

Model - und listenbasierte Software- entwicklung in der Industrieautomation

Bakkalaureatsarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Marco Handl

Matrikelnummer 1025510

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Kolleg. Dipl.-Ing., Bakk.techn. Andreas Fernbach

Wien, 04.07.2015


(Unterschrift Verfasser/in)

(Unterschrift Betreuer/in)

1 Zusammenfassung

Industrieautomation beschäftigt sich mit der Steuerung und Regelung industrieller Prozesse. Je nach Ebene der Automationspyramide [1] werden unterschiedliche Systeme verwendet. In der Steuerungsebene werden hierzu speicherprogrammierbare Steuerungen (SPS) eingesetzt. Bezogen auf die Anlagengröße kann sich in diesen die Länge des Quellcodes von wenigen Zeilen bis zu weit über 100.000 Zeilen ausdehnen. Diese Art von Software ist sehr eng mit elektrischen Signalen verflochten und basiert nicht auf objektorientierten Konzepten, was den Komplexitätsgrad weiter erhöht. Um dieser Komplexität entgegenzutreten und um die Strukturierung zu verbessern, werden Anlagen in einzelne Anlagenteile unterteilt. Weiters werden gleiche Funktionalitäten in Unterprogramme zusammengefasst und in verschiedenen Kontexten verwendet. Große Teile dieses Softwareentwicklungsprozesses sind von einer Vielzahl manueller Tätigkeiten geprägt, wovon jede eine hohe Fehleranfälligkeit birgt.

Ziel dieser Arbeit ist es, neue, praktikablere und fehlersichere Arbeitsweisen aufzuzeigen, durch die zusätzlich auch große Teile der Softwaredokumentation automatisch erstellt werden können. Hierzu sollen bewährte SPS-Softwarearchitektur-Konzepte herangezogen und klare Grenzen aufgezeigt werden, was automatisch generiert werden kann und welche Teile herkömmlich zu erstellen sind. Im Zuge der Arbeit soll ein Konzept entworfen werden, welches es erlaubt, gekapselte steuerungstechnische Funktionen für Motoren, Ventile und Messstellen auf einem höheren Abstraktionsniveau zu betrachten und zu verschalten.

Zur Evaluierung wurde eine "Proof-of-Concept"-Anwendung implementiert, welche es ermöglicht, aus einem bereits bestehenden Satz von Funktionsbausteinen Anlagenteile automatisch mit Ein- und Ausgängen zu verschalten und aus diesen entsprechenden SPS-Code zu generieren. Die praktische Umsetzung erfolgt anhand der Steuerungsplattform Siemens Simatic S7. Die zu erstellende Applikation ist in Form eines Eclipse Plugin entwickelt worden.

Aus Gründen der besseren Lesbarkeit wird auf die zusätzliche Formulierung der weiblichen Form verzichtet. Ich möchte deshalb darauf hinweisen, dass die ausschließliche Verwendung der männlichen Form explizit als geschlechtsunabhängig verstanden werden soll.

2 Inhaltsverzeichnis

1	Zusammenfassung.....	2
2	Inhaltsverzeichnis.....	3
3	Einführung.....	5
3.1	Automatisierungspyramide	6
3.2	Speicherprogrammierbare Steuerungen	7
3.2.1	Hardwarekomponenten in der Feldebene.....	7
3.2.2	Zyklen, Signale und Signallängen	9
3.2.3	Programmaufbau	10
3.2.4	Programmiersprachen	11
4	Umsetzung von Automatisierungsprojekten	12
4.1	Systeme zur Softwareentwicklung	12
4.2	Softwareerstellung	13
4.2.1	Erstellung eines Softwarekonzepts	13
4.2.2	Detail-Engineering, Implementierung, Vorabtest.....	13
4.3	Paradigmen.....	14
4.3.1	Wiederverwendbarkeit	14
4.3.2	Datenkapselung.....	14
4.3.3	Funktionskapselung.....	14
4.4	Systemeigenschaften.....	15
4.4.1	Betriebsarten.....	15
4.4.2	Aggregatsbausteine.....	16
4.4.3	Softwarearchitektur in SPS-Systemen	17
4.5	Anlagenkennzeichnungssysteme	19
4.6	Leittechnisches Mengengerüst	19
4.7	Aggregatsliste	20
4.8	Traceability	20
4.9	Kostenfaktoren	22
5	Listenbasierter Ansatz	23
5.1	Ziele	23
5.2	Generierung des Codes und Import in Entwicklungsumgebung.....	23
5.3	Qualitativer und quantitativer Nutzen	26
6	Proof of Concept	27
6.1	Eclipse-Plattform.....	27
6.2	Baustenaufbau und Bausteinschnittstellen in S7-Quellen	27

6.2.1	Allgemeiner Teil.....	28
6.2.2	Aufbau bausteinbezogener Variablen.....	28
6.2.3	Typen.....	30
6.2.4	Datenbausteine	30
6.2.5	Organisationsbausteine	31
6.2.6	Funktionen	32
6.2.7	Funktionsbausteine	33
6.3	Softwarebeschreibung.....	34
6.3.1	Erstellen eines Projektes und der Projektstruktur	36
6.3.2	Einlesen der Bausteine	36
6.3.3	Verschalten der Anlagen	39
6.3.4	Codegenerierung.....	40
7	Zusammenfassung und Ausblick	42
8	Literaturverzeichnis.....	43
9	Abbildungsverzeichnis.....	45

3 Einführung

In der Automation werden Arbeitsabläufe voll- bzw. semiautomatisch durchgeführt. Hierbei kommen reaktive Systeme zum Einsatz. Von Sensoren werden diverse Eingangssignale und Messwerte des physikalischen Prozesses ermittelt. Diese werden in entsprechender Software ausgewertet und in Algorithmen verarbeitet, welche dann in Form von Ausgangssignalen eine entsprechende physikalische Reaktion einleiten. Dieser Kreislauf ist in Abbildung 4.1 dargestellt. Je nachdem wieviel Einfluss menschliche Bediener in den Prozess haben, spricht man vom Grad der Automation. Die DIN IEC 60050-351 [2] definiert den Begriff „automatisch“ folgendermaßen: „ein Prozess oder eine Einrichtung bezeichnend, der oder die unter festgelegten Bedingungen ohne menschliches Eingreifen abläuft oder arbeitet.“

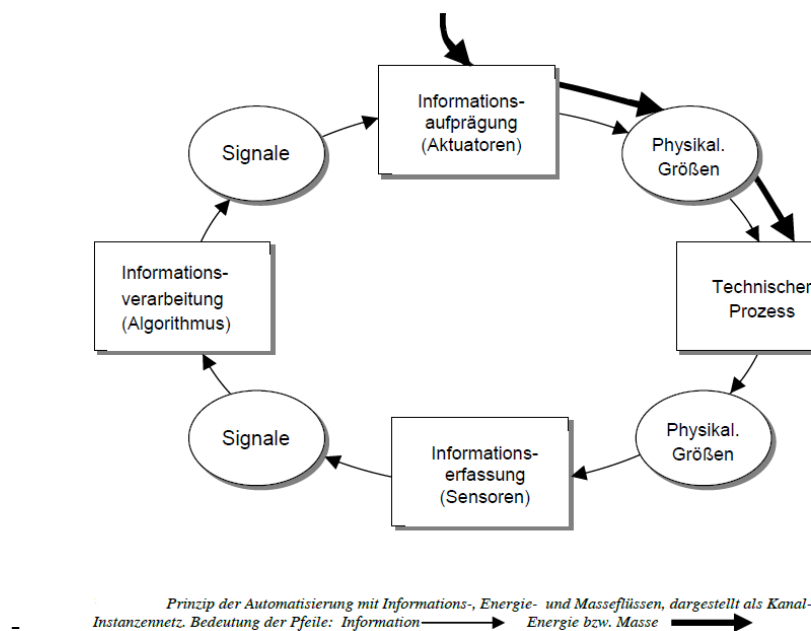


Abbildung 3.1 Reaktive Systeme in automatisierten Prozessen [3]

Automatisierungstechnik umgibt uns in vielen Bereichen des Alltags, ob zuhause oder in den Gebäuden, in denen wir arbeiten (Home and Buildingautomation), in Personenbeförderungsmitteln wie Straßenbahnen, U-Bahnen und PKWs (Automotive), in Verkehrsleitsystemen (Verkehrs- und Tunnelautomation) oder hinter den Produkten, die wir täglich verwenden.

Industrieautomation ist ein Überbegriff, welcher sich mit der Automation für produzierende Gewerbe beschäftigt. Wobei produzierende Gewerbe nach [4] wie folgt definiert sind.

Die amtliche Statistik grenzt die Wirtschaftsbereiche in Bergbau, verarbeitendes Gewerbe, Energie- und Wasserversorgung, Baugewerbe sowie die Betriebe des produzierenden Handwerks ab. Das produzierende Gewerbe kann gleichbedeutend mit der Industrie bzw. dem industriellen Sektor gesehen werden.

Industrielle Prozesse heben sich mitunter stark von anderen Sektoren in Form ihrer Aktuatoren ab. Physikalische Kräfte beziehungsweise Bewegungen werden durch ein breites Spektrum von Antrieben (Gleichstrom-, Asynchronmotoren, Servoantriebe, Schrittantriebe) sowie pneumatischen und hydraulischen Aggregaten umgesetzt. In der Schwerindustrie kommt noch eine große Bandbreite der Befuerungstechnologie hinzu.

3.1 Automatisierungspyramide

Die Automatisierungspyramide wie in Abbildung 3.2 dargestellt dient der Einordnung von Techniken und Systemen in der Automatisierungstechnik.

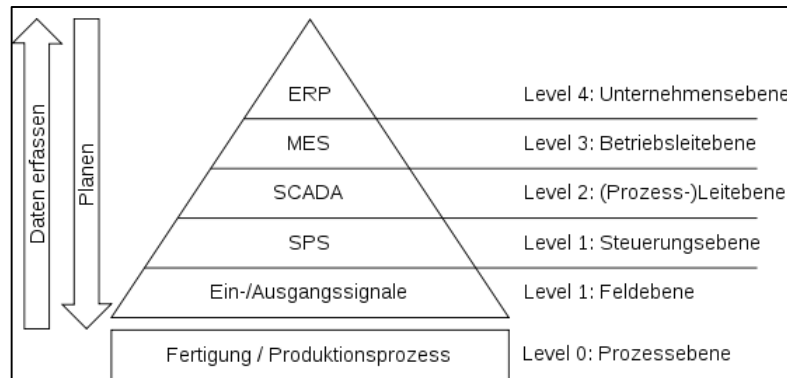


Abbildung 3.2 Automatisierungspyramide

Die Feldebene ist die Schnittstelle zwischen dem Automatisierungssystem und dem Produktionsprozess. Über die Elemente der Feldebene wirkt das Automatisierungssystem über Aktoren auf den Produktionsprozess ein. In der Feldebene angeordnete Sensoren erfassen die erforderlichen Messgrößen für die Abarbeitung der Regelungs- und Steuerungsaufgaben und geben diese an die Steuerungsebene zurück. [5]

Die Steuerungsebene (SPS) bildet die Schaltzentrale von Automatisierungslösungen. Sie übernimmt die Regelungs- und Steuerungsaufgaben, welche die gesamte Maschine oder Anlage betreffen. Sie stellt sicher, dass die eigentliche Aufgabe einer Maschine oder Anlage erfüllt wird. [5]

Oberhalb der Steuerungsebene befindet sich die Leitebene oder Prozessleitebene (Supervisory Control and Data Acquisition, SCADA). In dieser Ebene werden Informationen über den Prozesszustand von einem oder mehreren Steuerungssystemen zusammengetragen und als einheitlicher Prozess betrachtet. Diese Ebene beschäftigt sich mit der Steuerung und Darstellung des Gesamtprozesses.

Die Betriebsleitebene (Manufacturing Execution System, MES) betrachtet den Prozess noch abstrakter. Dadurch können sehr breite Prozessparameter betrachtet werden, wie Material- oder Qualitätsmanagement oder der Key Performance Indicator (KPI) [6]. Dieser bezeichnet in der Betriebswirtschaftslehre eine Kennzahl, anhand deren der Fortschritt oder der Erfüllungsgrad hinsichtlich wichtiger Zielsetzungen oder kritischer Erfolgsfaktoren innerhalb einer Organisation gemessen und/oder ermittelt werden kann.

An der Spitze der Pyramide befindet sich die Unternehmensebene (Enterprise Resource Planning, ERP). Hier kommen Systeme wie SAP zum Einsatz, welche unter anderem die Bestell- und Versandabwicklung durchführen und Informationen von untergeordneten Systemen beziehen bzw. weiterreichen.

Im Rahmen dieser Arbeit werde ich mich auf die Steuerungs- und Feldebene konzentrieren, wobei die hier vorgestellten Konzepte ebenso auf Teile der Prozessleitebene (Bedienebene) anwendbar sind.

3.2 Speicherprogrammierbare Steuerungen

Die speicherprogrammierbare Steuerung (kurz SPS oder umgangssprachlich Steuerung) stellt die zentrale Recheneinheit auf der Steuerungsebene dar. Diese Steuerungen kennzeichnen sich besonders durch ihre Robustheit gegen mechanische und elektromagnetische Belastungen. Dadurch stellen sie die Grundlage für einen 24/7 Betrieb dar. Je nach Projektgröße und Anforderung werden unterschiedliche SPS-Typen eingesetzt, oder es wird ein Verband von mehreren Steuerungen verwendet, wobei jede in sich klare Aufgabenbereiche aufweist und nur über schmale Schnittstellen mit den anderen kommuniziert. Wie man aus der Automationspyramide ablesen kann, stellen diese Steuerungen das Bindeglied zwischen den elektrischen Signalen und den IT-Prozessen dar. Das SPS-Programm dient in erster Linie dem Steuern und Regeln des physikalischen Prozesses. Durch die Leit- bzw. Bedienebene können übergeordnete Prozesse oder Bediener in den Produktionsprozess eingreifen bzw. diesen steuern. Dazu werden den einzelnen, untergelagerten speicherprogrammierbaren Steuerungen neue Sollwertparameter übertragen, um zum Beispiel einen Chargenwechsel herbeizuführen. Eine andere Herangehensweise ist, dass die Steuerungsprogramme aktiv bei der übergelagerten Ebene nachfragen, wie einzelne Bauteile bearbeitet werden müssen. Die Leitebene liefert danach prompt den entsprechenden Parametersatz. Dieses Verfahren wird oftmals eingesetzt, wenn die Steuerungsebene über ein Typerkennungssystem verfügt und gleichzeitig verschiedene Werkstücktypen über eine Fertigungsstraße laufen. Dies bringt den Vorteil hoher Flexibilität mit sich.

Störungen, die im laufenden Prozess auftreten, werden vom Programm des Steuerungssystem erkannt, entsprechend verarbeitet und an die übergeordnete Leit- und Bedienebene gemeldet bzw. zur Abholung bereitgestellt. Da Störungen innerhalb des Prozesses mit höherer Priorität behandelt werden müssen, sollen diese speziell für den Anlagenbediener gut und leicht verständlich visualisiert werden. Jedoch ist für den Betreiber auch der aktuelle Anlagenzustand (Prozessfortschritt, Aggregatzustände, etc.) von Bedeutung, dieser ist vom Steuerungssystem ebenfalls zu erfassen und auf definierten Schnittstellen bereitzustellen. Welche Werte und Signale wie bereitgestellt werden, ist dem Entwickler überlassen und wird nicht vom Steuerungssystem vorgegeben.

3.2.1 Hardwarekomponenten in der Feldebene

Speicherprogrammierbare Steuerungen kann man grob in Kompakt-Geräte und Modular-Geräte einteilen. Kompaktsteuerungen bestehen aus einer Recheneinheit, einer definierten Anzahl von Ein-/Ausgängen, diversen Schnittstellen und sind nur bedingt bis gar nicht erweiterbar. Ihr Einsatzgebiet ist eher in Klein- bis Kleinstanlagen angesiedelt, oder in Zukaufkomponenten, welche über schmale Schnittstellen in die Gesamtanlage eingebunden werden. Modular aufgebaute Steuerungen bestehen, wie aus dem Namen schlüssig hervorgeht, aus einzelnen Modulen, wodurch sich die Hardware optimal an die Anforderungen anpassen lässt. Hierdurch wird auch eine einfache Erweiterung gewährleistet.

Ein standardmäßiger Aufbau einer modularen SPS besteht aus einer Spannungsversorgung, einem CPU-Modul, einem oder mehreren Modulen für Kommunikationsprozessoren, diversen Ein-/Ausgangsmodule (EA-Module) und Sondermodulen wie Zählerbaugruppen, Reglerbaugruppen oder Positionierbaugruppen. Diese Baugruppen werden mittels eines Bussystems (Rückwandbus) zu einem gemeinsamen Rack verbunden.

Die Kommunikationsprozessoren werden oftmals direkt am CPU-Modul verbaut, da ansonsten der Rückwandbus zum Flaschenhals wird. Um Leitungslängen und Verkabelungsaufwand zu minimieren,

werden die EA-Module gerne im Feld verteilt. Diese sind dann an die CPU mittels Bussystemen angebunden. Man spricht hierbei von dezentraler Peripherie.

Bei den EA-Modulen unterscheidet man Digitalbaugruppen und Analogbaugruppen. Digitale Ein-/Ausgabebaugruppen stellen 8-32 digitale Ein-/Ausgänge pro Modul für das Automatisierungssystem bereit, wobei üblicherweise Anzahlen verwendet werden, die durch acht teilbar sind. Über diese Baugruppen können digitale Sensoren und Aktoren angeschlossen werden. So können an Eingabebaugruppen z.B. Schalter/Taster, 2-Draht-Näherungsschalter (Reed-Kontakt, etc.) oder Schützrückmeldungen angeschlossen werden. Die Ausgabebaugruppen eignen sich z.B. für den Anschluss von Magnetventilen, Schützen, Kleinmotoren, Lampen oder Motorstartern. Als gängige Steuerspannung haben sich 24VDC eingebürgert, obwohl es auch hier ein weites Spektrum von Modulen mit unterschiedlichen Steuerspannungen gibt. Man spricht bei diesen digitalen Signalen von wert- und zeitdiskreten Signalen.

Komplexere Aufgaben erfordern die Verarbeitung von analogen Prozesssignalen. Über die Analogbaugruppen können analoge Aktoren und Sensoren ohne zusätzliche Verstärker an ein Automatisierungssystem angeschlossen werden. Analogsignale wie 0-10V oder 0/4..20mA werden durch das Analogmodul ausgewertet und digitalisiert. Diese physikalischen Größen (Strom, Spannung) werden sensorabhängig (aktiv/passiv) bereitgestellt und ausgewertet, hierbei werden diese je nach Auflösung der Karte in einen bestimmten ganzzahligen Wertebereich umgewandelt. Üblich sind hier 8-15Bit Auflösung, eine 8Bit Auflösung entspricht einem Wertebereich von 0-255, 15Bit entsprechen hingegen einer Wertebereich von 0-32767. Der Einsatz von Analogsignalen wird durch die immer günstiger werdenden Feldbusanbindungen vermehrt in den Hintergrund gedrängt. Mittels Feldbusen können weitere Geräte an das SPS-System angebunden werden. Dadurch werden Frequenzumformer, Stellantriebe, diverse Messtechnik, Ventilinseln und noch vieles mehr direkt vom SPS-Programm steuerbar. Auch der Austausch von großen Datenmengen zwischen den Geräten und der Steuerung wird dadurch ermöglicht, was die Chance eröffnet, Parameterwerte der Feldgeräte transparenter der Leitebene zu übermitteln.

Der Hardwareaufbau muss im Automatisierungssystem softwaremäßig konfiguriert werden. In dieser Konfiguration werden den einzelnen Komponenten (EA-Module, Feldbusgeräte,...) Ein- und Ausgangsadressen (Absolutadressen) zugewiesen. Diese nutzt das Steuerungssystem, um mit der physikalischen Umwelt zu interagieren. In einem weiteren Abstraktionsschritt werden an die Adressen symbolische Name vergeben, diese stehen dann im SPS-Programm als globale Variablen zur Verfügung. Diese Variablen stellen die Schnittstelle zwischen Steuerungsebene und Feldebene dar. Für den weiteren Verlauf möchte ich den Begriff der FeldEAs einführen, hiermit sind Ein-/Ausgänge der Feldebene bzw. der Schnittstelle gemeint.

Eine weitere Gruppe von Steuerungen stellen die sogenannten Soft-SPS dar. Dies sind Industrie-PCs, auf welchen ein Steuerungsprogramm ausgeführt wird und mittels entsprechenden Schnittstellen auf die Feldgeräte zugegriffen wird. Soft-SPS stellen besondere Anforderungen an das Betriebssystem. Hierbei muss ein spezifiziertes Wiederanlaufverhalten und ein Echtzeitverhalten garantiert werden. Viele Hersteller wie Beckhoff¹ setzen sehr erfolgreich auf diese Art der Steuerungen.

¹ www.beckhoff.at

3.2.2 Zyklen, Signale und Signallängen

Steuerungstechnische Programme zeichnen sich durch ihren zyklischen Betrieb aus. Im einfachsten Fall wird ein Hauptprogramm nach der Hochlaufroutine in einer Endlosschleife gestartet. Ein SPS-Zyklus setzt sich jedoch aus mehr als dem reinen Programmzyklus zusammen. Zu Beginn eines jeden Zyklus werden die Eingangssignale von der Feldebene eingelesen und auf das Prozessabbild der Eingänge (PAE) übertragen. Mit dieser Momentaufnahme des Anlagenzustands wird das Programm bearbeitet, welches das Prozessabbild der Ausgänge (PAA) befüllt. Nach dem Beenden des Programms wird das PAA auf die realen Ausgänge (Feldebene) übertragen und ein neuer Zyklus beginnt. Man spricht hierbei von einem SPS-Zyklus. In Abbildung 3.3 wird dieser dargestellt. Dadurch, dass ein Mapping der tatsächlichen Eingänge auf das PAE stattfindet und innerhalb des Programms nur mit dem PAE gearbeitet wird, gewährleistet dies, dass innerhalb eines Programmzyklus in jeder Codezeile in der ein bestimmter Eingang verwendet wird, dieser den gleichen Zustand besitzt, ansonsten würde es zu unvorhersehbaren Effekten im Programm kommen.

Neben dem zyklischen Programm gibt es noch weitere azyklische Programmteile, welche durch Interrupts aktiviert werden. Die Interrupt-Routinen haben unterschiedliche Prioritäten, wodurch beim Auftreten mehrerer Interrupts eine festgelegte Reihenfolge eingehalten wird und klar feststeht, welche Interrupt-Routine von welcher unterbrochen wird. Diese Unterbrechungsroutinen werden von unterschiedlichen Ereignissen gesteuert. Fast alle Systeme bieten diverse Weckalarme, welche einen Echtzeitbetrieb ermöglichen. Weitere Ereignisse sind z.B. der Ausfall von Peripheriekomponenten oder das Auftreten eines Programmfehlers. Welche Interrupt Routinen verfügbar sind, hängt von dem Steuerungssystem ab.

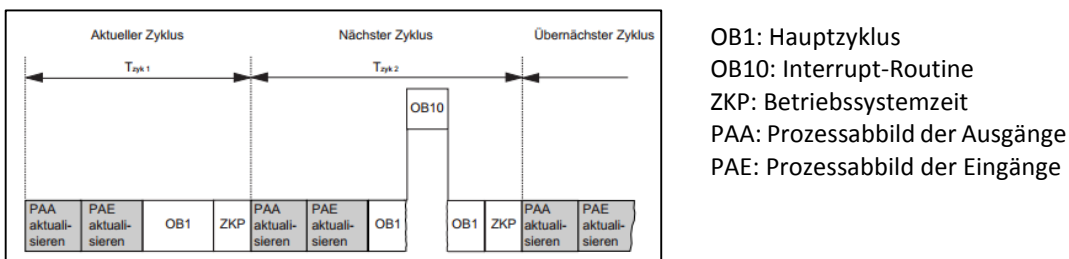


Abbildung 3.3 Zyklusabbild Siemens S7

Dadurch, dass Eingänge nur zu Beginn eines jeden Zyklus gelesen werden, kann es passieren, dass sehr schnelle Signalwechsel von Eingängen nicht erkannt werden. Um nun die Mindestsignallänge eines Signals zu ermitteln, gibt es folgende Formel:

$$t_{min} = t_{cycle} + \sum_{k=0}^n t_{Interrupt} \quad \{n = \text{Anzahl der Interruptroutinen}\}$$

Wobei t_{min} die Mindestzeit, t_{cycle} die Zykluszeit ohne Unterbrechungen und $t_{interrupt}$ die Dauer der jeweiligen Interrupt-Routine darstellt. Falls ein Signal eine Signallänge kleiner t_{min} besitzt, kann man diese Signale in einer schnelleren Taskklasse überwachen. Diese schnellere Taskklasse entspricht wiederum einer Interrupt-Routine, welche öfters aufgerufen wird. Dadurch verringert man jedoch die Geschwindigkeit des Gesamtzyklus. Um dies zu vermeiden, gibt es spezielle Zählerbaugruppen, welche schnelle Signale überwachen. Der Hauptzyklus greift danach auf die Anzahl der Signalwechsel zurück.

3.2.3 Programmaufbau

Damit Programme übersichtlich und wartbar bleiben, muss eine Strukturierung des Codes erfolgen. Hierzu wird dieser in einzelne Programmbausteine gegliedert. Diese Programmbausteine werden laut IEC 61131-3 [7] auch als Programmorganisationseinheit (POE) bezeichnet. In Programmbausteinen sind einzelne Funktionalitäten realisiert, welche am Ende zu einer funktionierenden Gesamtanlage verschalten werden.

Laut [8] gibt es drei Arten von POEs: Funktionen (FUN), Funktionsbausteine (FB) und Programme (PROG), welche in dieser Reihenfolge mit zunehmender Funktionalität versehen sind. FUNs unterscheiden sich von FBs vor allem dadurch, dass sie bei gleichen Eingangswerten immer dasselbe Ergebnis zurückliefern, sie sind somit zustandslos. Dagegen arbeiten FBs jeweils auf einem eigenen Datensatz, sie können sich also Zustandsinformationen merken (Instanzbildung). Programme (PROG) bilden den Kopf eines SPS-Anwenderprogramms und besitzen die Möglichkeit, auf SPS-Peripherie zuzugreifen bzw. diese den übrigen POEs zur Verfügung zu stellen.

Jeder Baustein verfügt über eine definierte Schnittstelle. Diese Schnittstelle gliedert sich in Eingabeparameter (INPUTS), Ausgabeparameter (OUTPUTS) und Ein-/Ausgabeparameter (INOUTS). Funktionsbausteine besitzen zusätzlich noch statische Variablen (STATS). Für den weiteren Verlauf möchte ich hierfür den Namen BausteinEAs einführen, dieser Begriff soll als Überbegriff für inputs, outputs und inouts fungieren. Diese Schnittstelle wird bei Bausteinerstellung vom Programmierer festgelegt und dient dazu, dass der Baustein in beliebigen Systemkontext aufgerufen werden kann.

Um Daten im Programm verwalten zu können, stellen Steuerungssysteme den globalen Merkerbereich zur Verfügung. Weiters können Daten auch strukturierter in globalen Datenbausteinen (DB) abgelegt werden. Um Daten lokal innerhalb eines Bausteins kurzzeitig abzulegen und zu verarbeiten, gibt es den Lokaldatenstack.

Das Betriebssystem des Steuerungssystems, auch Firmware genannt, stellt verschiedene Ereignisse zur Verfügung, in welchen die einzelnen Bausteine aufgerufen werden können. Abbildung 4.4 zeigt diesen hierarchischen Aufbau.

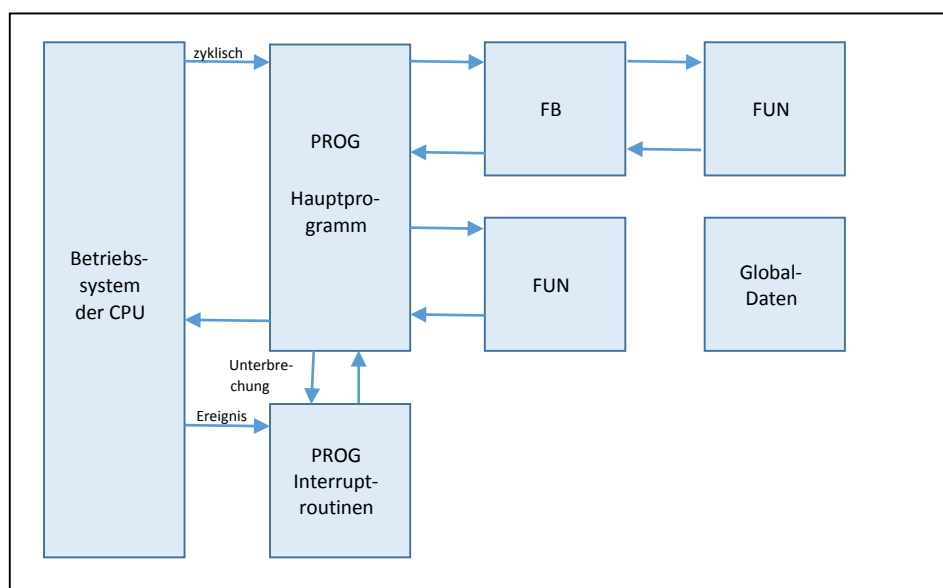


Abbildung 3.4 Aufrufstruktur POEs

3.2.4 Programmiersprachen

Die IEC Norm 61131 ist eine mehrteilige Norm für die Entwicklung und Erstellung von SPS-Systemen. Hier werden Richtlinien für die Erstellung von Hard- und Software festgelegt. Im dritten Teil dieser Norm 61131-3 [7] sind die Richtlinien zur SPS-Programmierung verankert. Da es jedoch schon viele Automatisierungssysteme vor der Zeit der Norm gegeben hat und die Hersteller gewisse Abwärtskompatibilitäten mit ihren Systemen nicht verlieren wollten, ist diese Norm meist nicht vollständig umgesetzt. Bei vielen Systemherstellern kann hierbei die Mnemonik von systemeigenen Sprachbefehlen auf normkonforme Sprachbefehle umgestellt werden und die genormten Funktionalitäten (wie z.B. die Stringfunktionen LEN, LEFT, RIGHT) werden in eigenen Bibliotheken angeboten.

Die IEC 61131-3 sieht als Programmiersprachen sowohl grafische als auch textbasierte Sprachen für die Programmierung von SPS-Systemen vor. Grafische Sprachen bieten den besonderen Vorteil der Einfachheit und Übersichtlichkeit bei der Entwicklung und der Fehleranalyse von logischen Verknüpfungen. Diese werden gerne zur Programmierung von Steuerungsabläufen verwendet und zum Teil auch vom Endkunden gefordert. Bei größeren Berechnungen werden diese Darstellungsarten jedoch sehr unübersichtlich, hier eignen sich hingegen textbasierte Sprachen, welche auch zum Handling größerer Datenmengen herangezogen werden.

Als grafische Sprachen hat die IEC 61131-3 den Kontaktplan (KOP), die Funktionsbausteinsprache (FBS) und die Ablaufsprache (AS) vorgesehen, wobei in der Ablaufsprache auch textuelle Elemente vorkommen. Als rein textbasierte Sprache sind die Anweisungsliste (AWL) und der strukturierte Text (ST) genormt worden.

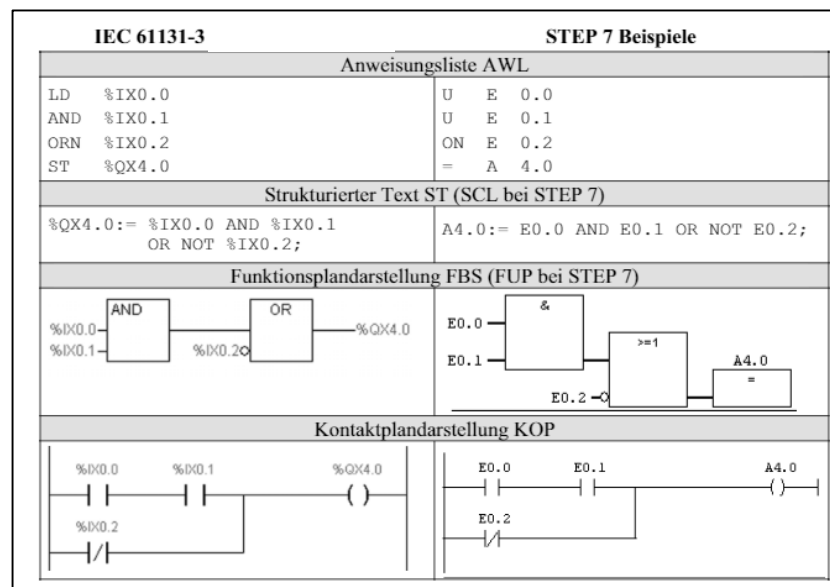
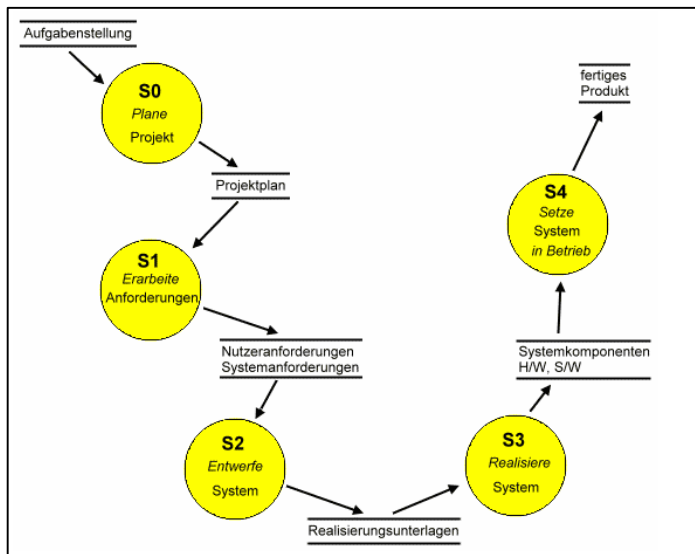


Abbildung 3.5 Übersicht Sprachen aus IEC 61131-3 [9]

4 Umsetzung von Automatisierungsprojekten

Automatisierungsprojekte sind interdisziplinäre Projekte, welche Maschinenbau, Elektrotechnik und Informationstechnologie zusammenführen. Die Kombination aus diesen drei Feldern wird unter dem Begriff Mechatronik zusammengefasst. Jedoch werden im Regelfall die einzelnen Teilgebiete nicht von einer Person durchgeführt, sondern je nach Projektgröße von einzelnen spezialisierten Teams, wobei jedes einen konkreten Output liefert, auf dem das nächste Team aufbaut. All diese Aufgaben müssen im Vorgehensmodell (siehe Abbildung 4.1) berücksichtigt werden.



In der Entwurfsphase arbeiten die drei Disziplinen parallel.

Die Maschinenbauabteilung bzw. der mechanische Konstrukteur liefert eine Konstruktionszeichnung, basierend auf dieser erstellt der elektrotechnische Konstrukteur die elektrische Konstruktionszeichnung. Mit diesen beiden Entwürfen erstellt der Softwaretechniker die Applikationssoftware. Gegen Ende der Entwicklungs- und Realisierungsphase kommt die Inbetriebnahme, wo alle Komponenten zueinander getestet werden.

Abbildung 4.1 Vorgehensmodell Automatisierungstechnik nach ISO9001 [10]

4.1 Systeme zur Softwareentwicklung

Aufgrund der Individualität der Steuerungssysteme und ihrer freien Konfigurierbarkeit bieten Steuerungshersteller eigene Entwicklungsumgebungen (Integrated Development Environment kurz IDE) für ihre Systeme an. Diese sind optimal auf die jeweilige Produktreihe abgestimmt und unterstützen die Entwickler durch eine große Anzahl an Features bei der Konfiguration, Programmierung, sowie beim Testen und letztendlich bei der Inbetriebnahme der Endanwendung. Da oftmals auch Bedien- und Beobachtungsgeräte (BuB) oder SCADA-Systeme mit dieser IDE entwickelt werden, entstehen für den Endanwender große Vorteile, was die Interaktion der einzelnen Teilkomponenten betrifft. So können Variablen in Bediengeräten direkt aus dem Steuerungsprogramm importiert oder es kann direkt darauf zugegriffen werden. Man spricht hierbei von Durchgängigkeit.

Manche IDEs setzten sich am Markt auf breitem Feld durch, sodass Hardwarehersteller ihre Systeme insofern umstellten, damit diese mit den IDEs kompatibel wurden. Einer dieser Quasistandards ist von Siemens Step7. Diese Entwicklungsumgebung wurde geschaffen, um Steuerungen der Simatic-S7 Reihe zu programmieren. Hersteller wie VIPA² oder Saja-Burgess³ haben Controller entwickelt, welche sich ebenfalls mit dieser Software programmieren lassen.

² www.vipa.at

³ www.saia-pcd.com

4.2 Softwareerstellung

4.2.1 Erstellung eines Softwarekonzepts

Wie aus [9] hervorgeht, kann man die Konzepterstellung in folgende Bereiche unterteilen:

- Darstellung der mechanischen und elektrischen Elemente der Anlage in Technologieschemata (abstraktes Schema aller softwarerelevanten Aggregate).
- Zerlegen der Anlage in Funktionseinheiten und Festlegen der zugehörigen Sensoren und Aktoren (Zerlegung der gesamten Domain in Subdomains).
- Für jede einzelne Funktionseinheit einen Ablauf-Funktionsplan (Ablaufkette) entwerfen.
- Den übergeordneten Ablauf-Funktionsplan (Haupt-Ablaufkette) für die Koordination der einzelnen Funktionseinheiten entwerfen.

Je nach Bedarf können die beiden letzten Schritte auch vertauscht oder zeitgleich ausgeführt werden. Der Vorteil dieser Methode ist nicht nur eine übersichtlichere Darstellung des gesamten Steuerungsablaufs, sondern auch der, dass die einzelnen Ablaufketten leichter handhabbar, änderbar und bei gleichen Funktionseinheiten auch kopierbar sind.

Die ersten beiden Punkte fallen in den Bereich des Basic Engineering, Punkt drei und vier befinden sich schon im Bereich des Detail-Engineering, diese müssen nicht unbedingt bei der Softwarekonzepterstellung erzeugt werden.

4.2.2 Detail-Engineering, Implementierung, Vorabtest

Jede Funktionseinheit, welche im Softwarekonzept erstellt wurde, stellt ein Feature dar. Diese sind so einzuteilen, dass sie separat entwickelt werden können und nach Fertigstellung in den Gesamtprozess leicht einzugliedern sind. Je nach Anlagengröße können hier entsprechende Vorgehensmodelle gewählt werden, um einen qualitativ hochwertigen Softwareentwicklungsprozess zu garantieren. Jedoch bieten sich agile Modelle wie SCRUM [11] an.

Wichtig ist, dass jede Funktionseinheit die drei Stadien Detail-Engineering, Implementierung und Vorabtest durchläuft. Der Vorabtest gestaltet sich ohne das Vorhandensein der endgültigen Hardware oftmals als Herausforderung. Hierbei sollte man bei der Entwicklung der einzelnen Komponenten auf Testbarkeit achten. Weitere Informationen zum Thema Testen von Software findet man in [12].

Zentrales Element bei der Entwicklung der Anwendersoftware ist die IDE. Dies kann in folgende Unterpunkte aufgeteilt werden.

- Konfiguration der Hardware
- Import der Ein- und Ausgangssymboliken von E-Konstruktion
- Umsetzung der Softwarearchitektur
- Implementierung der allgemeinen Teile (Uhrzeit, Meldesystem, Taktmerker, etc.)
- Implementierung, Testen der einzelnen Features
- Testen der Gesamtanlage
- Inbetriebnahme

4.3 Paradigmen

Paradigmen spiegeln die grundsätzliche Denkweise zur Erstellung von Software wider. Der Programmierung liegen je nach Design der einzelnen Programmiersprache verschiedene Prinzipien zugrunde. Diese sollen den Entwickler bei der Erstellung von wartbaren, verständlichen und lesbaren Code unterstützen, in manchen Fällen sogar zu einer bestimmten Herangehensweise bei der Lösung von Problemen zwingen. [13]

4.3.1 Wiederverwendbarkeit

Ein besonders erwähnenswerter Aspekt des Programmentwurfs ist die Wiederverwendbarkeit der entwickelten Bausteine. Wenn es gelingt, große Teile neu entwickelter Programme mit bereits vorhandenen und getesteten Bausteinen abdecken zu können, bedeutet dies eine erhebliche Zeiterparnis sowohl bei der Erstellung, beim Testen, als auch bei der Inbetriebnahme der Software, was infolgedessen auch zu einer Kostenersparnis führt. Daher gehört diese Programmierstrategie wohl zu einer der wichtigsten. Bei der Entwicklung neuer Bausteine ist darauf zu achten, dass diese norm- und bibliotheksgerecht nur unter Verwendung lokaler Variablen entwickelt werden, um eine Abgeschlossenheit zu garantieren [9].

4.3.2 Datenkapselung

Datenkapselung ist ein Prinzip, das in der objektorientierten Entwicklung eines der drei Hauptparadigmen darstellt. Unter Datenkapselung versteht man das Verbergen von Informationen hinter einem Satz von Operationen [14]. Jedoch gibt es in den klassischen SPS-Programmiersprachen keine Modifier um die Sichtbarkeit zu regeln, sondern nur lokale oder globale Variablen.

Um diesem Problem entgegenzuwirken, sind entsprechende Coding-Conventions einzuhalten. Objektdaten sollten grundsätzlich im Objekt gespeichert werden. Der Zugriff auf diese Objektdaten sollte nur über klar definierte Schnittstellen erfolgen (INPUTS, OUTPUTS, INOUTS, oder bestimmte Teile der statischen Parameter). Das Schreiben von vielen Stellen des Programms auf statische Parameter (stats) gilt hierbei als schlechter Stil, da es die Komplexität erhöht. Sind Daten von extern zu manipulieren, sollte dies an zentraler Stelle geschehen und die Schnittstelle innerhalb des statischen Bereichs klar abgegrenzt werden. Diese Abgrenzung kann mit Hilfe von strukturierten Datentypen (Stucts) klar gekennzeichnet werden.

4.3.3 Funktionskapselung

Wie oben erwähnt, ist Wiederverwendbarkeit eines der großen Ziele bei der Softwareentwicklung. Hierbei werden einzelne, häufig auftretende Funktionalitäten in Bausteine gekapselt. Man spricht dabei von Funktionskapselung. Die Herausforderung, die es hier zu bewältigen gilt ist, dass Bausteine einerseits sehr generisch zu erstellen sind, um sie vielfältig einsetzen zu können und andererseits projektbezogen, um nicht zu viel Aufwand betreiben zu müssen, um zusätzliche Signale zu verarbeiten.

Eine bewährte Methode ist es, diese Bausteine ebenfalls aus einzelnen Modulen aufzubauen (Abbildung 4.2), die je nach Anwendung miteinander verschalten werden können. So kann ein Motorbaustein zum Beispiel aus den Modulen Grundverriegelungen, Handbetrieb, Automatikbetrieb, Meldebaustein, Betriebsstundenzähler aufgebaut werden. All diese Einzelmodule können in einen großen Rahmen gepackt werden, welcher einen bestimmten Namen trägt und nach außen hin eine Schnittstelle bereitstellt. Diese ist letztlich vom Applikationshersteller zu versorgen. Dieses Konzept wird in Abbildung 4.2 dargestellt. Hier wird der Baustein Motor aus den Bausteinen „Hand“, „Auto“ und „Startmodul“ zusammengesetzt. Der Applikationshersteller muss sich um den inneren Aufbau des Bausteins nicht kümmern, sondern nur die Schnittstelle (BausteinEAs) des Motorbausteins richtig beschalten.

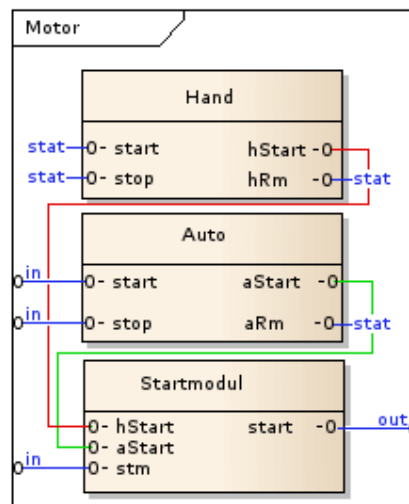


Abbildung 4.2 interne Verschaltung eines Motorbausteins

Für die Entwicklung und das Testen dieser Bausteine eignen sich die Entwicklungsumgebungen von den jeweiligen Herstellern hervorragend. Sind diese Bausteine einmal entwickelt und getestet, können diese in firmeneigene oder projektbezogene Bibliotheken aufgenommen werden und über Jahre von verschiedensten Entwicklern ohne weiteren Testaufwand verwendet werden.

4.4 Systemeigenschaften

Auch wenn man den Sondermaschinenbau immer als Entwicklung von Unikaten betrachtet, lassen sich doch von Maschine zu Maschine Parallelen ziehen und große Gemeinsamkeiten entdecken. Diese werde ich in diesem Kapitel genauer erläutern.

4.4.1 Betriebsarten

Jede automatisierte Anlage sollte sich zu jedem Zeitpunkt in einer bestimmten Betriebsart befinden. Je nach Anforderung unterscheidet man die Betriebsarten:

- Handbetrieb (Einrichtbetrieb, Manuell)
- Teilautomatikbetrieb
- Einzelschrittbetrieb
- Automatikbetrieb

Der Handbetrieb soll es dem Bediener ermöglichen, die Maschine/Anlage einzurichten. Hierbei sollen nur Basisverriegelungen aktiv sein, welche eine Gefährdung von Mensch und Maschine verhindern. Es ist darauf zu achten, dass die Anlage keine selbständigen Aktionen ausführt. Konträr zu dieser

Betriebsart ist der Automatikbetrieb. Hier lösen bestimmte Ereignisse im Umfeld der Maschine bestimmte Reaktionen ohne weiteres Zutun aus. Die beiden anderen Betriebsarten sind Sonderformen des Automatikbetriebs und dienen zum Test oder für Wartungsarbeiten. Beim Einzelschrittbetrieb muss nach jedem Schritt der Schrittkette vom Bediener ein Quittierungsbutton betätigt werden, damit der jeweilige Schritt verlassen wird und der nächste aktiv wird. Hierdurch kann man schnelle Prozessabfolgen langsam und kontrolliert testen. Der Teilautomatikbetrieb ist je nach Anlage sehr unterschiedlich, wobei einzelne Prozessfolgen automatisch durchlaufen werden, jedoch der Bediener zu gewissen Ereignissen in den Prozess eingreifen muss.

Anlagenspezifisch können einzelne Aggregate oder Aggregatsgruppen die Betriebsart wechseln, hierbei spricht man von Betriebsartengruppen. Üblicherweise entspricht eine Betriebsartengruppe einer Subdomain. Die Umschaltung ist an bestimmte Ereignisse geknüpft.

4.4.2 Aggregatsbausteine

Bei der Kapselung von Funktionen setzte sich der Ansatz durch, dass jedes komplexere Aggregat (Motor, Messstellen, etc.) zu einem Aggregatsbaustein zusammengefasst wird. Als komplexes Aggregat versteht man alle Aggregate, welche von mehreren Signalen und Anlagenzuständen abhängig sind. Auch versucht man hier, eine virtuelle Nachbildung der realen Anforderungen zu schaffen. Wenn ein Motor in der Realität existiert, sollte dieser auch virtuell in Form eines Bausteins existieren.

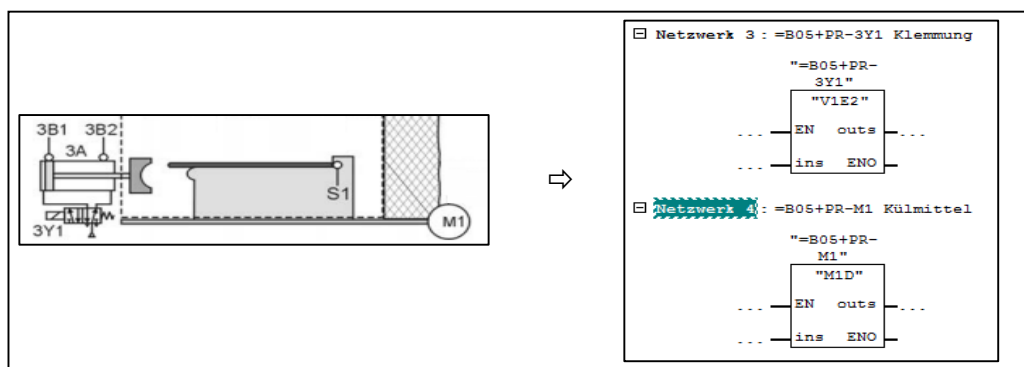


Abbildung 4.3 Überführung realer Objekte in virtuelle

Die schematische Anlagendarstellung in Abbildung 4.3 (linke Seite) lässt erkennen, dass der hier gezeigte Anlagenteil aus einem Ventil (3Y1) mit zwei Endlagen (3B1 und 3B2), einem Motor (M1) und einem Sensor (S1) besteht. Wobei das Ventil und der Motor hierbei komplexere Aggregate darstellen, diese werden softwareseitig in Form eines Aggregatsbausteins repräsentiert (rechte Seite von Abbildung 4.3).

Als Aktoren kennzeichnen sich zwei Superklassen ab, welche sich wieder in Subklassen unterteilen lassen.

- Ventile
 - 3/2 Wege
 - 5/2 Wege
 - Regelventile
 - ...
- Motoren
 - 1 / 2 Drehrichtungen
 - Servo
 - Umrichter
 - ...

Diese beiden Gruppen decken fast die gesamten Aktoren in der Automatisierungswelt ab. Wobei auch hier zwischen den Superklassen große Ähnlichkeiten herrschen, so kann das Handmodul eines Motors mit einer Drehrichtung, welcher die Befehle „START“ und „STOP“ besitzt, ebenso bei einem

3/2 Wegeventil verwendet werden. Es sind lediglich die Bezeichnungen der Befehle auf „Arbeitsstellung“ und „Grundstellung“ zu ändern. Wenn man diese Gedanken weiterführt, kann man hierbei sehr generische Schnittstellen schaffen, bei denen es im Endeffekt keine Rolle spielt, welche Aggregatstypen verschalten werden.

Die Sensorebene unterteilt sich in Messeinrichtungen und digitale Signale. Digitale Signale werden entweder Objekten zugeordnet, dienen als allgemeine domainspezifische Signale, oder fungieren als globale Signale. Im ersten Fall verarbeiten die jeweiligen Aktorbausteine die Signale (siehe Abbildung 4.3 3B1/3B2), in den beiden anderen Fällen muss dies der Entwickler explizit handhaben. Messeinrichtungen werden, wenn der Messwert nicht explizit einem Aktor zugewiesen werden kann (z.B. Stellwert bei Regelklappen, oder Frequenz bei Frequenzumrichtern), durch einen eigenen Baustein repräsentiert. In diesem findet die Umrechnung auf den tatsächlichen Realwert statt, weiters werden oftmals auch bestimmte Grenzwerte definiert, ab welchem Wert Warnungen bzw. Störmeldungen generiert werden sollen. Ein gutes Beispiel ist hier eine Levelmessung in einem Tank, welcher 5000l fasst. Hier kann bei 4000l eine Vollwarnung und bei 4800l eine Vollstörung ausgegeben werden. Für diese Grenzwertevergabe bietet sich der statische Bereich der Bausteinschnittstelle gut an, da diese erfahrungsgemäß von der Bedien- und Leitebene parametrierbar werden.

Zwar gibt es eine große Menge von weiteren Geräten wie Befuerungssysteme, Pressen, Schraubtechnologie, etc., doch lassen sich auch diese auf generische Schnittstellen bringen.

4.4.3 Softwarearchitektur in SPS-Systemen

Die Softwarearchitektur beschreibt den Aufbau einer Software. Je nach Anwendungsgebiet werden entsprechende Architekturen verwendet. Wobei, außer bei sehr kleinen Projekten, immer eine Art der Schichtenarchitektur vorkommt.

Bei der Aufteilung der gesamten Automatisierungsaufgabe in kleinere Teilaufgaben gibt es prinzipiell zwei Wege:

- Eine **technologische Programmgliederung** lehnt sich stark an den Aufbau der zu steuernden Anlage an und unterteilt nach Anlage, Teilanlage und Komponente. Den einzelnen Programnteilen entsprechen einzelne Teile der Anlage oder des zu steuernden Prozesses. Beispiel: Der Anlagenteil „Zuförderband“ kann aus unterschiedlichen Förderelementen, Verschiebewagen und Hubstationen bestehen. Diese Teilanlagen bestehen wiederum aus einzelnen Komponenten wie Motoren, Ventile, Anzeigeelemente.
- Eine **funktionelle Programmgliederung** richtet sich nach der auszuführenden Steuerungsfunktion; die unterlagerten Bausteine enthalten dann das Programm der Teilfunktionen. Beispiel: Die Funktion „Meldungserfassung“ kann aus Meldungsaufbereitung, Meldungsspeicherung und Meldungsausgabe bestehen [15]. Oder die Funktion „Pneumatik“ enthält alle pneumatischen Aktoren der Anlage.

Oftmals werden diese beiden Gliederungsarten auch nebeneinander verwendet. So werden Anlagenteile nach einer technologischen Programmgliederung unterteilt. Innerhalb dieser Anlagenteile werden diese funktionell aufgeteilt.

Abbildung 4.4 zeigt die übliche Softwarearchitektur eines SPS-Programms:

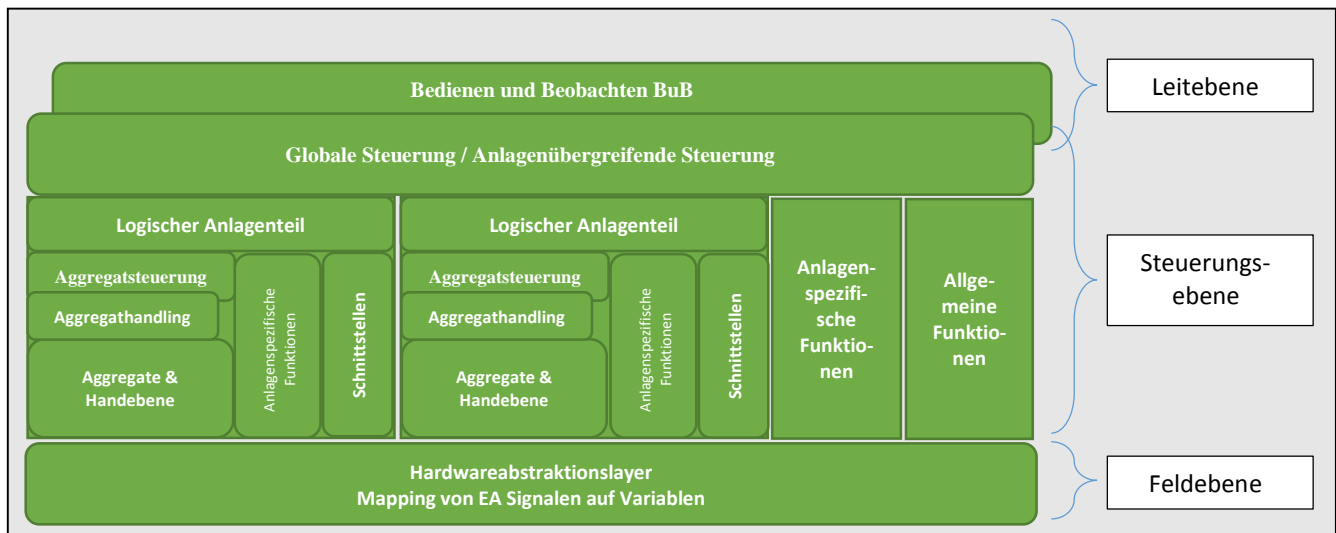


Abbildung 4.4 Technologische Programmgliederung unter Verwendung einer Schichtenarchitektur

Der Hardwareabstraktionslayer stellt eine Abstraktion der absoluten Adressen dar, hier wird jeder Adresse ein symbolischer Name zugewiesen, ein sogenannter Variablenname.

Die gesamte Anlage wird in logische Anlagenteile gegliedert, welche weitgehend abgekapselt sind. Innerhalb dieser Anlagenteile befinden sich wieder weitere Schichten. Der Block Aggregate & Handebene spiegelt den Aufruf der Aggregatsbausteine des jeweiligen Anlagenteils wider. Übergeordnet befindet sich die Aggregatssteuerung, wo meist in Schrittkettenform die automatischen Anlagenreaktionen programmiert sind. Dieser Block dient als übergeordneter Steuerungsapparat aller Aggregate dieser Anlage. Im Aggregathandling befinden sich allgemeine, anlagenglobale Teile, wie die Betriebsarten oder nicht aggregatsbezogene, anlagenspezifische Störmeldeauswertungen. Jede Anlage stellt den anderen Subsystemen diverse Schnittstellen (logische Anlagenteile, globale Steuerung, Bedien- und Beobachtungsebene, ...) zur Verfügung. Dies sind einerseits Anlagenbetriebszustände wie „Betriebsbereit“, „in Betrieb“ oder „Störung“, und andererseits Detailmeldungen, wie „Motorschuttschalten M1 ausgelöst“. Weiters werden auch diverse Handshakesignale (starte/stoppe Anlagenteil) oder die Handbedienung über diese Schnittstellen angeboten. All diese Informationen werden in entsprechend strukturierter Form in Datenbausteinen abgelegt und sind im Architekturdiagramm unter dem Block Schnittstellen zusammengefasst. Anlagenspezifische Funktionen dienen der jeweiligen Anlage als Hilfsfunktionen, um diverse Algorithmen besser kapseln zu können.

Auf selber Ebene wie die Anlagenteile befinden sich die allgemeinen Funktionen. Hier werden, wie der Name schon sagt, allgemeine Funktionen, die keiner Anlage zugewiesen werden können, programmiert (Interruptverhalten, Taktmerker, Uhrzeitsynchronisation, etc.). In den anlagenspezifischen Funktionen findet man hingegen Funktionalitäten, die alle bzw. eine große Schnittmenge der Anlagenteile betreffen.

Die Einheit Globale Steuerung steuert als übergeordnete Einheit die einzelnen Anlagenteile. Sie dient auch als Schnittstelle für die Leitebene. Die Bedien- und Beobachtungsebene ist eher auf derselben Ebene wie die globale Steuerung anzusiedeln, da diese sehr eng mit den entsprechenden Anlagen interagiert.

4.5 Anlagenkennzeichnungssysteme

Datenpunkte wie Eingänge oder Ausgänge werden von elektrotechnischen Konstrukteuren festgelegt und in Schaltplänen dokumentiert. Die Konstrukteure bestimmen die Adresse, den Namen, die Beschreibung und die Funktionalität. Wobei sie bei der Namensgebung speziellen Schlüsseln folgen. Zwei Systeme hierzu sind

- das Betriebsmittelkennzeichnungssystem (BMK) [16] und
- das Kraftwerkskennzeichnungssystem (KKS) [17].

Anhand dieser Schlüssel werden elektrischen Anlagenkomponenten und deren elektrotechnischen Schnittstellen einzigartige Schlüssel zugeteilt. Moderne CAD-Systeme (Computer Aided Design), wie E-Plan P8⁴ ermöglichen es, diese Adresslisten in eine Form zu exportieren, damit diese in SPS-Entwicklungsumgebungen oder in eine Drittsoftware importiert werden können.

=	Anlage	+	Ort	-	Betriebsmittelkennzeichen	:	Anschluss
---	--------	---	-----	---	---------------------------	---	-----------

Abbildung 4.5 Schlüssel Betriebsmittelkennzeichnungssystem (BMK)

Der Betriebsmittelkennzeichnungsschlüssel aus Abbildung 4.5 ist hierarchisch aufgebaut. Hierbei gliedert der Schlüsselteil „Anlage“ die Gesamtanlage in logische Anlagenteile. Anlagennamen sind zueinander projektweit eindeutig. Diese Anlagenteile werden mit der Ortskennzeichnung weiter unterteilt, diese ist anlagenweit eindeutig. Jedes Betriebsmittel ist innerhalb eines Ortes mit einem eindeutigen Betriebsmittelkennzeichen versehen. Diese Betriebsmittel haben diverse Anschlüsse, welche über ein Anschlusssymbol eindeutig zugeordnet werden können. Der so resultierende Schlüssel ist somit projektweit einzigartig.

4.6 Leittechnisches Mengengerüst

Dokumente in Listenform werden im Laufe eines Projekts zu Dutzenden erstellt. Wobei das leittechnische Mengengerüst [18] aus technischer Sicht eines der wichtigsten ist. Bei der Erstellung des leittechnischen Mengengerüsts wird zunächst anhand des Anlagenschemas ermittelt, welche Prozessgrößen (z. B. Druck, Temperatur, Füllstand etc.) wie zu verarbeiten sind (z.B. messen, steuern, regeln, anzeigen, überwachen, stellen usw.). Anschließend werden auf dieser Basis nach Auslegung und Dimensionierung von Mess- bzw. Stelleinrichtungen sowie Prozesssensorik und Bussystemen in Anlehnung an die Struktur von Automatisierungsanlagen die erforderlichen Mengen von Geräten zur

- Informationserfassung (Sensorik, Messeinrichtungen),
- Informationsverarbeitung (Baugruppen speicherprogrammierbarer Steuerungen bzw. Prozessleitsysteme einschließlich Bedien- und Beobachtungseinrichtungen, Kompaktregler, Wandler und Rechenglieder),
- Informationsausgabe (Stelleinrichtungen)

bestimmt.

⁴ www.eplan.at

4.7 Aggregatsliste

Dem leittechnischen Mengengerüst werden im Laufe des Projekts immer weitere Informationen hinzugefügt. Der Softwareentwickler erstellt anhand dieser Liste eine Aggregatsliste und teilt hierbei die einzelnen Aggregate den Anlagenteilen zu. Dabei werden der Aggregatstyp und die erforderlichen FeldEAs vergeben.

Das resultierende Dokument stellt das Fundament der späteren Softwareentwicklung dar, dieses wird ebenso zum Testen der Anlage weiterverwendet und dient dem Endkunden als Dokumentation.

Bezeichnung	BMK	Aggregatstyp	SMOVE_NR	F	VW_Auto	Auto_Start	Auto_Stop	VW_Hand	ES_Wrk	ES_Bas	VW_Sp	WwW	Anl	K_Wrk
Klemmung	=B05+PR-3Y1	V1E2	1	E342.0	E342.1	E342.2	E342.3	E342.4	E342.5	E342.6	E342.7	E343.0	E343.1	A63.3
Presse	=B05+PR-3Y2	VM2E2	2											
Kühlmittel	=B05+PR-M1	M1D	3											

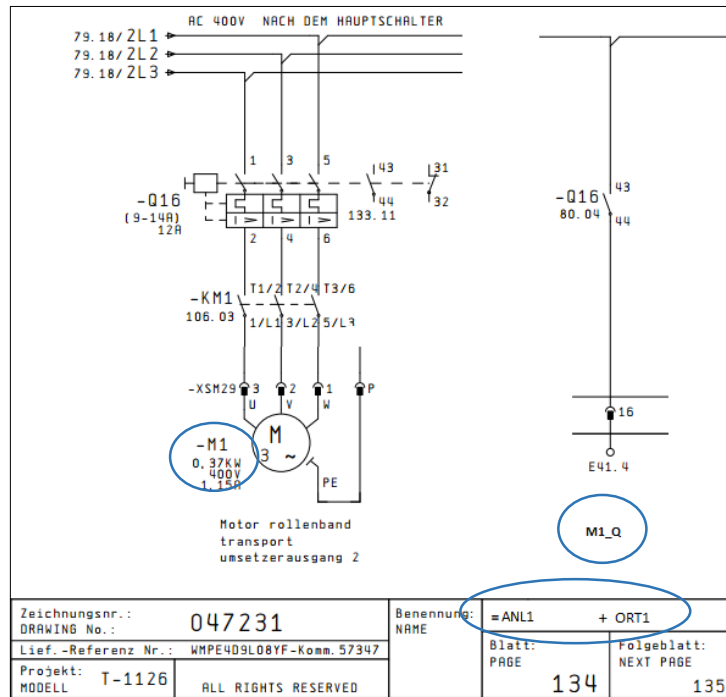
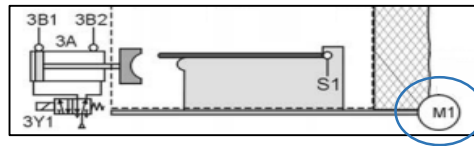
Abbildung 4.6 Aggregatsliste als Basis zur Softwareentwicklung

Sowohl die Anlagenaufteilung als auch die anlagenzugehörigen Aggregate und deren FeldEAs sind hier auf einer allgemein verständlichen Basis dargestellt. Die hier zusammengetragenen Informationen stellen eine enorme Wissensrepräsentation dar. Große Teile der Software lassen sich daraus ableiten. So beginnt der Anwendungssoftwareherstellen damit die Softwarearchitektur anhand dieses Dokuments aufzubauen und die entsprechenden Aggregate je Anlage zu instanziiieren und mit ihren FeldEAs zu verschalten.

4.8 Traceability

Der dokumentenübergreifende Informationserhalt von Konstruktionszeichnung bis zur fertigen Softwarelösung wird als Traceability bzw. Nachverfolgbarkeit bezeichnet. Abbildung 4.7 stellt diesen Informationsverlauf beispielhaft dar. Hierbei wird ein Aggregat aus der mechanischen in die elektrotechnische Konstruktionszeichnung überführt. Von dieser werden alle Informationen in die Aggregatsliste aufgenommen und letztendlich in die SPS Software unter Beibehaltung der Schlüssel überführt.

Diese Nachverfolgbarkeit kann in Automationsprojekten über das Anlagenkennzeichnungssysteme (Kapitel 4.5) realisiert werden. Um diese Nachverfolgbarkeit im SPS-Programm zu realisieren, sind Instanzen von Aggregatsbausteinen mit denselben Schlüsseln zu versorgen wie das reale Aggregat (siehe Abbildung 4.7). Die symbolischen Namen der BausteinEAs sind mit den entsprechenden Anschlussymbolen (:Anschluss) der Schlüssel zu versehen. Der instanziierte Aggregatsbaustein enthält somit die Informationen über Anlage, Ort und Betriebsmittel. Zusammen mit der Symbolik der BausteinEAs spiegelt dies das gesamte Betriebsmittelkennzeichen eines FeldEAs wider. Diese Informationen werden ebenfalls in der Aggregatsliste wiedergegeben.



=Anl1+ORT1.xsec					
General				M1D."M1D"	
S...E	Text	BMK	Type	Q	S_Rep
1	Motor	M1	M1D."M1D"	=Anl1+ORT1_M1_Q	=Anl1+ORT1_M1_S_Rep

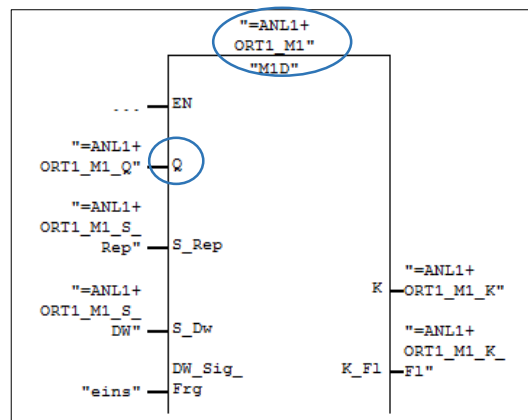


Abbildung 4.7 Traceability

4.9 Kostenfaktoren

Die Zeit der Softwareerstellung ist sicherlich ein maßgeblicher Faktor in der Kostenrechnung eines Projekts. Betrachtet man das Architekturmodell aus Kapitel 4.4.3 so kann man folgende Arbeitspakete schnüren aus welchen sich die benötigten Zeiten ableiten lassen. Bausteinentwicklung t_{dev} und Erstellung der globalen Funktionen t_{global} , diese Zeitwerte sind als konstant anzusehen. Weiters kommen noch die Zeiten zur Erstellung der Softwarearchitektur t_{arch} , die zur Erstellung der anlagen-spezifischen Funktionen t_{secGen} , jene für das Aggregathandling t_{agg} , die des Automatikablaufes $t_{secAuto}$ und die Schnittstellenerstellung t_{interf} hinzu. Diese Faktoren hängen von der Anzahl der logischen Anlagenteile m ab. Einen weiteren wesentlichen Zeitfaktor stellt das Verschalten der Aggregatsbausteine t_{block} dar, da dieser linear zur Anzahl der Aggregatsbausteine n wächst.

Der Gesamtzeitaufwand t_{ges} berechnet sich somit nach der Formel:

$$t_{ges} = t_{block} * n + (t_{arch} + t_{agg} + t_{secGen} + t_{secAuto} + t_{interf}) * m + t_{global} + t_{dev}$$

Da das Verschalten der Aggregatsbausteine eine ausschlaggebende Rolle spielt, betrachten wir dies genauer. Ein durchschnittlicher Aggregatsbaustein besitzt eine Schnittstelle von zehn bis 25 Baustein-EAs und eine statische Schnittstelle für Handebene und Parameter, die es zu verschalten gilt. Hierbei sind folgende Arbeitsschritte zu durchlaufen.

1. Zuteilung des Aggregats zu einer Anlage (1 Min.)
2. Manuelles Ausfüllen der Aggregatsliste (2 Min.)
3. Erstellen eines neuen Netzwerks + Beschriftung mit BMK und Klartext (1 Min.)
4. Baustein aufrufen und korrekte Instanz erstellen (1 Min.)
5. Instanz laut Schlüssel beschriften (1 Min.)
6. Schnittstellenbereiche für Globaldaten erstellen (3 Min.)
7. Schnittstelle zu Automatikenebene erstellen (1 Min.)
8. Bausteinschnittstelle beschalten (4 Min. bei vorliegender ausgearbeiteter Aggregatsliste)

Dabei kommt man auf einen Zeitfaktor von ca. 14 Min./Aggregat (gemessen durch den Autor). Bei manueller Bearbeitung schafft somit ein durchschnittlicher Entwickler ca. 4 Aggregate/h. Größere Anlagen haben nicht selten 100-200 Aggregate pro SPS, dies entspricht somit 3-6 Personentage an Entwicklungskosten.

Ein weiterer Faktor ist die Verfügbarkeit der Anlagen. Betrachtet man Motorproduktionslinien, bei denen alle 20 Sekunden (durchschnittliche Taktzeit) ein fertiges Produkt im Wert von 5.000-20.000€ die Linie verlässt, verursacht ein Linienstillstand aufgrund eines Softwarefehlers enorme Kosten. Speziell da diese Fehler im laufenden Prozess kaum in annehmbarer Zeit gefunden und behoben werden können. Dadurch ist die Fehlertoleranz sehr gering. Längere Maschinenstillstände, für Wartungsarbeiten oder, um neue Maschinen einzupflegen, erzeugen enorme Kosten und müssen daher in sehr kurzer Zeit erledigt. Durch diesen Zeitmangel gilt es besonders in der Softwareerstellungssphase qualitativ hochwertige Software zu erzeugen. Dies gilt für die Fehlerfreiheit, die verwendeten Softwarekonzepte (Architektur, Aufbau), die Wartbarkeit und die Kommentierung.

Speziell das Verschalten der FeldEAs auf die Aggregatsbausteine ist bei manueller Ausführung sehr fehleranfällig, da hier Anlagenteile oftmals kopiert werden und mit „Suchen und Ersetzen“ die entsprechenden Symbole oftmals nicht korrekt angepasst werden oder vieles übersehen wird. Fehler in dieser Ebene treten erst zu Tage, wenn man die tatsächliche Hardware zur Verfügung hat, wodurch oftmals der Fehler bis zur Inbetriebnahme beim Kunden verschleppt wird.

5 Listenbasierter Ansatz

5.1 Ziele

Ziel dieses Ansatzes ist es nun die mühsam zusammengetragenen Informationen, welche sich in der Aggregatsliste widerspiegeln und die Konzepte aus Kapitel 4 zu nutzen, um fehlerfrei arbeitende Automatismen herzustellen, welche einen großen Teil der fertigen Applikationssoftware erstellen. Diese Algorithmen beschalten akkurat getestete Bausteine und bereiten dem Entwickler jene Bereiche vor, welche noch manuell zu bearbeiten sind. Dadurch kann die Fehlerquote maßgeblich gesenkt werden und der Entwicklungs- und Testaufwand verringert werden.

Bei Einhaltung der in Kapitel 4.8 eingeführten Konvention lässt sich leicht ein Automatismus konzipieren, welcher die Zuordnung von FeldEAs zu entsprechenden virtuellen Aggregaten automatisch herstellt. Hierbei muss jedoch die Ein- und Ausgangsliste von der E-Konstruktion im Aggregatslistenfile zur Verarbeitung zur Verfügung stehen. Bei korrekter Vergabe der Symbolnamen durch den E-Konstrukteur nach dem jeweiligen Kennzeichnungssystem ist somit ein automatisches Ausfüllen der Aggregatsliste möglich, wodurch eine weitere Fehlerquelle durch das manuelle Bearbeiten der Liste vermieden wird.

Das Zusammenführen dieser beiden Listen ermöglicht auch einfache Plausibilitätschecks innerhalb der Aggregatsliste. So können Ausgangssymbole oder Rückmeldungen nicht mehreren Aggregaten zugewiesen werden. Auch Auswertungen über nicht verwendete Symbole geben Aufschluss über etwaige Fehler in der Benennung der Symbole oder es werden Aggregate aufgezeigt, welche in der Aggregatsliste vergessen wurden.

5.2 Generierung des Codes und Import in Entwicklungsumgebung

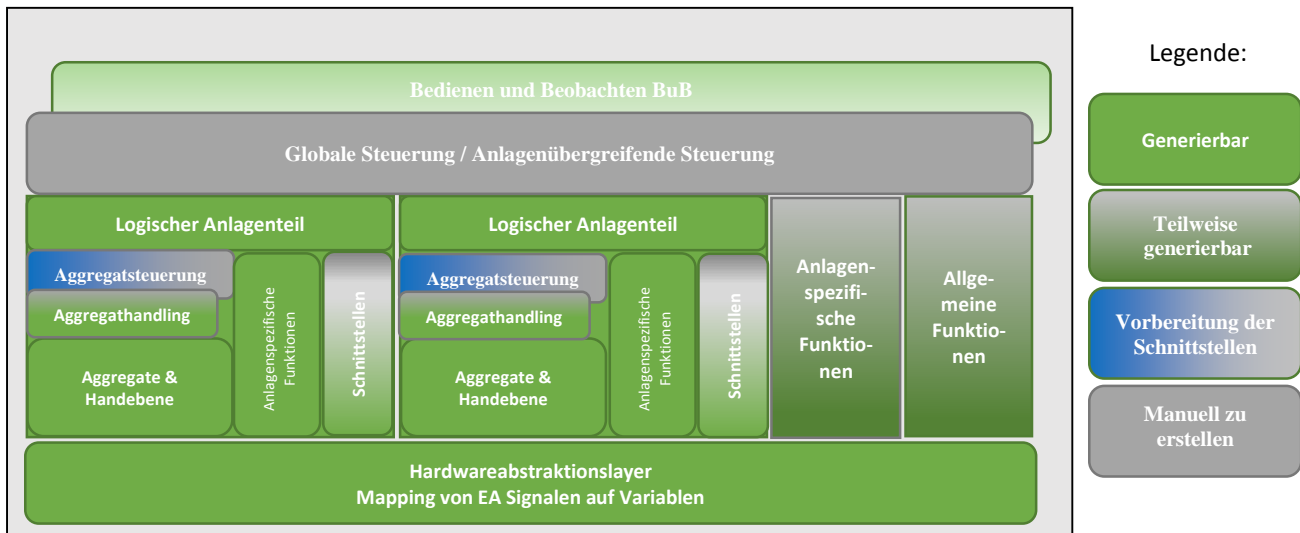


Abbildung 5.1 Generierbare Anlagenteile

Wie oben schon erwähnt, stellt eine fertige Aggregatsliste große Teile der Softwarelösung dar. So befinden sich in ihr alle FeldEAs, die Domänenaufteilung, sämtliche Aggregate und deren Belegung. Es würde sich daher anbieten, diese Informationen in Code zu überführen und somit ein fehlerfreies Grundtemplate zum Projektstart zu erstellen. Auch nur die Erzeugung von einzelnen Anlagenteilen ist hierbei denkbar, somit kann ein Projekt Stück für Stück wachsen, was einem agilen Ansatz zugutekommt.

Wie weit man bei dieser automatischen Erstellung gehen kann, ist von den jeweiligen Firmen abhängig und wie ausgeprägt ihre Standards sind. Ein mögliches Beispiel zeigt Abbildung 5.1.

Betrachtet man eine Architektur wie unter 4.4.3 beschrieben, so sieht man, dass sich rund 60 bis 70 Prozent der endgültigen Software aus der Aggregatsliste vollständig bzw. teilweise ableiten lassen.

Der Hardwareabstraktionslayer befindet sich vollständig in der Aggregatsliste und kann somit leicht in ein Format überführt werden (Abbildung 5.2), welches vom Zielsystem bzw. Zielsoftwaresystem importiert werden kann.

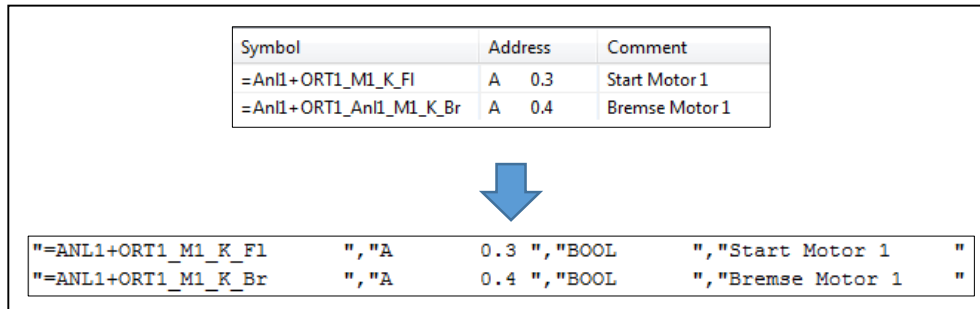


Abbildung 5.2 Überführung der Symbole

Die Aggregate und Handebene sind hierbei schon eine größere Herausforderung. Hier ist die Beschaltung und Instanziierung der Aggregatsbausteine umzusetzen. Die Schwierigkeit ist, dass die Informationen nicht rein von den FeldEAs kommen, sondern sehr viele Signale anlagenspezifische bzw. globale Signale sind, welche erst im Laufe des Projekts entstehen. Hinzu kommt, dass es eine große Menge von Bausteinen gibt, welche Entwickler im Laufe der Zeit erstellen, die es einzupflegen gilt. Wie ich zu einem späteren Zeitpunkt im Proof of Concept zeige, kann man auch hierbei einerseits sehr flexibel bleiben und doch einen hohen Automatisierungsgrad erreichen. Die Handebene ist meist in den Aggregatsbausteinen direkt implementiert und wird über entsprechende Schnittstellen direkt von der BuB-Ebene oder über einen steuerungseitigen Multiplexer angesprochen, wodurch diese hierbei nicht berücksichtigt ist.

Der Block Aggregatssteuerung beinhaltet das Anlagenverhalten und die Programmsteuerung, da diese je nach Anlage verschieden ist, ist es hier schwierig, einen Automatismus zu erstellen. Es gibt hierbei zwar verschiedene Ansätze (man siehe [19]), diese Abläufe von externen Softwaretools generieren zu lassen, jedoch ist es immer mit einer manuellen Tätigkeit verbunden. Was allerdings automatisch vorbereitet werden kann, ist die Schnittstelle zum Aggregatsbereich. Da die Aggregate der untergeordneten Schicht bekannt sind und ebenso bekannt ist, welche Erwartungen sie an die Aggregatssteuerung richten, kann hier die Schnittstelle bereits vollständig vorbereitet werden. Weiters kann hier bereits eine Schnittstellenverschaltung zwischen Aggregatssteuerungsblock und Aggregatsbausteinen vorgenommen werden.

Dadurch, dass die Stör-, Warn-, und Betriebseinzelmeldungen von den Aggregatsrückmeldungen abgeleitet werden können, können im Schnittstellenbereich die entsprechenden Stör-, Warn-, oder Betriebsmeldungsstrukturen automatisch erstellt werden (Abbildung 5.3). Aus diesen strukturierten Einzelmeldungen können in weiterer Folge sehr leicht Summenmeldungen und Anlagenbetriebszustände abgeleitet werden. Auch wenn das Projekt innerhalb der IDE nach der Generierung erweitert wird und Störmeldungen hinzukommen, haben diese Strukturauswertungen weiterhin Gültigkeit, wenn diese Meldungen in die richtigen Bereiche eingegliedert werden.

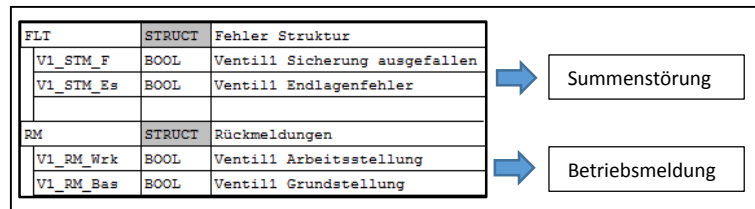


Abbildung 5.3 Automatische Ableitung von Summenmeldungen aus bestehenden Strukturen

Da bei einer Generierung diese Strukturen automatisch erstellt und befüllt werden, können auch Bausteine, die diese Strukturen auswerten und überwachen, automatisch in das Programm eingefügt und verschalten werden. Der Aufruf dieser Bausteine befindet sich in der Architekturübersicht unter Aggregatshandling, die benötigten Auswertungsbausteine in den anlagenspezifischen Funktionen. Im Aggregatshandling werden auch die Betriebsarten verschalten, diese werden hierbei von der physikalischen Ebene abstrahiert. Diese Betriebsartenumschaltung kann nur schwer automatisch erzeugt werden, da hier auf den tatsächlichen Hardwareaufbau eingegangen werden muss. Jedoch kann ebenso eine entsprechende Vorbereitung der Bausteinausgänge gemacht werden und innerhalb des Aggregatshandlings können die jeweiligen lokal-Variablen bereits verschalten werden (siehe hierzu Abbildung 5.4).

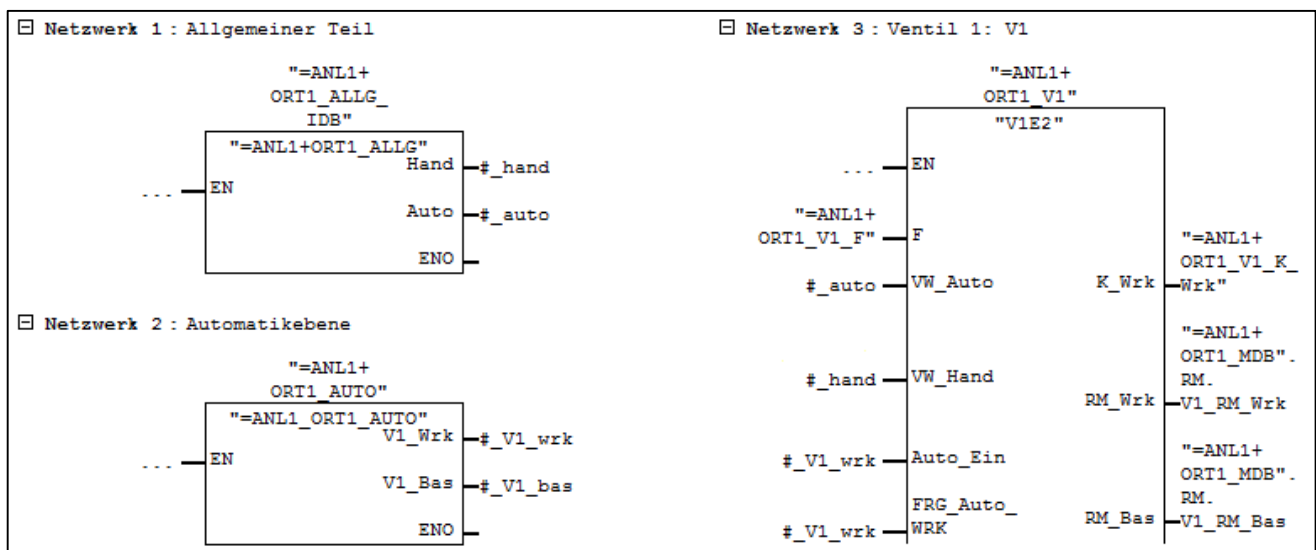


Abbildung 5.4 Verschaltung der Betriebsarten und Automatikschnittstelle auf Aggregatsbausteine

Da auch allgemeine Funktionalitäten wie Uhrzeitsynchronisation, Systemfehlerauswertung oder Taktgenerierung bei jedem Projekt zum Einsatz kommen, kann auch auf dieser Ebene einiges vorbereitet werden.

Durch diese Methodik verringert sich der tatsächliche Entwicklungsaufwand, welcher innerhalb der Entwicklungsumgebung zu erstellen ist, auf ein Minimum. Lediglich die Automatikabläufe der Aggregatssteuerung, Teile der anlagenspezifischen bzw. allgemeinen Funktionen, welche keiner Anlage zugeordnet werden können und die globale Steuerung sind hierbei noch manuell zu bearbeiten.

Verschiedene Systemhersteller bieten unterschiedliche Verfahren an, um extern erzeugte Codes in die Entwicklungsumgebung einschleusen zu können. Der Simatic-Manager (IDE für Siemens Steuerungssysteme) besitzt hierzu Quelldateien, dies sind einfache Textdateien aus welchen bei syntaktischer Korrektheit der ladbare Softwarecode übersetzt werden kann. Symboliken der globalen Variablen wie FeldEAs können ebenso importiert werden.

5.3 Qualitativer und quantitativer Nutzen

Ein automatisiertes Vorgehen macht den Entwicklungsprozess nicht nur effizienter, sondern liefert auch Software in höherer Qualität, da die Fehlerrate durch das Ausschalten einer menschlichen Bearbeitung eliminiert wird. Des Weiteren wird die Wartbarkeit erhöht, da alle Projekte gleich aufgebaut sind. Auch die in Kapitel 4.8 beschriebene Traceability wird durch dieses Vorgehen verbessert, da ohne menschliches Zutun alle Informationen, welche der CAD-Konstrukteur liefert, an die Software durchgereicht werden.

Betrachtet man nach der Einführung dieser Automatismen die Zeitaufwandsschätzung von Kapitel 4.9 erneut so ergibt sich für t_{block} :

1. Zuteilung des Aggregats zu einer Anlage (0 Min.)
⇒ Wird automatisch vom CAD-System geliefert und importiert
2. Manuelles Ausfüllen der Aggregatsliste (0 Min.)
⇒ Automatisches ausfüllen der Liste möglich
3. Erstellen eines neuen Netzwerks + Beschriftung mit BMK und Klartext (0 Min.)
⇒ Automatische Erstellung durch das System
4. Baustein aufrufen und korrekte Instanz erstellen (0 Min.)
⇒ Automatische Erstellung durch das System
5. Instanz laut Schlüssel beschriften (0 Min.)
⇒ Automatische Erstellung durch das System
6. Schnittstellenbereiche für Globaldaten erstellen (0 Min.)
⇒ Automatische Erstellung durch das System
7. Schnittstelle zu Automatikenebene erstellen (0 Min.)
⇒ Automatische Erstellung durch das System
8. Bausteinschnittstelle beschalten (0 Min.)
⇒ Automatische Erstellung durch das System

Bei sehr ausgereiften Standards erstellt das System vollständig die Architektur sowie die Anlagenspezifischen Funktionen, große Teile des Aggregathandling und bereitet die Automatikenebene vor.

Der Gesamtzeitaufwand t_{ges} berechnet sich somit nach der Formel:

$$t_{ges} = (0,1 * t_{agg} + 0,95 * t_{secAuto} + 0,1 * t_{interf} + t_{toolSec}) * m + t_{dev} + t_{toolGen}$$

Hierbei sieht man, dass der Gesamtaufwand nun nicht mehr von der Anzahl der Bausteine abhängig ist sondern nur noch von der Anzahl der Anlagenteile m . Wobei sich auch hier die Steigung der Funktion geändert hat, da große Teile bereits vollständig oder teilweise vom System vorbereitet wurden. Der jeweilige Zahlenwert vor t_{agg} (Zeit für das Aggregathandling), $t_{secAuto}$ (Zeit für den Automatikablauf) und t_{interf} (Zeit für die Schnittstellenerstellung) stellt einen Prozentwert dar, welcher wieder spiegelt, wieviel die einzelnen Bereiche noch manuell nachgearbeitet werden müssen (Diese Werte wurden vom Autor abgeschätzt). Hinzugekommen zu der Zeitberechnung ist das allgemeine Handling des Drittsystems $t_{toolGen}$ und ein anlagenabhängiger Teil $t_{toolSec}$.

6 Proof of Concept

Um zu zeigen, dass dieses Konzept machbar ist, wurde auf Basis der Eclipse-Plattform ein Software-tool mit folgenden Features erstellt:

- Einlesen von bestehenden Siemens-Step7-Bausteinen
- Auf einer listenbasierten Oberfläche diese Bausteine automatisch mit Ein-/Ausgängen verschalten
- Generierung eines auf Simatic Step 7 lauffähigen Softwarecodes

6.1 Eclipse-Plattform

Die Eclipse-Plattform stellt eine sehr stabile und breit verwendbare quelloffene Entwicklungsumgebung dar. Sie beheimatet nicht nur eine der erfolgreichsten Java IDEs, sondern durch ihren modularen Aufbau ermöglicht sie es Entwicklern, eigene, komponentenbasierende Systeme zu erstellen. Die Plattform stellt eigene, schnell zu verwendende und zuverlässige User-Interface-Komponenten über diverse APIs zur Verfügung und besitzt ein weites Featurespektrum, welches nur angepasst werden muss.

Firmen wie IBM, SAP und Google nutzen dieses Framework als Basis für ihre Produkte. Daher ist sichergestellt, dass Eclipse noch lange existiert und eine große Entwicklergemeinde beheimatet, wodurch auch guter Support sichergestellt ist.

6.2 Bausteinaufbau und Bausteinschnittstellen in S7-Quellen

Da für das Einlesen und Generieren der Bausteinen- und Datentypaufbau des Zielsystems wichtig ist, möchte ich diese hier kurz umreißen.

In Siemens-Zielsystemen lassen sich folgende Arten von Bausteinen unterscheiden: Organisationsbausteine, Funktionen, Funktionsbausteine, Typen und Datenbausteine. Diese Bausteine können innerhalb der IDE mit grafischem oder textuellem Code befüllt werden. Es besteht auch die Möglichkeit die Bausteine via Textfiles zu exportieren, bzw. zu importieren. Solche Textfiles werden als Quellen bezeichnet. Diese Files sind die Schlüsselkomponenten, um extern generierte Codes in das System einschleusen zu können. Ebenso kann man mithilfe dieser Files Bausteinvorlagen aus dem System exportieren und fertig verschaltete Bausteine importieren.

Man unterscheidet AWL-Quellen (*.awl) und SCL-Quellen (*.scl), wobei die AWL-Quelle einen Assembler-Code und die SCL-Quelle einen Hochsprachencode beinhaltet. Beide besitzen jedoch dieselben Schlüsselwörter, wodurch die einzelnen Bausteinteilbereiche beschrieben werden. Auch die Variablendefinition ist nach dem gleichen Schema aufgebaut. Jeder Bausteintyp hat unterschiedliche Teilbereiche, jedoch einen gemeinsamen Datenkopf.

Im weiteren Verlauf werde diese Quelldateien auch als Bausteinquellen oder zielsystemabhängige Quelldateien bezeichnet. Wobei diese Begriffe als Synonyme zu verstehen sind.

6.2.1 Allgemeiner Teil

Im allgemeinen Teil (dargestellt in Abbildung 6.1) befinden sich die gemeinsamen Daten jedes Bausteins. Dies umfasst den Bausteintypen, den symbolischen Namen, Symbolkommentar, Bausteincommentar, einen Headernamen, Versionsnummer, Bausteinfamilie, Autor, eine Liste von Eigenschaften und eine Liste von Bausteinattributen, welche systemintern bestimmte Reaktionen auslösen. Begrenzt wird ein Baustein durch den Bausteintypen, gefolgt von seinem Namen oder Adresse bis zum Schlüsselwort END_<Datentyp>.

<pre><DATENTYP> <Symbolic Name> TITLE = <Blocktitel> //Block Comment [{List of attributes}] AUTHOR : <'Name of the author'> FAMILY : <'Blockfamily'> NAME : <'Blockname'> VERSION : <Version> [{List of properties}] <SOURCE> END_<DATENTYP></pre>	<pre>DATA_BLOCK DB 2 TITLE = //Block Comment { S7_m_c := 'true' } AUTHOR : 'Author' FAMILY : 'Family' NAME : 'Name(HE)' VERSION : 0.1 UNLINKED READ_ONLY NON_RETAIN STRUCT _int { S7_m_c := 'true' } : INT ; END_STRUCT ; BEGIN _int := 0; END_DATA_BLOCK</pre>
--	--

Abbildung 6.1 Schematischer Aufbau und konkrete Implementierung des allgemeinen Teils

6.2.2 Aufbau bausteinbezogener Variablen

Man unterscheidet in der Siemensarchitektur die elementaren Datentypen und die komplexen Datentypen. Elementare Datentypen weisen eine maximale Länge von vier Bytes auf. Hier unterscheidet man: BOOL, BYTE, CHAR, WORD, DWORD, INT, DINT, REAL, S5TIME, TIME, DATE, TIME_OF_DAY, BLOCK_FB, BLOCK_FC, BLOCK_DB, BLOCK_SDB, TIMER, COUNTER, POINTER. Als komplexe Datentypen sind dem System bekannt: STRING, ARRAY, STRUCT, ANY, DATE_AND_TIME und eigene Typen. Diese weisen bzw. können eine Länge größer vier Bytes aufweisen.

Der Aufbau bausteinbezogener Variablen (BausteinEAs) verläuft nach dem Schema von Abbildung 6.2.

```
//Maximaler Ausbau einer Variablendefinition
<Tagname>[{List of attributes}]:<datatype>[{length}]OF <Datatype>[:='String'];//Comment

//Aufbau eines Elementaren Datentyps
<Tagname>[{List of attributes}]:<datatype>;//Comment
Beispiel:
variablenname{S7_m_c := 'true'} : INT; //Variable vom Typ Integer

//Aufbau des Komplexen Datentyp Array
<Tagname>[{List of attributes}]:<datatype> [from .. to] OF <Datatype>;//Comment
Beispiel:
arrayname{S7_m_c := 'true'} : ARRAY [1..10] OF INT; //Variable vom Typ Integer

//Aufbau des Komplexen Datentyp Struct
<Tagname>[{List of attributes}]:STRUCT;//Comment
Beliebige Datentypen
END_STRUCT

Beispiel:
strukturname : STRUCT
    _Bool : BOOL ;
    _Byte : BYTE ;
    _DateAndTime : DATE_AND_TIME ;
    _String : STRING [254] ;
    _Array : ARRAY [1 .. 2 ] OF INT ;
END_STRUCT ;
```

Abbildung 6.2 Bausteinbezogene Variablen (BausteinEAs)

Um die Schnittstellen des Bausteins mit den entsprechenden Datentypen zu versorgen, musste hierbei eine entsprechende Typstruktur geschaffen werden (Abbildung 6.3), wobei es elementare Datentypen und zusammengesetzte Datentypen zu unterscheiden gilt. Bei zusammengesetzten Datentypen wie „Struct“ oder „Array“ ist darauf zu achten, dass diese auch mehrfach ineinander geschachtelt werden können (Abbildung 6.4).

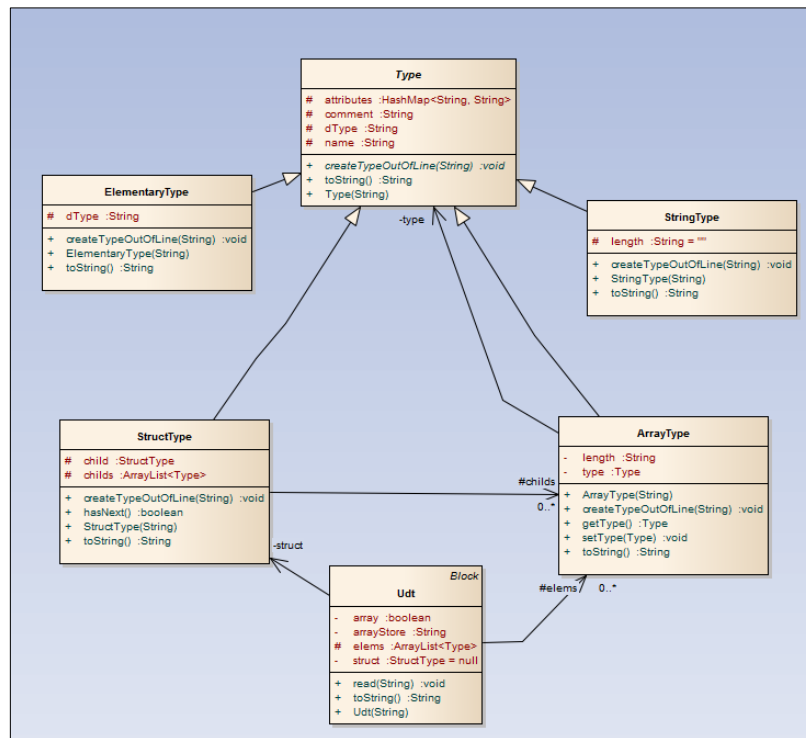


Abbildung 6.3 Typstruktur der SPS-Typen

```

VAR_IN_OUT
inout : ARRAY [0 .. 2 ] OF
STRUCT
zweite : ARRAY [0 .. 3 ] OF
STRUCT
st1 : STRUCT
elem1 : INT ;
elem2 : BOOL ;
elemAr : ARRAY [2 .. 4 ] OF
INT ;
END_STRUCT ;
st2 : STRUCT
elem1 : INT ;
elemAr : ARRAY [2 .. 4 ] OF
INT ;
elem2 : BOOL ;
END_STRUCT ;
END_STRUCT ;
END_VAR

```

Abbildung 6.4 Mehrfachverschachtelung von Arrays und Structs

6.2.3 Typen

Typen dienen dazu, eigene strukturierte Datentypen zu erstellen. Wie der Name schon anklingen lässt, geschieht dies in Form einer Struktur. Typen bestehen hierbei nur aus dem beschreibenden Teil der Struktur, also der Deklaration. Diese neuen Typen können anschließend im gesamten Programm verwendet werden. Es ist auch möglich, ganze Datenbausteine von ihnen abzuleiten. Der Aufbau ist in Abbildung 6.5 dargestellt

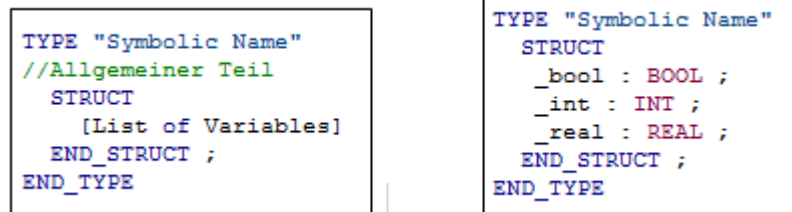


Abbildung 6.5 Schematischer Aufbau und konkrete Implementierung von Typen

6.2.4 Datenbausteine

Datenbausteine werden verwendet, um globale Daten an zentraler Stelle zu verwalten und diese entsprechend zu strukturieren. Datenbausteine bestehen aus einem strukturellen Teil, dem deklarativen Teil und einem darauffolgenden, definierenden Teil. Eine weitere Option ist, dass man Datenbausteine von Funktionsbausteinen ableiten kann. Hier wurde die Möglichkeit geschaffen, Objektdaten zusammenzufassen und Funktionsbausteine mehrfach unter anderem Kontext zu instanziiieren. Bei diesem Vorgehen liefert die Schnittstelle des Funktionsbausteins die Typdeklaration und im Datenbaustein findet die Typdefinition statt. Nach gleichem Prinzip funktioniert die Ableitung von Typen.

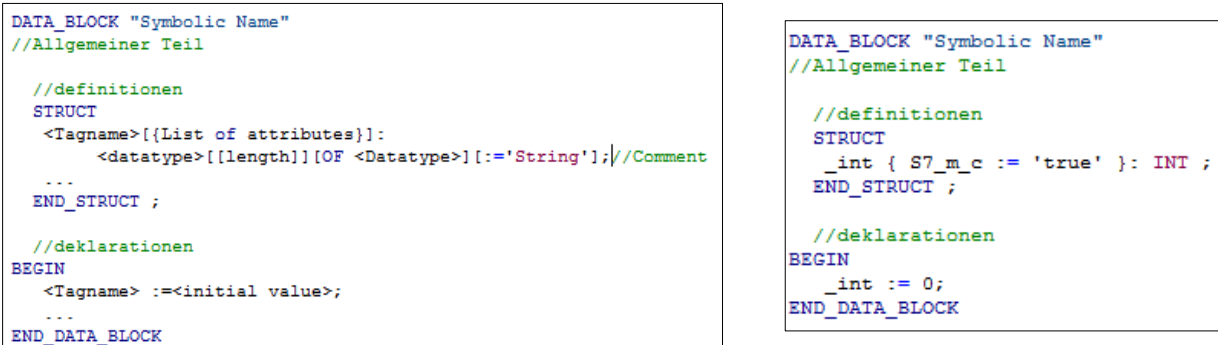


Abbildung 6.6 Schematischer Aufbau und konkrete Implementierung von Datenbausteinen

6.2.5 Organisationsbausteine

Organisationsbausteine, wie in Abbildung 6.7 dargestellt, werden von der Firmware der Steuerung ereignisgesteuert aufgerufen. Sie besitzen keinerlei Ein- oder Ausgabeparameter sondern lediglich einen temporären Variablenbereich, in welchem oftmals Daten von der Firmware über den Lokaldatenstack an das Anwenderprogramm übergeben werden. Sie dienen somit als Schnittstelle zwischen dem Anwenderprogramm und dem Betriebssystem. Organisationsbausteine verfügen im Bereich der Schnittstellendefinition zwei Schlüsselwörter. VAR_TEMP kennzeichnet den Beginn der temporären Variablen und END_VAR das Ende eines Variablenblocks. Das Schlüsselwort BEGIN kennzeichnet das Ende des Variablenbereichs und den Beginn des Programms.

```
ORGANIZATION_BLOCK "Main"
//Allgemein

VAR_TEMP
    OB1_EV_CLASS : BYTE ; //Bits 0-3 = 1 (Coming event), Bits 4-7 = 1 (Event class 1)
    OB1_SCAN_1 : BYTE ; //1 (Cold restart scan 1 of OB 1), 3 (Scan 2-n of OB 1)
    OB1_PRIORITY : BYTE ; //Priority of OB Execution
    OB1_OB_NUMBR : BYTE ; //1 (Organization block 1, OB1)
    OB1_RESERVED_1 : BYTE ; //Reserved for system
    OB1_RESERVED_2 : BYTE ; //Reserved for system
    OB1_PREV_CYCLE : INT ; //Cycle time of previous OB1 scan (milliseconds)
    OB1_MIN_CYCLE : INT ; //Minimum cycle time of OB1 (milliseconds)
    OB1_MAX_CYCLE : INT ; //Maximum cycle time of OB1 (milliseconds)
    OB1_DATE_TIME : DATE_AND_TIME ; //Date and time OB1 started
END_VAR
BEGIN

NETWORK
TITLE =call Functionblock
CALL FB1,DB1;

END_ORGANIZATION_BLOCK
```

Abbildung 6.7 Aufbau Organisationsbaustein OB1

6.2.6 Funktionen

Eine Funktion besitzt laut Definition mehrere Eingabeparameter und genau einen Ausgabeparameter. Streng formal ist eine Funktion eine Abbildung von $U^n \rightarrow U$, wobei U das mögliche Werteverseum darstellt. Funktionen werden in der Siemenswelt nicht so strikt definiert, obwohl dies ebenso möglich ist. Funktionen können hier neben dem klassischen Rückgabeparameter beliebig viele Rückgabeparameter besitzen. Der klassische Rückgabeparameter wird hierbei als Ret_Val bezeichnet und die zusätzlichen als VAR_OUTPUT. Die Eingabeparameter tragen die Bezeichnung VAR_INPUT. Einen weiteren Schnittstellenbereich stellen die Ein-/Ausgabeparameter dar, welche sowohl eingelesen als auch geschrieben werden können. Diese tragen die Bezeichnung VAR_IN_OUT. Auch wenn die Gruppe der temporären Variablen nicht zur Schnittstelle des Bausteins gehört, werden diese im Bausteinkopf angeführt. Die gesamte Bausteinschnittstelle wird am Lokaldatenstack abgelegt, wodurch sie keine wertspeichernde Eigenschaft über den Baustein hinaus besitzt.

Bei Funktionen kann auch das Schlüsselwort NETWORK verwendet werden. Dieses grenzt eine logische Teilgruppe des Funktionscodes ein. Ein Netzwerk besitzt einen Netzwerktitel und einen Netzwerkkommentar. Funktionscode darf in AWL-Quellen nicht außerhalb von Netzwerken stehen, in SCL-Quellen gibt es diese Einschränkung nicht. Abbildung 6.8 zeigt beispielhaft eine konkrete Implementierung einer Funktion.

```
FUNCTION FC 1 : BOOL
//Allgemein

VAR_INPUT
    inBool : BOOL ;
    inByte : BYTE ;
END_VAR
VAR_OUTPUT
    outBool : BOOL ;
    outByte : BYTE ;
END_VAR
VAR_IN_OUT
    inoutBool : BOOL ;
    inoutByte : BYTE ;
END_VAR
VAR_TEMP
    tempBool : BOOL ;
    tempByte : BYTE ;
END_VAR
BEGIN

NETWORK
TITLE =Netzwerktitel NW1

    U    #inBool;
    =    #outBool;
    L    #inByte;
    SLW 3;
    T    #outByte

END_FUNCTION
```

Abbildung 6.8 Aufbau Funktionsbaustein

6.2.7 Funktionsbausteine

Funktionsbausteine weisen einen weiteren Block im Schnittstellenbereich auf, den VAR. In diesem Bereich können Daten abgelegt werden, welche beim nächsten Aufruf wieder benötigt werden, wie Zählwerte, Zeiten, etc. Neben dieser offensichtlichen Erweiterung gibt es zwischen Funktion und Funktionsbaustein noch den wesentlichen Unterschied, dass Funktionsbausteine mit Datenbausteinen gekoppelt sind, wodurch diese ein nicht-flüchtiges Verhalten aufweisen. Hierdurch ändert sich auch der Datenaustausch im Hintergrund. Bei Funktionen geschieht die Übergabe von Formalparametern über den Lokaldatenstack, bei Funktionsbausteinen wird dies über den Datenbaustein realisiert. Dadurch ändert sich im Wesentlichen, dass jede Schnittstellenvariable eine globale Adresse bezieht und man sie mit Startwerten versehen kann. Somit kann ein nicht beschalteter Eingang zu keinem unerwarteten Ergebnis führen.

Hierbei werden die Bereiche VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT und VAR des Funktionsbausteins als Typdeklaration und der gekoppelte Datenbaustein als Typdefinition benutzt. Diesen Vorgang nennt man Instanziierung. Temporäre Variablen werden wie beim FC am Lokaldatenstack abgelegt. Abbildung 6.9 zeigt beispielhaft eine konkrete Implementierung eines Funktionsbausteins.

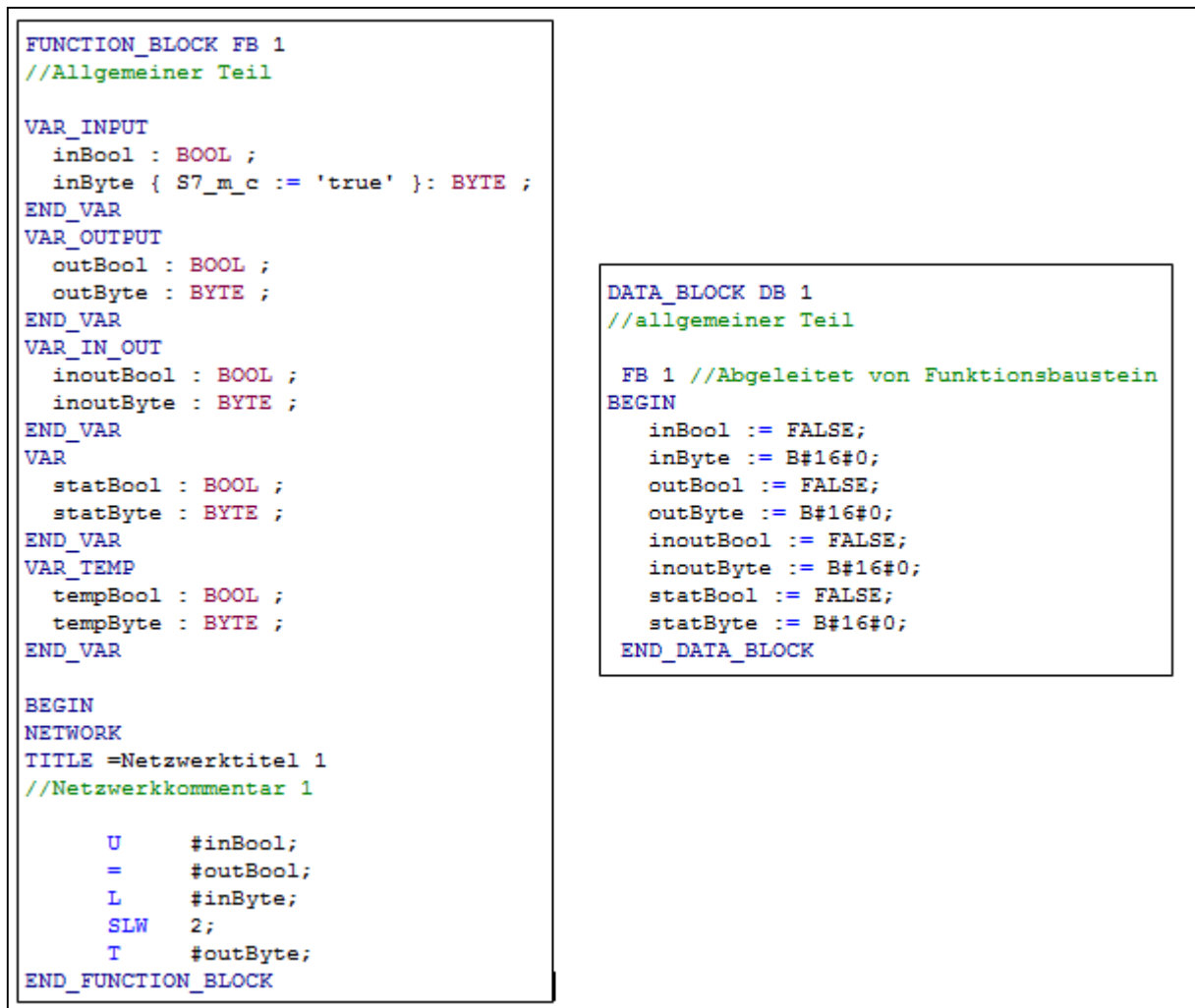


Abbildung 6.9 Funktionsbaustein mit zugehörigem Datenbaustein

6.3 Softwarebeschreibung

Das Softwaretool besteht aus drei Kernkomponenten (Abbildung 6.10), welche alle zueinander lose gekoppelt sind.

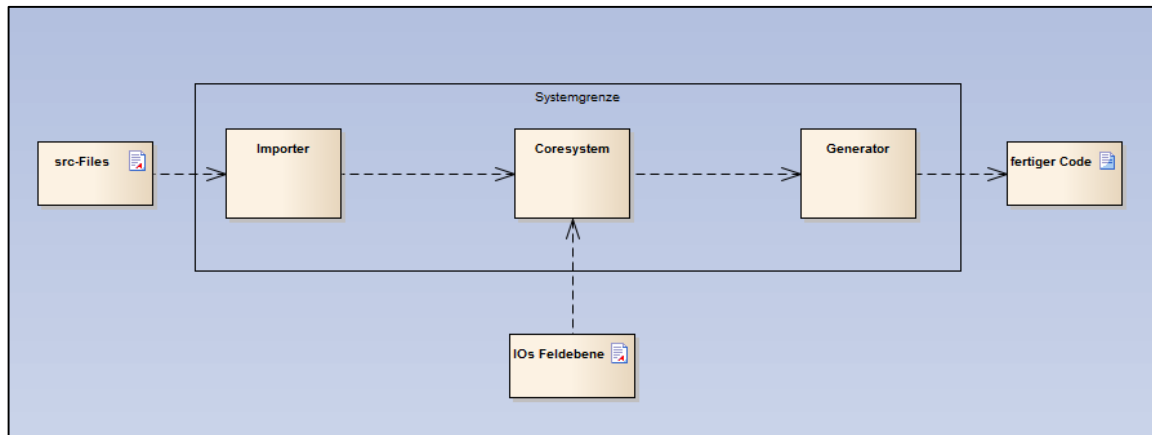


Abbildung 6.10 Komponenten des Systems

- Der Importer: hier werden zielsystemabhängige Quelldateien (siehe Kapitel 6.2) eingelesen und abstrahiert
- Das Coresystem: verarbeitet die abstrakten Daten systemunabhängig und erzeugt die Verschaltungen durch Hinzugabe der FelDEAs und Informationen des Users.
- Der Generator: erzeugt aus dem Output des Coresystems und des Importers wieder zielsystemabhängigen Quellcode

Alle drei Komponenten können unabhängig voneinander ausgetauscht werden und es können auch mehrere Subsysteme nebeneinander existieren. Je nach Zielsystem oder gewähltem Standard können dann verschiedene Importer bzw. Generatoren verwendet werden. Da das Coresystem weitgehend systemunabhängig ist und nur Daten auf XML-Basis verarbeitet, ist dies allgemein gültig.

Hier nun eine genauere Betrachtung des Systems unter Miteinbeziehung eines Users.

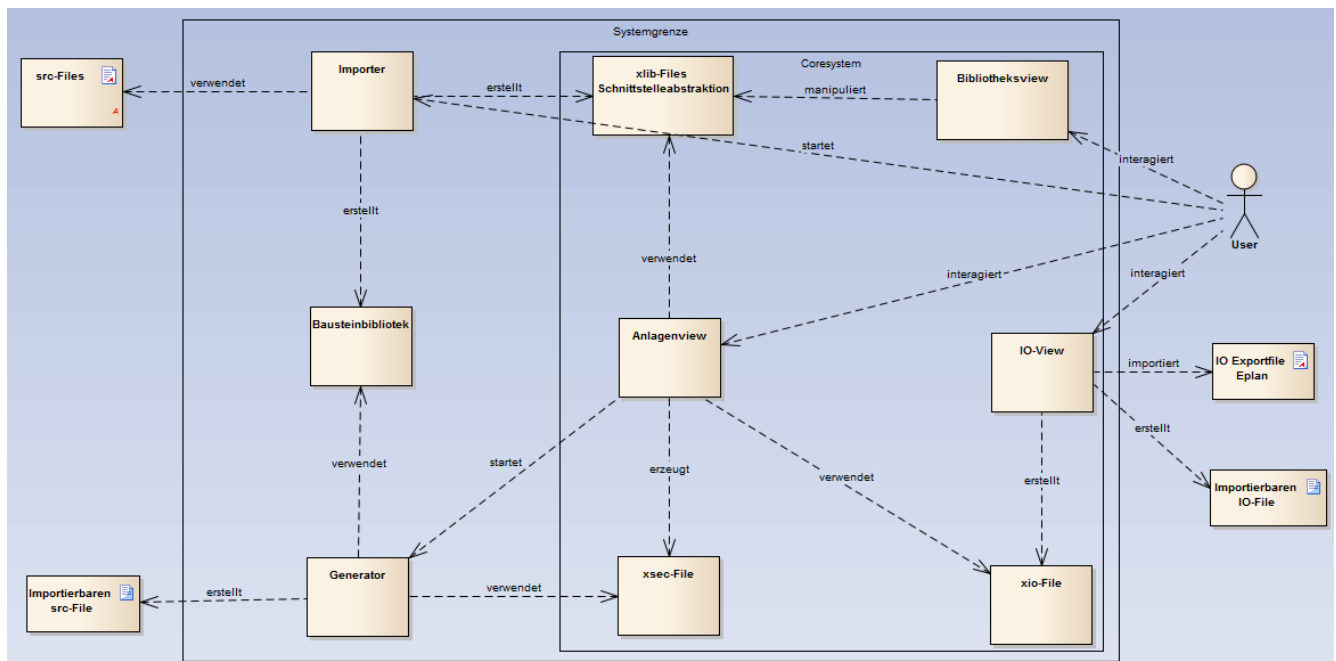


Abbildung 6.11 Funktionsprinzip des Softwareprodukts

Abbildung 6.11 zeigt die Interaktionen der einzelnen Komponenten und des Users. Der folgende Text beschreibt dies verbal. Systemabhängige Begriffe werden in den darauffolgenden Kapiteln genauer erläutert. Der User startet den Importvorgang. Hierbei werden zielsystemabhängige Quelldateien (siehe Kapitel 6.2) in das System eingelesen. Der Importer erstellt in weiterer Folge einerseits die Bausteinbibliothek und andererseits abstrahiert dieser die jeweiligen Schnittstellen der eingelesenen Bausteine und legt diese im XML-Format ab. Jeder BausteinEA wird um zusätzliche XML-Attribute erweitert. Mithilfe des Bibliotheks-View kann der User die XML-Attribute der Schnittstellen insofern manipulieren, dass er dem System mitteilt, um welche Kategorie von BausteinEA es sich handelt (FeldEA, Lokalvariable, Globalvariable, ...). Der User kann mithilfe des IO-View FeldEAs von der E-Konstruktion in das System importieren oder selbst Variablen erstellen. Ebenfalls über den IO-View können diese wieder in Zielsystem konformer Weise exportiert werden. Der IO-View legt die importierten Daten im XML-Format im System ab. Am Anlagenview erstellt der User für jedes Betriebsmittel (Aggregat) innerhalb der Anlage einen Eintrag und vergibt den entsprechenden Aggregatstyp, das Betriebsmittelkennzeichen und einen Kommentar. Sind alle Betriebsmittel aufgenommen, kann die Liste automatisch vom System komplementiert werden. Bei dieser Komplementierung wird der IO-File ausgelesen und alle FeldEAs werden auf die entsprechenden Aggregate (BausteinEAs) verschalten (siehe Kapitel 5.2). Ist die Liste vollständig ausgefüllt, kann der User über den Anlagenview den Export starten. Der Generator nimmt hierzu die Bausteinbibliothek und den entsprechenden Anlagenfile und erzeugt daraus einen zielsystemkonformen Steuerungscode.

6.3.1 Erstellen eines Projektes und der Projektstruktur

Mithilfe des Projektwizards kann ein neues Projekt erstellt werden. Hierbei werden der Projektname und der Name der ersten SPS vergeben. Einem Projekt können weitere SPSn hinzugefügt werden.

Dem System sind zum momentanen Zeitpunkt folgende Fileformate bekannt.

- *.xio: IO-File, in diesem werden globale Variablen wie Merker oder FeldEAs deklariert.
- *.xsec: Anlagenfiles, jeder logische Anlagenteil (siehe Kapitel 4.4.3) wird in Form eines Anlagenfiles repräsentiert. Auf dem Anlagenfile werden alle anlagenzugehörigen Betriebsmittel (Aggregate) aufgelistet mit zugehörigen FeldEAs. Ein Anlagenfile kann somit als Untermenge der Aggregatsliste gesehen werden.
- *.xlib: Libraryconfigurationfile, diese Datei verwaltet die abstrakten Schnittstellen der eingelesenen Bausteine und die zusätzlichen XML-Attribute der BausteinEAs (siehe hierzu Kapitel 6.3.2).

Ein Projekt, wie in Abbildung 6.12 dargestellt, gliedert sich in vier Bereiche:

- Allg: Ablage Projektallgemeine Files (IO-Files, Beschreibungen, etc.)
- Anl: Ablage der Anlagenfiles
- Bin: Ablage vom System erzeugter Exportfiles (IO-Exportfiles, Anlagenexportfiles)
- Libs: Ablage der Projektlibrary (xlib, awl-Quellen, scl-Quellen)

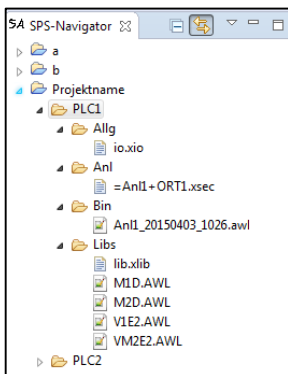


Abbildung 6.12 Projektübersicht

Alle Files können mithilfe eines Wizards erstellt, projektintern oder projektübergreifend kopiert werden. Da jeder File auf XML beruht, können diese Files auch von anderen Programmen bearbeitet werden. Somit wird es ermöglicht, dass IO-Files oder die Anlagenaufteilung von CAD-Systemen erzeugt werden und hier nur noch importiert werden müssen.

6.3.2 Einlesen der Bausteine

Damit Bausteine eingelesen werden können, müssen diese aus dem Steuerungssystem über Quellen exportiert werden. Danach können mithilfe eines Wizards ein oder mehrere Files in das System eingelesen werden. Die eingelesenen Bausteine werden danach projektglobal im System abgelegt und können ab diesem Zeitpunkt verwendet werden.

Beim Einlesen wird automatisch erkannt, um welchen Bausteintypen es sich handelt und ein entsprechend neues Objekt erzeugt. Die neu erzeugte Instanz wird in einer Hashmap abgelegt, wobei der Key eine Kombination aus Filename und Blockname ist. Wie oben bereits beschrieben, wird neben dem Baustein auch eine abstrakte Schnittstellenbeschreibung des Bausteins im System abgelegt.

Der Einleseprozess gliedert sich in mehrere Phasen, welche in Abbildung 6.13 dargestellt sind.

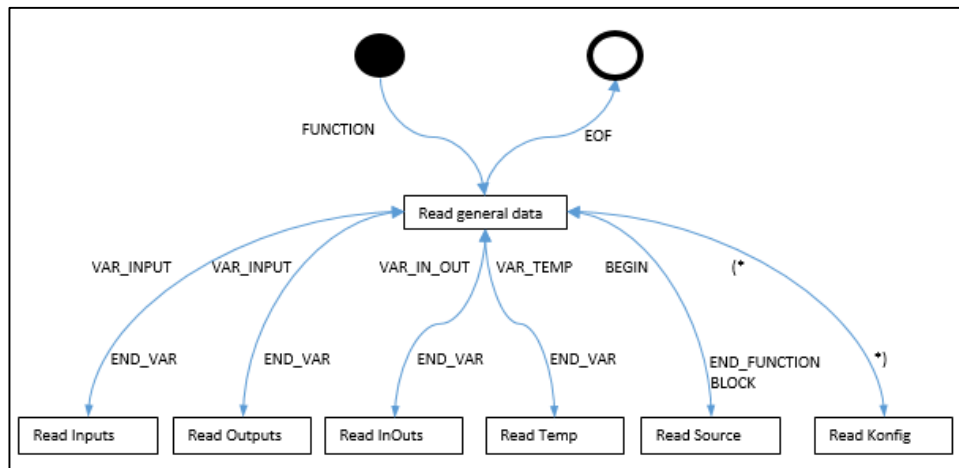


Abbildung 6.13 Grafische Darstellung des Einleseprozess

Jeder Teilblock wird durch spezielle Schlüsselwortpaare erkannt und abgearbeitet, wobei nicht immer alle Teilblöcke existieren müssen. Jedes Schlüsselwortpaar besteht immer aus einem öffnenden und einem schließenden Tag. Falls Schlüsselwörter mehrmals verwendet werden oder zwei öffnende Tags hintereinander folgen, wird der Importvorgang mit entsprechender Fehlermeldung gestoppt.

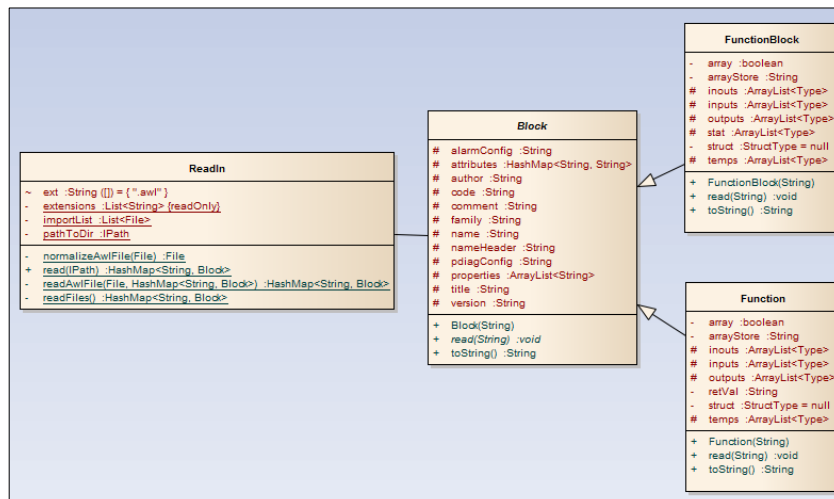


Abbildung 6.14 Typstruktur und Aufrufstruktur von Blöcken

Die beim Einlesen entstandene Schnittstellenbeschreibung der Bausteine wird im XML-Format abgelegt und um zwei Attribute erweitert (Abbildung 6.15). Das Attribut „type“ gibt Auskunft, ob es sich bei dem BausteinEA um einen IN, einen OUT oder einen INOUT handelt. Dieser wird beim Einlesen fix vergeben. Das Attribut „category“ speichert Informationen, woher es im späteren Prozess seinen Verschaltungswert bekommt. Dies muss vom User projektweit einmalig pro Aggregatstyp festgelegt werden. Hierzu dient der Library-Editor, welcher *.xlib Files standardmäßig öffnet. Je nach Kategorie wird im späteren Verlauf die IO-Liste nach einem passenden Partner durchsucht, im Schnittstellenbereich ein Datenpunkt erzeugt und verschalten oder eine lokale Variable erstellt.

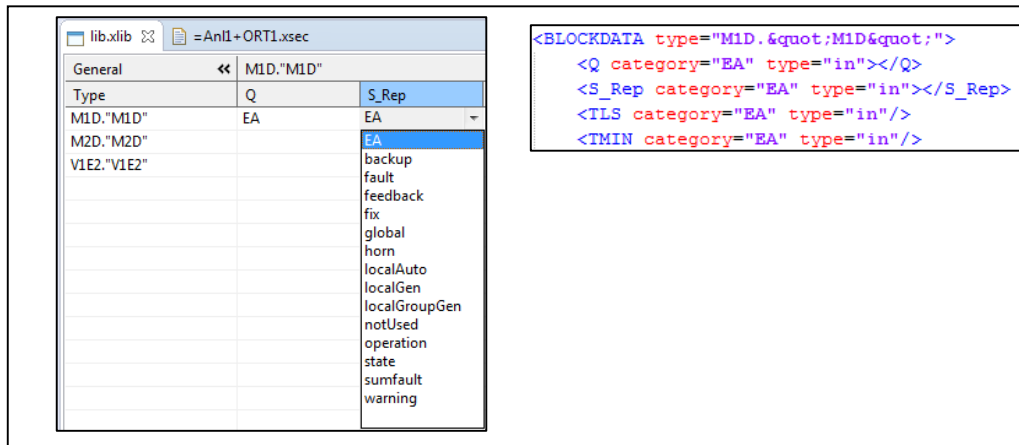


Abbildung 6.15 Festlegung der Kategorie und Abbildung als XML

Folgende Kategorien sind bislang im System vorhanden und erzeugen bei dem momentan implementieren Generator folgende Reaktion.

- EA⁵: Verschaltungspartner befindet sich in der Feldebene.
- Fix⁵: Wert ist ein Fixwert, welcher am Anlagen-View vergeben wird.
- Global⁵: Wert ist eine globale Variable welche im xio-File vorhanden ist.
- Fault: Die Fehlerstruktur im Schnittstellenbereich wird beim Generieren automatisch um einen Datenpunkt erweitert und dieser wird auf dem BausteinEA verschalten.
- Warning: Wie Fault, jedoch innerhalb der Warnungsstruktur.
- Feedback: Wie Fault, jedoch innerhalb der Rückmeldungsstruktur.
- Sumfault: Wie Fault, jedoch innerhalb der Summenfehlerstruktur.
- Horn: Wie Fault, jedoch innerhalb der Hupenstruktur.
- State: Wie Fault, jedoch innerhalb der Betriebszustandsstruktur.
- Operation: Wie Fault, jedoch innerhalb der Operationsstruktur.
- localAuto: Wert kommt aus der Automatiebene. Hierbei wird im Automatikbaustein eine Ausgangvariable (outs) definiert, im Baustein Aggregatshandling wird eine lokale Variable (temp) erzeugt und diese auf die beiden Bausteine verschalten.
- localGen: Wie localAuto, jedoch hier mit dem allgemeinen Baustein.
- localGroupGen: Hier wird eine Ausgangvariable im allgemeinen Baustein erzeugt, im Baustein Aggregatshandling wird eine Variable erstellt und diese auf alle BausteinEAs verschalten, welche als localGroupGen markiert sind und diesen Namen besitzen. Beispiele hierfür sind „Hand“ oder „Auto“.
- Backup: BausteinEA dient zu Backupzwecken.
- notUsed: Werden nicht weiter verarbeitet und werden mit einem null-Wert belegt.


Defaultmäßig werden alle BausteinEAs beim Einlesen auf EA gesetzt.


⁵ Kategorien, welche im Anlagenview angezeigt werden

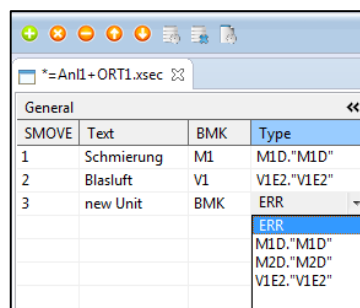
6.3.3 Verschalten der Anlagen

Die Anlagen- oder Sectionfiles dienen der Speicherung und Anzeige eines logischen Anlagenteils. Hier werden alle Betriebsmittel aufgelistet, welche zu einem logischen Anlagenteil gehören (Abbildung 6.16). Der Name des Anlagefiles dient hierbei als Prefix des Betriebsmittelkennzeichens.

Besitzt beispielsweise der Anlagenfile den Namen =Anl1+Ort1 und es sind 2 Betriebsmittel mit den Betriebsmittelkennzeichen M1 und V1 projektiert, so ergeben sich die tatsächlichen Betriebsmittelkennzeichen =Anl1+Ort1-M1 und =Anl1+Ort1-V1.

Der Anlagenfile wird mithilfe des Anlageneditors grafisch in tabellarischer Form dargestellt. Dieser Editor dient nicht nur zur Anzeige, sondern vielmehr zum Verschalten der FeldEAs zu den Baustein-EAs bzw. zur Vergabe von Fixwerten. In weiterer Folge wird von hier auch der Generierungsprozess eingeleitet. Durch das -Icon in der Toolbar wird dem Anlagenfile ein neues Betriebsmittel hinzugefügt. Vom User sind die Felder Beschreibung, Betriebsmittelkennzeichen und die Bausteintypen zu editieren. Als Bausteintypen können alle Aggregatstypen vergeben werden, welche sich in der Library befinden, diese stehen im entsprechenden Feld als Dropdown Menü zur Verfügung. Steht der Bausteintyp fest und wird dieser Typ das erste Mal in der Anlage verwendet, wird die Tabelle um alle BausteinEAs erweitert, welche die Kategorie EA, global oder fix, besitzen.


Die Verschaltung der BausteinEAs kann entweder manuell oder automatisch geschehen. Durch Betätigen des -Icons wird ein Fileauswahldialog gestartet, bei dem man den entsprechenden IO-File wählen kann, welchen man verschalten möchte. Der Defaultpfad ist hierbei das Verzeichnis „Allg“ des aktuellen Projekts, es kann aber auch jeder andere Dateipfad gewählt werden. Wurde die gewünschte Datei ausgewählt, startet der Ausfüllprozess. Hierbei wird für jeden Wert, welcher in der Tabelle dargestellt wird, die IO-Liste nach einem entsprechenden Partner durchsucht. Der erwartete, symbolische Name setzt sich aus Tabellennamen + BMK + BausteinEA-Name zusammen. Wird dieser Name in der IO-Liste gefunden, so wird der Wert in die Tabelle eingetragen bzw. in das Model im Hintergrund. Alle Werte, welche in der IO-Liste gefunden wurden und bereits in der Tabelle eingetragen sind, werden überschrieben. Alle anderen Zellen bleiben unberührt. Jede Zelle kann auch nachträglich manuell bearbeitet werden.



SMOVE	Text	BMK	Type
1	Schmierung	M1	M1D."M1D"
2	Blasluft	V1	V1E2."V1E2"
3	new Unit	BMK	ERR
			M1D."M1D"
			M2D."M2D"
			V1E2."V1E2"

Abbildung 6.16 Hinzufügen neuer Betriebsmittel zu einer Anlage

6.3.4 Codegenerierung

Die Generierung einer Anlage in Zielcode wird vom Anlagenview aus initiiert. Durch Betätigen des  -Icons wird ein Ordnerauswahldialog gestartet, in welchem festgelegt ist, wo der Exportfile abgelegt werden soll. Nach der Wahl des gewünschten Verzeichnisses (default Wert: Bin-Verzeichnis des Projekts), startet der Exportvorgang. Hierbei werden alle Bausteine, welche für einen übersetzbaren Quellcode benötigt werden, in eine Textdatei gespeichert. Anschließend kann der Entwickler die erstellte Datei in die Zielsystemsoftware importieren und übersetzen. Abbildung 6.17 stellt den Ablauf des Generierungsprozesses dar.

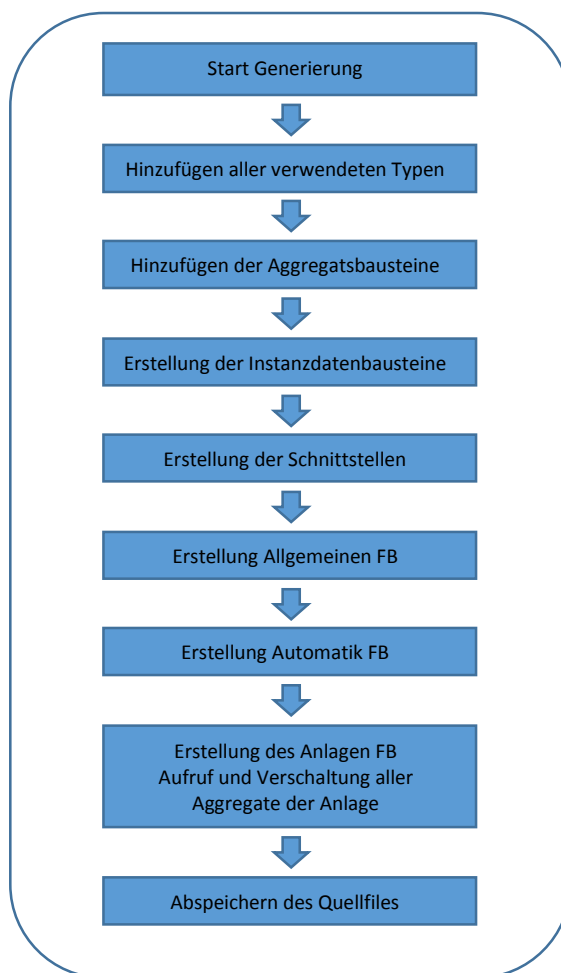


Abbildung 6.17 Modellierung des Generierungsvorgangs

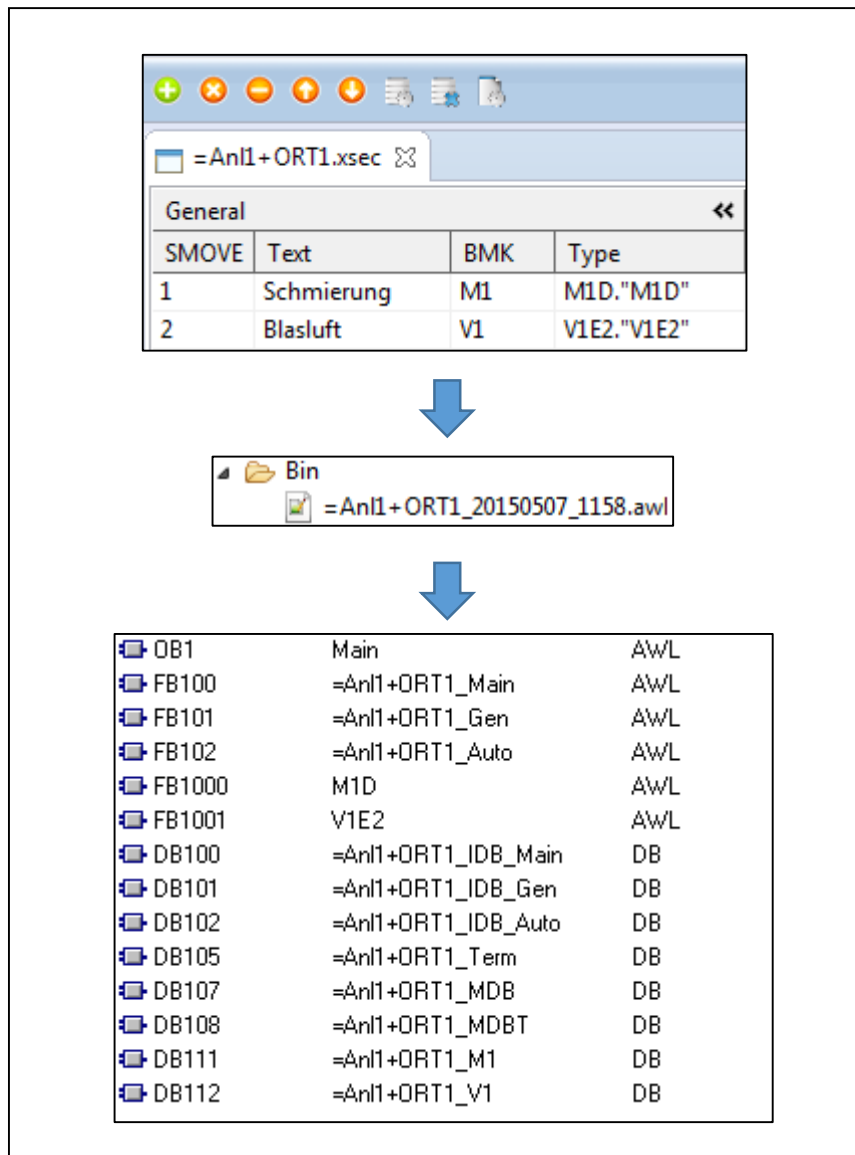


Abbildung 6.18 Erzeugung des Zielcodes

Abbildung 6.18 stellt grafisch die Erzeugung von Zielcode durch den Softwaregenerator dar. Hierbei wird die Generierung des Anlagenteils „=Anl1+Ort1“ wie oben beschrieben gestartet. Das Resultat ist eine AWL-Quelle (*.awl), welche den Namen des Anlagenteils plus das aktuelle Datum und die Uhrzeit des Generierungsvorgangs trägt. Diese Quelldatei kann in die Zielsystemsoftware importiert und übersetzt werden. Dadurch werden alle benötigten Bausteine im System erstellt, befüllt und verschalten. Im oben gezeigten Bild stellen die Funktionsbausteine FB100-FB102 die Anlagenfunktionen wie unter 4.4.3 beschrieben dar und die Datenbausteine DB100-DB108 beinhalten den Anlagenschnittstellenbereich. Die Funktionsbausteine FB1000 und FB1001 sind die Aggregatsbausteine für einen Motor und ein Ventil. Diese werden im FB100 aufgerufen und mittels der Datenbausteine DB111 und DB112 instanziiert.

7 Zusammenfassung und Ausblick

Der hier vorgestellte Ansatz zeigt, wie man durch geeignete Anlagenabstraktionen, unter Zuhilfenahme bestehender Architekturansätze und durch enge Zusammenarbeit mit Konstruktionsabteilungen die Gemeinsamkeiten von Steuerungsanlagen effizient nutzen kann, um qualitativ hochwertige Software zu erzeugen. Dieser Ansatz verschafft zusätzlich auch einen wirtschaftlichen Vorteil gegenüber den Mitbewerbern und diktiert bis zu einem gewissen Maß den Softwareaufbau, wodurch firmeninterne Standards besser eingehalten werden. Dass diese Idee auch realistisch umgesetzt werden kann, wurde mithilfe des Proof of Concept auch weitgehend und praxisnahe belegt. Auf die Modularität und Flexibilität der Softwarekomponenten wurde hier besonders Wert gelegt, sodass neue Bausteine leicht und ohne Programmieraufwand hinzugefügt werden können.

Das erstellte Plug-In stellt eine gute Basis für eine entsprechende Weiterentwicklung dieser Ansätze dar. Weitere Steuerungssysteme können leicht in das System eingebunden werden, welche individuell über das Factory-Pattern aufgerufen werden. Bestehende Bausteinobjekte können von einem System eingelesen werden und über einen entsprechenden Adapter in ein anderes System überführt werden.

Auch über Reverse/Roundtrip Engineering könnte im Zuge dessen nachgedacht werden, bei dem man gesamte Systeme einliest, diese entsprechend erweitert und wieder exportiert. Somit wird ein paralleles Arbeiten mit IDE und Generierungstool ermöglicht.

Moderne Softwareentwicklung im Bereich der Hochsprachen mit Stichworten wie UML sind sicherlich wegweisend auch für die Softwareentwicklung im SPS Bereich. Neue Konzepte erfordern immer wieder neue Herangehensweisen, aber wie man hier sieht, kann man auch bei bestehenden Systemen in neuen Bahnen denken und dadurch neue Möglichkeiten eröffnen.

8 Literaturverzeichnis

- [1] *DIN EN 62264-1: Integration von Unternehmensführungs- und Leitsystemen*, Deutsches Institut für Normung, 2014.
- [2] *DIN IEC 60050-351: Internationales Elektrotechnisches Wörterbuch*, Deutsches Institut für Normung, 2014.
- [3] L. Litz, Grundlagen der Automatisierungstechnik, Regelungssysteme - Steuerungssysteme - Hybride Systeme, 2. Hrsg., Rosenheimer Straße 145, D-81671 München: Oldenbourg Verlag München, 2013.
- [4] Duden Wirtschaft von A bis Z, Grundlagenwissen für Schule und Studium, Beruf und Alltag, 5. Hrsg., Bouche'straße 12, 12435 Berlin: Dudenverlag, 2013.
- [5] J. Weidauer, Elektrische Antriebstechnik, Grundlagen, Auslegung, Anwendung, Lösungen, Erlangen: Publicis Corporate Publishing, Erlangen, 2008.
- [6] Gabler Wirtschaftslexikon, Wiesbaden: Springer, 2013.
- [7] *DIN EN 61131-3: Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen*, Deutsches Institut für Normung, 2014.
- [8] K. H. John und M. Tiefelkamp, SPS-Programmierung mit IEC 61131-3, Berlin Heidelberg: Springer-Verlag, 2009.
- [9] G. Wellenreuther und D. Zastrow, Automatisieren mit SPS - Theorie und Praxis, 4 Hrsg., Wiesbaden: Vieweg+Teubner, 2008.
- [10] *DIN EN ISO 9001: Qualitätsmanagementsysteme – Erfolg durch Qualität*, DIN EN ISO, 2014.
- [11] S. Alliance, „Scrum Alliance,“ [Online]. Available: <https://www.scrumalliance.org/>. [Zugriff am 03 2015].
- [12] D. Winkeler, A. Schatten, S. Biffel, M. Demolsky, E. Gostischa-Franta und T. Östreicher, Best Practice Software-Engineering, Heidelberg: Spektrum Verlag, 2010.
- [13] A. M. Böhm und B. Jungkunz, Grundkurs IT-Berufe, Vieweg, 2005.
- [14] R. Schiedermeier, Programmieren mit Java, München: Pearson Studium, 2010.
- [15] H. Berger, Automatisieren mit Simatic, Erlangen: Publicis Publishing, 2010.
- [16] *DIN EN 81346: Industrielle Systeme, Anlagen und Ausrüstungen und Industrieprodukte - Strukturierungsprinzipien und Referenzkennzeichnung*, Deutsches Institut für Normung, 2015.

- [17] *DIN 6779: Kennzeichnungssystematik für technische Produkte und technische Produktdokumentation*, Deutsches Institut für Normung, 2011.
- [18] T. Bindel und D. Hofmann, *Projektierung von Automatisierungsanlagen*, Wiesbaden: Vieweg+Teubner, 2009.
- [19] M. von Detten, *Templatebasierte Codegenerierung für Speicherprogrammierbare Steuerungen*, Paderborn: Universität Paderborn, 2006.
- [20] H. Berger, *Automatisieren mit Step 7 in AWL und SCL*, 5. Hrsg., Erlangen: Publicis Corporate Publishing, 2006.
- [21] L. Vogel, *Eclipse 4 RCP - The complete guide to Eclipse application development*, Lars Vogel, 2013.
- [22] C. Pörnbacher, „Modellgetriebene Entwicklung der Steuerungssoftware automatisierter Fertigungssysteme,“ Herbert Utz Verlag GmbH, München, 2010.
- [23] Kagermann und Henning, *Umsetzungsempfehlungen*, Frankfurt/Main: acatech – Deutsche Akademie der Technikwissenschaften, 2013.
- [24] *DIN EN 61158: Industrielle Kommunikationsnetze - Feldbusse*, Deutsches Institut für Normung, 2015.
- [25] M. Metter und R. Bucher, *Industrial Ethernet in der Automatisierungstechnik*, 2. Hrsg., Erlangen: Publicis Corporate Publishing, 2007.
- [26] A. Blewitt, *Eclipse 4 Plug-in Development by Example*, Birmingham: Livery Place, 2013.
- [27] „VGB PowerTech,“ VGB PowerTech, [Online]. Available: <http://www.vgb.org/>.

9 Abbildungsverzeichnis

Abbildung 3.1 Reaktive Systeme in automatisierten Prozessen [3]	5
Abbildung 3.2 Automatisierungspyramide	6
Abbildung 3.3 Zyklusabbild Siemens S7	9
Abbildung 3.4 Aufrufstruktur POEs	10
Abbildung 3.5 Übersicht Sprachen aus IEC 61131-3 [9].....	11
Abbildung 4.1 Vorgehensmodel Automatisierungstechnik nach ISO9001 [10].....	12
Abbildung 4.2 interne Verschaltung eines Motorbausteins	15
Abbildung 4.3 Überführung realer Objekte in virtuelle	16
Abbildung 4.4 Technologische Programmgliederung unter Verwendung einer Schichtenarchitektur	18
Abbildung 4.5 Schlüssel Betriebsmittelkennzeichnungssystem (BMK)	19
Abbildung 4.6 Aggregatsliste als Basis zur Softwareentwicklung	20
Abbildung 4.7 Traceability.....	21
Abbildung 5.1 Generierbare Anlagenteile.....	23
Abbildung 5.2 Überführung der Symbole	24
Abbildung 5.3 Automatische Ableitung von Summenmeldungen aus bestehenden Strukturen.....	25
Abbildung 5.4 Verschaltung der Betriebsarten und Automatikschnittstelle auf Aggregatsbausteine	25
Abbildung 6.1 Schematischer Aufbau und konkrete Implementierung des allgemeinen Teils.....	28
Abbildung 6.2 Bausteinbezogene Variablen (BausteinEAs)	28
Abbildung 6.3 Typstruktur der SPS-Typen	29
Abbildung 6.4 Mehrfachverschachtelung von Arrays und Structs.....	29
Abbildung 6.5 Schematischer Aufbau und konkrete Implementierung von Typen.....	30
Abbildung 6.6 Schematischer Aufbau und konkrete Implementierung von Datenbausteinen.....	30
Abbildung 6.7 Aufbau Organisationsbaustein OB1	31
Abbildung 6.8 Aufbau Funktionsbaustein	32
Abbildung 6.9 Funktionsbaustein mit zugehörigem Datenbaustein	33
Abbildung 6.10 Komponenten des Systems.....	34
Abbildung 6.11 Funktionsprinzip des Softwareprodukts	34
Abbildung 6.12 Projektübersicht.....	36
Abbildung 6.13 Grafische Darstellung des Einleseprozess.....	37
Abbildung 6.14 Typstruktur und Aufrufstruktur von Blöcken	37
Abbildung 6.15 Festlegung der Kategorie und Abbildung als XML	38
Abbildung 6.16 Hinzufügen neuer Betriebsmittel zu einer Anlage.....	39
Abbildung 6.17 Modellierung des Generierungsvorgangs.....	40
Abbildung 6.18 Erzeugung des Zielcodes	41