Die Logik dieser Bausteine wertet Signale der Sicherheitstechnik aus und gibt über Freigabesignale die Antriebstechnik und andere Betriebsmittel, wie die Fahrbahnklappe, frei. Die Hauptkomponenten werden durch den Liftantrieb sowie die Not-Halt Kreise und den jeweiligen Zutrittstüren gebildet. Ein Bewegen des Liftes ist nur dann möglich, wenn der Not-Halt Kreis des Liftes quittiert ist und die Zutrittstüren des Liftes verriegelt sind. Wird eines dieser Kriterien nicht erfüllt, stoppt der Lift sofort und kann erst nach der Wiederherstellung der Betriebssicherheit, sprich Not-Halt quittiert und Schutzbereich in Ordnung, bewegt werden.

Die für den sichereren Betrieb des Liftes benötigten Sicherheitsfunktionen werden mit Simatic Safety realisiert, damit die Anlage bei einem Eintreten von gefahrbringenden Ereignissen in einen sicheren Zustand gebracht bzw. in einem solchen gehalten werden kann. Simatic Safety setzt sich aus einem sicherheitsgerichteten Anwenderprogramm, welches umgangssprachlich als Sicherheitsprogramm bezeichnet wird, und einer fehlersicheren Peripherie (F-Peripherie) zusammen. Die Realisierung des Sicherheitsprogrammes und das Einlesen und Ausgeben der F-Peripherie ist nur mit einer fehlersicheren Steuerung (F-CPU) möglich.

Die F-Peripherie gewährleistet, dass Eingangsinformationen wie Not-Halt oder Türkontaktschalter fehlersicher für die Verarbeitung im Anwenderprogramm zur Verfügung stehen. Die Programmierung des Sicherheitsprogrammes kann nur in den Programmiersprachen Funktionsplan und Kontaktplan erfolgen. Zusätzlich erfolgt eine Strukturierung des Sicherheitsprogrammes in ein oder mehrere fehlersichere Ablaufgruppen (F-Ablaufgruppen). Die Gruppen enthalten das Anwenderprogramm, aber auch durch das System automatisch generierte fehlersichere Bausteine (F-Bausteine). Die Abbildung 6 zeigt die schematische Zusammensetzung einer F-Ablaufgruppe.

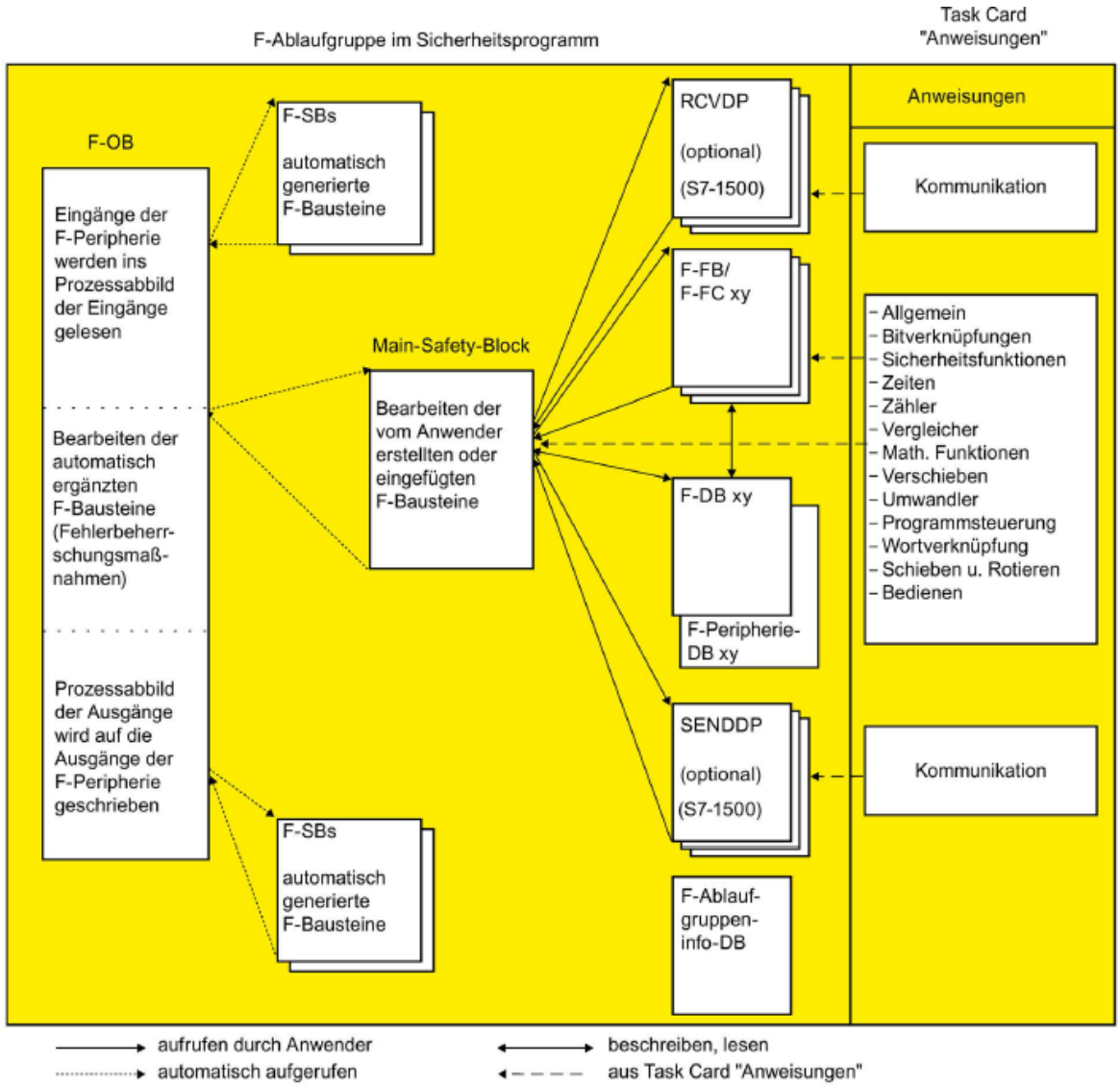


Abbildung 6: Schematischer Aufbau einer F-Ablaufgruppe Quelle: Siemens (2017c), Online-Quelle [4.12.2017], S. 106.

Die derzeitigen Umsetzungsvarianten des Anwender- sowie auch des Sicherheitsprogrammes, wie im folgenden Unterkapitel angeführt, können den Anforderungen eines übersichtlichen und nachvollziehbaren Steuerungsprogrammes nicht gerecht werden. Daher ist eine Überarbeitung des Konzeptes notwendig.

2.4 Aktuelle Programmierung

Derzeit befinden sich zwei unterschiedliche Programmiervarianten im Einsatz. Die erste Variante wird durch die ursprüngliche Programmierung des alten Systems mit einer Siemens 300er-Steuerung abgebildet. Diese Programmierung stellt eine sehr aufwendige, komplexe und kaum nachvollziehbare Programmiermethode dar. In diesem Ansatz erfolgt der Großteil der Programmierung in SCL-Quellen, welche wiederum im Programm instanziiert werden. Der große Nachteil an diesem Verfahren stellt die Programmnachverfolgung dar, da Quellen es nicht ermöglichen Querverweise des Programmes anzuzeigen, was wiederum dazu führt, dass der Programmaufbau schwer nachvollzogen werden kann. Daraus leitet sich zugleich der nächste schwerwiegende Nachteil ab, denn das Programm hat keine einheitliche Struktur. Je nach Anwendungsfall ist das Programm unterschiedlich aufgebaut, was wiederum

zu Folge hat, dass die Programmierung meist nur vom Ersteller richtig gelesen werden kann. Somit können Änderungen und Fehlerbehebungen oft nur unter hohem Risiko durchgeführt werden. Außerdem wird durch die Anwendung von Quellen und deren Instanziierungen vielfach auf indirekte Adressierungen und direkte Schreib- und Lesezugriffe der Instanzvariablen zurückgegriffen. Eine derartige Programmierung stellt ein hohes Gefahrenpotenzial bei Programmänderungen von laufenden Anlagen dar. Werden unter anderem Instanz-Datenbausteine geändert, muss das gesamte Anwenderprogramm übersetzt und neu in die Steuerung übertragen werden, da es dazu kommen kann, dass sich Zugriffe verändern und somit eine korrekte Abarbeitung nicht mehr gewährleistet werden kann.

Beim Wechsel auf die neue Steuerungsgeneration von Siemens wurde beschlossen, die Programmierung komplett zu überarbeiten. Dabei wurde gänzlich auf die Programmierung von Quellen verzichtet. Der Aufbau erfolgt modular, somit wird bereits hier eine Art von Objekten geschaffen. Alle Programmteile, welche zur Steuerung des Liftes benötigt werden, werden in eigenen Bausteinen zusammengefasst. Dennoch führte dieser Entwicklungsschritt noch nicht zum gewünschten Ziel, da die Ansteuerung des Liftes und der Feldgeräte zwar durch eigene Bausteine abgebildet, aber der Aufbau dahinter zu komplex realisiert wurde. So werden Steuersignale an verschiedenen Stellen mehrfach beschrieben, was die Programmnachvollziehbarkeit wiederum schmälert.

Da beide Varianten nicht das gewünschte Ziel eines einfach handzuhabenden SPS-Programmes erfüllen, wird ein weiterer Schritt zur Entwicklung der Programmierung getätigt. Dieser soll in Richtung objektorientierter Programmierung durchgeführt werden.

3 OBJEKTORIENTIERTE STEUERUNGSPROGRAMMIERUNG

Vor mehr als 30 Jahren entwickelte sich die objektorientierte Programmierung. Es dauerte jedoch noch einige Zeit, bis OOP auch in der kommerziellen Programmierung Einzug hielt und schlussendlich den Weg zu Automatisierungssystemen fand. Nach wie vor müssen Automatisierungslösungen strukturiert aufgebaut werden, um den Anwender eine gewisse Transparenz der Software zu ermöglichen. Außerdem sollen Anwendungen auch wartungsfreundlich und erweiterbar gestaltet werden. Diese Ziele können mittels strukturierter und grafischer Programmierung erreicht werden, indem jedem Objekt einer Anlage, sei es ein Ventil, ein Antrieb oder dergleichen, ein Objekt in der Software zugewiesen wird. Bereits in der IEC 61131-3:2003 wird die Wiederverwendbarkeit und Objektorientierung von Funktionsbausteinen beschrieben, aber erst durch die IEC 61131-3:2013 wird objektorientierte SPS-Programmierung (OOSP) definiert. Beispiele zur OOSP in der Anlagenprogrammierung sind derzeit jedoch selten.

Durch den Einzug der objektorientierten Steuerungsprogrammierung steht ein Paradigmenwechsel bevor, welcher in den nächsten Jahren einen starken Einfluss auf die Programmierung von Steuerungssystemen nehmen wird. Programmierumgebungen wie CoDeSys unterstützen bereits Teile der OOSP, TIA Portal ermöglicht jedoch eine direkte Anwendung der OOP in der Steuerungsprogrammierung noch nicht. Um die Programmierung in TIA Portal annähernd objektorientiert ausführen zu können stehen derzeit mehrere verschiedene Varianten zur Verfügung. Bevor diese genauer untersucht werden ist es notwendig die Grundlagen der OOP sowie die Definitionen nach IEC 61131-3 zu behandeln.

3.1 Grundlagen der objektorientierten Programmierung

Oft wird die strukturierte Programmierung als Vorreiter der objektorientierten Programmierung bezeichnet, diese Behauptung ist für sich nicht ganz korrekt. Zutreffender ist die Bezeichnung, dass strukturierte wie auch objektorientierte Programmierung fundamentale Paradigmen darstellen, welche gleichberechtigt koexistieren. Bereits Ende der Sechzigerjahre wurde die strukturierte Programmierung entwickelt und kann als eine hierarchische Programmorganisation bezeichnet werden, welche auf logischen Programmeinheiten mit zentraler Steuerung sowie einer begrenzten Datenverfügbarkeit aufsetzt. Das Ziel der strukturierten Programmierung wird dadurch beschrieben, Algorithmen so aufzubauen, dass der Ablauf einfach zu erfassen und änderbar ist. Das bedeutet, es muss eine bestmögliche Anordnung des Codes erzielt werden, damit die Transparenz sowie Testbarkeit und die Wiederverwendbarkeit effizient genützt werden können. Diese Ziele und grundlegenden Eigenschaften der strukturierten Programmierung lassen sich durch die objektorientierte Programmierung abbilden. Dennoch entwickelte sich die OOP erst einige Jahre später. Erst mit dem Beginn des Windows-Zeitalters konnte sich auch die objektorientierte Programmierung etablieren.[15]

Alan Kay, einer der Mitentwickler der objektorientierten Programmierung, definiert diese wie folgt:[16]

[15] Vgl. Doberenz/Gewinnus (2013), S. 156 f.

[16] Ram (2004), Online-Quelle [4.12.2017].

„Ein *Objekt* ist ein gedachter Baustein, der Zustände und Prozeduren enthält und auf den von außen nur durch Zusendung von *Nachrichten* zugegriffen werden kann. Das Objekt entscheidet, wie es (im Rahmen seiner Spezifikation) auf eine bestimmte Nachricht reagiert (Laufzeitmodell).

Bei der *objektorientierten Programmierung* beschreiben Programme, unter welchen Bedingungen zur Laufzeit welche Nachrichten an Objektausdrücke gesendet werden: Hierzu gibt es eine Versendungsbestimmung, die den Empfängerobjektausdruck und die zu versendende Nachricht festlegt. Diese Versendungsbestimmung kann auch insgesamt als ein Ausdruck gelten, dessen Wert dann vom Empfängerobjekt (als eine Art von Antwort) festgelegt wird (Quelltextmodell).

Es muß möglich sein, daß erst spätestmöglich (also zur Laufzeit während der Auswertung der Versendungsbestimmung) festgelegt wird, welches Objekt eine bestimmte Nachricht erhält (späte Bindung): Hierzu kann das Empfängerobjekt in der Versendungsbestimmung selber wieder durch einen Ausdruck angegeben werden, der erst spätestmöglich zur Laufzeit ausgewertet wird (Laufzeit-Polymorphie).“

Um ein besseres Verständnis der Definition von Alan Kay, aber auch der objektorientierten Programmierung zu erlangen, erweist es sich als sinnvoll, die grundlegenden Begriffe der OOP zu betrachten, welche im Nachfolgenden charakterisiert werden:[17]

- *Objekt*

Ein Objekt bezeichnet die Kapselung bzw. Zusammenfassung von Daten und Funktionalitäten, welches verwendet wird, um Objekte des Alltages zur Datenverarbeitung abzubilden. Objekte stellen Gruppen dar, welche Eigenschaften und Methoden besitzen die logisch zusammengehören und auch manipuliert werden können. Zudem besitzen sie auch Ereignisse, auf welche im Code reagiert werden kann.

- *Klasse*

Jedes Objekt benötigt einen Bauplan, dieser wird durch die Klasse abgebildet. Auf der Grundlage dieses Bauplanes wird zur Programmlaufzeit das Objekt generiert. Mit der Klasse wird definiert, wie das Objekt aussieht und wie es sich verhält. Eine Klasse bezeichnet somit eine Softwarekonstruktion, welche die Definition der Eigenschaften, Methoden wie auch der Ergebnisse vornimmt, ohne dass dabei ein Objekt erzeugt wird.

- *Instanz*

Erst mit der Instanziierung der Klasse wird ein Objekt erzeugt, wobei mehrere Objekte mit einer einzigen Klassendefinition erstellt werden können.

[17] Vgl. Doberenz/Gewinnus (2013), S. 157 ff.

- *Kapselung*

Mit der Kapselung von Objekten wird die Möglichkeit geschaffen, die Implementierung der Klasse sauber von der Schnittstelle abzutrennen. Durch dieses Vorgehen werden die innere Struktur und deren Daten sowie auch Methoden geschützt. Über die Schnittstelle können Manipulationen des Objektes erfolgen, sofern dafür öffentliche Methoden zur Verfügung gestellt werden.

- *Wiederverwendbarkeit*

Klassen schaffen die Opportunität der Wiederverwendbarkeit, das bedeutet, eine Klasse kann mehrfach in verschiedenen Teilen der Programmierung verwendet werden. Daraus folgend reduziert sich der Wartungsaufwand, aber auch der redundante Code einer Anwendung.

- *Vererbung*

Die echte Vererbung, auch Implementierungsvererbung genannt, erlaubt es Klassen von anderen Klassen abzuleiten. Dabei werden die Schnittstelle und der dazugehörige Code an die abgeleitete Klasse vererbt. Somit kann eine Klasse, die Oberklasse, als Basis für weitere Klassen, die Unterklassen, dienen. Mittels Vererbung werden Unterklassen erzeugt, welche die Eigenschaften und Methoden der Oberklasse erhalten. Darüber hinaus können in der Unterklasse zu den erhaltenen Eigenschaften der Oberklasse weitere hinzugefügt oder auch die der Oberklasse überschrieben werden.

- *Polymorphie*

Polymorphie bezeichnet die Vielgestaltigkeit einer Methode. Hierbei wird eine Methode für verschiedene Objekte angewandt, wobei die Ausführung für jedes Objekt unterschiedlich erfolgen kann. Das bedeutet, durch die Polymorphie wird die Möglichkeit geschaffen, Eigenschaften und Methoden einer Klasse mit unterschiedlichen Ausführungen (Implementierungen) aufzurufen.

Verschiedenen Programmiersprachen wie C++, Java oder Borland Delphi, bauen auf der OOP auf. Die Steuerungsprogrammierung hingegen verwendet herstellerabhängig eigene Programmierumgebungen mit welchen Programmiersprachen realisiert werden, die in der nachfolgend angeführten Norm IEC 61131-3 definiert sind.

3.2 Programmierung nach IEC 61131

Bereits 1977 wurde mit der IEC 848 der erste Grundstein zur Entstehung der IEC 61131 gelegt. Heute werden durch diese Norm eine Zusammenfassung sowie Fortschreibung von verschiedenen Normen dargestellt. Die IEC 61131 selbst stellt die erste international und industriell akzeptierte Norm zu Speicher-Programmierbaren Steuerungen dar. Der Inhalt erstreckt sich über Verschriften zu Zeichencodes bis hin zum Aufbau von grafischen Programmdarstellungen und unterteilt sich, wie in Tabelle 2 gezeigt, derzeit in neun Teile.[18]

[18] Vgl. John/Tiegelkamp (2008), S. 14.

Teil	Bezeichnung	Beschreibung
1	Allgemeine Informationen	Allgemeine Begriffe sowie Funktionsbeschreibung zur Unterscheidung von Speicher-Programmierbaren Steuerungen zu anderen Systemen
2	Betriebsmittelanforderungen und Prüfungen	Definiert die Anforderungen an die Hardware und an die Peripheriegeräte
3	Programmiersprachen	Behandelt die grundlegende Software-Architektur und Programmiersprachen zur Programmierung von SPSen
4	Anwenderrichtlinien	Leitfaden zur Unterstützung des SPS-Anwenders während der Projektphase
5	Kommunikation	Beschreibt den Signalaustausch von SPSen untereinander sowie zu anderen Geräten
6	Funktionale Sicherheit	Wie DIN EN 61131-1, Definition der Anforderungen von SPS-en und den Peripheriegeräten, welche jedoch als Teil eines sicherheitstechnischen Systems vorgesehen sind.
7	Fuzzy-Control-Programmierung	Beschreibt eine Sprache zur Programmierung von Fuzzy-Control-Applikationen mit SPSen
8	Richtlinie zur Anwendung und Implementierung von Programmiersprachen	Technischer Bericht zur Anwendung und Umsetzung der in Teil 3 definierten Programmiersprachen
9	IO-Link	Beschreibt die Technologie zur digitalen Punkt-zu-Punkt Schnittstelle für Sensoren und Aktoren.

Tabelle 2: Teile der IEC 61131, Quelle: John/Tiegelkamp (2008), S. 15 f.

Aus Tabelle 2 kann entnommen werden, dass die Beschreibung der Programmierung in Teil 3 – Programmiersprachen der Norm erfolgt. Dieser Teil der IEC 61131 spezifiziert die Syntax sowie die Semantik für eine einheitliche Programmiersprache für SPSen.

Einer der wichtigen Vorreiter der heutigen ICE 61131-3 wird durch die DIN 19239 - Messen, Steuern, Regeln; Steuerungstechnik; Speicherprogrammierte Steuerungen; Programmierung dargestellt. Die Norm definiert die Programmierung von SPSen mittels der Programmiersprachen Funktionsplan, Kontaktplan und Anweisungsliste. Zudem wurden bereits in dieser Norm unterschiedliche Bausteintypen, wie in Abbildung 7 dargestellt, definiert. Die DIN 19239 wurde im Jahr 1994 mit der Einführung der IEC 61131-3 abgelöst, und somit auch die Definition der Bausteintypen angepasst.

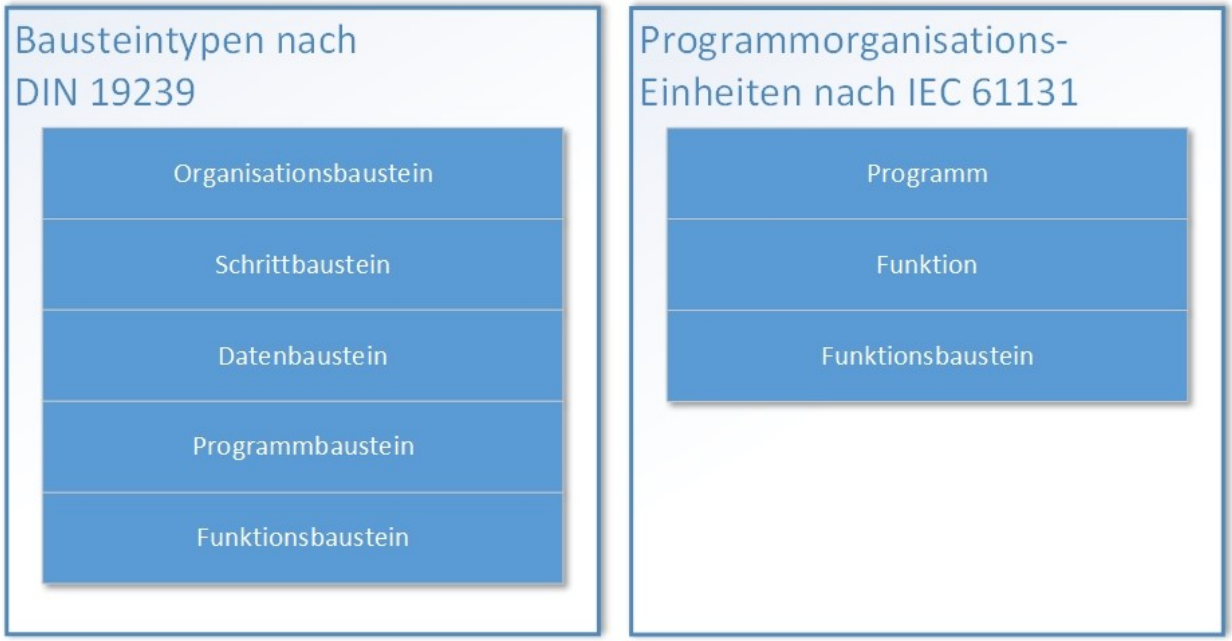


Abbildung 7: Bausteintypen nach DIN 19239 und IEC 61131, Quelle: John/Tiegelkamp (2008), S. 30. (leicht modifiziert).

Wie aus Abbildung 7 hervorgeht, werden durch die DIN 19239 die fünf definierten Bausteine in der IEC 61131 auf drei reduziert. Die Programmorganisationseinheit beschreibt, wie bereits in Abschnitt 2.2.2 angeführt, jene Bausteine, welche für den Projektaufbau zur Verfügung stehen und unterscheidet die nachfolgenden POE-Typen:

- *Funktion*

Stellt eine programmierbare Einheit ohne Gedächtnis dar. Das bedeutet, der Baustein verfügt über keine statischen Variablen. Daraus resultiert, dass der Baustein bei gleichen Zuständen der Eingangsparameter das gleiche Ergebnis als Funktionswert liefert.[19]

- *Funktionsbaustein*

Im Gegensatz zur Funktion verfügt der Funktionsbaustein über statische Variablen und somit auch über ein Gedächtnis. Folglich liefert ein FB bei gleichen Zuständen der Eingangsparameter abhängig von den internen Variablen unterschiedliche Ergebnisse als Ausgangsparametern. Der Funktionsbaustein entspricht somit auch den herkömmlichen FB aus der DIN 19239.[20]

- *Programm*

Das Programm stellt in der POE die oberste Ebene dar. Diverse Systeme setzen diesen Typ als Hauptprogramm zur Organisation des Anwenderprogramms ein. Siemens Steuerungen organisieren das Programm zum Beispiel durch Organisationsbausteine, welche die Aufgabe haben, Aufrufe der Funktionen und Funktionsbausteine auszuführen. Die Umsetzung erfolgt identisch zu der eines Funktionsbausteines.[21]

[19] Vgl. John/Tiegelkamp (2008), S. 31.

[20] Vgl. John/Tiegelkamp (2008), S. 31.

[21] Vgl. Wellenreuther/Zastrow (2008), S. 20.

Der grundlegende Aufbau der POE-Typen wird in der nachfolgenden Abbildung 8 veranschaulicht. Eine POE definiert sich somit über die Angabe des POE-Typs und Namen sowie dem Deklarationsteil mit der Variablendeklaration und dem abschließenden Anweisungsteil, welcher als POE-Rumpf bezeichnet wird.

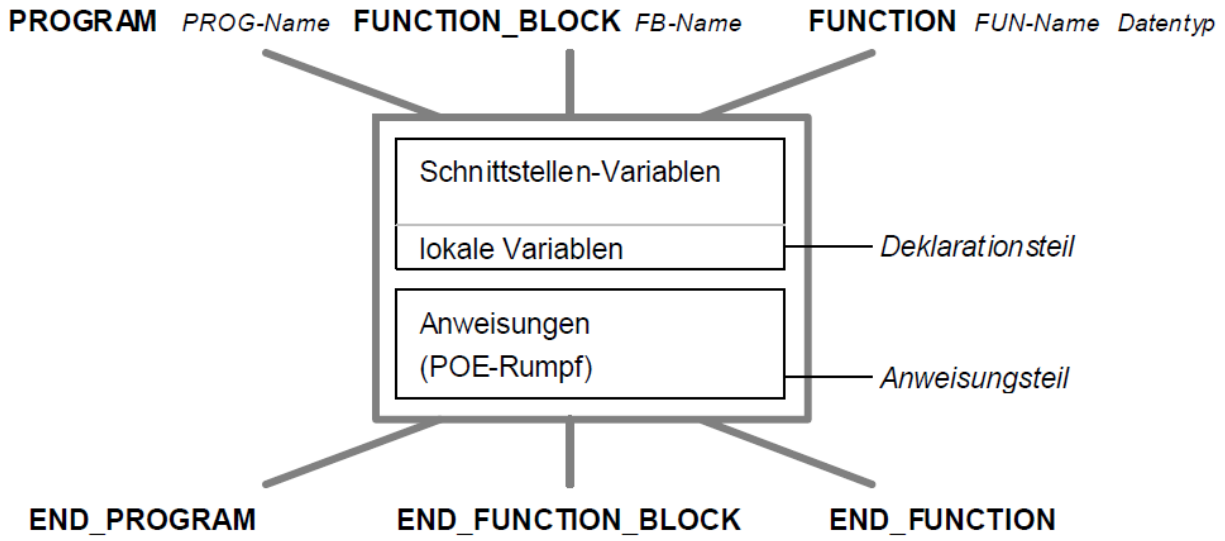


Abbildung 8: Aufbau der POE – Typen nach IEC 61131-3, Quelle: John/Tiegelkamp (2008), S. 32.

Die IEC 61131-3 bildet keine direkte Vorschrift für Anwender sowie auch nicht für SPS-Hersteller, vielmehr ist dieser Teil der Norm als eine Art Richtlinie anzusehen und bildet so einen Maßstab zur Orientierung, inwieweit sich die Programmierung an die Norm hält. Darüber hinaus wird in den ersten Ausführungen der Norm die objektorientierte Programmierung nicht behandelt. Erst mit der Ausführung IEC 61131:2003 wird dieses Thema bezüglich Wiederverwendbarkeit und Objektorientierung von Funktionsbausteinen erwähnt. Mit der Ausgabe 2013 fand die objektorientierte Programmierung ebenfalls einen ausführlichen Einzug in die IEC 61131-3. In den nachfolgenden Unterkapiteln werden die Ausführungen nach Ausgabe 2003 und 2013 genauer betrachtet, um für das Kapitel 4 ein Grundverständnis zu schaffen.

3.3 Objektorientierte Programmierung nach IEC 61131-3:2003

Diese Ausgabe der Norm beinhaltet eine Beschreibung der Wiederverwendbarkeit und Objektorientierung von Funktionsbausteinen. Bezüglich objektorientierter Programmierung kann jedoch nur ein Ansatz geschaffen werden. Das zentrale Element wird durch den Funktionsbaustein gebildet, welcher ein fundamentales Hilfsmittel zur Programmstrukturierung darstellt. Einen weiteren bedeutsamen Aspekt repräsentiert die Instanziierung.

Durch eine Instanz wird eine Art Gedächtnis erzeugt. Mittels der mehrfachen Verwendung von Funktionsbausteinen und deren Instanziierung wird eine Datenkopie im Speicher des Instanz-Datenbausteines erstellt, in welcher eine Kopie aller Eingangs- und Ausgangsparameter sowie auch der statischen (lokalen) Variablen hinterlegt wird, siehe hierzu Abbildung 9. Dabei handelt es sich um fest zugeordnete Speicherbereiche, welche als statisch deklariert sind.[22]

[22] Vgl. John/Tiegelkamp (2008), S. 46.

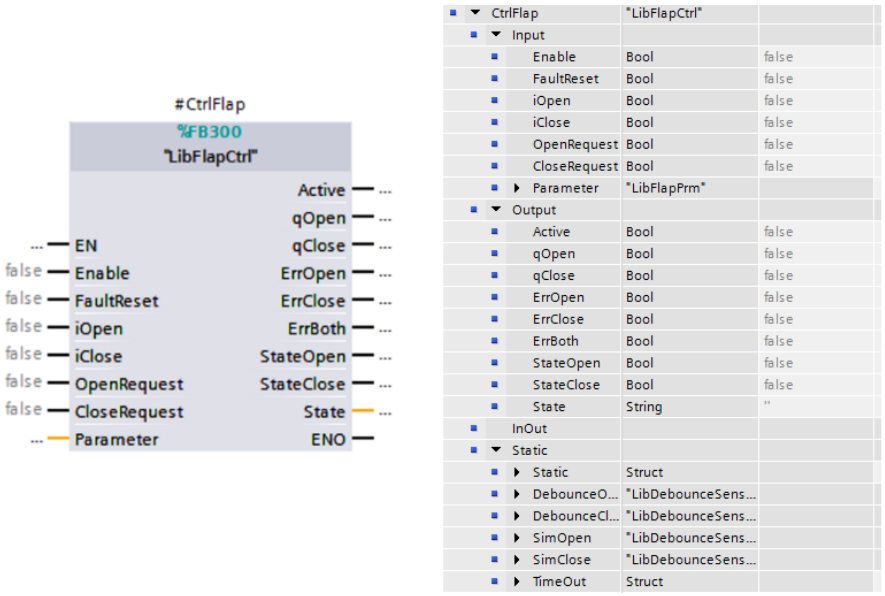


Abbildung 9: Funktionsbaustein mit Instanz-Daten Auszug, Quelle: Eigene Darstellung.

Die Instanziierung legt somit den Grundstein der Wiederverwendbarkeit von Funktionsbausteinen. Wiederum unterliegen diese gewissen Bedingungen, damit eine mehrfache Verwendung überhaupt zustande kommen kann. So ist unter anderem eine feste Zuordnung von lokalen Variablen zu SPS-Adressen nicht gestattet. Darüber hinaus ist auch eine Verwendung von Globalen Variablen in den Funktionsbausteinen unzulässig. Variablen dieser Art können mittels FB-Schnittstelle (Eingangsparameter), welche wiederum auch in der Instanz abgebildet wird, dem Baustein zur Verfügung gestellt werden. Zudem bietet diese Anwendung keine Möglichkeit der Vererbungseigenschaft, wodurch ein grundlegendes Prinzip der OOP nicht erfüllt werden kann. Wie jedoch zuvor beschrieben, kann ein Funktionsbaustein, wenn ein Variablenzugriff nur über die Schnittstelle gestattet ist, als gekapselt bezeichnet werden. Grundlegend dafür ist die Eigenschaft, dass Daten nur über die definierte FB-Schnittstelle übergeben werden und somit die Funktion des Funktionsbausteines nicht von globalen Variablen oder der Peripherie abhängig ist.[23]

Durch die Programmstrukturierung und den Speicher in Verbindung mit der Wiederverwendbarkeit des Funktionsbausteines stellt der FB eine eigenständige und nach außen hin gekapselte Datenstruktur dar. Zudem verfügt die Datenstruktur, welche der FB-Instanz entspricht, über eine definierte Berechnungsvorschrift, welche durch den Anweisungsteil des Funktionsbausteines dargestellt wird. Die FB-Instanz kann im Gegensatz zu normalen Datenstrukturen aufgerufen werden, woraus resultiert, dass ein FB beliebig oft mit eigenen Instanzen gebildet werden kann, ohne dass sich diese dabei gegenseitig beeinflussen, da jeder Instanz ein eindeutiger Name sowie Speicherbereich zugeordnet werden muss. Aufgrund dessen können Funktionsbausteine nach IEC 61131-3 auch als objektorientiert bezeichnet werden, jedoch kann diese Definition nicht mit den Sprachmitteln, über welche moderne objektorientierte Programmiersprachen wie C# verfügen, gleichgestellt werden.[24]

[23] Vgl. John/Tiegelkamp (2008), S. 48.

[24] Vgl. John/Tiegelkamp (2008), S. 48 ff.

Beispielhafte Vorgehensweisen zur Anwendung und Umsetzung dieser Methode werden in Unterkapitel 4.1 ausgeführt. Zuvor wird jedoch noch eine weitere Option der Programmierung behandelt, um auch die Möglichkeit der objektorientierten SPS-Programmierung nach IEC 61131-3:2013 aufzuzeigen.

3.4 Objektorientierte Programmierung nach IEC 61131-3:2013

Aufbauend auf dem Ansatz der Wiederverwendbarkeit und Objektorientierung von Funktionsbausteinen der Norm IEC 61131-3:2003 wird in der IEC 61131-3:2013 diese Zugangsweise erweitert, um somit das Paradigma der Objektorientierung abzubilden. Hierzu wird auf ein Konzept zurückgegriffen, welches für Klassen definiert wurde. Bevor weiter auf die objektorientierten Eigenschaften von Funktionsbausteinen eingegangen werden kann, ist es notwendig, das Konzept der Klasse der IEC 61131-3:2013 zu erörtern.

In der IEC 61131-3:2013 stellt die Klasse eine Programmorganisationseinheit dar, welche die objektorientierten Paradigmen unterstützt. Die Klasse definiert eine Datenstruktur, welche in interne und öffentliche Variablen aufgeteilt ist. Aufbauend auf den Elementen dieser Struktur werden Methoden ausgeführt. Um Methoden der Klasse aufrufen oder auf Variablen zugreifen zu können, muss diese zuvor instanziiert werden. Dabei ist zu beachten, dass eine Klasse keinen herkömmlichen Funktionsbaustein darstellt, sondern sich in mehreren Punkten von diesem unterscheidet. So werden Klassen mit den Schlüsselwörtern `CLASS` und `END_CLASS` definiert. Außerdem sind die von Funktionsbaustein bekannten Variablenbereiche *VAR_INPUT*, *VAR_OUTPUT* sowie *VAR_IN_OUT* und *VAR_TEMP* für Klassen nicht zulässig, stattdessen erfolgt die Deklaration in einer *VAR*-Sektion. Zudem besitzt die Klasse selbst keinen Rumpf (Anweisungsteil – siehe Abbildung 8), und kann dadurch nur Methoden definieren. Außerdem ist es nicht möglich, die Instanz einer Klasse aufzurufen, nur Methodenaufrufe der Klasse können umgesetzt werden.[25]

Da der Ansatz des objektorientierten Funktionsbausteines (OOFB) auf die Definition der Klasse zurückzuführen ist, werden auch bei OOFB Methoden angewandt. Im Gegensatz zur Klasse kann der FB jedoch über einen Rumpf und auch über Methoden verfügen. Die folgenden drei Typen des Funktionsbausteines können demnach unterschieden werden:[26]

- *Funktionsbaustein nur mit Rumpf*

Diese Art des Funktionsbausteines stellt den herkömmlichen FB dar, welcher durch die IEC 61131-3:2003 definiert ist.

- *Funktionsbaustein mit Rumpf und Methoden*

In einem derartigen FB müssen Methoden einen Zugriff auf ihre eigenen lokalen Variablen, aber auch auf die Variablen des Funktionsbausteines, welche in den Sektionen *VAR_INPUT*, *VAR_OUTPUT*, *VAR_IN_OUT* und *VAR_TEMP* definiert sind, verfügen.

[25] Vgl. EN 61131-3:2013 (2014), S. 132 ff.

[26] Vgl. EN 61131-3:2013 (2014), S. 160 ff.

- *Funktionsbaustein nur mit Methoden*

Werden nur Methoden im Funktionsbaustein definiert, so besitzt der FB einen leeren Rumpf, somit kann der Funktionsbaustein wie eine Klasse deklariert werden.

Im nachfolgenden Kapitel soll aufgezeigt werden, mit welchen Methoden derzeit objektorientierte Programmierung mit Siemens TIA Portal unter Betrachtung der Normen IEC 61131-3:2003 und IEC 61131-3:2013 ermöglicht werden kann.

4 OBJEKTORIENTIERTE PROGRAMMIERUNG MIT SIEMENS TIA PORTAL

Zuvor wurde bereits darauf verwiesen, dass TIA Portal OOP noch nicht unterstützt. Daraus muss aber nicht zwingend abgeleitet werden, dass ein SPS-Programm nicht objektorientiert aufgebaut werden kann. Hierfür bietet es sich an, die drei nachfolgend beschriebenen Methoden genauer zu betrachten. Die Methoden *OOP mit Funktionsbausteinen* sowie *OOP mit SCL* stellen Programmieransätze dar, welche keine direkte objektorientierte Programmierung unterstützen. Die erste Variante bezieht sich auf die Definition der IEC 61131-3:2003 und baut auf der Wiederverwendbarkeit und Objektorientierung von Funktionsbausteinen auf. Hingegen nähert sich letztere durch eine mehr oder weniger aufwendige Programmierung an die objektorientierte Programmierung an. Beide Methoden stellen wie im Nachfolgenden beschrieben keine echte OOP dar, hingegen wird durch *OOP mit SIMOTION* die Möglichkeit der OOP mit TIA Portal geschaffen. Dazu wird jedoch wie in Unterkapitel 4.3 beschrieben eine Erweiterung der Software SIMOTION und auch die zugehörige Hardware benötigt.

4.1 OOP mit Funktionsbausteinen

Bereits SIMATIC STEP7 ermöglichte die ersten Schritte in Richtung OOP, beruhend auf der in der IEC 61131:2003 beschriebenen Anwendung der Wiederverwendbarkeit und Objektorientierung von Funktionsbausteinen. Dennoch müssen einige Abstriche gemacht werden, denn auf Vererbung und Überschreibung von Methoden muss gänzlich verzichtet werden, was auch leider derzeit bei TIA Portal der Fall ist. Der Programmaufbau wird modular umgesetzt. Das bedeutet, Anwendungen oder auch Elemente wie Betriebsmittel werden einmal in einem Funktionsbaustein programmiert und können mit Hilfe der Instanziierung immer wieder verwendet werden. Dadurch wird es möglich, mehrere reale Objekte mit ein und demselben Funktionsbaustein zu realisieren.[27]

An dieser Stelle sei jedoch noch angemerkt, dass in diesem Zusammenhang die objektorientierte Programmierung nach der IEC 61131-3:2003 angewandt wird. Dieser Zugang zur OOP kann nur durch die Einhaltung von gewissen Regeln Ansätze der OOP erfüllen. Daraus resultierend kann dieses Vorgehen nicht mit Programmiersprachen wie C# verglichen werden, welche den vollständigen Umfang der OOP bieten.

4.1.1 Wiederverwendung des Funktionsbausteines

Damit eine Wiederverwendung eines Funktionsbausteines erfolgen kann, muss dieser einmal erstellt und die Aufrufe mittels Instanziierung umgesetzt werden. Mit diesem Vorgehen wird, wie schon zuvor beschrieben, eine Kopie des Funktionsbausteines erstellt, welche in dem zugehörigen Instanz-Datenbaustein gespeichert wird. Genauer gesagt wird eine Variable des Datentyps des verwendeten Funktionsbausteines im Speicher des aufrufenden Instanz-Datenbausteines erzeugt. In diesem Speicherbereich liegen die Daten des FBs. Da auf den Speicherbereich zugegriffen werden kann, ist es

[27] Vgl. Braun/Horn (2015), S. 51.

nicht möglich, die Ein- bzw. Ausgangsschnittstellen als öffentlich zu bezeichnen, da auch auf Teile der lokalen Variablen (statische Variablen) direkt über den Instanz-Datenbaustein zugegriffen werden kann. Jedoch stellt das eine Verletzung der Definition für Variablenzugriffe dar. Wie bereits in Unterkapitel 3.3 angemerkt, sollen Variablen nur über die Eingangsschnittstelle eingebracht werden, um damit den Funktionsbaustein als gekapselt betrachten zu können. Daher ist es nicht erlaubt, lesende oder schreibende Zugriffe auf Instanzvariablen außerhalb des Funktionsbausteines auszuführen, hierzu erfolgt unter Kapitel 6 noch eine ausführliche Beschreibung.

In Abbildung 10 wird beispielhaft die Instanziierung des Bausteines zur Ansteuerung der Fahrbahnklappe dargestellt. Aus dem Bezeichner des Bausteines geht bereits hervor, dass es sich hierbei um einen Bibliotheksbaustein handelt, welcher für mehrfache Verwendung mittels Instanziierung vorgesehen ist.

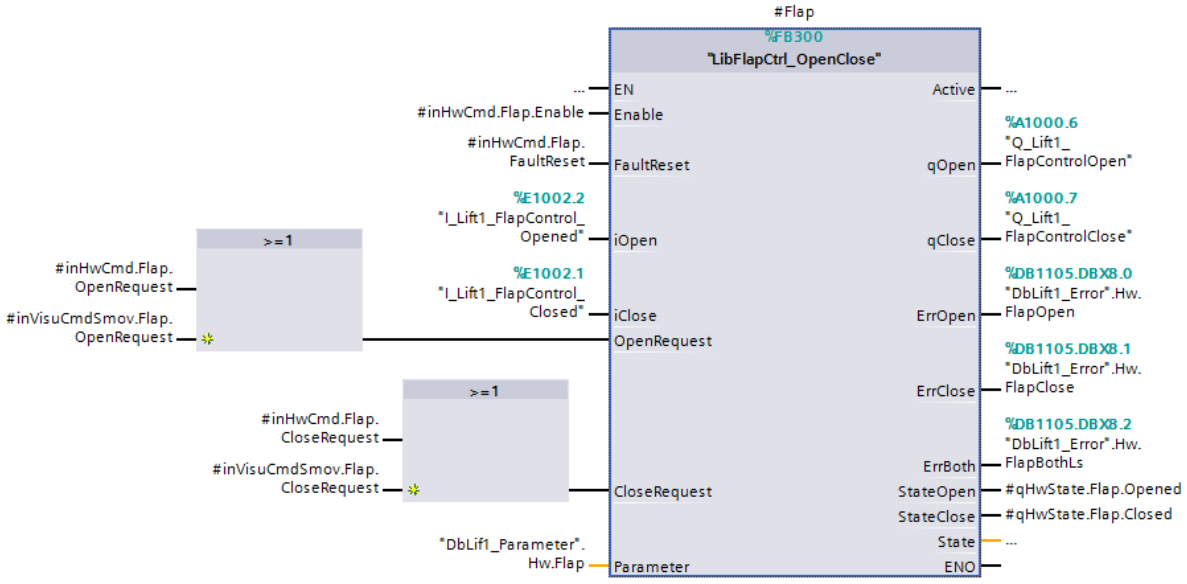


Abbildung 10: Beispielhafte Instanziierung des Fahrbahnklappenbausteines, Quelle: Eigene Darstellung.

Dieser Funktionsbaustein wird in einem übergeordneten Baustein, dem Aufruf-Funktionsbaustein, als Instanz angelegt und somit eine Kopie des Bausteines erstellt. Durch dieses Vorgehen werden die Daten des Aufrufes von der Definition des Bibliotheksbausteines getrennt und im Instanz-Datenbaustein des Aufruf-Funktionsbausteines erzeugt. Die zu dem in Abbildung 10 abgebildeten FB zugehörigen Daten im Instanz-Datenbaustein werden in der Abbildung 11 grafisch aufgelistet. Es ist jedoch nicht zwingend notwendig, dass die Instanziierung durch eine statische Variable in einem übergeordneten Baustein erfolgt, sondern auch eine Instanz mit einem eigenen Instanz-Datenbaustein ist möglich, jedoch erhöht diese Variante die Anzahl an benötigten Bausteinen drastisch.

Flap	"LibFlapCtrl_OpenClose"	11978.0	
Input			
Enable	Bool	11978.0	false
FaultReset	Bool	11978.1	false
iOpen	Bool	11978.2	false
iClose	Bool	11978.3	false
OpenRequest	Bool	11978.4	false
CloseRequest	Bool	11978.5	false
Parameter	"LibFlapPrm_OpenClose"	11980.0	
Output			
Active	Bool	12026.0	false
qOpen	Bool	12026.1	false
qClose	Bool	12026.2	false
ErrOpen	Bool	12026.3	false
ErrClose	Bool	12026.4	false
ErrBoth	Bool	12026.5	false
StateOpen	Bool	12026.6	false
StateClose	Bool	12026.7	false
State	String	12028.0	''
InOut			
Static			
Static	Struct	12284.0	
DebounceOpen	"LibDebounceSensor"		
DebounceClose	"LibDebounceSensor"		
SimOpen	"LibDebounceSensor"		
SimClose	"LibDebounceSensor"		
TimeOut	Struct	12298.0	

Abbildung 11: Beispielhafte Darstellung der Instanz-Daten des Fahrbahnklappenbausteines, Quelle: Eigene Darstellung.

Damit ein Daten- bzw. Befehls- und Statusaustausch zu anderen Softwarekomponenten erfolgen kann, ist es notwendig Schnittstellen zu definieren, welche durch die Ein- und Ausgangsattribute des Funktionsbausteines abgebildet werden. Dadurch entsteht die Möglichkeit das Objekt über eine übergeordnete Schnittstelle wie ein Human Maschine Interface oder durch Schrittkettensignale zu steuern. Ein Zugriff auf diese Schnittstelle ist prinzipiell für alle weiteren Softwarekomponenten möglich, wodurch sich diese zwangsläufig als öffentlicher Parameter darstellt. Die Schnittstelle selbst bildet das Bindeglied zur Logik des Bausteines. Durch die Verknüpfung der Signale und der Bausteinlogik werden bestimmte Funktionen ausgeführt oder Ereignisse ausgelöst. Die Ergebnisse hängen jedoch zusätzlich nicht nur von den Eingangsparametern des Bausteines ab, sondern auch die Zustände der statischen Variablen sind ausschlaggebend für die ausgeführten Ereignisse.

Da die In- und Output-Parameter des Funktionsbausteines zur Instanz gehören, ist eine vollständige Beschaltung des Bausteines nicht notwendig. Das bedeutet, an der Aufrufschnittstelle müssen nicht alle Parameterschnittstellen belegt werden. Durch die Instanziierung kann auch ein Zugriff an einer anderen Stelle erfolgen, hierbei wird, wie beispielhaft in Abbildung 12 gezeigt, direkt auf die Instanz zugegriffen.

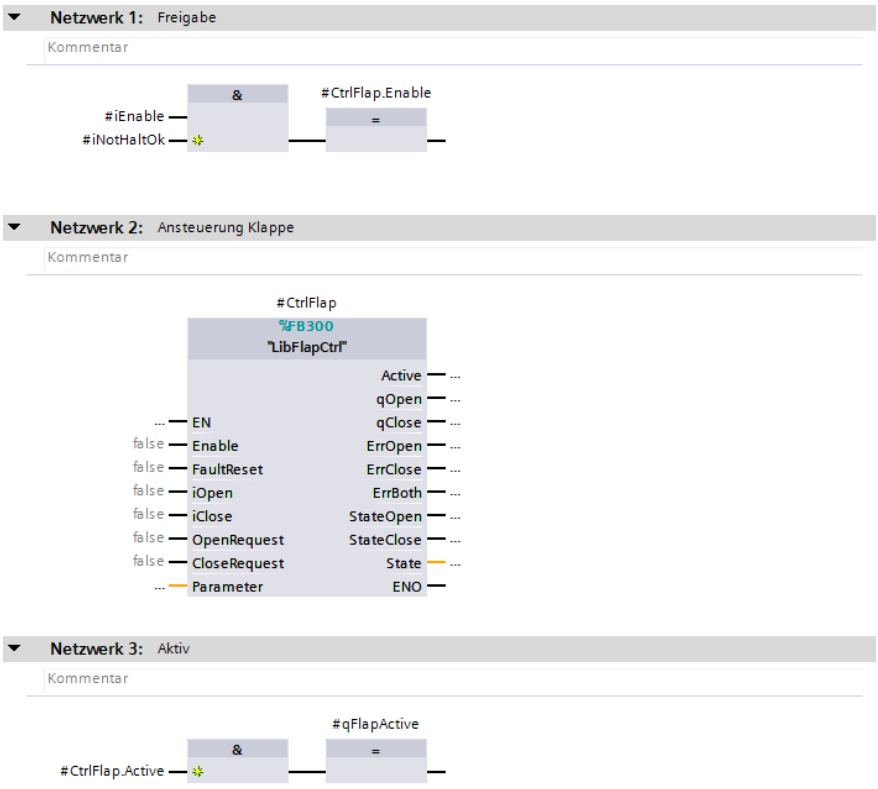


Abbildung 12: Beispielhafte Darstellung eines Zugriffes auf Ein- und Ausgangsparameter, Quelle: Eigene Darstellung.

Bei dieser Anwendung erfolgt ein schreibender Zugriff vor und ein lesender nach Aufruf der Instanz. Das Lesen bzw. Schreiben der Parameter der Instanz erfolgt durch die Angabe <Instanzname>.<Variable> wie z.B. in Abbildung 12 in den Netzwerken 1 und 3 dargestellt. Derartige Anwendung bieten jedoch den schwerwiegenden Nachteil, dass mit solchen Zugriffen das Programm unübersichtlich gestalten und die Fehlersuche erschwert wird, daher sollte auf diese Möglichkeit verzichtet werden.[28]

Grundsätzlich stellt diese Art der Programmierung eine derzeit sehr übliche Variante dar, dennoch bietet der Ansatz nicht nur Vorteile, sondern birgt auch einige schwerwiegende Nachteile. Nur durch strenge Regeln und der strikten Einhaltung dieser kann sichergestellt werden, dass Programme, welche mit modularer Programmierung erstellt wurden, strukturiert und erweiterbar aufgebaut werden und somit auch Ansätze der objektorientierten Programmierung aufweisen.

4.1.2 Fazit

Eine Steuerungsprogrammierung mittels OOP mit Funktionsbausteinen bringt sowohl Vorteile als auch Nachteile mit sich. Durch die nachfolgende Betrachtung dieser Variante sollen die Eigenschaften der Programmiermethode nochmals zusammengefasst dargestellt werden.

[28] Vgl. Hofer (2014), S. 54.

- *Weiterentwicklung der Software*

Eine Weiterentwicklung der Software ist in vielen Fällen unumgänglich. Vor allem in der Anlagen- bzw. Maschinenprogrammierung müssen immer wieder Anpassungen in der Programmierung vorgenommen werden. Diese können aus unterschiedlichsten Gründen wie zum Beispiel mechanische Änderungen oder Komponententausch hervorgerufen werden. Dennoch stellt dies ein großes Problem der modularen Programmierung dar, da die hier herkömmliche Vorgehensweis durch Kopieren und Anpassen dargestellt wird. Dabei werden, wie beispielhaft in Abbildung 14 gezeigt, von einem Baustein ausgehend verschiedenste weitere Bausteine erstellt. Diese Methode führt oft zu schnellen Ergebnissen aber auch dazu, das unzählige Derivate der Ausgangssoftware geschaffen werden, deren Verwaltung beinahe unmöglich erscheint. Dennoch bieten sich durch die Anwendung von TIA Portal bereits Möglichkeiten, Bausteine versioniert zu entwickeln und diese mit eigenen Bibliotheken zu verwalten.

- *Fehlerbehebung*

Bereits zuvor wurde beschrieben, dass eine Verwaltung der Software in der modularen Programmierung sich als besonders schwierig erweist. Zusätzlich entsteht durch dieses Problem eine weitere Problematik, und zwar die der Fehlerbehebung. Zur Weiterentwicklung einer Software muss davon ausgegangen werden, dass die Ausgangssoftware bereits umfangreich getestet wurde und auch einsatzbereit ist. In vielen Fällen werden bestimmte Fehler erst nach einiger Zeit durch kaum nachvollziehbare Situationen aufgedeckt. In dieser Zeit kann es schon zur Weiterentwicklung des Bausteines gekommen sein, und somit pflanzt sich der Fehler in nachfolgenden Ableitungen fort. Daher ist es zwingend notwendig, Fehlerbehebungen genaustens zu dokumentieren. Zudem soll mit Bibliotheken, wie in Abschnitt 2.2.3 beschrieben, gearbeitet werden, um einen Wildwuchs verschiedenster Softwarelösungen zu vermeiden und somit auch sicherzustellen, dass nicht bereits durchgeführte Änderungen wieder rückgängig gemacht werden.

- *Strukturierung und Nachvollziehbarkeit*

Eine Programmierung der OOP mit Funktionsbausteinen bezieht sich vorwiegend auf die Verwendung von Funktionsbausteinen, welche mittels Bibliotheken verwaltet werden können. In den Funktionsbausteinen werden die Anwendungen zur Steuerung des Programmes und der Hardwareobjekte abgebildet. Durch einen strukturierten Aufbau der Funktionsbausteine und somit auch des Anwenderprogrammes ist es möglich ein Programm einfach und nachvollziehbar aufzubauen. Die Vorteile ergeben sich aus der Verwendung der Funktionsbausteine. Mit der Instanziierung der Bausteine eröffnet sich die Möglichkeit der mehrfachen Verwendung, wodurch eine zentrale Verwaltung mit der Projektbibliothek hervorragend genützt werden kann. Darüber hinaus realisiert jeweils ein FB eine definierte Aufgabe oder ein reales Objekt in der Steuerung, wodurch die Nachvollziehbarkeit zusätzlich gesteigert wird.

4.2 OOP mit Structured Control Language

Bereits in der Kapiteleinleitung wurde angemerkt, dass SIMATIC STEP7 und TIA Portal keine objektorientierte Programmierung unterstützen, dennoch ist es möglich, auf mehr oder weniger aufwendigen Wegen OOP mit Structured Control Language abzubilden. Diese Methode verlangt jedoch, dass Ableitungen und dergleichen manuell ausprogrammiert werden, was bei Compilern für

objektorientierte Hochsprachen automatisch umgesetzt wird. Die Basis dieser Anwendung liegt darin, Klassen und Objekte mit den herkömmlichen Bausteinen wie Funktionsbaustein, Funktion und Datenbaustein nachzustellen. Dabei wird durch eine spezielle Programmierung versucht, Funktionsbausteine, aber auch Funktionen in SCL mit einer Klasse der OOP gleichzustellen. Bei diesem Vorgehen definiert der Autor des Buches *SCL und OOP mit dem TIA Portal* Johannes Hofer eine Klasse wie folgt.

„Eine Klasse besteht aus einer Datenstruktur und Funktionen, auch Methoden oder Memberfunktion genannt. Die Daten bezeichnet man auch als Attribute. Zudem gibt es Schutzmechanismen für die Methoden und Attribute.“[29]

Diese Definition erfordert eine eigene Programmierung, da in der herkömmlichen SPS-Programmierung nicht von Methoden oder Memberfunktionen gesprochen wird. Im nachfolgenden Abschnitt 4.2.1 wird gezeigt, wie ein Funktionsbaustein aufgebaut werden kann, um diesen als Klasse zu interpretieren.

4.2.1 Funktionsbaustein als Klasse

Zur Erstellung einer Klasse mittels Funktionsbausteines müssen die in Unterkapitel 4.2 definierten Kriterien erfüllt werden. Daraus geht hervor, dass ein FB, welcher als Klasse abgebildet wird, Funktionen, in diesem Fall als Methoden bezeichnet, sowie auch Daten, welche als Attribute bezeichnet werden, beinhaltet. Zusätzlich müssen für ebendiese Methoden und Attribute Zugriffsspezifikationen, also Schutzmechanismen, möglich sein. Ein solcher Aufbau wird in Abbildung 13 beispielhaft dargestellt.

```
// Beispiel: Grundlage zur Programmierung einer Klasse

CASE #FunctionNumber OF         // Eingangs Parameter - Funktions Nummer
    "Function_1":               // Konstante - Methode 1
        GOTO METHODE_1;
    "Function_2":               // Konstante - Methode 2
        GOTO METHODE_2;
    "Function_3":               // Konstante - Methode 3
        GOTO METHODE_3;
    ELSE
        RETURN;
END_CASE;

// Methode 1 - Rückgabe Wert der Variable Function_1
METHODE_1:
#staticFunctionNumber := "Function_1";
RETURN;

// Methode 2 - Rückgabe Wert der Variable Function_2
METHODE_2:
#staticFunctionNumber := "Function_2";
RETURN;

// Methode 3 - Rückgabe Wert der Variable Function_3
METHODE_3:
#staticFunctionNumber := "Function_3";
RETURN;
```

```
// Aufruf in Main Program Sweep (OB1)
// Aufruf des "Klassen-FBs" mit Methode 1
      CALL  "FirstClass", "DI_FirstClass"
         FunctionNumber :="FirstClassDaten".FunctionNumber.Function_1

// Aufruf des "Klassen-FBs" mit Methode 2
      CALL  "FirstClass", "DI_FirstClass"
         FunctionNumber :="FirstClassDaten".FunctionNumber.Function_2

// Aufruf des "Klassen-FBs" mit Methode 3
      CALL  "FirstClass", "DI_FirstClass"
         FunctionNumber :="FirstClassDaten".FunctionNumber.Function_3

```

Abbildung 13: Beispielhafte Ausführung einer Klasse mit SCL, Quelle: Eigene Darstellung.

Der linke Bereich der Abbildung 13 zeigt einen Funktionsbaustein als Klasse. Der Baustein wird als *FB1* ausgeführt und trägt die Bezeichnung `FirstClass`, die wiederum im rechten Teil der Abbildung, welche den *Main* Organisationsbaustein darstellt, wiederkehrt. Im Funktionsbaustein `FirstClass` werden alle benötigten Eigenschaften, welche zur Definition der Klasse nötig sind, erfüllt. So werden Attribute durch

[29] Hofer (2014), S. 237.

die Variable `staticFunctionNumber` und die Methode durch `METHODE_1`, `METHODE_2` und `METHODE_3` abgebildet. Der Zugriffsspezifizierer kann durch die Variable `staticFunctionNumber` als *public* zum Ausdruck gebracht werden, da diese einen zugewiesenen Speicherbereich in der zugehörigen Instanz besitzt. Hier muss jedoch angemerkt werden, dass ein Zugriff auf diese Variablen nicht erfolgen sollte, da dadurch schwerwiegende Fehler produzierte werden können, welche kaum zu debuggen sind. Würde diese Variable aber nicht im Bereich der statischen, sondern in dem der temporären Variablen definiert werden, dann kann diese nicht mehr als *public* bezeichnet werden, da temporäre Variablen keinen definierten Speicher in einer Instanz besitzen. Diese Variablen sind nur während der Abarbeitung der Klasse (dem Funktionsbaustein) gültig und stehen auch nur dieser zur Verfügung, und werden daher als *private* bezeichnet.

Bereits zuvor wurde darauf verwiesen, dass die Klasse `FirstClass` im rechten Teil der Abbildung wiederkehren wird. Der Aufruf der Klasse erfolgt im Organisationsbaustein 1 und mittels des Attributes `FunctionNumber` werden die einzelnen Methoden ausgeführt.

Der beschriebene Aufbau soll verdeutlichen, wie ein Funktionsbaustein als Klasse mittels SCL aufgebaut werden kann. Diese Möglichkeit bietet natürlich nicht nur SCL, sondern kann auch mit den Programmiersprachen Funktionsplan und Anweisungsliste realisiert werden. Dennoch soll dies nur beispielhaft das Gerüst einer Klasse darstellen. Umfangreichere Anwendungen, welche zum Beispiel Funktionalitäten wie Vererbung oder Polymorphie aufweisen, gestalten eine solche Realisierung sehr schnell als besonders komplex und sind für unerfahrene Anwender nicht besonders einfach anzuwenden.

4.2.2 Vererbung mit SCL und Multiinstanzen

Einen wichtigen Punkt in der Steuerungsprogrammierung stellt die Wiederverwendbarkeit von bereits vorhandenen und getesteten Applikationen dar. In vielen Fällen muss die Software jedoch angepasst werden. Anpassungen erfolgen meist durch Kopieren von bestehenden Lösungen und Adaption neuer Anwendungen. Aus Abbildung 14 geht schnell hervor, dass dieses Variante der Versionierung bzw. Erweiterung sehr rasch unüberschaubar wird. Vor allem besteht die Gefahr, dass bereits vorhandene und getestete Applikationen wieder verloren gehen, da Anpassungen mit falschen oder veralteten Versionen durchgeführt wurden.

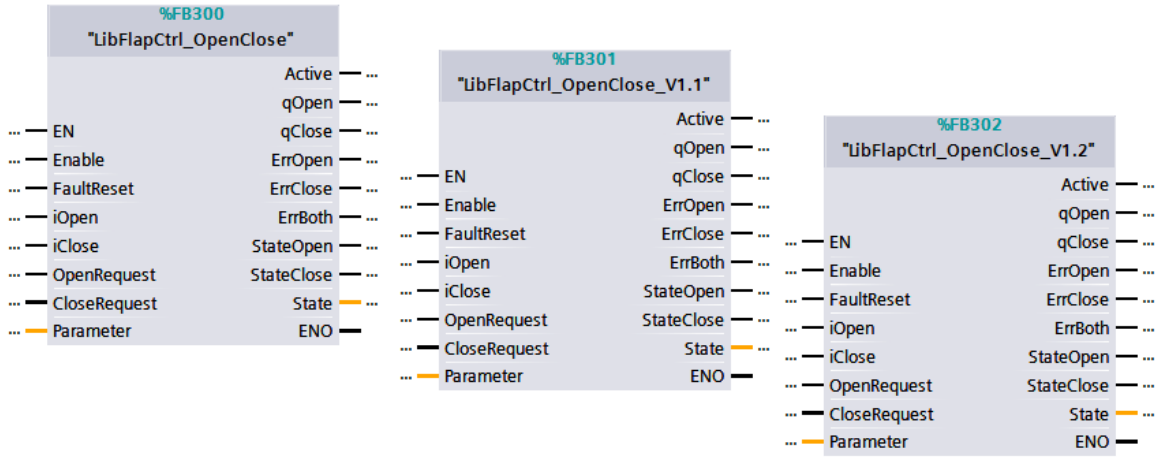


Abbildung 14: Funktionsbaustein in verschiedenen Versionen, Quelle: Eigene Darstellung.

Unter anderem bietet sich in der Softwarearchitektur die Möglichkeit der Wiederverwendbarkeit von bestehender Software durch Vererbung. Bei dieser Methode erbt eine abgeleitete Klasse alle Eigenschaften und Funktionalitäten der Basisklasse. In der SPS-Programmierung wachsen somit Funktionsbausteine zusammen und stellen nach der Instanziierung eine neue Klasse dar. Erst durch die Instanziierung kann der Anwender auf Methoden der Klassen zugreifen, da diese mit der Generierung der Instanz erzeugt werden.[30]

Um das Vorgehen zur Vererbung mit Multiinstanzen genauer beschreiben zu können, wird nachfolgend in Abbildung 15 ein Auszug aus einem Programm gezeigt, mit welchem die Skalierung von Messwerten durch die Ableitung eines Funktionsbausteines realisiert wird.

OB1 - MAIN

Netzwerk 2: Ableitung mit Multiinstanz

Kommentar

```
1  // Ableitung mit Multiinstanz Temperatur
2        CALL  "ClassTemperatur", "ClassTemperatur_DI"      %FB1, %DB1
3           AP_Error    :="Error"                            %MD10
4           AP_Messwert :="Messwert"                         %MD14
5  // Ableitung mit Multiinstanz Druck
6        CALL  "ClassDruck", "ClassDruck_DI"                %FB2, %DB2
7           AP_Error    :="Error"                            %MD10
8           AP_Messwert :="Messwert"                         %MD14
9  // Ableitung mit Multiinstanz Distanz
10       CALL  "ClassDistanz", "CLassDistanz_DI"            %FB3, %DB3
11          AP_Error    :="Error"                            %MD10
12          AP_Messwert :="Messwert"                         %MD14
13
```

FB1 - ClassTemperatur

```
1  // ClassTemperatur
2  // Messstelle Temperatur:ClassSkalierung
3
4  #Rohwert := "PortTemp";  // Übergabe des Temperatur-Rohwertes
5
6  // Ableitung der Klasse Skalierung
7  #ClassSkalierung(EP_Ableitung:="ClassSkalierung",
8                   EP_Funktion:="SKALIERUNG_TEMPERTATUR",
9                   EP_Wert:=#Rohwert,
10                  AP_Error=>#AP_Error,
11                  AP_Messwert=>#AP_Messwert);
12
```

Abbildung 15: Ableitung der Klasse Skalierung, Quelle: Eigen Darstellung.

OB1 – MAIN stellt den Aufruf der Ableitungen dar. Es werden die drei Methoden Temperatur, Druck und Distanz jeweils mit dem Baustein `ClassSkalierung` aufgerufen. Beispielhaft wird die Ausführung `ClassTemperatur` gezeigt, die wiederum die Basisklasse `ClassSkalierung` verwendet. In der Basisklasse sind Methoden zur Skalierung der Temperatur, des Druckes und der Distanz ausgeführt. Der Aufruf der gewünschten Methode wird über den Eingangsparameter `EP_Funktion` übergeben. In der Basisklasse erfolgt die Ausführung der parametrierten Methode, wie in Abbildung 16 dargestellt.

[30] Vgl. Hofer (2014), S. 315 ff.

```
CASE #EP_Funktion OF
    "SKALIERUNG_TEMPERTATUR":                // Anwenderkonstante für Skalierung Temperatur
      IF( #SichtAufWert[1] = 16#4 ) THEN     // Datentyp Word
        #tempWord := WORD_TO_BLOCK_DB(INT_TO_WORD(#fbNr)).DW(WORD_TO_INT(#adr));
        #AP_Messwert := (100.0/32767)* INT_TO_REAL(WORD_TO_INT(#tempWord));
        RETURN;
      END_IF;
      #AP_Error := #FALSCHER_DATENTYP;       // Fehlerausgabe
    "SKALIERUNG_DRUCK":                      // Anwenderkonstante für Skalierung Druck
      IF( #SichtAufWert[1] = 16#2 ) THEN     // Datentyp Byte
        #tempWord := WORD_TO_BLOCK_DB(INT_TO_WORD(#fbNr)).DB(WORD_TO_INT(#adr));
        #AP_Messwert := (200.0/255.0)* INT_TO_REAL(WORD_TO_INT(#tempWord));
        RETURN;
      END_IF;
      #AP_Error := #FALSCHER_DATENTYP;       // Fehlerausgabe
    "SKALIERUNG_DISTANZ":                    // Anwenderkonstante für Skalierung Distanz
      IF( #SichtAufWert[1] = 16#6 ) THEN     // Datenty Dword
        #tempDWord := WORD_TO_BLOCK_DB(INT_TO_WORD(#fbNr)).DD(WORD_TO_INT(#adr));
        #AP_Messwert := DINT_TO_REAL(DWORD_TO_DINT(#tempDWord)) * 0.555;
        IF( #AP_Messwert <=0 )THEN #AP_Error := #SENSOR_GESTOERT; END_IF;
        RETURN;
      END_IF;
      #AP_Error := #FALSCHER_DATENTYP;       // Fehlerausgabe
  ELSE
    // fehlermeldung
    #AP_Error := #EP_Funktion*-1;
    RETURN;
END_CASE;
```

Abbildung 16: Methodenaufruf der Basisklasse, Quelle: Eigene Darstellung.

4.2.3 Fazit

TIA Portal bietet keine Möglichkeit der objektorientierten Programmierung. Durch eine mehr oder weniger aufwendige Programmierung und einer weichen Definition von Klassen und Zugriffsspezifizierer kann mit etwas Fantasie eine Art OOP mit SCL geschaffen werden. Die Möglichkeit der Wiederverwendbarkeit wird auch durch eine modulare Programmierung der *OOP mit Funktionsbausteinen*, wie in Unterkapitel 4.1 beschrieben, geboten. Auch hier stellen sich dieselben Probleme ein, wie bereits in Abschnitt 4.1.2 angemerkt. Für Weiterentwicklungen ist es notwendig, dass die Basisklasse ausführlich getestet wurde. Zusätzlich entsteht auch das Problem der Versionierung. Diesem kann aber durch Versionierung mit Bibliotheken entgegengewirkt werden.

Aus dieser Variation der Programmierung entsteht kein zwingender Vorteil gegenüber der Variante *OOP mit Funktionsbausteinen*. Im Gegenteil, diese Ausführung stellt sich weitaus komplexer für Anwender mit geringen Programmierkenntnissen dar, wobei bei der nachfolgenden Option der Programmierung *OOP mit SIMOTION* auch eine gewisse Vorkenntnis der Programmierung vorausgesetzt werden muss.

4.3 OOP mit SIMOTION

SIMOTION stellt ein Motion-Control-System von Siemens dar. Das Hauptaugenmerk liegt dabei auf bewegungsführenden Anwendungen. Zur Projektierung eines SIMOTION Systems wird das Engineering-System SCOUT benötigt, welches alle Tools und Editoren zur Programmierung und Konfigurierung von SIMOTION Applikationen umfasst. Zudem wird für die Umsetzung eine eigene Hardware benötigt, welche in Kombination mit SIMOTION SCOUT eine objektorientierte Programmierung ermöglicht.

Die Umsetzung der objektorientierten Programmierung mit SIMOTION basiert auf der in der Norm IEC 61131-3:2013 ausgeführten Neuerung des Sprachumfanges objektorientierter Programmierung. Die Norm

definiert zwei Konzepte zur Umsetzung der OOP. Das erste Konzept beschreibt die Anwendung der OOP mit objektorientierten Funktionsbausteinen, welche die gesamte Funktionalität übernehmen und somit die Klassen ersetzen. Die Implementierung der OOP mittels Klassen stellt das zweite Konzept da, worauf auch das Steuerungssystem SIMOTION aufsetzt. Zudem bildet die OOP mit Klassen eine gängige Umsetzungsmethode. Die Erweiterung der objektorientierten Programmierung erfolgt mittels SIMOTION zweistufig. Die erste Stufe baut auf der Realisierung von Klassen mit Methoden auf und mit deren verbundenem Vererbungsapparat. Zusätzlich wird auch die Umsetzung von Interfaces und deren Mechanismen inklusive der Vererbung zur Verfügung gestellt, wodurch bereits die wesentlichen Funktionen der OOP abgebildet werden. Stufe zwei stellt die Komplementierung der IEC 61131-3 bezüglich OOP dar, wird in dieser Arbeit jedoch nicht weiter ausgeführt, denn diese Stufe befasst sich ausführlich mit der Einführung der Referenz, die im Folgenden keine Anwendung findet.[31]

In den nachfolgenden Abschnitten werden grundlegende Informationen zur OOP mit SIMOTION angeführt, um abschließend ein Beispiel zur Umsetzung der objektorientierten Programmierung ausführen zu können.

4.3.1 Klassen in SIMOTION

Objektorientierte Programmierung mit SIMOTION baut auf dem Konzept der Klasse auf. Eine Klasse setzt sich aus Attributen und Methoden zusammen und definiert den Bauplan für Objekte. Attribute bezeichnen Eigenschaften und Variablen, Methoden beschreiben bestimmte Verhaltensweisen. Eine Klasse stellt den Datentyp eines Objektes dar, wodurch diese als komplexer Datentyp, wie etwa eine Struktur, betrachtet werden kann. Zusätzlich bietet die Klasse die Möglichkeit zur Definierung von Algorithmen, wodurch sie sich wiederum von Strukturen unterscheidet. Die Programmierung erfolgt in Structured Text in sogenannten ST-Quellen, dabei beginnt die Klasse mit dem Schlüsselwort `CLASS` gefolgt von <ClassName> und wird mit `END_CLASS` beendet. Im Gegensatz zu FBs ist eine Verwendung der Attribute *VAR_INPUT*, *VAR_OUTPUT* sowie *VAR_IN_OUT* und *VAR_TEMP* im Deklarationsteil der Klasse nicht gestattet, es besteht nur die Möglichkeit Variablen zu vereinbaren, wie bereits in Unterkapitel 3.4 beschrieben.[32]

4.3.2 Methoden in SIMOTION

Die Programmierung der Klasse erfolgt durch Methoden, die wie Funktionen angesehen werden können, da auch sie mit einer Variablenschnittstelle und dem ausführbaren Programm ausgestattet werden. Wie auch schon bei der Klasse muss die Definition der Methode über Schlüsselwörter erfolgen. Der Anfang der Methode mit wird `METHOD <MethodName>` eingeleitet und das Ende mit `END_METHOD` gekennzeichnet. Die Ausführung ist ebenfalls der einer Funktion ähnlich. Beim Aufruf wird die Methode über die definierte Variablenschnittstelle mit Daten beladen und über die Ausgangsschnittstelle (Rückgabewert oder *VAR_OUTPUT*) werden die Ergebnisse ausgegeben. Die Daten werden hierfür im Stack der Central

[31] Vgl. Braun/Horn (2015), S. 49 f.

[32] Vgl. Braun/Horn (2015), S. 67.

Processing Unit (CPU) hinterlegt und nach dem Verlassen wieder gelöscht. Das bedeutet, die Daten der Methode sind wie auch bei Funktionen nur zur Zeit des Aufrufes gültig.[33]

Über Schlüsselwörter kann für jede einzelne Methode einer Klasse festgelegt werden, wie bzw. von wo aus auf Methoden zugegriffen werden darf. Die Definition erfolgt durch Hinzufügen eines Schlüsselwortes vor dem Methodennamen bei der Methodendefinition (`METHOD <SCHLÜSSELWORT> <MethodName>`). Mit der nachfolgenden Tabelle 3 werden die zur Verfügung stehenden Schlüsselwörter und deren Funktionen aufgelistet.[34]

Schlüsselwort	Beschreibung
PROTECTED	Durch dieses Schlüsselwort wird angegeben, dass die Methode nur innerhalb der eigenen Klasse sowie in deren Ableitungen aufgerufen werden kann. PROTECTED stellt den Default Schlüssel dar und kann somit auch weggelassen werden, das bedeutet, alle einfach deklarierte Methoden werden standardmäßig als PROTECTED ausgeführt.
PUBLIC	Über diesen Bezeichner wir festgelegt, dass die beschriebene Methode von überall, wo die zugehörige Klasse verwendet wird, aufgerufen werden kann.
PRIVATE	Mittels Zugriffsbezeichner PRIVATE wird determiniert, dass nur innerhalb der Klasse die Methode zur Verfügung steht. Entgegen einer PROTECTED Methode ist bei einer PRIVATE Methode der Zugriff aus einer abgeleiteten Methode nicht möglich.
INTERNAL	Dieses Schlüsselwort findet Anwendung bei implementierten Namensräumen, sogenannten Namespaces. Durch INTERNAL wird angegeben, dass eine Methode innerhalb des Namensraumes, in der die Klasse definiert ist aufgerufen werden kann

Tabelle 3: Zugriffsspezifikationen von Methoden, Quell: Braun/Horn (2015), S. 71.

Die Verwendung von Methoden erfolgt gleichermaßen wie von Funktionen. Der Methodenaufruf kann in grafischer Programmierung mit Aufrufboxen umgesetzt werden, hierzu muss lediglich der Klassenname und der Methodenname angegeben werden oder, wie auch nachfolgend dargestellt, in Structured Text. Der nachfolgende Programmauszug soll dazu dienen, die Anwendung von Methoden und Klassen zu verdeutlichen.

[33] Vgl. Braun/Horn (2015), S. 70.

[34] Vgl. Braun/Horn (2015), S. 71.

```
// ===============================================================
// Class Counter
CLASS COUNTER
    VAR
        CV:INT;             // Current value of counter
    END_VAR

    VAR OVERRIDE
        MAX_Val:INT := 999;
        MIN_Val:INT := 0;
    END_VAR
    //#########################################################
    //Public Method UP
    METHOD PUBLIC UP: INT
        VAR_INPUT           // Definition of Input variables
            INC:INT:=1;
        END_VAR
        VAR_OUTPUT          // Definition of Output variables
            QU:BOOL;
        END_VAR

        // Upper limit detection
        IF CV <= MAX_Val - INC THEN
            CV := CV + INC; // Count up of current value
            QU := FALSE;
        ELSE
            QU := TRUE;     // upper limit reached
        END_IF;
        UP := CV;           // Result of method
    END_METHOD
    //#########################################################
END_CLASS
// ===============================================================
// Program
PROGRAM CallCounter_ST
    VAR
        C1:COUNTER:=(MAX_Val:=1000,MIN_VAL:=0);
        CountOut: INT;
        LockCnt: BOOL;
    END_VAR
    //#########################################################
    // Call COUNTER
    IF Locking = FALSE THEN
        CountOut:=C1.UP(INC:= 1, QU=>LockCnt); // increment
    END_IF;
END_PROGRAM
// ===============================================================
```

Der Programmauszug zeigt Teile eines einfachen Beispiels einer Zähler Klasse, die eine Methode mit der Bezeichnung UP beinhaltet. Mit der Methode UP wird ein Hochzählen des Zählwertes `CV` bis zum Erreichen der Grenze `Max_Val` ausgeführt. Bei erreichter Obergrenze wird die Ausgangsvariable `QU` gesetzt.

Die Nutzung der Klasse `Counter` mit der Methode `UP` erfolgt durch das Programm `CallCounter_ST`, welches die Instanz `C1` der Klasse `COUNTER` erzeugt. Diese Klasse kann wiederum, wie im nachfolgenden Abschnitt 4.3.3 beschrieben, abgeleitet werden.

4.3.3 Vererbung in SIMOTION

Die objektorientierte Programmierung bietet die Möglichkeit der Vererbung, das bedeutet eine neue Klasse kann von einer bestehenden abgeleitet werden. Durch die Ableitung werden von der Ursprungsklasse alle Eigenschaften an die neue Klasse vererbt. Mit dieser Methode werden alle Variablen, Eigenschaften und Methoden der Ursprungsklasse, auch Basisklasse genannt, der abgeleiteten Klasse zur Verfügung gestellt. Der Quellcode der Basisklasse ist in der Ableitung nicht einsehbar. Dennoch besteht die Möglichkeit die abgeleitete Klasse erneut abzuleiten umso weitere Unterklassen zu erzeugen, wie in Abbildung 17 grafisch verdeutlicht wird. Jedoch ist eine Mehrfachvererbung nicht erlaubt.[35]

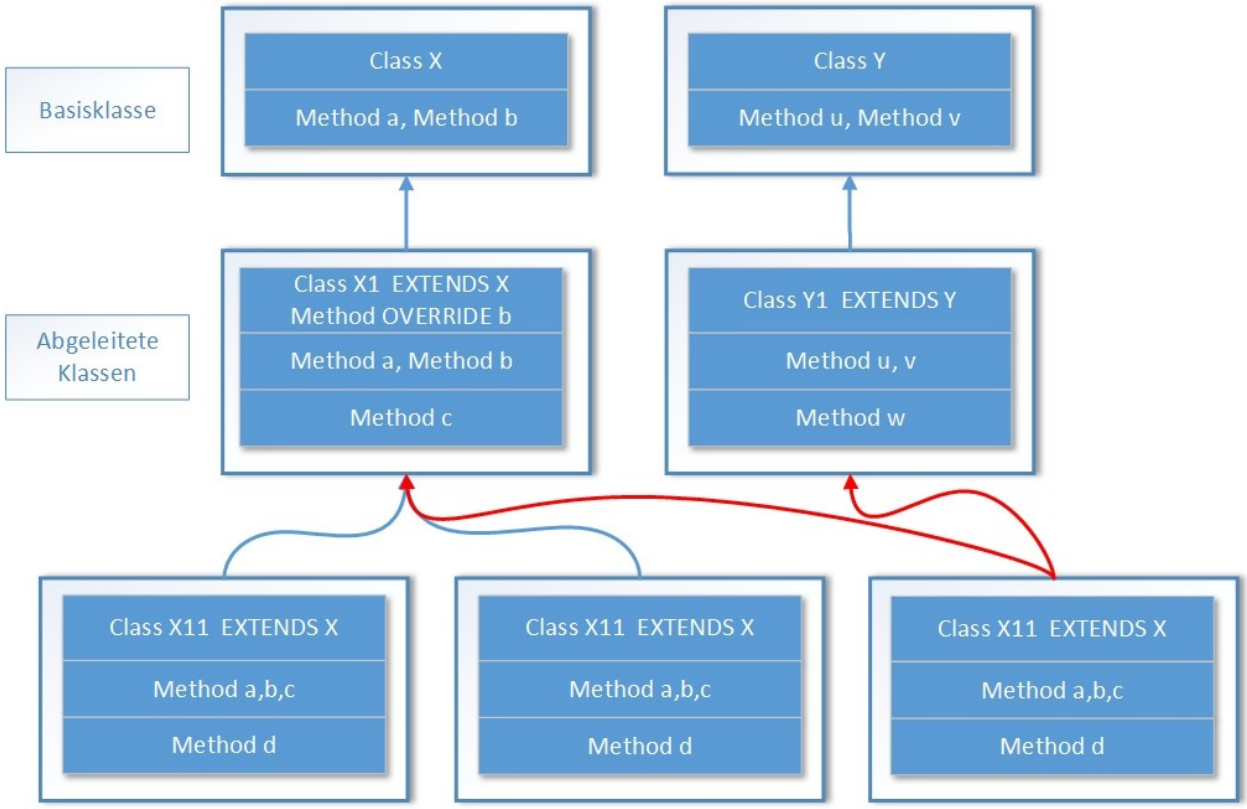


Abbildung 17: Ableitung von Klassen, Quelle: Braun/Horn (2015), S. 78. (leicht modifiziert).

Wie in der Abbildung 17 gezeigt, wird die Basisklasse durch *Class X* repräsentiert, von welcher wiederum durch das Schlüsselwort *EXTENDS X* die Klasse *Class X1* abgeleitet wird. Alle Eigenschaften wie Variablen und Methoden der Ursprungsklasse werden durch die Vererbung der abgeleiteten Klasse zur Verfügung gestellt. Das kann jedoch nur als sinnvoll erachtet werden, wenn die durch die Ableitung entstandene neue Klasse an neue Anforderungen angepasst werden kann. Die Möglichkeit dazu wird durch das Überschreiben von Methoden erzielt. Mit dem Schlüsselwort *OVERRIDE* wird in der neuen Klasse angegeben, dass eine Methode, in Abbildung 17 die Methode b, zu überschreiben ist, also mit neuen Funktionen abgebildet wird. Es ist jedoch zu beachten, dass die Methode eine neue Implementierung erhalten kann, aber die Schnittstellendefinition und der Name nicht verändert werden dürfen. Mit dem nachfolgenden Programmauszug soll anhand des in Abschnitt 4.3.2 dargestellten Beispiels die Ableitung

[35] Vgl. Braun/Horn (2015), S. 78.

der Klasse COUNTER erfolgen. Die abgeleitete Klasse zählt im Gegensatz zur Basisklasse in Fünferschritten.[36]

```
//#########################################################
CLASS COUNTER_5STEP EXTENDS COUNTER
    METHOD PUBLIC OVERRIDE UP: INT // Method override
        VAR_INPUT
            INC:INT:=1;
        END_VAR
        VAR_OUTPUT
            QU:BOOL;
        END_VAR
        UP:=SUPER.UP(INC:=INC*5,QU=>QU);
    END_METHOD
END_CLASS
//#########################################################
```

Die Klasse COUNTER_5STEP wird durch das Schlüsselwort EXTENDS von der Basisklasse COUNTER abgeleitet und erbt damit alle Variablen und Methoden dieser Klasse. Um die Klasse in Fünferschritten hochzählen lassen zu können, muss die Methode UP der Basisklasse überschrieben werden. Das Überschreiben der Methode wird mit dem Statement METHOD PUBLIC OVERRIDE UP ausgeführt. Eine Änderung der Methodensignatur darf dabei nicht erfolgen, somit müssen Name, Rückgabewert aber auch die Definition der Ein- und Ausgangs-Parameter unverändert bleiben. Zum Zählen in der gewählten Schrittweite, wird durch die Methode UP der Ableitung die Methode UP der Basisklasse aufgerufen. Dies erfolgt durch das Schlüsselwort SUPER, mit zusätzlicher Multiplikation des Übergabewertes INC. Somit werden zwei der grundlegenden Eigenschaften der OOP, die Vererbung und das Überschreiben von Methoden, abgebildet.[37]

4.3.4 Fazit

Im Gegensatz zu den zuvor ausgeführten Programmiermethoden bietet objektorientierte Programmierung mit SIMOTION alle Funktionen der OOP bezogen auf die IEC 61131-3:2013 an. Der Ansatz liegt in der Programmierung von Klassen, was wiederum eine sehr gängige Methode der OOP darstellt. Da SIMOTION den kompletten Umfang der OOP bereitstellt, können Klassen mit Methoden wie auch die Vererbung ohne Umwege realisiert werden. Zudem können mittels Schlüsselwörtern Zugriffsrechte auf Methoden definiert werden, was bei einer Programmierung in TIA Portal nicht der Fall ist. Daneben wird die Möglichkeit der Vererbung von Klassen geboten, mit welcher auch ein Überschreiben bzw. auch Erweitern von Methoden umgesetzt werden kann.

Der Nachteil dieser Anwendung ergibt sich aus dem Aufwand der Programmierung. Die Methoden *OOP mit Funktionsbausteinen* und *OOP mit SCL* werden in TIA Portal ausgeführt. Für die Programmierung in SIMOTION wird ein zusätzliches Software-Paket sowie eine eigene Hardware, sogenannte SIMOTION-

[36] Vgl. Braun/Horn (2015), S. 78 f.

[37] Vgl. Braun/Horn (2015), S. 79.

Geräte, benötigt. Zusätzlich muss eine gewisse Vorkenntnis im Umgang mit der Programmierung von SIMOTION Systemen vorausgesetzt werden, um somit Programme objektorientiert aufbauen zu können.

Daraus resultierend bietet die objektorientierte Programmierung mit SIMOTION gegenüber der herkömmlichen Programmierung in TIA Portal den Vorteil, dass eine Programmierung im vollen Umfang der OOP möglich ist. Nachteilig zu betrachten ist jedoch die Notwendigkeit eigner Soft- und auch Hardware sowie die umfangreiche Kenntnis zur Programmierung von SIMOTION Systemen. Um die Vor- und Nachteile der jeweiligen Methoden aufzuzeigen, erfolgt nachfolgend eine Evaluierung dieser.

4.4 Evaluierung

In den vorangegangenen Unterkapiteln wurden drei verschiedene Programmieransätze zu objektorientierter Programmierung von Siemens Steuerungen behandelt. Damit eine bestmögliche Umsetzung der Lift-Programmierung erfolgen kann, ist es notwendig, aus diesen Methoden die am besten geeignete zu erheben. Die Auswahl wird mit einer Evaluierung nach einem Punktesystem getroffen. Vorab werden acht Evaluierungseigenschaften definiert und auf die jeweiligen Methoden angewandt. Durch das Punktesystem ist es möglich je Eigenschaft einen bis zehn Punkte zu vergeben. Ein Punkt drückt dabei aus, dass die Eigenschaft überhaupt nicht erfüllt wird, zehn Punkte hingegen zeigen an, dass eine Erfüllung zu Gänze erfolgt. Daraus folgt, dass gesamt je Methode maximal 80 Punkte erreicht werden können. Zur übersichtlichen Gegenüberstellung der Methoden, wird das Ergebnis grafisch wie auch numerisch dargestellt. Die grafische Darstellung soll in erster Linie dazu dienen, die Vorteile und Nachteile der einzelnen Systeme aufzuzeigen und das Ergebnis zudem noch verdeutlichen. Das nummerische Ergebnis dient dazu, eine Aussage treffen zu können, welches System am besten zur Umsetzung erscheint.

Bevor die Evaluierung durchgeführt werden kann, ist es notwendig die zur Bewertung herangezogenen Eigenschaften zu definieren. Diese werden nachfolgend definier und weiter ausgeführt, worauf die jeweilige Eigenschaft sich bezieht.

- *Erfüllung der objektorientierten Programmierung*

Mit dieser Eigenschaft wird angegeben, inwieweit die Ansätze der OOP mit der jeweiligen Methode umgesetzt werden können.

- *Programmiereinfachheit*

Die Einfachheit beschreibt, wie einfach sich die Erstellung eines Programmes mit der gewählten Programmiervariante gestaltet.

- *Programmübersicht*

Die Programmübersicht stellt einen der wichtigsten Punkte der Programmierung dar. Die Übersicht ist von grundlegender Bedeutung für die einfache Nachvollziehbarkeit des Programms und damit auch für die Anwendbarkeit durch unerfahrene Programmierer.

- *Einsteigertauglichkeit*

Diese Eigenschaft ist gewissermaßen ein weiterführender Punkt der Programmübersicht und drückt aus, wie einfach die Anwendung der Methode für Programmiereinsteiger anwendbar ist.

- *Wiederverwendbarkeit*

Eine der Haupteigenschaften der objektorientierten Programmierung ist die Wiederverwendbarkeit von bereits erstellten Klassen und Objekten. Da TIA Portal die OOP noch nicht unterstützt müssen die Ansätze über Umwege erreicht werden. Die Eigenschaft zeigt auf, inwieweit bereits erstellte Programmteile wiederverwendet werden können.

- *Erweiterbarkeit*

Wie auch die Wiederverwendbarkeit stellt die Erweiterbarkeit von Programmteilen einen wichtigen Aspekt der Programmierung dar. Diese Eigenschaft drückt die Möglichkeit zur fortlaufenden Programmentwicklung aus.

- *Wartungsfreundlichkeit*

Nicht nur Wiederverwendbarkeit und Erweiterbarkeit sind für die bestehende Programmierung von enormer Bedeutung, sondern auch die Wartung dieser. Das Punktesystem beschreibt hierzu, wie einfach sich die Änderungen von bestehenden Programmteilen gestalten.

- *Software-Pakete*

Abschließend werden die zur Programmierung benötigten Softwarepakete bewertet, da diese durchaus maßgeblich die vorausgesetzten Programmierkenntnisse des Anwenders und die Kosten der Programmierung beeinflussen.

Diese Eigenschaften und das Punktesystem finden in der Tabelle 4 Anwendung, um damit die Bewertung der Programmiermethoden *OOP mit Funktionsbausteinen*, *OOP mit SCL* und *OOP mit SIMOTION* durchzuführen. Ziel ist es, numerisch die am besten geeignete Methode zu erheben und zusätzlich anhand der Daten der Tabelle die Programmiervarianten auch grafisch gegenüberzustellen um die Vor- und Nachteile aufzuzeigen.

	Erfüllung der OOP	Programmier-einfachheit	Programm-übersicht	Einsteiger-tauglichkeit	Wieder-verwendbarkeit	Erweiterbarkeit	Wartungs-freundlichkeit	Software-Pakete	*Ergebnis*
OOP mit FBs	5	9	8	8	6	8	8	10	***62***
OOP mit SCL	6	3	4	4	4	5	5	10	***41***
OOP mit Simotion	10	5	6	3	8	7	7	3	***49***

Tabelle 4: Methoden-Bewertung, Quelle: Eigene Darstellung.

Aus der Tabelle geht hervor, dass die Methode *OOP mit Funktionsbausteinen* von gesamt möglichen 80 Punkten 62 Punkte erreicht. Die Variante *OOP mit SIMOTION* erzielte 49 Punkte und *OOP mit SCL* hingegen nur 41 Punkte. Das bedeutet, die Methode *OOP mit Funktionsbausteinen* stellt die am besten zur Realisierung geeignete Variante des Shuttle-Lift Steuerungsprogrammes dar. Diese Aussage soll mit der Abbildung 18 noch grafisch verdeutlicht werden, um somit die Vor- und Nachteile der jeweiligen Methoden zu veranschaulichen. Aufbauend auf diesem Ergebnis, werden im folgenden Kapitel die Software-Architektur sowie auch die notwendigen Programmierrichtlinien zur Realisierung des YLOG-Shuttle Systems definiert.

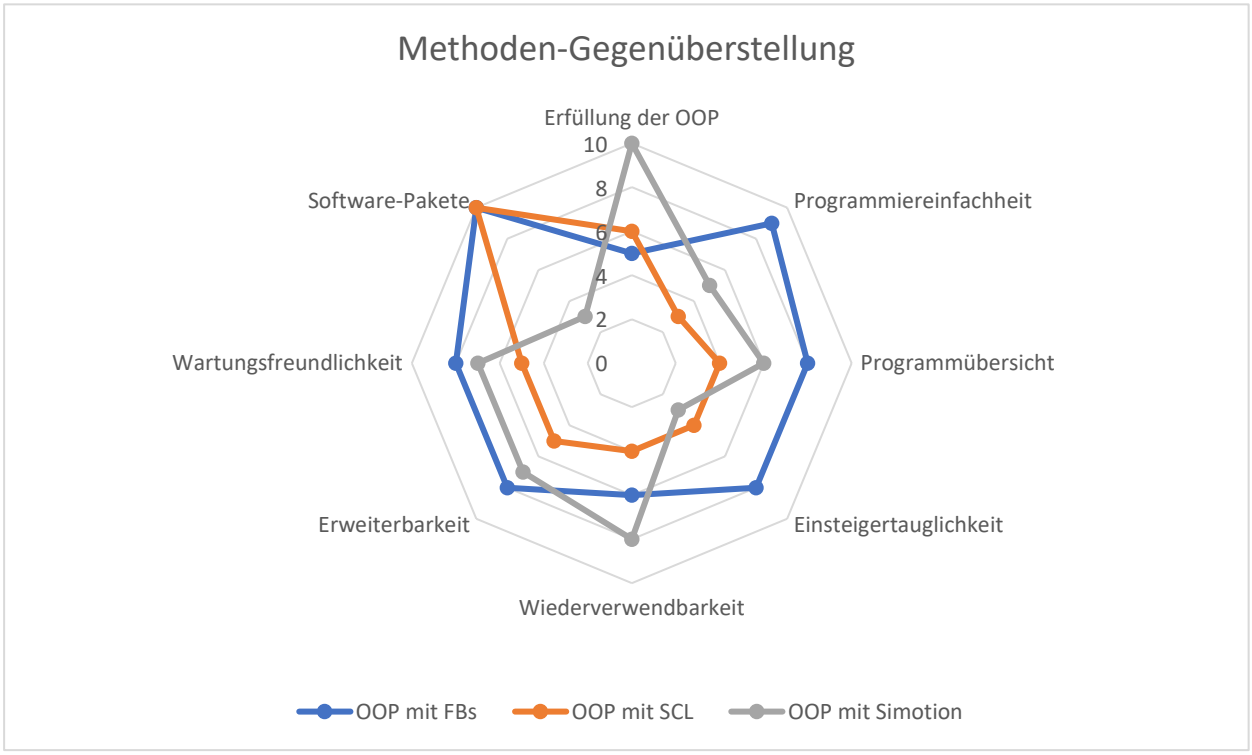


Abbildung 18: Methoden-Gegenüberstellung, Quelle: Eigene Darstellung.

5 SOFTWARE-ARCHITEKTUR

Über die Zeit hinweg altert nicht nur Hardware, sondern auch Software durchläuft einen eigenen Lebenszyklus der einem Alterungsprozess gleicht. Dieser Prozess ist auf die rasante Weiterentwicklung der Technik sowie auf die sich ständig ändernden Anforderungen an die Systeme zurückzuführen. Um dem Altern der Software entgegen zu wirken und somit auch den Lebenszyklus zu erweitern, ist es notwendig, Programme erweiterbar bzw. weiterentwickelbar und nachvollziehbar auszuführen. Diese Anforderungen können durch den Ansatz des objektorientierten Entwurfes erreicht werden.[38]

Damit das in Kapitel 2 beschriebene System den Anforderungen entsprechen kann, muss die Software strukturiert geplant werden. Abbildung 19 beschreibt den strukturierten Aufbau eines Liftes des YLOG-Shuttle System Anwenderprogrammes, welches zusammenfassend als Lift-Objekt bezeichnet wird. Nachfolgend wird auf das Lift-Objekt sowie die einzelnen Strukturteile näher eingegangen.

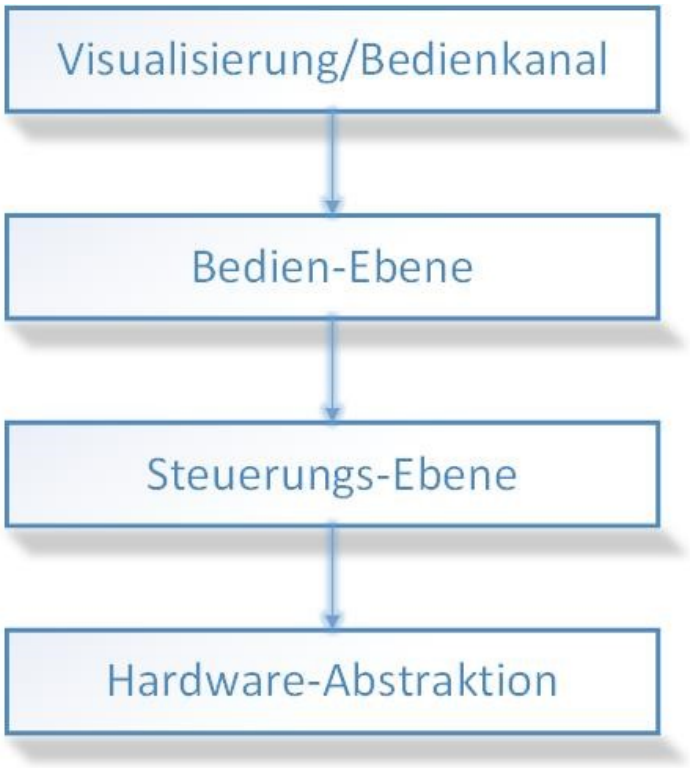


Abbildung 19: Grundstruktur des Lift-Anwenderprogrammes, Quelle: Eigene Darstellung.

Das Lift-Objekt umfasst, wie in Abbildung 19 gezeigt, alle Komponenten die zur Steuerung des Liftes notwendig sind. Dieses Objekt setzt sich zusammen aus der Hardware-Abstraktion, die zur Ansteuerung der Betriebsmittel über die Peripherie dient, der Steuerungs-Ebene des Liftes, welche durch mehrere Module abgebildet wird und über eine Schnittstelle zur Hardware-Abstraktion die Betriebsmittel steuert, sowie übergeordnet der Bedien-Ebene, welche über eine weitere Schnittstelle zur Bedienung der Steuerungs-Ebene verwendet wird und den Datenaustausch zur Visualisierung realisiert.

[38] Vgl. Goll (2014), S. 2 ff.

5.1 Datenaustausch zwischen den Ebenen

Wie bereits zuvor beschrieben, setzt sich das Lift-Objekt aus den Modulen Hardware-, Steuerungs-, Bedien- und Visualisierungs-Ebene zusammen. Damit der reibungslose Betrieb des Liftes und ein einfaches Austauschen bzw. Erweitern der Module ermöglicht werden kann, ist es notwendig, Schnittstellen zum Datenaustausch zwischen den einzelnen Ebenen zu definieren.

Für den Datenaustausch wird auf die Definition des Laufzeitmodelles von Alan Kay zurückgegriffen. Das besagt, dass ein Objekt als ein gedachter Baustein definiert wird, auf welchen von außen durch die Zusendung von Nachrichten zugegriffen werden kann. Zudem entscheidet das Objekt, wie es auf zugesendete Nachrichten reagiert. Eben dieser Ansatz soll durch die Einführung der unterschiedlichen Ebenen erzielt werden. Wie in der Abbildung 20 gezeigt erfolgt die Kommunikation zwischen den Ebenen durch die Zusendung von Nachrichten (in Abbildung 20 als Kommando dargestellt). Dabei wird jede Ebene durch einen Funktionsbaustein mit der zugehörigen Instanz dargestellt. Dieser Funktionsbaustein beinhaltet verschiedene Programmteile im Anweisungsteil (oder auch als Rumpf bezeichnet) sowie auch Zustände, mit welchen auf die zugesendete Nachricht individuell reagiert wird.

Wie der Grafik zu entnehmen ist, erfolgt die Kommunikation immer in zwei Richtungen. Die jeweils übergeordnete Ebene übergibt an die untergeordnete mittels der Kommando-Struktur die Befehle. Über die Status-Struktur werden die jeweiligen Zustände rückgemeldet bzw. diverse Befehle quittiert. Der Vorteil erweist sich dadurch, dass die einzelnen Ebenen untereinander nicht voneinander abhängig sind. Somit können die Module getauscht, geändert oder auch erweitert werden ohne dass davon eine andere Ebene betroffen ist, vorausgesetzt natürlich, die Schnittstelle wird nicht verändert.

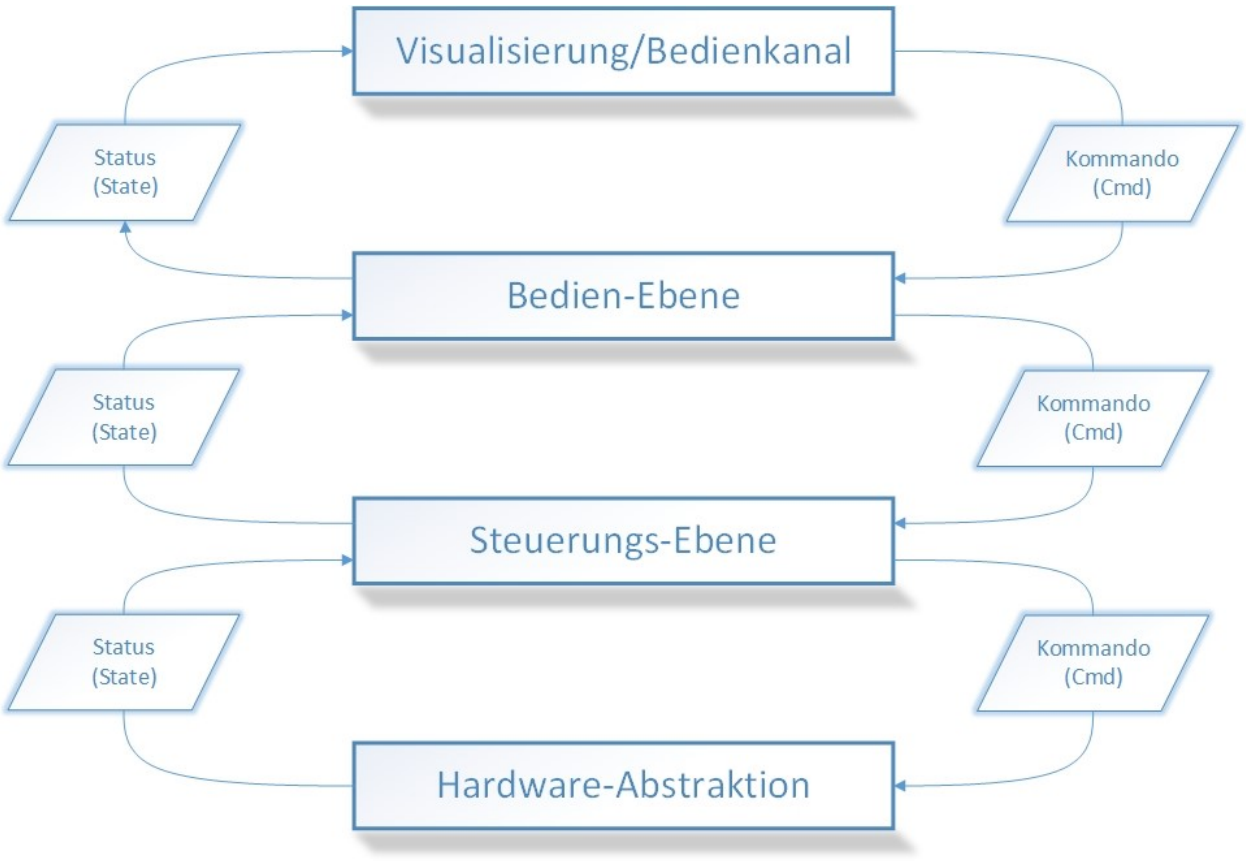


Abbildung 20: Datenaustausch zwischen den Ebenen, Quelle: Eigene Darstellung.

5.2 Hardware-Abstraktion

Dieser Programmteil wird wie zuvor beschrieben zur Steuerung der Hardwarekomponenten eingesetzt und stellt somit im Lift-Objekt die unterste Ebene dar. Der Grundgedanke liegt darin, eine Schnittstelle der einzelnen Komponenten zur Steuerungs-Ebene zu schaffen. Diese definierte Kommunikation zwischen Hardware-Abstraktion und Steuerungs-Ebene führt zu dem Vorteil des einfachen Komponentenaustausches. Das bedeutet, sollte es zum Einsatz einer neuen Hardwarekomponente kommen, muss diese so ausgeführt werden, dass ein Datenaustausch über die Schnittstelle erfolgen kann. Somit sind keine Anpassungen auf der Steuerungs-Ebene notwendig, was wiederum zu einem einfach erweiterbaren bzw. weiterentwickelbaren Konzept führt. Der Signalaustausch zwischen den Ebenen erfolgt bidirektional. Durch die Steuerungsebene werden die jeweiligen Steuersignale über eine Kommando-Schnittstelle an das zugehörige Hardwareobjekt (HW-Objekt) übergeben und stellen somit Befehle dar, welche durch das zuständige HW-Objekt unter der Betrachtung der aktuellen Zustände ausgeführt werden. Zu jedem Zeitpunkt meldet das HW-Objekt den aktuellen Zustand über eine Status-Schnittstelle an die Steuerungsebene zurück.

Ein weiterer Effekt der Aufgliederung zwischen Hardware-Abstraktion und Steuerungs-Ebene ist die strikte Trennung zwischen Ansteuerung und Ausführung, wodurch automatisch eine Hierarchie eingefügt wird, welche die Strukturierung und Nachvollziehbarkeit der Software deutlich verbessert.

5.3 Steuerungs-Ebene

Die Steuerungs-Ebene dient zur Ansteuerung der unterlagerten Hardware-Abstraktion und ist der Bedien-Ebene untergeordnet. Mittels Bedien-Ebene werden die Betriebsarten (Operation Modes) an die Ebene übergeben und verschiedene Statusinformationen an die Bedien-Ebene rückgemeldet. Wie zuvor unter 5.2 beschrieben erfolgt der Datenaustausch zwischen Steuerungs- und Hardware-Abstraktion ebenfalls in beide Richtungen, von der Steuerungs-Ebene werden Steuerbefehle an die Hardware-Abstraktion übergeben und entgegengesetzt werden Statusinformationen empfangen. Somit fungiert die Steuerungs-Ebene als das Bindeglied zwischen Hardware- und Bedien-Ebene. Der Aufbau dieser Ebene ist, wie in Abbildung 21 dargestellt, modular ausgeführt.

Abbildung 21: Schematischer Aufbau der Steuerungs-Ebene, Quelle: Eigene Darstellung.

Durch den modularen Aufbau der Ebene wird jede der nachfolgenden angeführten Betriebsarten durch einen eigenen Funktionsbaustein, welcher eine Art Methode repräsentiert, abgebildet. Da jede Methode durch einen eigenen Funktionsbaustein dargestellt wird, muss zudem eine Instanziierung erfolgen, damit diese auch für das Anwenderprogramm verfügbar wird. Durch die Zusendung von Kommandos der Bedien-Ebene werden die jeweiligen Methoden (Betriebsarten) ausgeführt. Zudem werden durch die Hardware-Abstraktion Zustände übermittelt, wodurch die Methode entscheidet, wie sie reagiert. Somit wird die Programmierung übersichtlicher ausgeführt und ermöglicht zusätzlich eine einfache Handhabung bzw. Erweiterbarkeit der Ebene. Darüber hinaus bietet dieser Aufbau den Vorteil, dass jede Betriebsart für sich unabhängig von anderen weiterentwickelt bzw. an neue Anforderungen angepasst werden kann.

- *Handbetrieb – Steuerung Lift – Tippbetrieb kontinuierlich*

Der Tippbetrieb ermöglicht ein einfaches Tippen des Liftes. Die Bewegungen aufwärts bzw. abwärts werden durch das Betätigen der Tasten *Tippen Auf* oder *Tippen Ab* auf der Visualisierung ausgeführt. Solange eine der Tasten aktiv ist, wird eine kontinuierliche Bewegung in die gewählte Fahrtrichtung zwischen den positiven und negativen Endlagenschaltern ausgeführt.

- *Handbetrieb – Steuerung Lift – Positionieren auf Position*

Durch das Positionieren auf Position wird die Möglichkeit geschaffen, eine Position, welche über ein Eingabefeld in Millimeter vorgegeben wird, anzufahren. Während des Positioniervorganges muss die Taste *Positionieren* solange aktiv bleiben, bis die Zielposition erreicht wird. Wird die Taste vor dem Erreichen der Position wieder freigegeben, stoppt die Bewegung.

- *Handbetrieb – Steuerung Lift – Positionieren auf Level*

Positionieren auf Level bildet die gleiche Funktionalität wie Positionieren auf Position ab, wobei nicht eine Position in Millimeter eingegeben werden muss, sondern die Positionierung mittels Selektierung eines Levels erfolgt, welchem eine Position bereits zugewiesen wurde.

- *Testbetrieb – Level Auf und Ab (Up and Down)*

Diese Betriebsart wird wie auch die nachfolgende (Testbetrieb – Level Zufall) für verschiedene Dauerlauftests verwendet. So können Tests über längere Zeit ohne ein übergeordnetes System, wie es für den Automatikbetrieb benötigt wird, durchgeführt werden. In diesem Modus werden aufsteigend bzw. absteigend bis zum Erreichen des höchsten bzw. des niedrigsten Levels ein Level nach dem anderen angefahren. Wurde das oberste bzw. das unterste Level erreicht, startet der Testlauf wieder von dem untersten bzw. dem obersten Level. Zudem ist es möglich die Funktion der Fahrbahnklappe für diese Betriebsart an- bzw. abzuwählen um auch diese unter annähernd realen Bedingungen zu testen.

- *Testbetrieb – Level Zufall (Random Mode)*

Im Gegensatz zum Testbetrieb – Level Auf und Ab werden in diesem Modus verschiedene Levels über einen Zufallsgenerator angesteuert. Der Lift verweilt im Zufallsmodus eine zufällige Zeit in einem Level, bis eine Positionierung zum nächsten erfolgt. Die Betriebsart bietet die Möglichkeit, den Lift beinahe unter den gleichen Bedingungen wie im Automatikbetrieb zu testen, nur das Be- und Entladen des Liftes kann nicht simuliert werden. Diese Betriebsart bietet ebenfalls die Möglichkeit der An- bzw. Abwahl der Fahrbahnklappenfunktion.

- *Automatik*

Der Automatikbetrieb stellt die Hauptbetriebsart des Liftes dar. In diesem Modus erfolgt die Positionierung durch Fahrbefehle, welche durch das übergeordnete Lagerverwaltungs-System erteilt werden. Nach dem Erreichen des Levels wird der Befehl von der Steuerung quittiert. Nach der Quittierung können Be- bzw. Entladevorgänge ausgeführt werden, die ebenfalls quittiert werden müssen. Somit wird sichergestellt, dass das übergeordnete System immer über die Informationen zum aktuellen Zustandes des Liftes verfügt.

5.4 Bedien-Ebene (Operation Mode)

Die oberste Ebene im SPS-Programm wird von der Bedien-Ebene gebildet. Sie stellt die Schnittstelle zu der dem Anwenderprogramm übergeordnete Ebene, der Visualisierung, und der unterlagerten Steuerungs-Ebene dar. Die Aufgabe der Ebene ist es, die von der Visualisierungs-Ebene zur Verfügung gestellten Befehle auszuwerten. Auf den Schnittstellen zwischen Bedien- und Visualisierung werden unter anderem allgemeine Signale übertragen, welche zum Betrieb des Liftes notwendig sind, aber auch Signale, die die Steuerung des Liftes betreffen. Erst in dieser Ebene erfolgt die Auswertung der Betriebsart und somit die Aktivierung der jeweiligen Methode in der Steuerungs-Ebene. Zur Visualisierung von verschiedenen Statusinformationen sammelt die Bedien-Ebene die Zustände von den jeweiligen untergeordneten Ebenen zusammen und stellt diese der Visualisierungs-Ebene zur Verfügung. Somit kann eine saubere Trennung der Ebenen beibehalten und das Programm durchwegs strukturiert ausgeführt werden

Zusätzlich bildet die Bedien-Ebene allgemeine Befehle, wie Fehlerquittierung und auch Not-Halt-Quittierung, aus den Befehlen der Visualisierung und bereitet diese für den jeweiligen Lift auf.

5.5 Visualisierungs-Ebene

Automatisierte System sind ohne eine Schnittstelle zwischen Mensch und Maschine heute kaum noch vorstellbar. Ein sogenanntes Human Maschine Interface erlaubet es, wichtige Statusinformationen wiederzugeben aber auch eine Steuerung des Systems. Zur Steuerung des Liftes, ist es unabdinglich, dass die Visualisierung mit der Steuerung kommuniziert. Damit dies erfolgen kann, wird die Visualisierungs-Ebene eingeführt. Über diese Schnittstelle werden alle Steuerbefehle der HMI an die Steuerung übergeben und auch aktuelle Zustände an das HMI-Gerät zurückgemeldet, um verschiedene Zustände zu visualisieren. Dabei werden von dieser Ebene die folgend angeführten Aufgaben erfüllt.

- *Einzelbetrieb*

Bereits in Unterkapitel 5.3 wurden die verschiedenen Betriebsarten zur Liftsteuerung aufgelistet. Diese realisieren jeweils einen Betrieb, in welchem mehrere Betriebsmittel, in erster Linie der Liftantrieb und die Fahrbahnklappe, gemeinsam agieren. Um einfache Einstell- oder auch Wartungsarbeiten an den verschiedenen Komponenten vornehmen zu können, wird eine einfache Möglichkeit zur Steuerung von einzelnen Hardwarekomponenten benötigt, welche durch den Einzelbetrieb abgebildet wird.

- *Statusanzeige*

Jedes automatisierte System muss dem Bediener Statusinformationen zur Verfügung stellen, was unter anderem durch das Anzeigen von Stör- bzw. Betriebsmeldungen erfolgen kann. Des Weiteren werden verschiedenste Statusinformationen der Betriebsmittel ebenfalls visualisiert.

- *Liftsteuerung*

Zur Inbetriebnahme sowie auch zu Wartungsarbeiten am Lift werden diverse Betriebsmodi benötigt. Dem Bediener werden durch die Visualisierung die Steuerung des Liftes und die Umschaltung der verschiedenen Betriebsarten ermöglicht.

- *Parameterverwaltung*

Für den Betrieb des Liftes sind verschiedenste Parameter notwendig. Diese beinhalten zum Beispiel die Level-Positionen, Verfahrgeschwindigkeiten, Beschleunigungen und Verzögerungen des Liftes aber auch weitere Parameter für die Ansteuerung von Betriebsmitteln. Um eine übersichtliche und zentrale Verwaltung der Parameter zu ermöglichen, werden alle Parameter in einem eigenen Menüpunkt auf der Visualisierung abgebildet.

- *Fehlerbehebung*

Wünschenswert ist natürlich ein fehlerfreier Betrieb einer Anlage, jedoch können immer wieder Störungen oder Betriebsmeldungen auftreten. In der Visualisierung werden die Meldungen in Klartext angezeigt und ermöglichen somit den Anlagenbediener eine einfache und schnelle Fehlerbehebung. Des Weiterem wird durch die Visualisierung auch die Hardware dargestellt, wodurch Hardwareausfälle in Klartext aber auch grafisch dargestellt werden können.

Zur Realisierung der hier ausgeführten Software-Architektur und der Erstellung eines Anwenderprogrammes unter Ansätzen der objektorientierten Programmierung ist es notwendig Programmierrichtlinien zu definieren. Diese Richtlinien werden im nachfolgenden Kapitel festgelegt und näher ausgeführt.

6 PROGRAMMIERRICHTLINIEN

Jedes Unternehmen, welches sich mit der Programmierung von Speicher-Programmierbaren Steuerungen auseinandersetzt definiert eigene Programmierrichtlinien und pflegt diese. Zusätzlich werden noch von vielen Steuerungsherstellern Empfehlungen zur Steuerungsprogrammierung ausgegeben.

Einer der Schlüssel zur Wiederverwendbarkeit von Steuerungssoftwarelösungen liegt darin, die Programmierung nachvollziehbar aufzubauen. Dieser Grundstein wird nicht erst im Laufe der Programmerstellung gelegt, sondern schon mit der Definition der Programmierrichtlinie. Während der Erstellung des Anwenderprogrammes hat der Programmierer dafür Sorge zu tragen, dass die in der Richtlinie verankerten Regeln umgesetzt und eingehalten werden. Nur wenn dies der Fall ist, kann eine Softwarelösung erstellt werden, die für verschiedene Anwender lesbar, nachvollziehbar und schnell verständlich ist.

In vielen Unternehmen existieren Programmierrichtlinien ebensolange wie auch die Programmierung selbst. Sollten diese nicht vorhanden sein oder müssen sie durch einen Hersteller- bzw. Steuerungsgenerationenwechsel (z.B.: STEP7 auf TIA Portal) neu definiert werden, bietet sich an, einen Leitfaden des jeweiligen Steuerungsherstellers zu verwenden. In diesem konkreten Fall handelt es sich um eine Siemens TIA Portal-Steuerung, daher wird auch auf das Dokument *Programmierstyleguide für S7-1200/S7-1500* von Siemens zurückgegriffen. Dieser Styleguide von Siemens wurde zur Programmierung von Anwenderprogrammen und Bibliotheken in den in der IEC 61131-3 definierten Programmiersprachen erstellt. Das Dokument dient als Leitfaden, nicht alles kann und soll aus diesem Dokument zur Definition der Richtlinie übernommen werden. Der Styleguide kann nicht für alle Anwendungen passen, zudem sollte auch eigene Erfahrung zur Programmierung eingebracht werden. Unabhängig davon muss das Ziel sein, durch die Programmierrichtlinie einen einheitlichen und durchgängigen Stil der Programmierung zu schaffen, um somit das Anwenderprogramm lesbarer und verständlicher zu gestalten. Aufbauend darauf wird die Möglichkeit geboten, dass mehrere verschiedene Anwender mit der gleichen Programmierung arbeiten können. Werden diese Punkte eingehalten, so kann davon ausgegangen werden, dass die Software so aufgebaut wird, dass eine einfache Wartung von unterschiedlichen Anwendern möglich ist, aber auch ein wichtiger Schritt in Richtung Wiederverwendbarkeit von Programmteilen getätigt wird.[39]

In den nachfolgenden Unterkapiteln werden Richtlinien definiert, welche zur Programmierung des YLOG-Shuttle-Systems angewandt werden. Diese Richtlinien dienen dazu, dass die Regeln zur *OOP mit Funktionsbausteinen* von Siemens Steuerungen eingehalten werden.

6.1 Hardware und Peripherie

Bereits unter Abschnitt 2.2.1 wurde angeführt, dass TIA Portal-Steuerungen über mehrere Schnittstellen zum Datenaustausch mit der Umwelt verfügen. Durch diese Hardware-Schnittstellen (ProfiNet oder Profibus DP) werden die zur Verfügung gestellten Ein- und Ausgänge in der Peripherie der Steuerung

[39] Vgl. Siemens (2016), Online-Quelle [4.12.2017], S. 4 f.

abgebildet. Nachfolgend wird ein beispielhafter Auszug der Hardwarekonfiguration des Liftes dargestellt, sowie das Konzept zur Bezeichnung von Ein- und Ausgängen beschrieben.

6.1.1 Hardwarekonfiguration

Werden in der Hardwarekonfiguration weitere Feldgeräte hinzugefügt, muss diesen eine eindeutige Bezeichnung zugeordnet werden, um damit die Diagnosemöglichkeiten des Systems optimal nutzen zu können. Abbildung 22 zeigt hierzu beispielhaft die Konfiguration einer ET200SP Baugruppe. Der Abbildung ist zu entnehmen, dass jedes Modul über eine eindeutige Kennzeichnung verfügt.

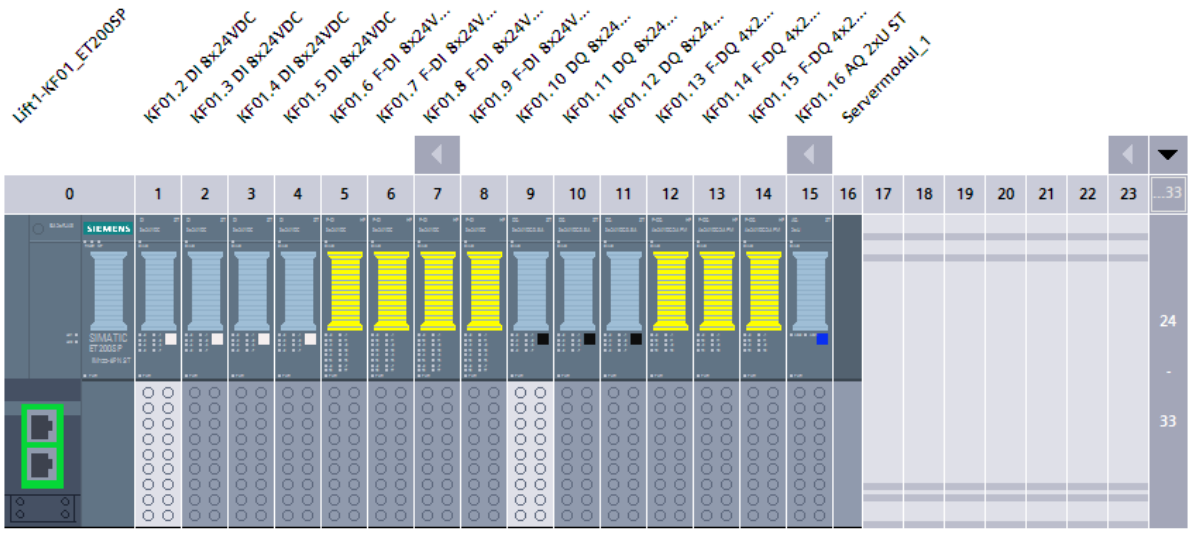


Abbildung 22: Beispielhafte Konfiguration einer ET200SP Baugruppe, Quelle: Eigene Darstellung.

Ein Großteil der Kommunikation zu Feldgeräten und anderen Systemen wird bereits über ProfiNet realisiert, was wiederum zu einer schrittweisen Ablösung von ProfiBus DP führt. Ein großer Vorteil von ProfiNet liegt in der einfachen und umfangreichen Diagnoseeigenschaft der Module. Um Statusmeldungen zu Feldgeräten unmissverständlich zuordnen zu können, ist es unbedingt notwendig, in der Hardwarekonfiguration dem Gerät eine eindeutige, verständliche und nachvollziehbare Bezeichnung zu zuteilen. Diese Bezeichnung setzt sich, wie in Tabelle 5 als Referenz dargestellt, aus einer Zuordnung zum Lift sowie einer elektrischen Ortskennung und abschließend der Bezeichnung des Feldgerätes und eventuell einer weiteren Zusatzkennung, wie zum Beispiel Import oder Export, zusammen.

Zuordnung	Elektrische Ortskennung	Bezeichnung	Zusatzkennung
Lift 1	KF02	PLC	
Lift 1	TA01	Lift Antrieb	
Lift 1	TA02	FU	Import

Tabelle 5: Beispielhafte Bezeichnungen der Hardware Konfiguration, Quelle: Eigene Darstellung.

Eine gute Diagnose-Eigenschaft ist nicht nur für Feldgeräte notwendig, auch für die Ein- und Ausgänge soll eine solche Möglichkeit geboten werden, wie im folgenden Abschnitt gezeigt wird.

6.1.2 Ein- und Ausgänge

Grundsätzlich sollen Symboliken in englischer Sprache verfasst werden und müssen eindeutig sein. Das bedeutet, dass keine mehrfachen Bezeichnungen, auch wenn diese sich durch Groß- und Kleinschreibung unterscheiden, erlaubt sind. TIA Portal erlaubt zudem sprachspezifische Sonderzeichen wie zum Beispiel. ä, ö, ü usw., welche aber nicht in Symboliken verwendet werden. Ebenso dürfen keine Leerzeichen oder sonstige Sonderzeichen in Symboliken vorkommen.[40]

Seitens Siemens wird die Schreibweise *camelCasing* empfohlen. Kennzeichnend für *camelCasing* ist, dass grundsätzlich das erste Zeichen des Bezeichners kleingeschrieben wird, außerdem findet Groß- und Kleinschreibung innerhalb des Bezeichners Anwendung. Dabei wird jedes konstituierende Wort im Bezeichner mit einem Großbuchstaben begonnen wie zum Beispiel `staticMaxTemperatur`. Trotz der Empfehlung von Siemens wird in dieser Anwendung bewusst auf *camelCasing* für Variablenbezeichner verzichtet, da sich aus der Erfahrung gezeigt hat, dass diese Schreibweise in der Steuerungsprogrammierung für Variablen teilweise schwer lesbar ist. Aus diesem Grund wird auf einen der Programmiersprache C ähnlich verwendeten Stil zurückgegriffen, bei welchem eine Trennung durch einen Unterstrich erfolgt. Zusammengesetzte Bezeichnungen werden mittels *PascalCasing* dargestellt. Bei diesem Vorgehen werden jeweils die Anfangsbuchstaben der konstituierenden Wörter großgeschrieben. [41]

Die Symbolik setzt sich aus den folgenden Informationen zusammen und ermöglicht somit eine eindeutige Zuordnung.

- *EA-Bereich*

Eingänge werden mit „I“ gekennzeichnet, Ausgänge mit einem „Q“

- *Anlagen-Teil*

Kennzeichnet, welchen Bereich des Liftes das Symbol zugeordnet wird. Allgemeine Variablen werden mit „Main“ gekennzeichnet. Variablen, die einem Lift zugeordnet werden, müssen mit der Bezeichnung des Liftes, wie in Tabelle 6 gezeigt, ausgeführt werden.

[40] Vgl. Siemens (2016), Online-Quelle [4.12.2017], S. 11.

[41] Vgl. Wagner (2007), Online-Quelle [4.12.2017].

- *Bereich*

Weist dem Symbol den Bereich des Liftes wie Import oder Export zu.

- *Betriebsmittel*

Definiert das Symbol und ordnet diesem einem realen Objekt zu.

- *Zusatzinformation*

Verschafft Auskunft über die Funktionalität des Ein- bzw. Ausganges.

Mit dieser Regelung wird eine Symbolik erstellt, welche schon beim Lesen eine eindeutige Zuordnung erlaubt. Die nachfolgende Tabelle 6 zeigt hierzu beispielhaft, wie die Symbolik aus den definierten Regeln und verschiedenen Informationen zusammensetzt wird. In dieser Tabelle wird aus Darstellungsgründen Trennzeichen mit TZ abgekürzt.

EA-Bereich	TZ	Anlagen Teil	TZ	Bereich	TZ	Betriebs Mittel	TZ	Zusatz Information
I	_	Lift1	_	Import	_	LightBarrier	_	GapControl
I	_	Lift1	_	Export	_	LightBarrier	_	GapControl
I	_	Lift1	_	Export	_	Pusher	_	Occupied
I	_	Main	_			Fuse	_	Ok

Tabelle 6: Beispielhafte Zusammensetzung der Ein- und Ausgangssymbolik, Quelle: Eigene Darstellung.

6.2 Bausteine

Die IEC 61131-3 definiert wie bereits in Unterkapitel 3.2 beschrieben, die drei Bausteintypen Funktion, Funktionsbaustein und Programm. Die Programmierung des Anwenderprogrammes erfolgt auch mit TIA Portal durch Funktionen und Funktionsbausteinen, jedoch wird die Organisation des Programmes nicht, wie in der IEC 61131-3 beschrieben mit der POE Programm sondern mit Organisationsbausteinen (siehe Unterabschnitt 2.2.2.1) umgesetzt. Zudem werden für das Anwenderprogramm Datenbausteine zur Verfügung gestellt, welche zur Instanziierung von Funktionsbausteinen oder zur Sicherung von Daten in der Steuerung verwendet werden können. Dabei können Datenbausteine, welche als Datenspeicher dienen, mittels PLC-Datentypen einfach und ausführlich strukturiert werden. Im Nachfolgendem werden Regeln zur Programmierung von Funktionen und Funktionsbausteinen definiert, um eine bestmögliche Umsetzung der Methode *OOP mit Funktionsbausteinen* garantieren zu können.

6.2.1 Funktionsbaustein und Funktion

Zur Programmierung von Anwenderprogrammen stellen Funktionsbausteine wie auch Funktionen ein besonders wichtiges Werkzeug dar, was auch in dieser Anwendung der Fall ist. Viele Applikationen können nur mittels Funktionsbausteinen realisiert werden, für viele andere sind aber auch Funktionen völlig ausreichend. Aus diesem Grund muss zuerst bedacht werden, welche Funktionalitäten der Baustein erfüllen muss und ob dafür zwingend der Einsatz eines Funktionsbausteines notwendig ist, oder auch eine

Funktion als vollkommen ausreichend erscheint. Unabhängig davon, ob ein FB oder ein FC zur Anwendung kommt, stellen die nachfolgenden Regeln einen wichtigen Ansatz zur Realisierung der OOP mit Funktionsbausteinen dar. Ohne die Einhaltung dieser kann nicht sichergestellt werden, dass die Definition der objektorientierten Programmierung nach IEC 61131-3:2003 eingehalten werden kann.

- *Baustein Symbolik*

Aus der Bausteinbezeichnung muss eindeutig nachvollzogen werden können, zu welcher Familie der Baustein gehört und welche Aufgabe dieser erfüllt.

Der Bezeichner setzt sich aus den folgenden Gruppen zusammen

- *Bausteinart*
  - Funktionsbaustein (gekennzeichnet als Fb)
  - Funktion (gekennzeichnet als Fc)
  - Bibliotheksbaustein (gekennzeichnet als Lib)
- *Familie*
  - Lift 1
  - Motor
  - Valve usw.
- *Funktion*
  - Call
  - Control
  - Error usw.

So werden Funktionsbausteine die zur Familie Lift gehören, wie in der folgenden Tabelle 6 beispielhaft dargestellt, bezeichnet.

Bausteinart	Familie	Funktion	Bausteinbezeichnung	Beschreibung
Fb	Lift1	Call	FbLift1Call	Lift 1 – Aufrufender Baustein
Fb	Lift1	Control	FbLift1Control	Lift 1 – Steuerung
Fb	Lift1	Error	FbLift1Error	Lift 1 – Fehler Auswertung

Tabelle 7: Beispielhafte Bezeichnungen von Funktionsbausteinen, Quelle: Eigene Darstellung.

Funktionsbausteine, aber auch Funktionen, welche als Bibliotheksbausteine verwaltet werden, müssen folglich auch als solche gekennzeichnet werden. Ebenfalls muss vorausgesetzt werden, dass aus der Bausteinbezeichnung des Bausteins eine sofortige Zuordnung getroffen werden kann. Eine derartige Zuordnung wird durch die Verwendung von Bausteinbeschreibungen vereinfacht. Zudem besteht mit TIA Portal der Vorteil, dass eine Sprachumschaltung unterstützt wird, wodurch Projekttexte wie Bausteinbeschreibungen, mehrsprachig ausgeführt werden können. Tabelle 8 veranschaulicht, wie Bezeichner für Bibliotheksbausteine beispielhaft aussehen können.

Bausteinart	Familie	Funktion	Bausteinbezeichnung	Beschreibung
Lib	Flap	Control	LibFlapControl	Flap – Klappen Steuerung
Lib	Sensor	Debounce	LibSensorDebounce	Sensor – Sensor Entprellung
Lib	Operation	Mode	LibOperationMode	Operation – Betriebsarten

Tabelle 8: Beispielhafte Bezeichnungen von Bibliotheksbaustein, Quelle: Eigene Darstellung.

- *Bausteinkopf*

Aus der Erfahrung zeigt es sich als besonders sinnvoll, wichtige Informationen zu Bausteinen, unabhängig davon, ob es sich hierbei um einen Organisations- oder Funktionsbaustein sowie eine Funktion handelt, im ersten Kommentarfeld des Bausteines zu vermerken. Mit diesem Eintrag im Baustein muss die Möglichkeit geboten werden, sofort den Verwendungszweck sowie den Ersteller des Bausteines erheben zu können. Zudem müssen weitere Informationen wie das Erstellungsdatum vermerkt sein. Darüber hinaus unterliegen Anwenderbausteine einem Entwicklungsprozess, daher wird festgelegt, dass Änderungen zumindest mit einem Datum sowie einem Kommentar zum Änderungsgrund versehen werden. Zudem muss der Programmierer vermerket werden, um somit eine Nachverfolgung von Änderungs- und Entwicklungsprozessen zu vereinfachen. Nachfolgend wird gezeigt, wie das Kommentarfeld ausgeführt werden soll.

```
================================================================================
Knapp Industry Solutions Copyright (c) 2017
--------------------------------------------------------------------------------
Function:     Template for Function Blocks
Autor:        Roland Robosch
Creation:     26.08.2017
--------------------------------------------------------------------------------
Change Log Table
................................................................................
Date                Comment                              Editor
26.08.2017          main functions                       Roland Robosch
================================================================================
```

Außerdem unterstützt TIA-Portal die Verwaltung von Bausteinen mittels Bibliotheken, welche eine Versionierung von Bausteinen vereinfachen.

- *Ein- und Ausgangsschnittstelle*

Eingänge stellen eine Schnittstelle zu anderen Objekten dar und versorgen somit den Baustein mit den für die Abarbeitung nötigen Informationen. Ausgänge dienen als Rückmeldung an die Umwelt. Die einzelnen Bezeichner der Schnittstelle werden mit dem unter Abschnitt 6.1.2 erwähnten Verfahren *PascalCasing* beschrieben. Bei dieser Methode werden jeweils die Anfangsbuchstaben großgeschrieben. Zur Kennung einer Ein- bzw. Ausgangsvariable wird hierzu vor dem Bezeichner für einen Eingang ein „in“ und für einen

Ausgang ein „q“ gesetzt. Diese Notation verbessert die Lesbarkeit des Programmcodes und vereinfacht zudem die Programmierung durch die optimierte Verwendung der Autovervollständigung.

- *Statische und temporäre Variablen*

Funktionsbausteine besitzen im Gegensatz zu einer Funktion einen Speicher, welcher in einem Instanz-Datenbaustein hinterlegt ist. Dieser ermöglicht es Zustände von internen Variablen, sogenannte statische Variablen, über einen SPS-Zyklus hinaus zu sichern. Hingegen werden temporäre Variablen in jedem Zyklus neu gebildet und sind somit auch nur in diesem gültig. Zur Kennung von statischen und temporären Variablen werden diese speziell in der Schnittstelle des Bausteines ausgeführt. In Abbildung 23 wird gezeigt, wie statische und temporäre Variablen in einem Funktionsbaustein strukturiert werden. Statische Variablen werden in einer Struktur mit der Bezeichnung `static` zusammengefasst, welche wiederum in weitere Strukturen bezogen auf die Datentypen unterteilt wird. Dasselbe Vorgehen wird auch bei temporären Variablen angewandt, welche in der Struktur `_temp` zusammengefasst werden. Der große Vorteil dieser Strukturierung liegt darin, dass der Zugriff auf die Variable durch die Autovervollständigung von TIA Portal sehr einfach und schnell erfolgen kann. Ein weiterer Aspekt ist die Lesbarkeit des Codes. Durch diese Art der Bezeichnung wird auf den ersten Blick verständlich, um welche Type von Variable es sich handelt, und ermöglicht eine sofortige Identifizierung des Datentyps.

5		Static							
6		static	Struct	...	☑	☑	☑	☐	static variables
7		Bool	Struct	...	☑	☑	☑	☐	Bool
8		Byte	Struct	...	☑	☑	☑	☐	Byte
9		Int	Struct	...	☑	☑	☑	☐	Intager
10		Dint	Struct	...	☑	☑	☑	☐	Double Intager
11		Real	Struct	...	☑	☑	☑	☐	Real
12		Temp			☐	☐	☐	☐	
13		_temp	Struct	...	☐	☐	☐	☐	temporary variables
14		Bool	Struct	...	☐	☐	☐	☐	Bool
15		Byte	Struct	...	☐	☐	☐	☐	Byte
16		Int	Struct	...	☐	☐	☐	☐	Intager
17		Dint	Struct	...	☐	☐	☐	☐	Double Intager
18		Real	Struct	...	☐	☐	☐	☐	Real

Abbildung 23: Variablen Struktur Funktionsbaustein, Quelle: Eigene Darstellung.

- *Variablenzugriffe*

Sofern es sich nicht um einen aufrufenden Baustein handelt, ist nur die Verwendung von internen Variablen gestattet. Das bedeutet, Zugriffe auf globale Datenbausteine und Variablentabellen sind nicht erlaubt. Dadurch wird sichergestellt, dass bei Mehrfachverwendung von FBs und FCs keine fehlerhaften Programmausführungen durch falsche Variablenzustände zustande kommen. Um weiteren Fehlern vorzubeugen ist ein Schreib- wie auch Lesezugriff auf Instanzvariablen strengstens untersagt. Werden Instanzvariablen außerhalb des Bausteines geschrieben oder gelesen, besteht die Gefahr, dass diese nicht aktuell sind, da zuvor eine Änderung der Schnittstelle erfolgt sein könnte. Solche Zugriffe können zu absolut undefinierten Zuständen führen und sind daher verboten.

- *Aufruf*

Funktionsbausteine, welche nicht als Stationsaufruf ausgeführt sind, werden als Multi-Instanz im aufrufenden Baustein ausgeführt. Dadurch können gekapselte Funktionen wie zum Beispiel die Steuerung des Liftes erstellt werden.

- *Quellen*

Um ein einfaches Debugging sicherzustellen, muss auf den Import bzw. die Verwendung von Quellen verzichtet werden. Es werden ausschließlich Bausteine programmiert und Standardbausteine wie zum Beispiel der Ansteuerungsbaustein der Fahrbahnklappe werden durch die Bibliothek verwaltet.

- *Programmierung*

Siemens empfiehlt aufgrund des Performancevorteiles die gesamte Programmierung in SCL auszuführen. Sollte es jedoch zu Aufrufen von einzelnen Bausteinen kommen, wie es unter anderem bei den Stationsaufrufen der Fall ist, werden KOP und FUP empfohlen. Zudem bietet für bestimmte Anwendungen auch die Anweisungsliste nach wie vor diverse Vorteile. Dahingehend ist es dem Programmierer selbst überlassen, welche Programmiersprache gewählt wird, abgesehen von Organisationsbausteinen, welche nicht in SCL programmiert werden. Was jedoch nicht der Fall sein darf, ist, dass sich in einem Baustein verschiedene Netzwerke welche in SCL, KOP, FUP oder AWL programmiert wurden, wiederfinden. Prinzipiell sollen Bausteine, die Programmaufrufe bzw. eine Vielzahl an Statusabfragen beinhalten, in FUP erstellt werden. Komplexe Bausteine, wie sie zum Beispiel zur Ansteuerung von Hardware-Komponenten benötigt werden, sollen in SCL erstellt werden. Zusätzlich sollen keine Netzwerke in Bausteinen entstehen, welche sich über mehrere Seiten erstrecken und somit nicht mehr lesbar und nachvollziehbar sind, wodurch diese vor allem auch kein Debugging mehr ermöglichen.

- *Initialisierung von Variablen*

Um undefinierte Zustände durch nicht initialisierte temporäre Variablen zu verhindern, wird im ersten Netzwerk des Bausteines eine Initialisierung dieser vorgenommen. Dies gilt prinzipiell für alle Bausteine. In Kombination mit der Strukturierung der temporären Variablen gestaltet sich die Initialisierung dieser sehr einfach, wie in Abbildung 24 gezeigt. Zudem wird durch dieses Vorgehen der Vorteil geschaffen, dass neue hinzugefügte Variablen automatisch initialisiert werden.

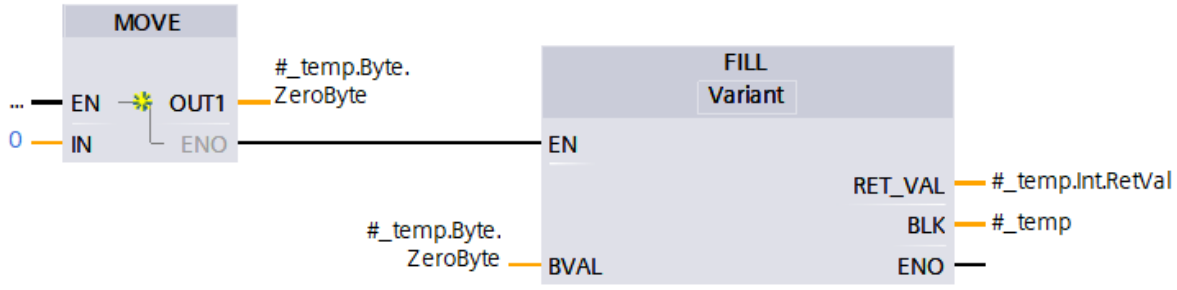
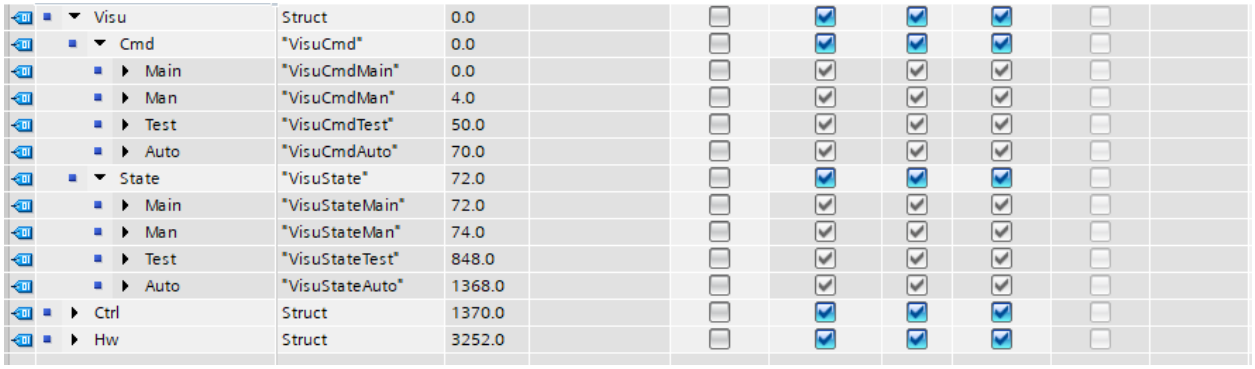


Abbildung 24: Initialisierung temporärer Variablen, Quelle: Eigene Darstellung.

6.2.2 Datenbausteine und Datentypen

Datenbausteine dienen zur Speichererweiterung des Anwenderprogrammes. Diese sollen grundsätzlich strukturiert ausgeführt werden. Das bedeutet, der DB soll mittels Strukturen, in Abhängigkeit zu deren Verwendung, untergliedert werden (siehe Abbildung 25).

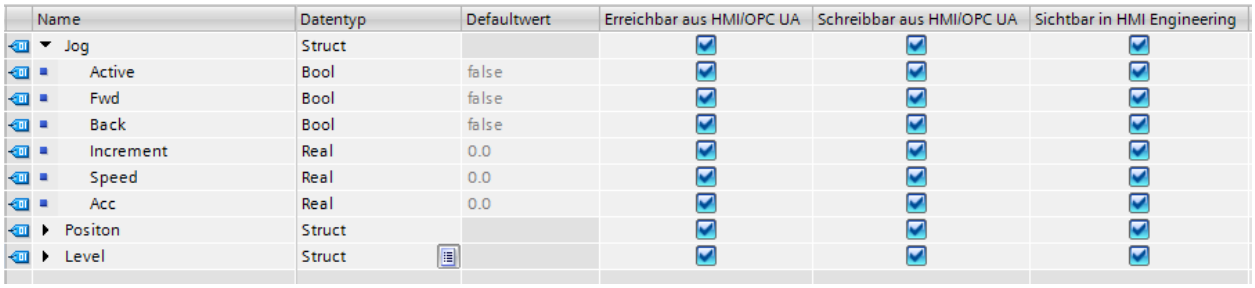


Visu	Struct	0.0	☐	☑	☑	☑	☐
Cmd	"VisuCmd"	0.0	☐	☑	☑	☑	☐
Main	"VisuCmdMain"	0.0	☐	☑	☑	☑	☐
Man	"VisuCmdMan"	4.0	☐	☑	☑	☑	☐
Test	"VisuCmdTest"	50.0	☐	☑	☑	☑	☐
Auto	"VisuCmdAuto"	70.0	☐	☑	☑	☑	☐
State	"VisuState"	72.0	☐	☑	☑	☑	☐
Main	"VisuStateMain"	72.0	☐	☑	☑	☑	☐
Man	"VisuStateMan"	74.0	☐	☑	☑	☑	☐
Test	"VisuStateTest"	848.0	☐	☑	☑	☑	☐
Auto	"VisuStateAuto"	1368.0	☐	☑	☑	☑	☐
Ctrl	Struct	1370.0	☐	☑	☑	☑	☐
Hw	Struct	3252.0	☐	☑	☑	☑	☐

Abbildung 25: Strukturierung eines Datenbausteines, Quelle: Eigene Darstellung.

Zur Vereinfachung der Strukturierung des Datenbausteines sollen PLC-Datentypen verwendet werden. Diese ermöglichen eine zusätzliche Gliederung und bieten zudem die Möglichkeit Strukturen anwendungsbezogen, wie in Abbildung 26 gezeigt, zu erstellen. Daraus wird ein weiterer Vorteil für die Autovervollständigung gewonnen, da durch die Gliederung eine einfache Orientierung in den jeweiligen Datenbausteinen schon während der Programmierung ermöglicht wird.

Die Bezeichnungen in den Strukturen wird mittels *PascalCasing* realisiert, da durch diese Schreibweise die Lesbarkeit in den Strukturen sowie auch im Code vereinfacht wird.



Name	Datentyp	Defaultwert	Erreichbar aus HMI/OPC UA	Schreibbar aus HMI/OPC UA	Sichtbar in HMI Engineering
Jog	Struct		☑	☑	☑
Active	Bool	false	☑	☑	☑
Fwd	Bool	false	☑	☑	☑
Back	Bool	false	☑	☑	☑
Increment	Real	0.0	☑	☑	☑
Speed	Real	0.0	☑	☑	☑
Acc	Real	0.0	☑	☑	☑
Positon	Struct		☑	☑	☑
Level	Struct		☑	☑	☑

Abbildung 26: Beispielhafter Aufbau eines PLC-Datentyps, Quelle: Eigene Darstellung.

7 PROGRAMMIERUNG

Mittels der in Unterkapitel 4.4 durchgeführten Evaluierung, zeigt sich, dass eine Umsetzung der Programmierung nach der Methode *OOP mit Funktionsbausteinen*, welche in Unterkapitel 4.1 angeführt ist, am besten zur Programmierung eines Shuttle-Liftes geeignet ist. Aufbauend auf dieser Variante wurde eine Programmier- sowie auch Visualisierungsrichtlinie definiert, mit welcher die nachfolgend dargestellte Programmierung erfolgt. Als erster Schritt wird ein Einblick in die allgemeine Struktur der Lift-Software geschaffen. Aufsetzend auf dieser werden die einzelnen Ebenen und deren Funktionen sowie auch die Programmierung ausführlich behandelt. Abschließend wird das Zusammenspiel zwischen SPS-Programm und Visualisierung sowie auch das Konzept der Fehlerauswertung erläutert.

7.1 Lift

Zur Steuerung eines Shuttle-Liftes werden mehrere verschiedene Komponenten benötigt. Die Hauptbestandteile des Liftes werden bereits in Kapitel 2 beschrieben. Damit der Lift jedoch einen Shuttletransport zwischen unterschiedlichen Ebenen eines Regales durchführen kann, ist es notwendig, diese Komponenten zu einer gesamten mit einander funktionierenden Einheit zusammen zu setzen. Dies erfolgt durch die Programmierung des SPS-Programmes. Dabei werden verschiedene Schichten der Programmierung eingeführt, um somit das gesamte System nahe an dem realen Objekt Lift abzubilden.

7.1.1 Programmaufbau

Alan Kay definierte ein Objekt, als einen Baustein, auf welchen durch die Zusendung von Nachrichten zugegriffen werden kann. Dabei wird vom Objekt durch die jeweiligen Zustände und der programmierten Logik entschieden, wie auf die jeweilige Nachricht reagiert wird. Dieser Ansatz wird in der Programmierung der Steuerungssoftware des Liftes realisiert, welche im Nachfolgenden behandelt wird.

Die Ausführung des Programmes wird, wie in Abbildung 27 veranschaulicht, durch den Organisationsbaustein 1 umgesetzt. Der OB1 stellt dabei das Main-Programm dar und führt den Aufruf des Liftes aus. Werden mehr als ein Lift programmiert, müssen alle weiteren Aufrufe ebenfalls, wie in Abbildung 27 beispielhaft abgebildet, im OB1 erfolgen.

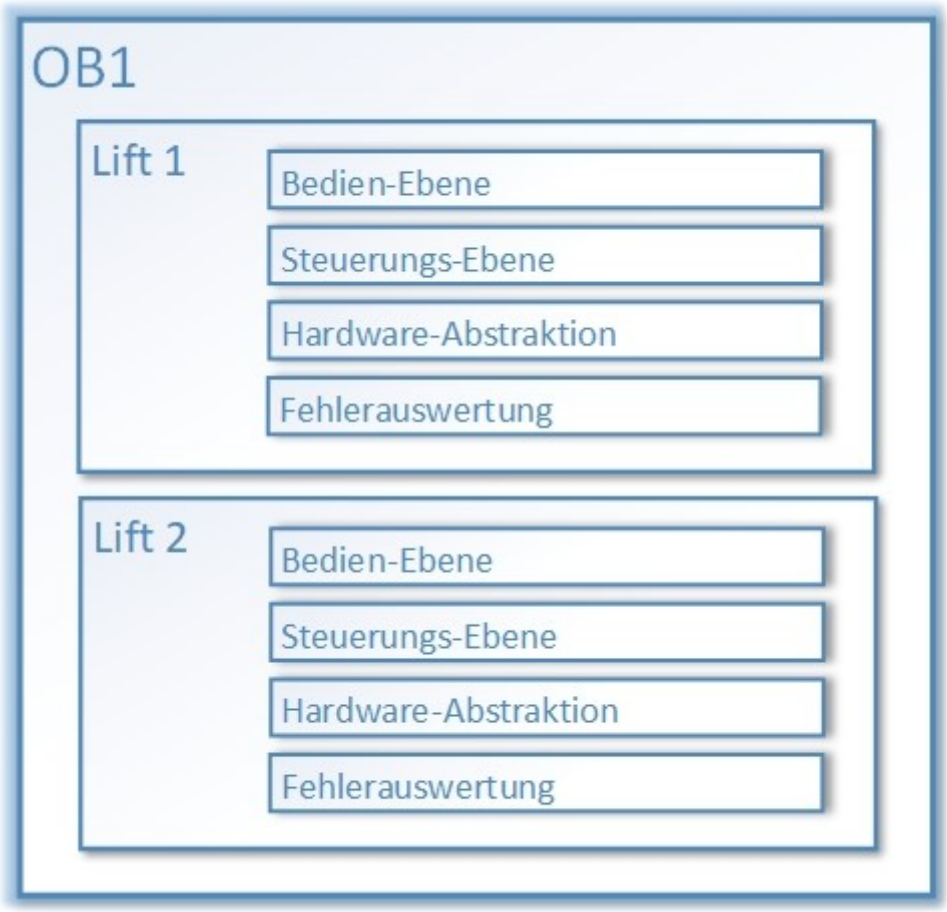


Abbildung 27: Programmaufbau OB1, Quelle: Eigene Darstellung.

Der OB1 führt den Aufruf des jeweiligen Liftes aus. Der zugehörige grundlegende Programmaufbau wurde bereits zuvor in Kapitel 5 angeführt. Die Programmierung des Liftes findet in einem eigenen Funktionsblock statt, welcher in der Abbildung 27 beispielhaft als Lift 1 oder Lift 2 dargestellt wird. Dieser FB bildet den aufrufenden Baustein des jeweiligen Liftes. Überdies gliedern sich diese Bausteine in die weiteren Schichten Bedien- und Steuerungs-Ebene sowie Hardware-Abstraktion des jeweiligen Liftes. Nicht in die Abarbeitung des OB1 fällt die oberste Ebene, welche als Visualisierungs-Ebene bezeichnet wird, über jene werden Steuerungsbefehl, im Sinne der OOP die Nachrichten, an den untergeordneten Betriebsarten-Layer übermittelt. Durch die interne Logik des Betriebsartenbausteines werden die von der Logik abhängigen Nachrichten an die darunter angeordnete Schicht über die Kommando-Schnittstelle weitergeleitet. Dieses Objekt meldet über die Status-Schnittstelle, siehe hierzu Abbildung 28, an das übergeordnete Objekt den aktuellen Status zurück. Weiterführend werden von der Steuerungs-Ebene an die unterste Ebene, welche als Hardware-Abstraktion bezeichnet wird, Nachrichten in Abhängigkeit von Zustand und Logik übermittelt. Zusätzlich erhält die Hardware-Abstraktion Nachrichten von der Visualisierungs-Ebene, um eine direkte Steuerung der verschiedenen Betriebsmittel im Einzelbetrieb realisieren zu können. Um den jeweiligen Zustand der Betriebsmittel zu visualisieren, meldet diese nicht nur den Status an die Steuerungs-Ebene, sondern auch direkt an die Visualisierungs-Ebene zurück.

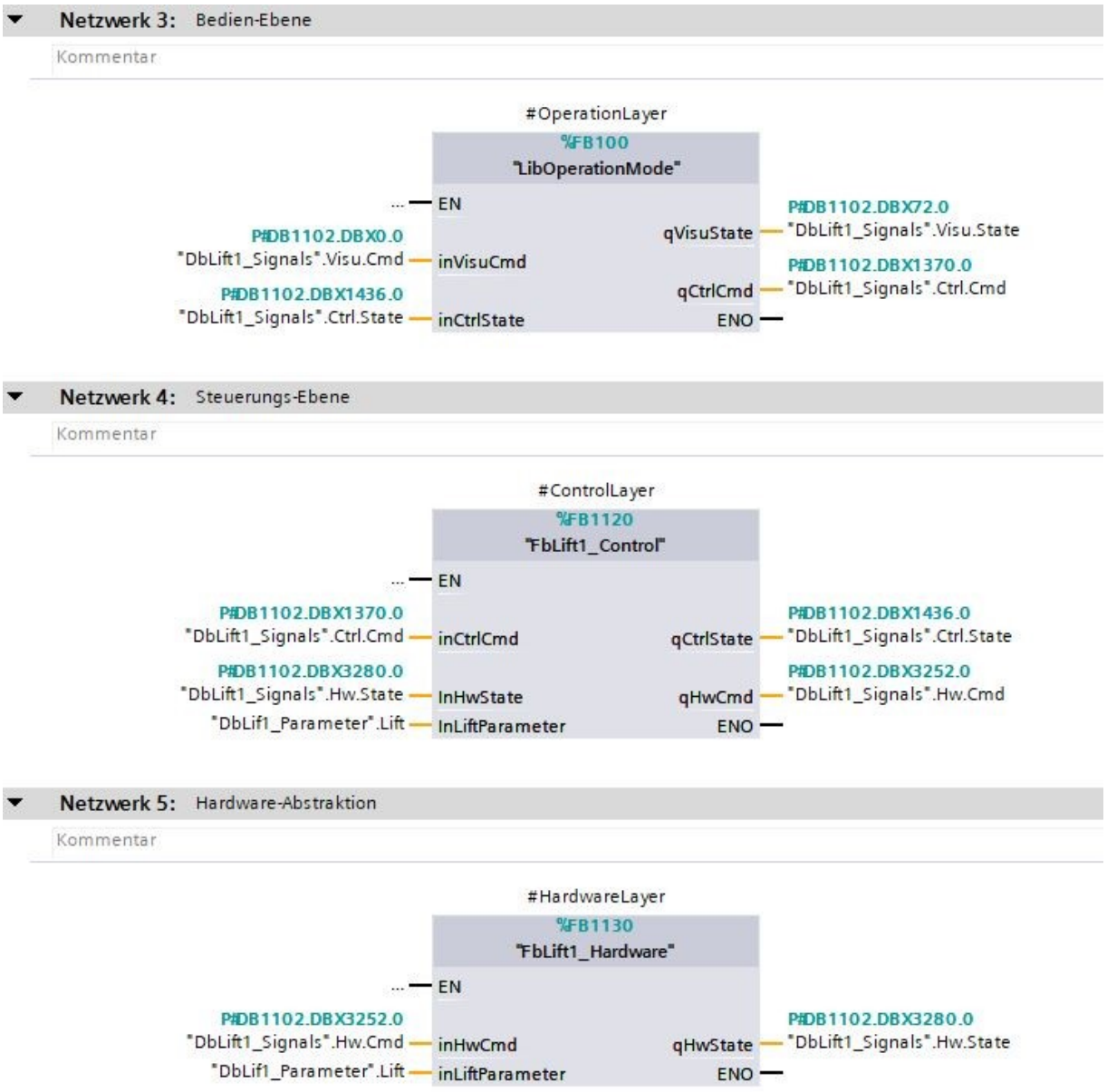


Abbildung 28: Aufrufender Baustein der Liftsteuerung, Quelle: Eigene Darstellung.

Dadurch, dass der Datenaustausch zwischen den Ebenen, wie bereits in Abbildung 20 im Unterkapitel 5.2 gezeigt, über die Kommando- und die Status-Schnittstelle erfolgt, müssen diese Schnittstellen definiert werden. Die Schnittstellen werden durch PLC-Datentypen, welche in der Bibliothek verwaltet werden, organisiert. Dabei unterteilen sich diese Typen jeweils in Kommando (Cmd) und Status (State) Datentypen. Das Vorgehen bildet den großen Vorteil, dass die Ebenen voneinander unabhängig sind. Somit kann von jeder Ebene die Logik individuell angepasst werden. Die Schnittstelle hingegen darf nicht verändert werden. Dieser Ansatz kann durch eine weitgehende Betrachtung auch auf die objektorientierte Programmierung zurückgeführt werden. So erlaubt die OOP bei der Vererbung, wie in Abschnitt 4.3.3 vermerkt, eine Anpassung der Logik, jedoch darf dabei die Schnittstelle sowie der Name des Objektes nicht verändert werden. Würde im Anwenderprogramm die Schnittstelle verändert werden, dann muss dies über die Datentypen erfolgen, welche durch die Bibliothek verwaltet werden, was wiederum zur Folge hätte, dass diese im gesamten Projekt automatisch aktualisiert wird. Auch eine Änderung der Objektbezeichnung hätte in dieser Ausführung der Programmierung keine größeren Auswirkungen. Der Objektaufruf erfolgt durch

eine Instanz, welche im aufrufenden Funktionsbaustein definiert ist. Würde der Bezeichner verändert werden, hätte das eine Aktualisierung der Instanz des aufrufenden Bausteines und des dazugehörigen Instanz-Datenbausteines zur Folge. Dies stellt jedoch kein Problem für das weitere Programm dar, da bereits in den Richtlinien festgelegt wurde, dass ein direkter Zugriff auf Instanz-Daten außerhalb der Instanz nicht gestattet ist. Des Weiteren stellt eine Veränderung der Schnittstelle, wie sich im nachfolgenden Abschnitt zeigen wird, für die Visualisierung kein großes Problem dar.

Aus der Abbildung 28 ist ersichtlich, dass die Schnittstellen der jeweiligen Objekte mit dem jeweilig dazugehörigen Datentyp versorgt werden müssen. Eine mögliche Variante wäre, direkt auf die Ein- bzw. Ausgangsschnittstelle der jeweiligen Objektinstanz zuzugreifen, was jedoch nicht erwünscht ist und auch andere Nachteile mit sich führen würde. Aus diesem Grund werden die Schnittstellen der verschiedenen Ebenen, wie in der nachfolgenden Abbildung 29 veranschaulicht, in einem eigenen Datenbaustein hinterlegt.

DbLift1_Signals

	Name	Datentyp	Offset	Startwert	Remanenz	Erreichbar a..	Schrei...	Sichtbar i...	Einstellwert
1	Static				☐	☐	☐	☐	☐
2	Visu	Struct	0.0		☐	☑	☑	☑	☐
3	Cmd	"VisuCmd"	0.0		☐	☑	☑	☑	☐
4	State	"VisuState"	72.0		☐	☑	☑	☑	☐
5	Ctrl	Struct	1370.0		☐	☑	☑	☑	☐
6	Cmd	"CtrlCmd"	1370.0		☐	☑	☑	☑	☐
7	State	"CtrlState"	1436.0		☐	☑	☑	☑	☐
8	Hw	Struct	3252.0		☐	☑	☑	☑	☐
9	Cmd	"HwCmd"	3252.0		☐	☑	☑	☑	☐
10	State	"HwState"	3280.0		☐	☑	☑	☑	☐

Abbildung 29: Aufbau des Datenbausteines für Liftsignale, Quelle: Eigene Darstellung.

Da die Signale im Datenbaustein hinterlegt sind, kann auf diese durch die Programmierung theoretisch in allen Bausteinen lesend sowie auch schreibend zugegriffen werden, was dennoch nur im jeweiligen aufrufenden Baustein erfolgen soll, um einen strukturierten Programmaufbau beizubehalten. Ferner kann durch dieses Vorgehen unter anderem ein definierter Schreib- sowie auch Lesezugriff von der Visualisierungs-Ebene auf die Kommando- und Status-Schnittstelle erfolgen.

7.1.2 Visualisierungs-Ebene

Die oberste Ebene wird durch die Visualisierungs-Ebene ausgeführt, mit welcher die Steuerung des gesamten Liftes erfolgt. Die Visualisierung wird nicht durch die Steuerung, sondern auf einem HMI-Gerät abgebildet. Das bedeutet, die verschiedenen Befehle werden nicht mittels Logik ausgelöst, sondern durch verschiede Eigenschaften diverser Objekte in der Visualisierung. Zur Erstellung der Visualisierung wird die Software TIA Portal WinCC Professional eingesetzt, mit der unter anderem Bildbausteine erstellt werden können.

Zur Steuerung des Liftes werden für jede einzelne Betriebsart eigene Bildbausteine erstellt. Diese bieten den Vorteil, den Projektierungsaufwand der Visualisierung zu minimieren. Bereits bei der Erstellung des Bildbausteines, wie in Abbildung 31 gezeigt, werden die in der Bibliothek vorhandenen Datentypen verwendet. Zusätzlich werden dem Bildbaustein weitere Eigenschaften hinzugefügt, welche mit den Datentypen der Bibliothek verknüpft werden. Auf diese Weise wird, ähnlich wie bei Funktionsbausteinen in

der Steuerung, dem Bildbaustein eine Schnittstelle zur Verfügung gestellt, welche durch die Instanziierung nach außen hin sichtbar wird.

Die Visualisierung soll in erster Linie zur Bedienung des Liftes dienen. Außerdem werden die Möglichkeiten geschaffen einzelne Betriebsmittel, wie zum Bespiel die Klappe, direkt zu steuern. Darüber hinaus muss es möglich sein, die für den Betrieb des Liftes notwendigen Parameter verwalten zu können und Diagnosemeldungen anzuzeigen.

Die Steuerung wird, wie zuvor schon beschrieben, über mehrere Betriebsarten umgesetzt, wobei jede Betriebsart durch ein eigenes Bild und einen eigenen Bildbaustein dargestellt wird. Abbildung 30 zeigt hierzu beispielhaft die Steuerung des Tipp-Betriebes.

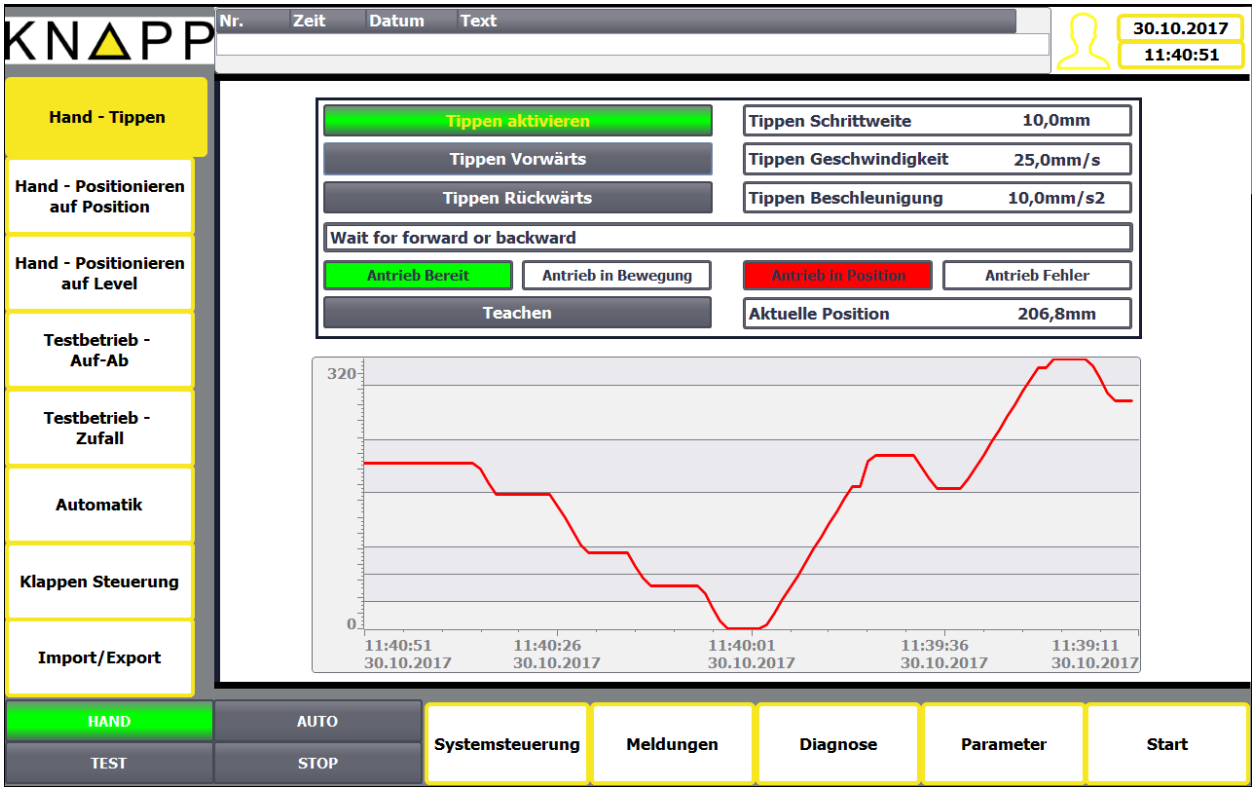


Abbildung 30: Bildbaustein Beispiel, Quelle: Eigene Darstellung.

Der Bedienteil der Betriebsart wird mit einem Bildbaustein abgebildet. Die Anwendung des Bildbausteines bietet zwei grundlegende Vorteile.

- *Mehrfachverwendung*

Durch den Einsatz von Bildbausteinen ist eine Änderung nur in der Bibliothek notwendig, die Aktualisierung der Instanzen des Bildbausteines wird mit der Versionsfreigabe umgesetzt.

- *Anwendung von Datentypen*

Bildbausteine verfügen über Schnittstellen, welche mit PLC-Datentypen des Anwenderprogrammes verknüpft werden können, wodurch die Parametrierung des Bildbausteines beschleunigt und vereinfacht wird.

Im Bildbaustein selbst können verschiedene Objekte wie Schaltflächen, Textfelder oder auch zum Beispiel Ein- und Ausgabefelder erstellt werden. Diese Objekte werden über deren Eigenschaften mit den Variablen der zuvor erzeugten Schnittstellen, in diesem Anwendungsfall die Schnittstellen *Cmd, State* und *HwState*, verknüpft.

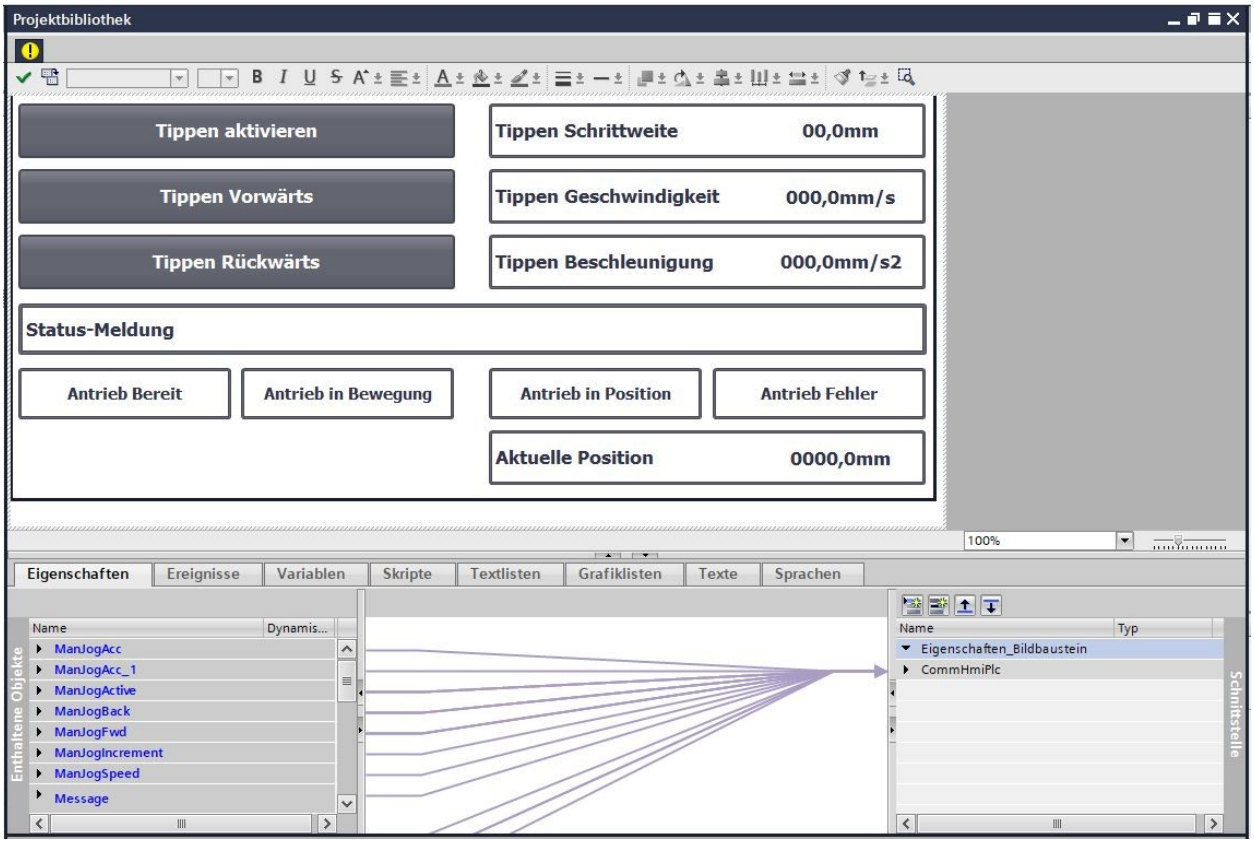


Abbildung 31: Bildbaustein in Bearbeitung des Tipp-Betriebes, Quelle: Eigene Darstellung.

In diesem Zustand handelt es sich nur um den Bildbaustein für sich alleine, wie auch bei Funktionsbausteinen in der Steuerungssoftware muss auch der Bildbaustein instanziiert werden, um diesen nützen zu können. Bei der Instanziierung wird auch hier gleich wie in der Steuerung eine Kopie des Objektes erzeugt, welche wie in der nachfolgenden Abbildung 32 alle Eigenschaften des Bildbausteines erbt.

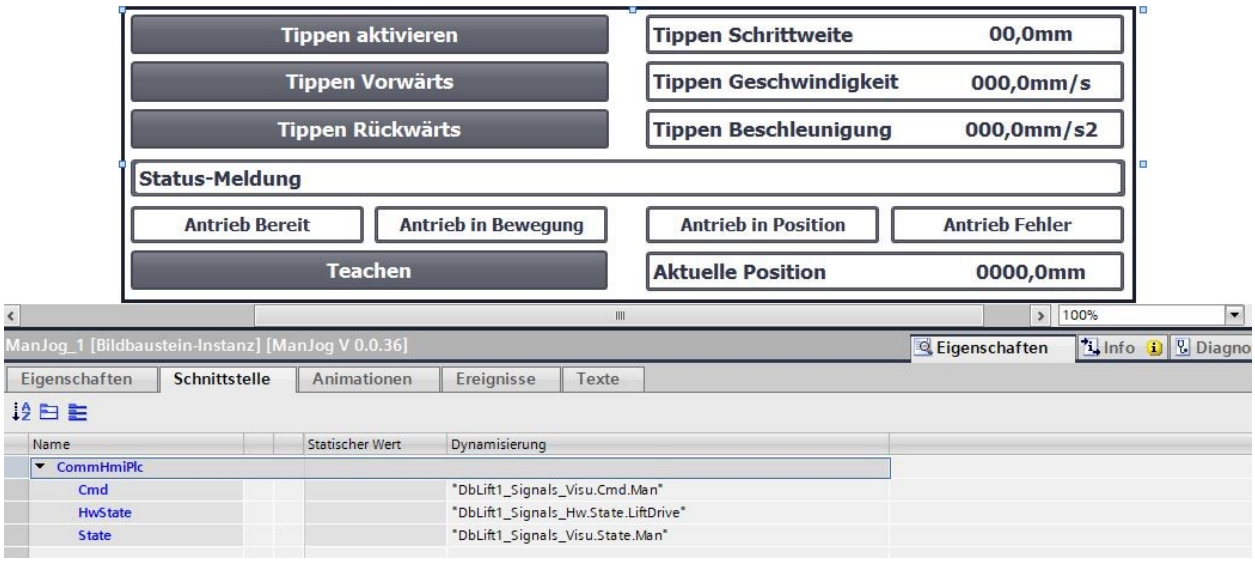


Abbildung 32: Instanziierter Bildbaustein des Tipp-Betriebes, Quelle: Eigene Darstellung.

Nach der Instanziierung des Bildbausteines werden ebenfalls Schnittstellen zur Verfügung gestellt, mit denen die Anbindung an die Steuersignale des Datenbausteines erfolgt. Dazu muss die Schnittstelle mit den zugehörigen Datentypen parametriert werden. In diesem Fall handelt es sich um den Tipp-Betrieb, welcher einen Teil des Handbetriebs darstellt. Somit werden die Schnittstellen *Cmd* und *State* mit den Handbetriebssignalen des Signal-Datenbausteines versorgt. In dieser Anwendung wird noch zusätzlich die Schnittstelle *HwState* verwendet, über die die Visualisierung der Antriebszustände realisiert wird.

Mit den übergebenen Signalen des Signal-DB werden die in Unterkapitel 5.3 festgelegten Betriebsarten durch die Steuerung umgesetzt. Eine genauere Beschreibung wird im nachfolgenden Abschnitt 7.1.3 ausgeführt. Die Visualisierungs-Ebene verfügt zudem über einen direkten Zugriff auf die Hardware-Abstraktion, um Betriebsmittel wie den Fahrbahnklappenmotor oder Ähnliches anzusteuern. Eine ausführliche Behandlung diesbezüglich erfolgt unter Abschnitt 7.1.5.

7.1.3 Bedien-Ebene

Die Bedien-Ebene bildet das Bindeglied zwischen der Visualisierung und der Lift-Steuerungs-Ebene. Alle Signale, welche von der Visualisierung an die Steuerung übergeben werden, werden in dieser Ebene für die Steuerungs-Ebene und allgemeine Programmteile des Liftes bereitgestellt. Ausgenommen hiervon sind die Steuersignale für den Einzelbetrieb, welche direkt in der Hardware-Abstraktion verarbeitet werden.

Der Datenaustausch zwischen der Visualisierungs- und der Bedienebene erfolgt über die Schnittstellen *VisuCmd* und *VisuState*. Die *Cmd*-Schnittstelle wird hierbei in die in der nachfolgenden Tabelle 9 angeführten Strukturen unterteilt.

Bezeichnung	Datentyp	Beschreibung
Main	VisuCmdMain	Diese Struktur dient zur Übergabe der allgemeinen Bediensignale, welche zur Anwahl der Betriebsarten, aber auch für den laufenden Betrieb des Liftes benötigt werden.
Man	VisuCmdMan	Mit dieser Struktur werden die Steuersignale zur Bedienung der Handbetriebsarten übergeben. Diese Struktur wird durch weitere Strukturen, welche für die einzelnen Betriebsarten des Handbetriebes benötigt werden, untergliedert.
Test	VisuCmdTest	Die Struktur Test dient zur Steuerung des Testbetriebes. Wie auch schon der Handbetrieb wird auch die Test-Struktur für die unterschiedlichen Betriebsmodi weiter unterteilt.
Auto	VisuCmdAuto	Mit der Struktur Auto werden die für den Automatikbetrieb benötigten Signale von der Visualisierung an die Steuerung übertragen.

Tabelle 9: Cmd-Struktur der Bedien-Ebene, Quelle: Eigene Darstellung.

Die PLC-Datentypen werden ebenfalls mit der Bibliothek verwaltet und ermöglichen außerdem eine einfache Erweiterung von Signalen. Was wiederum den Vorteil bietet, dass, sollte die Struktur geändert werden, diese nur projektweit aktualisiert werden muss, was bereits durch Freigabe der Änderung erfolgen kann. Mit der Bezeichnung des Datentyps, wie zum Beispiel „VisuCmdMan“ lässt sich daraus schließen, dass dieser Typ die Steuersignale der Visualisierung für den Handbetrieb beinhaltet. Darüber hinaus ermöglichen die Strukturen einen einfachen Austausch der Steuersignale und erleichtern zudem die Projektierung der Visualisierung mittels Bildbausteinen. Daher ist es nicht mehr notwendig, jedes Signal einzeln mit dem Bildbaustein zu verknüpfen, sondern es kann direkt die gesamte Struktur an den Bildbaustein, wie in Abbildung 32 gezeigt, angebunden werden. Das Steuern der einzelnen Signale erfolgt dann durch die definierten Eigenschaften des Bildbausteines. Zusätzlich werden zu den Steuersignalen auch Statussignale von der Steuerung an die Visualisierung über der *State*-Schnittstelle übermittelt. Der Aufbau dieser Schnittstelle gestaltet sich simultan zu der *Cmd*-Schnittstelle.

Wie bereits erwähnt, dient die Bedien-Ebene als Schnittstelle zwischen Visualisierungs- und Steuerungs-Ebene. Das hat zur Folge, dass auch ein Signalaustausch zwischen der Bedien- und Steuerungs-Ebene geschehen muss. Die Kommunikation der Ebenen wird ebenfalls über jeweils eine eigene *Cmd*- und *State*-Schnittstelle, wie in der Abbildung 33 gezeigt, ausgeführt. Die Steuersignale werden durch die Schnittstelle *qCtrlCmd* an die Steuerungsebene übergeben, welche wiederum mittels *inCtrlState* den aktuellen Zustand an die Ebene rückmeldet.

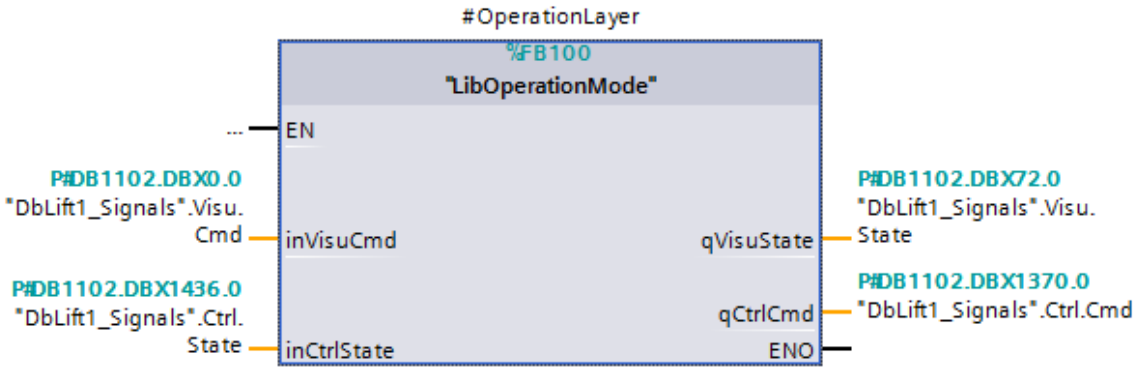


Abbildung 33: Aufruf Bedien-Ebene, Quelle: Eigene Darstellung.

Zur Auswertung, Ausführung und Steuerung von Befehlen und Signalen erfolgt die Programmierung des Bausteines mit der Programmiersprache Funktionsplan. Da in dieser Ebene viele verschiedene einzelne Befehle und auch Statusmeldungen ausgewertet werden, die wiederum neue Befehle generieren, bietet die Programmierung in FUP den Vorteil, dass die Zusammensetzung der einzelnen Signale schnell nachvollzogen und zudem auch ein Debugging des Programmes einfach durchgeführt werden kann. Abbildung 34 zeigt einen einfachen Auszug aus dem Programm der Bedien-Ebene, mit welchem der Tipp-Betrieb aktiviert wird. Daraus ist klar zu erkennen, welche Signale zur Aktivierung der Betriebsart benötigt werden. Im Beobachtungsmodus, welcher hier durch das Anzeigen von durchgehende grünen und strichlierten blauen Linien dargestellt wird, kann sofort erkannt werden, welche Signale noch benötigt werden, um die Aktion zum Setzen der Betriebsart-Aktivierung auszuführen.

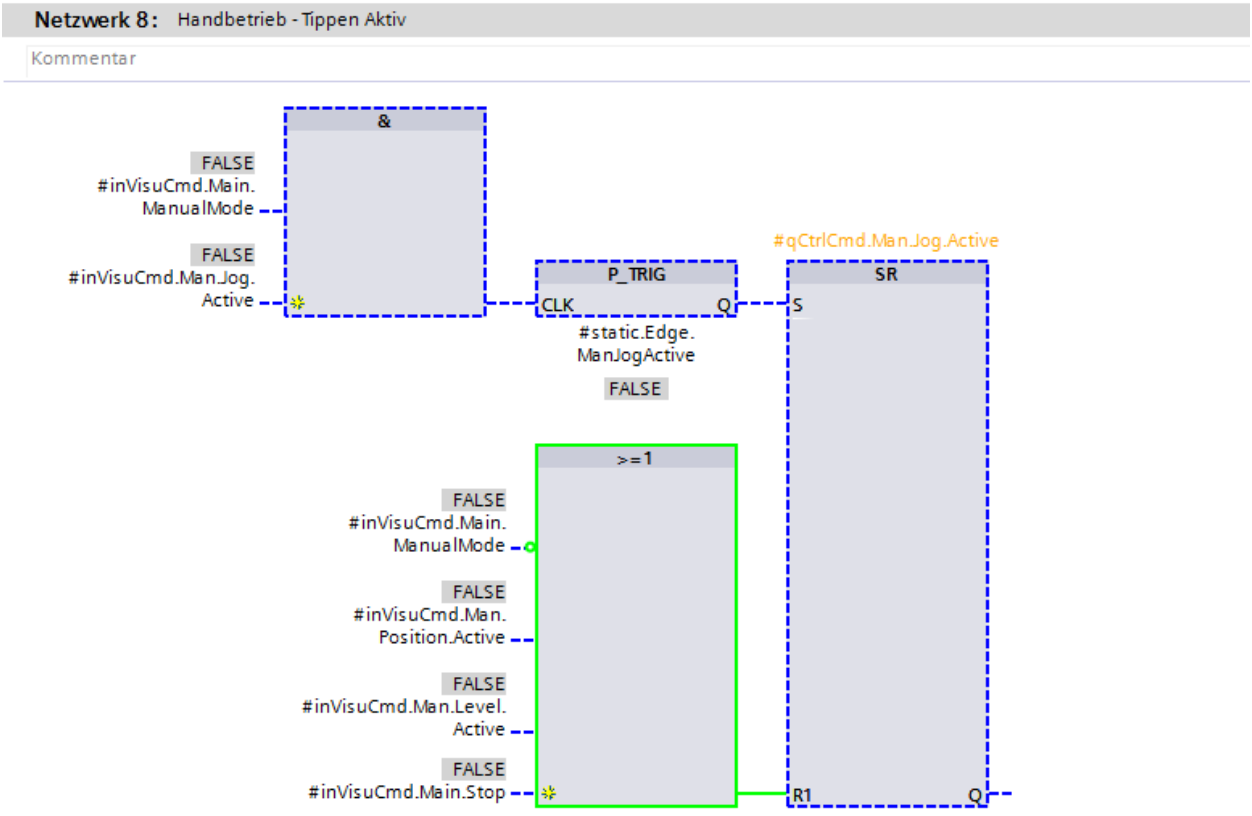


Abbildung 34: Programmauszug Bedien-Ebene, Quelle: Eigene Darstellung.

7.1.4 Steuerungs-Ebene

Die Ausführung der jeweiligen Betriebsarten des Liftes erfolgt bis auf den Einzelbetrieb durch die Steuerungs-Ebene. Wie auch schon in den Ebenen zuvor werden die Signale zur Steuerung, welche durch die Bedien-Ebene bereitgestellt werden, durch die *Cmd*-Schnittstelle und die Status-Signale über die *State*-Schnittstelle abgebildet.

Die Kommando-Schnittstelle des Funktionsbausteines `ControlLayer`, siehe Abbildung 28, aktiviert die jeweilige Betriebsart, die wiederum durch einen eigenen FB abgebildet wird. In dieser Ausführung könnte der ControlLayer-FB als Klasse Steuerung betrachtet werden und die untergeordneten Betriebsarten-FBs als Methoden. Die Programmierung der Betriebsartenbausteine erfolgt in SCL. Die Bausteine werden wie auch die Datentypen durch die Bibliothek verwaltet, wodurch eine Weiterentwicklung der Methoden ohne größere Probleme durchgeführt werden kann. Um einen Überblick des Konzeptes der Programmierung zu erlangen, wird nachfolgend der allgemeine Aufbau des Control-Layer sowie beispielhaft die Methode Tippen und deren Aufruf ausgeführt.

Die Klasse Steuerung bildet die durch die Bedien-Ebene bereitgestellten Betriebsarten ab. Damit das erfolgen kann, ist es notwendig, die verschiedenen Betriebsarten durch mehrere Methoden, welche jeweils zu einer Betriebsart gehören, abzubilden. Hierzu werden in der Klasse die folgend gelisteten Methoden zur Verfügung gestellt.

- *ManJog – Handbetrieb Tippen*

Bildet einen einfachen Tipp-Betrieb des Liftes. Durch das Betätigen der Taste Up oder Down auf der Visualisierung wird eine Auf- bzw. Abwärtsbewegung des Liftes ausgeführt. Zusätzlich werden dabei die Zustände der Klappen überwacht.

- *ManPosition – Handbetrieb Positionieren*

Der Lift kann mittels dieser Methode auf eine gewünschte Position in Millimeter positioniert werden.

- *ManLevel – Handbetrieb Positionieren auf ein Level*

In dieser Methode erfolgt die Positionierung auf ein Level, welches zuvor bereits in Millimeter gespeichert wurde.

- *TestUpDown – Testbetrieb Level Auf und Ab*

Dieser Betriebsmodus dient zum kontinuierlichem Anfahren verschiedener Level-Positionen in ansteigender sowie auch abfallender Reihenfolge.

- *TestRandom – Testbetrieb Positionieren auf Zufalls-Level*

Diese Methode bildet einen Zufallsbetrieb ab. Eine Art Zufallsgenerator wählt bei einer Anforderung ein Level aus, auf das der Lift positioniert wird.

- *Auto – Automatikbetrieb*

Der Automatikbetrieb stellt die Hauptbetriebsart des Systems dar. Dabei werden durch ein übergeordnetes System der Steuerung die Fahrbefehle vorgegeben. Die Steuerung quittiert nach der Ausführung den Befehl des übergeordneten Systems.

Alle Methoden werden durch eigene Funktionsbausteine, die, wie bereits zuvor angemerkt wurde, durch die Bibliothek verwaltet werden, abgebildet. Die Programmierung erfolgt in der Programmiersprache SCL. Die Ansteuerung der Hardwarekomponenten des Liftes findet mittels Schrittkettensignalen statt, welche mit der sogenannten CASE Anwendung gesteuert werden. Bei dieser Anwendung werde in Abhängigkeit eines numerischen Wertes eine oder mehrere Anweisungen abgearbeitet. Der Wert muss durch eine Ganzzahl abgebildet sein, in der Abbildung 35 stellt die Variable `#Static."Int".step` den Schrittzähler dar, welcher während der Abarbeitung mit einer Konstanten verglichen wird, dies wird in der Abbildung in der Code-Zeile 74 durch die Zahl `1` repräsentiert. Besitzen die CASE-Anweisungsvariable und die Konstante den gleichen Wert, so werden die Anweisungen, welche direkt nach der Konstante folgen, ausgeführt.

```
68  //============================================================================
69  CASE #Static."Int".step OF
70  //############################################################################
71
72  //----------------------------------------------------------------------------
73  // Case 1: Ready to select Operation Mode
74      1:
75          IF #inTestDownActive THEN
76              #State.Updown.Message := 'Wait for up or down';
77              #State.Updown.Active := TRUE;
78              IF #inTestUpActive OR #inTestDownActive AND #inSafetyDoorClosed THEN
79                  #Static."Int".step := 2;
80              END_IF;
81          ELSE
82              #State.Updown.Active := FALSE;
83          END_IF;
84
85          IF NOT #inSafetyDoorClosed THEN
86              #State.Updown.Message := 'Safety doors are not closed!';
87          END_IF;
88
89  //----------------------------------------------------------------------------
90  // Case : Get UpDown Level
91      2:
```

Abbildung 35: Programmausschnitt zur CASE-Anwendung, Quelle: Eigene Darstellung.

Die Funktionsbausteine müssen, wie in Abbildung 36 gezeigt, in der Klasse Steuerung instanziiert werden, damit sie auch als Methoden verwendet werden können. Die Abbildung zeigt hierzu die Instanziierung der Bausteine im Steuerungs-FB. Der Instanz-Name bezeichnet die zuvor definierten Betriebsarten. Der Datentyp verweist auf den verwendeten Bibliotheksbaustein.

FbLift1_Control

	Name	Datentyp	Off...	Default..	Erreich...	Sc...	Sicht...	Einste..	Überwach..	Kommentar
	Static				☐	☐	☐	☐		
	ManJog	"LibLiftCtrl_Man_Jog_Flap"	...		☑	☑	☑	☐		Handbetrieb - Tippen
	ManPosition	"LibLiftCtrl_Man_Pos_mm_Flap"	...		☑	☑	☑	☐		Handbetrieb - Positionieren
	ManLevel	"LibLiftCtrl_Man_PosLevel_Flap"	...		☑	☑	☑	☐		Handbetrieb - Positionieren auf Level
	TestUpDown	"LibLiftCtrl_Test_UpDown_Flap"	...		☑	☑	☑	☐		Testbetrieb - Auf und Ab
	TestRandom	"LibLiftCtrl_Test_Random"	...		☑	☑	☑	☐		Testbetrieb - Zufall
	Auto	"LibLiftCtrl_Auto_Flap"	...		☑	☑	☑	☐		Automatikbetrieb

Abbildung 36: Instanziierung der Betriebsarten-Methoden, Quelle: Eigene Darstellung.

Die Instanziierung der Methoden erstellt eine Kopie des Funktionsbausteines in der Klasse Steuerung, mit der die Parametrierung erfolgt. Abbildung 37 zeigt beispielhaft den Methodenaufruf *ManJog* zur Handbetriebsart Tippen.

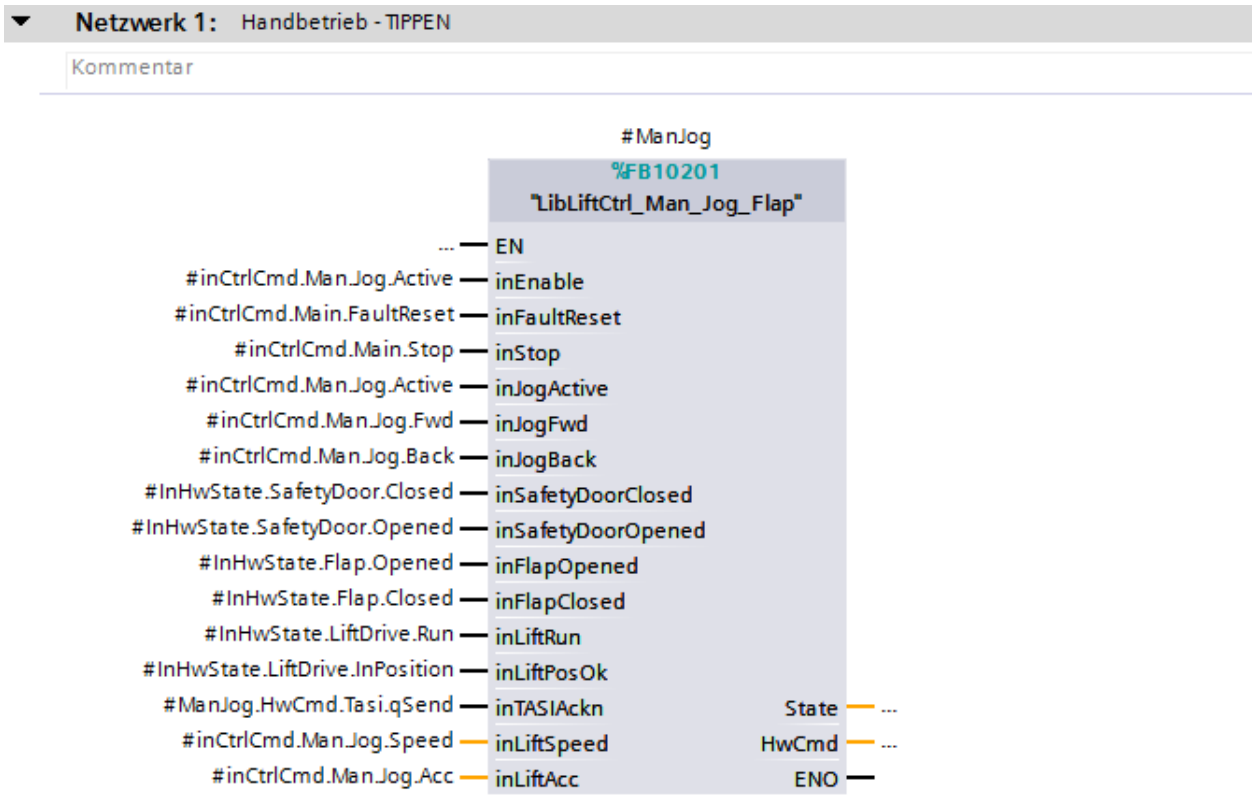


Abbildung 37: Methode Handbetrieb Tippen, Quelle: Eigene Darstellung.

Wie der Abbildung zu entnehmen ist, verfügt der Baustein über eine umfangreiche Anzahl an Eingangsparametern, welche zur Steuerung des Bausteines eingesetzt werden. Dabei werden die Schnittstellensignale, welche mit Cmd-Signalen beschalten sind, als eine Art Nachricht interpretiert, auf welche die Methode (die Instanz des FBs) dementsprechend reagiert. Über Schnittstellen, welche mit State-Signalen versorgt sind, werden Zustandsinformationen an den Funktionsbaustein übermittelt, auf welche der Baustein je nach Zustandsinformation und Nachricht reagiert.

Die Eingänge des Bausteines könnten ebenfalls durch Datentypen abgebildet werden, was aber aufgrund dessen nicht umgesetzt wurde, da nicht alle Methoden dieselben Eingangsparameter benötigen. Die Ausgangsparameter andererseits werden nur von den Schnittstellen *State* und *HwCmd* gebildet, da diese von allen Methoden angewandt werden. Die Ausgangsschnittstelle State bildet den aktuellen Status der gewählten Betriebsart, welche an die übergeordnete Ebene, der Bedien-Ebene, übermittelt wird. Hierzu ist zu beachten, dass alle Bausteine, welche zur Betriebsart Hand (Man) gehören, die gleichen Ausgangsparameter des Objektes Steuerung verwenden. Daher muss eine Unterscheidung erfolgen, welche Betriebsart aktiv ist und somit auch die Schnittstelle beschreibt. Die Auswahl sowie auch die Übergabe der Parameter erfolgen, wie in der folgenden Abbildung 38 veranschaulicht, mittels direktem Zugriff auf die Instanzvariablen. In diesem Fall verstößt dieses Vorgehen nicht gegen die Regelung der Programmierrichtlinien, die einen Schreib- bzw. Lesezugriff von Instanzvariablen untersagt, da der Zugriff auf die Variable innerhalb des Bausteines erfolgt, in dem die Instanzvariable definiert ist. Zudem wird in Abschnitt 4.1.1 angemerkt, dass derartige Zugriffe auf lokale Variablen die Nachvollziehbarkeit des Programmes schmälern. In dieser Anwendung macht ein derartiger Zugriff dennoch Sinn, da die Strukturen *State* und *HwCmd* nicht mittels weitere lokale Variablen zwischengespeichert werden müssen. Dadurch

können lokale Variablen auf einer überschaubaren Anzahl an Einträgen gehalten werden und reduzieren zudem den benötigten Speicherbedarf der Steuerungs-Ebene im Anwenderprogramm. Durch den Zugriff auf die Instanz-Schnittstelle der Methoden ist keine zusätzliche Aktualisierung eines Zwischenspeichers notwendig, sondern mit der Instanz-Aktualisierung wird diese Schnittstelle ebenfalls aktualisiert. Somit wird auch sichergestellt, dass die aktuellen Daten der Methode an die Schnittstelle der Steuerungs-Ebene übergeben werden.

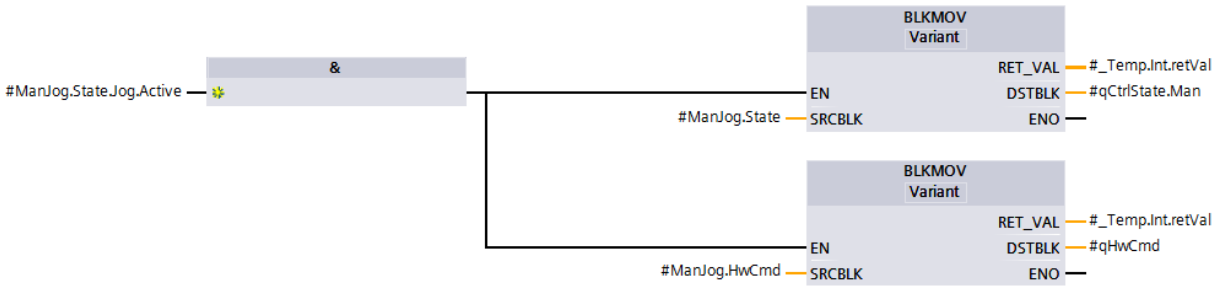


Abbildung 38: Parameter Übergabe, Quelle: Eigene Darstellung.

Aus der Abbildung geht hervor, dass sofern die Methode aktiv ist, die Ausgangsparameter der Instanz auf die Schnittstelle des Objektes geschrieben werden. Der Instanz-Zugriff auf die Ausgangsschnittstelle von *ManJog* erfolgt zum Beispiel mit `ManJog.State`. *ManJog* bezeichnet die Instanz und State spezifiziert die Parameter der Instanz.

Die Ausgangsschnittstelle *HwCmd* der Methode beinhaltet die Befehle zur Steuerung der Betriebsmittel, die in der Hardware-Abstraktion abgebildet werden. Die Übergabe der Signale an die Hardware-Abstraktion erfolgt über die Ausgangsschnittstelle *qHwCmd* des Objektes Steuerung. Da jedoch von allen Methoden auf diese Schnittstelle zugegriffen wird, muss hierbei ebenfalls eine Unterscheidung erfolgen, damit sichergestellt werden kann, dass die gültigen Steuerbefehle an die Hardware-Abstraktion übermittelt werden.

7.1.5 Hardware-Abstraktion

Die unterste Ebene der Liftsteuerung wird durch die Hardware-Abstraktion repräsentiert. Wie bereits die Bezeichnung Hardware-Abstraktion vermuten lässt, dient diese Ebene zur Ansteuerung von Hardwarekomponenten, also zur Steuerung von realen Objekten wie unter anderem der Antriebsmotor des Liftes oder der Fahrbahnklappe, welche auf der Peripherie der Steuerung abgebildet werden. Aus der Abbildung 28 geht wie schon bei den zuvor beschriebenen Ebenen hervor, dass der Signalaustausch zwischen der Hardware-Abstraktion und den übergeordneten Ebenen durch die Schnittstellen *Cmd* und *State* erfolgen. Zu diesen beiden Schnittstellen verfügt diese Schicht über eine zusätzliche Eingangsschnittstelle mit der Bezeichnung *VisuCmd*, mit der eine Steuerung der Betriebsmittel in einem Einzelbetrieb erfolgen kann. Das bedeutet, dass unter anderem die Fahrbahnklappe unabhängig vom aktuellen Zustand des Liftes gesteuert werden kann. Dies erleichtert die Inbetriebnahme sowie auch Wartungs- und Einstellarbeiten an verschiedenen Hardwarekomponenten. Ergänzend werden durch die Eingangsschnittstelle Betriebsmittelparameter zur Verfügung gestellt, die es ermöglichen, verschiedene Einstellungen der Betriebsmittel vorzunehmen.

Wie auch in der Steuerungs-Ebene werden in der Hardware-Abstraktion Bibliotheksbausteine zur Realisierung der Hardwarekomponenten verwendet. Erst durch die Instanziierung des Bausteines kann die Parametrierung des Bausteines erfolgen. Die Abbildung 39 zeigt hierzu den Aufruf der Instanz „Flap“ zur Ansteuerung der Fahrbahnklappe.

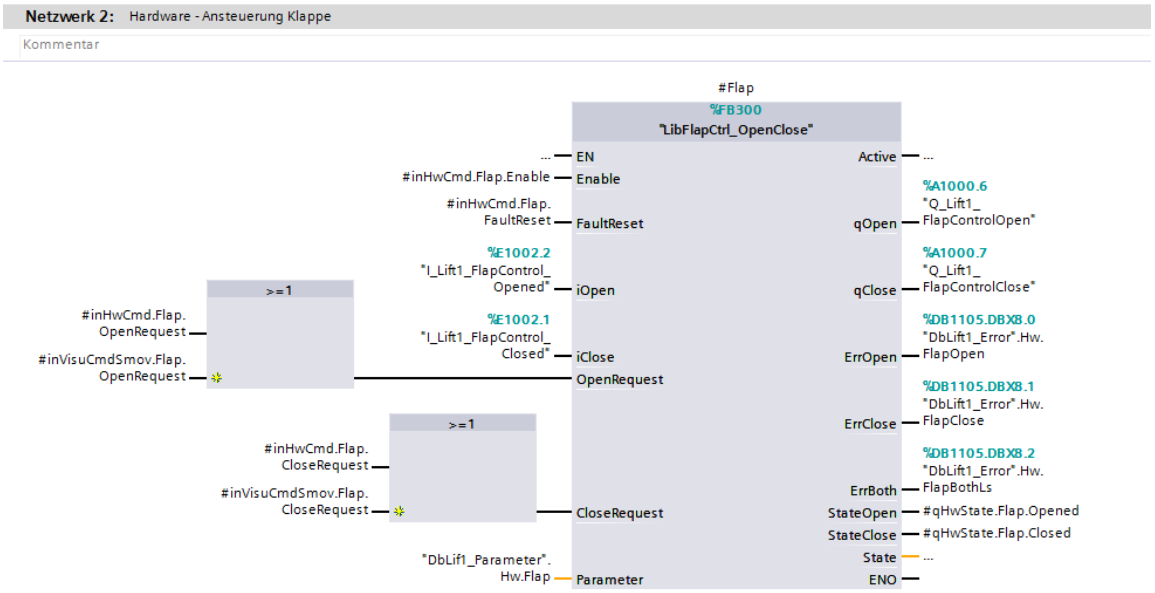


Abbildung 39: Instanziierung der Fahrbahnklappensteuerung in der Hardware-Abstraktion, Quelle: Eigene Darstellung.

Die Steuerung des Betriebsmittels erfolgt auch wie in der Steuerungsebene durch die Zusendung von Befehlen. Hierbei verfügt der Fahrbahnklappenbaustein über zwei mögliche ausführbare Anwendungen, Öffnen und Schließen. Durch die Zusendung einer Öffnen- bzw. Schließanforderung wird die Klappe in Abhängigkeit der Zustandsinformation geöffnet oder geschlossen. Das bedeutet, durch die Befehle und die Zustandsinformationen der Instanz des Fahrbahnklappenbausteines werden die Ausgänge zur Ansteuerung des realen Objektes realisiert. Die Ausführung der Befehle wird überwacht, um ein Fehlermeldung zu generieren, sollte eine Befehlsausführung nicht innerhalb einer definierten Zeit abgeschlossen sein.

7.2 Fehlerauswertung

Der Betrieb des Liftes soll die meiste Zeit im Automatikbetrieb und fehlerfrei erfolgen. Dennoch stellt der Shuttle-Lift ein sehr komplexes System für sich dar. Schon der Lift selbst mit den verbauten Komponenten bietet mehrere mögliche Fehlerquellen. Die Anzahl der Fehlerverursacher nimmt durch den Einsatz der Shuttlesysteme sowie auch dem übergeordneten System weiter zu. Damit nicht undefinierte Stillstände des Systems eintreten, ist es notwendig, potentielle Fehler durch die Programmierung zu erfassen und für den Bediener zu visualisieren.

Aus Gründen der Programmübersicht und Nachvollziehbarkeit werden eigene Datenbausteine erstellt, um die jeweiligen Fehler im SPS-Programm realisieren zu können. Dieser Baustein beinhaltet die gesamten Fehlermeldungen als einzelne Bits des jeweiligen Liftes. Den Fehlermeldungen werden Bezeichnungen zugeordnet, welche einen Rückschluss auf den Fehler erlauben, siehe Abbildung 40.

	Name	Datentyp	Offset	Startwert	Remanenz	Erreichbar a..	Schrei...	Sichtbar i...	Einstellwert	Überwach..	Kommentar
1	Static										
2	Main	Struct	0.0								
3	Visu	Struct	2.0								
4	Opm	Struct	4.0								
5	Ctrl	Struct	6.0								
6	Hw	Struct	8.0								
7	ErrorFlapOpen	Bool	8.0	false							Fehler beim Öffnen der Klappe
8	ErrorFlapClose	Bool	8.1	false							Fehler beim Schließen der Klappe
9	ErrorFlapBothLs	Bool	8.2	false							Fehler beide Endlagen der Klappe aktiv

Abbildung 40: Beispielhafte Darstellung eines Fehlermeldungs-Datenbausteins, Quelle: Eigene Darstellung.

Zusätzlich erfolgt eine Strukturierung der Fehlerbereiche bezogen auf die Programmteile, in welchen die Fehlermeldungen beschrieben werden. Dadurch ist es dem Programmierer möglich, sofort nachzuvollziehen, in welchem Teil des Programmes die Fehlermeldung definiert wurde und er muss sich nicht eventuell umständlich über mehrere Wege zur Meldung durchsuchen.

Die Fehlermeldungen werden in der Visualisierung in Klartext dem Bediener angezeigt. Das bedeutet, die Meldung muss in der Projektierung des Bediengerätes ebenfalls eingetragen werden. Dazu dienen eigene Variablen in der Visualisierung, welche über einen direkten Zugriff auf den zugehörigen Bereich des Fehlermeldungs-Datenbausteines des jeweiligen Liftes verfügen. Zu beachten ist, dass die Längeninformationen der Visualisierungsvariable mit der Länge der Struktur des Datenbausteines übereinstimmen. Zusätzlich muss die Meldung in der Visualisierungsprojektierung in den HMI-Meldungen eingetragen werden. Was so viel bedeutet, wie den gewünschten Meldetext sowie die zugehörige Trigger-Variable zu parametrieren. Der Trigger stellt die Visualisierungsvariable dar, welche den Zugriff auf die Variable des Fehlermeldungs-Datenbausteines ermöglicht. Ergänzend ist die Bit-Nummer der zu lesenden Variable, wie in Abbildung 41 beispielhaft dargestellt, anzugeben.

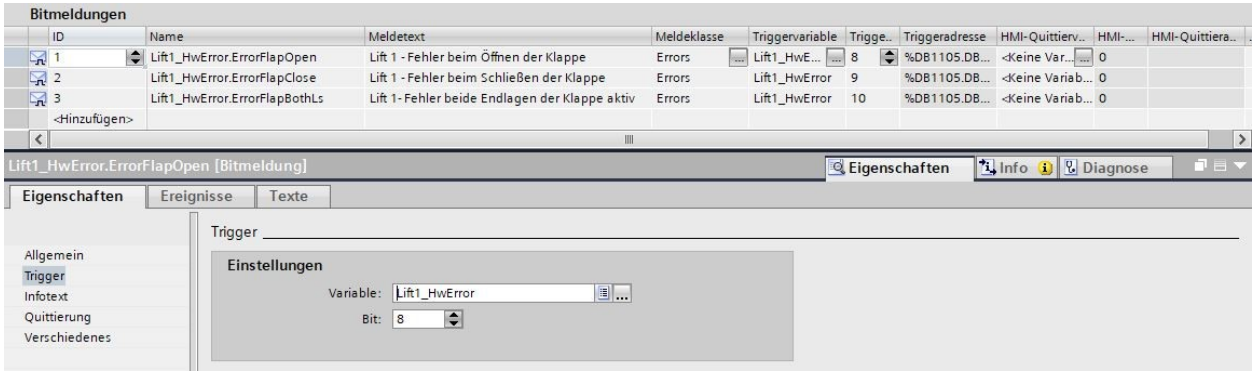


Abbildung 41: Parametrierung einer Bitmeldung, Quelle: Eigene Darstellung.

7.3 Programmaufruf

Der Programmaufbau wird bereits wie in Kapitel 5 beschrieben strukturiert ausgeführt, daher gestaltet sich der Aufruf des Programmes strukturiert. Der grundlegende Baustein für den Aufruf des Anwenderprogrammes wird, wie in Abbildung 42 dargestellt, durch den Organisationsbaustein 1 gebildet. Der OB1 wird in jedem Zyklus der Steuerung abgearbeitet, wird aber keinem festen Zyklus zugeordnet. Das heißt, je nachdem wie hoch die Auslastung der Steuerung ist wird der OB1 in kürzeren oder längeren Zeitabständen ausgeführt. Im Gegensatz dazu werden Programmteile welche in definierten Zeitabständen ausgeführt werden müssen, in eigenen Organisationsbausteinen aufgerufen. Diese bieten die Möglichkeit den Aufrufzyklus in den Bausteineinstellungen zu parametrieren.

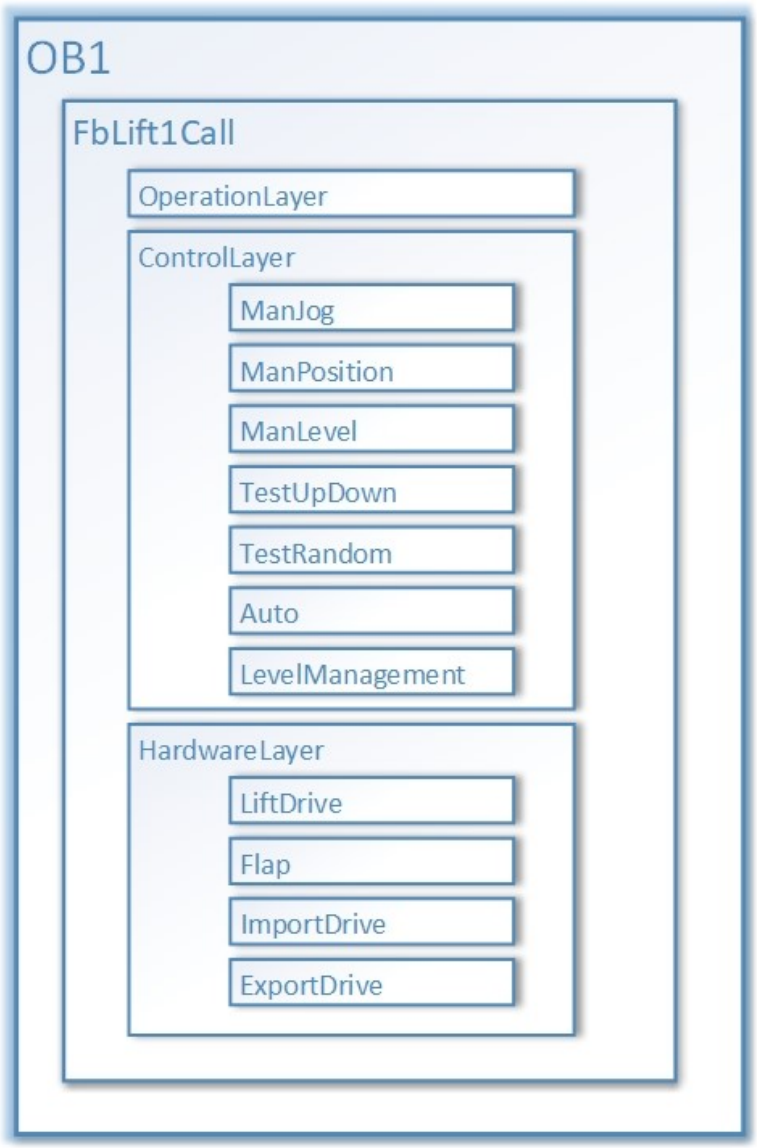


Abbildung 42: Programmaufruf, Quelle: Eigene Darstellung.

Der OB1 stellt den Main Teil der Programmierung dar. Alle Anwenderprogrammteile werden in diesem zusammengefasst und in der programmierten Reihenfolge abgearbeitet. Im strukturierten Programm des OB1 wird der aufrufende Baustein des Liftes 1, der Funktionsbaustein *FbLift1Call*, aufgerufen. Der Baustein verfügt über die in Kapitel 5 festgelegte Struktur. Die Aufrufe der jeweiligen Ebenen erfolgen in diesem Baustein mittels Instanzen. Daraus ergeben sich die Instanz *OperationLayer* zur Bedienung des Liftes sowie dem ControlLayer zur Steuerung des Liftes und dem *HardwareLayer* zur Ansteuerung der Hardwarekomponenten.

Der aus dieser Strukturierung resultierende Vorteil bildet sich darin ab, dass der OB1 nur für den Aufruf der jeweiligen Lift-Aufrufe zuständig ist, die Organisation der weiteren Programmteile erfolgt ausschließlich im aufrufenden Funktionsbaustein des jeweiligen Liftes. Diese Art der Programmierung bietet den weiteren Vorteil des modular aufgebauten Anwenderprogrammes. Alle Programmteile die für den Lift notwendig sind, werden in dem Lift-Aufrufbaustein zusammengefasst und nur dieser muss im OB1 instanziiert und ausgeführt werden. Resultierend daraus gestaltet sich das Programm übersichtlicher und kann einfacher nachvollzogen werden.

8 ERGEBNISSE UND AUSBLICK

Der Automatisierungsgrad nimmt nicht nur im Bereich der Logistik kontinuierlich zu, sondern in allen Bereichen der Industrie, wodurch natürlich auch die Nachfrage an Automatisierungslösungen weiter ansteigt. Damit die Programmierung von automatisierten Systemen effizient erfolgen kann, ist es notwendig, einen Programmierstandard zu schaffen, welcher einfach nachvollziehbar und verständlich aufgebaut ist.

Schon vor einigen Jahren trat die objektorientierte Programmierung ihren Triumphzug an, doch konnte sich diese bis heute noch nicht in der Programmierung von Automatisierungssystemen behaupten. Einige Systeme wie CoDeSys bieten bereits die Option der OOP an, doch der Marktführer Siemens verzichtet derzeit noch auf eine Realisierung der OOP in TIA Portal. Mit Siemens SIMOTION besteht die Möglichkeit einen großen Umfang der OOP zu nutzen, dafür sind jedoch eigene Hardware- und Software-Komponenten erforderlich. Zu dieser Option bestehen weitere Ansätze um Eigenschaften der OOP manuell in SCL zu realisieren, wodurch sich die Programmierung nicht zwingend vereinfacht oder nachvollziehbarer gestaltet. Eine weitere Möglichkeit besteht in der Umsetzung der IEC 61131-3:2003 definierten Wiederverwendbarkeit von Funktionsbausteinen. Diese Variante bietet für die Anwendung eines Shuttle-Liftes derzeit die meisten Vorteile im Bereich der Programmierung des Anwenderprogrammes.

Mit der Wiederverwendbarkeit von Funktionsbausteinen werden bereits gewisse Ansätze der OOP realisiert. Dennoch müssen auf bestimmte Vorteile, vor allem auf die Vererbung, verzichtet werden. Nichtsdestotrotz, bildet diese Programmiermethode einen gravierenden Vorteil, da der Programmaufbau strukturiert, modular und übersichtlich erfolgen kann. Dadurch wird es für den Programmierer einfacher, sich in ein bestehendes Programm einzulesen bzw. dieses zu bearbeiten und zu erweitern.

Durch den strukturierten Programmaufbau und die Einführung der Visualisierungs-, Bedien- und Steuerungs-Ebene sowie der Hardware-Abstraktion konnte eine saubere Auftrennung des Anwenderprogrammes erzielt werden. Da die Kommunikation zwischen den verschiedenen Ebenen über definierte Schnittstellen erfolgt, können die einzelnen Ebenen als gekapselt betrachtet werden. Daraus resultierend ergibt sich der Vorteil, dass sich Änderungen nur auf die jeweilige Ebene beziehen, sofern diese nicht die Schnittstelle betreffen. Somit ist es möglich ein standardisiertes Anwenderprogramm für den Shuttle Lift zu erstellen. Zudem wirken Änderungen der Hardwarekonfiguration nur auf die Hardware-Abstraktion, da die Kommunikation zwischen den Ebenen über definierte Schnittstellen erfolgt, sind derartige Anpassungen ohne Auswirkung auf das restliche Anwenderprogramm.

Aufgrund der definierten Programmierrichtlinien und des standardisierten Anwenderprogrammes, reduziert sich der Projektierungsaufwand, was wiederum dazu führt, dass Projekte effizienter abgewickelt werden können. Ein weiterer angenehmer Nebeneffekt eines standardisierten Anwenderprogrammes wird durch eine verkürzte Einführungsphase von neuen Mitarbeitern erzielt, da diese nicht mehr mit unterschiedlichsten Software-Versionen konfrontiert werden.

Für die Anwendung des Shuttle Liftes ist die Realisierung des Anwenderprogrammes mit der Methode OOP mit Funktionsbausteinen am besten geeignet, da die Umsetzung einfach, nachvollzieh- und erweiterbar umgesetzt werden kann. Leider wurde das System während der Erstellung dieser Arbeit von einem neuen abgelöst und wird in nächster Zeit so nicht umgesetzt werden. Dennoch konnten aus dieser Arbeit neue

Erkenntnisse zur Erstellung von Anwenderprogrammen gewonnen werden. Daraus resultierend soll die definierte Software-Architektur sowie die festgelegten Programmierrichtlinien als Grundlage für neu Anwendungen im Unternehmen Knapp Industry Solutions dienen.

Abschließend soll angeführt werden, dass bereits jetzt die OOP Programmierung einen starken Einfluss auf die Steuerungsprogrammierung nimmt, nicht zuletzt durch die Einführung in die IEC 61131-3, wodurch ein Paradigmenwechsel bevorsteht, welcher sich in den nächsten Jahren stark auf die Programmierung von Steuerungssystemen auswirken wird. Es bleibt jedoch abzuwarten, inwieweit sich die objektorientierte Programmierung im Bereich der Steuerungsprogrammierung als sinnvoll erweisen wird.

LITERATURVERZEICHNIS

Gedruckte Werke (6)

Braun, Michael; Horn, Wolfgang (2015): *Objektorientiertes Programmieren mit SIMOTION*, Publics Pixelpark Erlangen, Erlangen

Division Digital Factory (2017a): *PLC programmieren*, in: Siemens (Hrsg.): *Simatic - STEP 7 Professional V14 SP1*, ohne Verlagsangaben, Nürnberg

Division Digital Factory (2017b): *Bibliotheken verwenden*, in: Siemens (Hrsg.): *Simatic - STEP 7 Professional V14 SP1*, ohne Verlagsangabe, Nürnberg

Doberenz, Walter; Gewinnus, Thomas (2013): *Visual C# 2012 Grundlagen und Profiwissen*, Carl Hanser Verlag, München

Goll, Joachim (2014): *Architektur- und Entwurfsmuster der Softwaretechnik*, 2 Auflage, Springer Vieweg, Esslingen

Hofer, Johannes (2014): *SCL und OOP mit dem TIA Portal*, 2 Auflage, VDE VERLAG GMBH, Berlin

John, Karl; Tiegelkamp, Michael (2008): *SPS-Programmieren mit IEC 61131-3*, 4. Auflage, Springer Verlag Berlin Heidelberg, Berlin

Wellenreuther, Günter; Zastrow, Dieter (2008): *Automatisieren mit SPS – Theorie und Praxis*, 4. Auflage, Vieweg+Teubner GWV Fachverlage GmbH, Wiesbaden

Online-Quellen (6)

Ram, Stefan (2004): *Was ist objektorientierte Programmierung?* http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/begriff_objektorientierte_programmierung_de [Stand: 4.12.2017]

Siemens (2013): *Welche Organisationsbausteine können Sie in STEP 7 (TIA Portal) verwenden?* https://support.industry.siemens.com/cs/document/40654862/welche-organisationsbausteine-k%C3%B6nnen-sie-in-step-7-(tia-portal)-verwenden-?dti=0&lc=de-WW [Stand: 4.12.2017]

Siemens (2016): *Programmierstyleguide für S7-1200/1500* https://support.industry.siemens.com/cs/document/109478084/programmierstyleguide-f%C3%BCr-s7-1200-1500?dti=0&lc=de-WW [Stand: 4.12.2017]

Siemens (2017c): *Industrie Software SIMATIC Safety - Projektieren und Programmieren* https://cache.industry.siemens.com/dl/files/126/54110126/att_915536/v1/ProgFAILdeDE_de-DE.pdf [Stand: 4.12.2017]

Wagner, Herfried (2007): *Zur Sinnhaftigkeit der Camel Case-Konvention* https://dotnet.currifex.org/dotnet/articles/camelcase/ [Stand: 4.12.2017]

www.dasWirtschaftslexikon.com (2016): *Logistik* http://www.daswirtschaftslexikon.com/d/logistik/logistik.htm#hwprod-l0221L15 [Stand: 4.12.2017]

Normen (1)

Beuth (Hrsg.) (2014): *EN 61131-3:2013: Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen (IEC 61131-3:2013)*

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

AWL	Anweisungsliste
Cmd	Kommando
CPU	Central Processing Unit
DB	Datenbaustein
F-Ablaufgruppen	Fehlersichere Ablaufgruppen
FB	Funktionsbaustein
FC	Funktion
F-CPU	Fehlersichere Steuerung
F-Peripherie	Fehlersichere Peripherie
FUP	Funktionsplan
HMI	Human Maschine Interface
HW-Objekt	Hardwareobjekt
Instanz-DB	Instanz-Datenbaustein
KIN	KNAPP Industry Solutions
KOP	Kontaktplan
MFR	Materialflussrechner
OB	Organisationsbaustein
OB1	Organisationsbaustein 1
OOFB	Objektorientierter Funktionsbaustein
OOP	Objektorientierte Programmierung
OOSP	Objektorientierte SPS-Programmierung
POE	Programmorganisationseinheit
SCL	Structured Control Language
SPS	Speicher-Programmierbare Steuerung
State	Status