**Murwan Yousif Merghani Mohamed**

# Entwicklung eines objektorientierten SPS-Steuerungsprogramms mit Controller Development System (CoDeSys)

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Automatisierungstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Ulfert Meiners
Zweitgutachter: Prof. Dr. rer. nat. Henning Hasemann

Abgegeben am 23. September 2010

**Murwan Yousif Merghani Mohamed**

## Thema der Diplomarbeit

Entwicklung eines objektorientierten SPS-Steuerungsprogramms mit Controller Development System (CoDeSys)

## Stichworte

SPS-Steuerung, IEC 61131-3, CoDeSys, Strukturierter Text „ST“, Ablaufsprache „AS“, Objektorientierte Programmierung, Klassen, Objekte, Methoden, Instantiierung, Vererbung, Serielle Schnittstelle „RS-232“, PROFIBUS-DP, Triangulations-LASER

## Kurzzusammenfassung

Übereinstimmend mit den Anforderungen der Norm IEC 61131-3 wird in dieser Diplomarbeit anhand des Entwicklungssystems für Automatisierungssysteme „CoDeSys V3“ ein SPS-Steuerungsprogramm mit dem objektorientierten Ansatz für eine Sortieranlage entwickelt.

Zusätzlich wird eine einfache CoDeSys-Visualisierung zur Darstellung des Sortierprozesses der Sortieranlage integriert.

**Murwan Yousif Merghani Mohamed**

## Title of the paper

Development of an object-oriented PLC programm with Controller Development System (CoDeSys)

## Keywords

PLC-Controller, IEC 61131-3, CoDeSys, Structured Text “ST”, Sequential Function Chart “SFC”, Object-oriented programming, Classes, Objects, Methods, Instantiation, Inheritance, Serial port “RS-232”, PROFIBUS-DP, Triangulation-LASER

## Abstract

An object-oriented PLC-Controller-Programm for a sorting plant using the Controller Development System “CoDeSys V3” is developed in conformity with the requirements of the IEC 61131-3 standard.

Additionally a simple CoDeSys-Visualization is integrated for the display of the sorting process.

*Meiner Besten,*
*meiner lieben Familie,*
*den lieben Freunden*
*und Verwandten...*

# *Danksagung*

*An dieser Stelle möchte ich mich zunächst bei Herrn Prof. Dr.-Ing. Ulfert Meiners für das interessante Thema der Diplomarbeit bedanken. Für seine fachliche und kompetente Betreuung und seine zahlreichen wertvollen Hinweise bedanke ich mich ebenso. Mein Dank gilt auch Herrn Dipl.-Ing. Ulrich Krebbel für seine ständige Hilfsbereitschaft.*

*Des Weiteren bedanke ich mich sehr bei allen Korrekturlesern meiner Diplomarbeit, besonders:*

*Herrn Harro Wolter-Strindberg*

*Frau Sofie Olbers*

*Frau Lene Strindberg*

*Die vorliegende Diplomarbeit wäre mir ohne die Unterstützung meiner lieben Familien im Sudan und in Deutschland nicht gelungen. Vielen Dank!*

*Zu guter Letzt möchte ich mich bei meiner Besten, Sophie Strindberg, die mir stets mit Rat und Tat zur Seite stand, für ihre tatkräftige Unterstützung während meines gesamten Studiums an der Hochschule für Angewandte Wissenschaften Hamburg herzlichst bedanken.*

*Murwan Y. M. Mohamed*

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| AA | Analoger Ausgang |
| AE | Analoger Eingang |
| AI | Analog Input |
| AO | Analog Output |
| AS | Ablaufsprache |
| AWL | Anweisungsliste |
| CAN-Bus | Controller Area Network-Bus |
| CCD | Charge Coupled Device |
| CFC | Continuous Function Chart |
| CNC | Computerized Numerical Control |
| CoDeSys | Controller Development System |
| CPU | Central Processing Unit |
| CR | Carriage Return |
| DGND | Datenbezugspotential |
| DI | Digital Input |
| DIN | Deutsches Institut für Normung |
| DNC | Distributed Numerical Control |
| DO | Digital Output |
| E/A | Eingang/Ausgang |
| EIA | Electronic Industries Alliance |
| EN | Europäische Norm |
| EW | Eingangswort |
| FB | Funktionsbaustein |
| FBD | Function Block Diagram |
| FC | Funktion |
| FIFO | First In First Out |
| FS | Full Scale |
| FUP | Funktionsplan-Diagramm |
| GND | Ground |
| GRAFCET | Graphe Fonctionnel de Commande Etapes/Transitions |

| | |
|---|---|
| GVL | Globale Variablenliste |
| HMI | Human Machine Interface |
| I/O | Input/Output |
| IEC | International Electrotechnical Commission |
| IP | International Protection |
| ISO | International Standard Organisation |
| KOP | Kontaktplan |
| LF | Line Feed |
| LSB | Least Significant Bit |
| MMS | Mensch-Maschine-Schnittstelle |
| MSB | Most Significant Bit |
| OSI | Open Systems Interconnection |
| PAC | Programmable Automation Controller |
| PLC | Programmable Logic Controller |
| POE | Programmorganisationseinheit |
| POU | Program Organisation Unit |
| PRG | Programm |
| PROFIBUS-DP | Process Field Bus-Dezentrale Peripherie |
| RS | Recommended Standard |
| RxD | Receive Data |
| SFC | Sequential Function Chart |
| ST | Strukturierter Text |
| TR | Technischer Report |
| TxD | Transmit Data |
| UART | Universal Asynchronous Receiver Transmitter |
| UDT | User Defined Type |
| UML | Unified Modeling Language |
| USART | Universal Synchronous/Asynchronous Receiver Transmitter |
| USB | Universal Serial Bus |
| VP | Versorgungsspannungs-Pluspol |

# Abbildungsverzeichnis

# Tabellenverzeichnis

# 1 Einleitung

## 1.1 Motivation

Der Einsatz von speicherprogrammierbaren Steuerungen (SPSen) nimmt eine zentrale Position innerhalb moderner Automatisierungssysteme ein. Der heutige Markt wird von den unterschiedlichen SPS-Programmiersystemen der verschiedenen Hersteller geprägt.

Aus diesen Gründen erfordert die steigende Komplexität der SPS-Programme mehr denn je eine Standardisierung. Deshalb wurde unter Schirmherrschaft der *International Electrotechnical Commission (IEC)* der Standard IEC 61131-3 als drittes Teil der IEC 61131 Familie erarbeitet. Dieser internationale Standard IEC 61131 besteht aus folgenden sieben Teilen:

**Teil 1: Allgemeine Informationen (*General Informations*):** legt die in der Norm verwendeten Begriffe fest.

**Teil 2: Ausrüstung und Testanforderungen (*Equipment and Test Requirements*):** legt die SPS-Struktur und die physikalische Anforderungen fest wie Temperaturbereiche, Umgebungsbedingungen, Impedanzwerte für Analogeingänge etc.

**Teil 3: Programmiersprachen (*Programming Languages*):** legt die SPS-Programmiersprachen und zusätzlich die grafische Ablaufsprache (`AS`) fest.

**Teil 4: Anwenderrichtlinien (*User Guidelines*):** beschreibt die Anforderungen an die Dokumentation von Hard- und Software.

**Teil 5: Kommunikation (*Messaging Services*):** beschreibt einheitliche Bausteine für den Aufbau der Datenkommunikation zu beliebigen internen und externen Kommunikationsobjekten.

**Teil 7: Fuzzy-Control-Programmierung (*Fuzzy Control Programming*):** gibt dem Hersteller und dem Anwender ein gemeinsames Verständnis zur Integration von Fuzzy[1]-kontrollierten Anwendungen auf Basis der IEC 61131-3.

**Teil 8: Leitlinien für die Anwendung und Implementierung von Programmiersprachen für SPSen**

[1] Fuzzylogik (verschwommene Logik) ist eine Theorie, welche vor allem für die Modellierung von Unsicherheiten und Unschärfen von umgangssprachlichen Beschreibungen entwickelt wurde. Sie ist eine Verallgemeinerung der zweiwertigen Booleschen Logik. (Wikipedia, 2010, Fuzzylogik, Zugriff: 18.08.2010)

Aktuell arbeitet das Normungskomitee an der Erstellung der IEC 61131-6 (**Teil 6: Sicherheitsgerichtete SPS**). Ziel davon ist die Anforderung der Sicherheitsnorm IEC 61508 und der Maschinenanforderung IEC 62061 an speicherprogrammierbaren Steuerungen aufzubereiten. (John und Tiegelkamp, 2009)

Zur Durchsetzung der Norm wurde im Jahre 1992 die *Organisation PLCopen* gegründet. Sie ist eine internationale, hersteller- und produktunabhängige Vereinigung von SPS- und Software-Herstellern und Instituten. Ihr Ziel ist die Entwicklung und den Einsatz kompatibler Software für *PLCs* zu fördern. PLC steht für ***P**rogrammable **L**ogic **C**ontrol* und ist die weltweite Bezeichnung des im Deutschen gebräuchlichen Begriffes SPS. (Becker, 2007)

Die IEC-Norm vereint in sich die Erfahrungen, die auf dem Gebiet der Programmierung der speicherprogrammierbaren Steuerungen in den letzten 30 Jahren gemacht und zum Teil in nationalen Gremien standardisiert worden sind.

Seit der Verfügbarkeit der ersten Entwürfe des Standards werden von verschiedenen Herstellern der SPS-Programmiersysteme vielfältige Anstrengungen unternommen, um für ihre Produkte Entwicklungsumgebungen entsprechend der IEC 61131-3-Norm zu schaffen. Weit über 200 namhafte Gerätehersteller aus den unterschiedlichsten Branchen setzen das Softwaretool *CoDeSys* als Programmierschnittstelle für ihre Automatisierungsgeräte ein. Tausende Endanwendern aus dem Maschinen- und Anlagenbau sowie weiteren Industriezweigen nutzen das Programmiersystem *CoDeSys* bei ihrer täglichen Arbeit. Das sind mehr als bei jedem anderen IEC 61131-3 Entwicklungssystem in Europa. Damit ist *CoDeSys* faktisch Marktstandard. (3S, 2009)

In der Automatisierungstechnik sind SPSen die am häufigsten eingesetzten Automatisierungsgeräte. Wesentliche Kennzeichen speicherprogrammierbarer Steuerungen sind die zyklische Abarbeitung der Programme, die geeigneten Programmiersprachen der Norm IEC 61131-3 sowie die gute Anbindung der Sensoren und Aktoren über Ein- und Ausgangsbaugruppen oder einen Feldbus.

## 1.2 Aufgabenstellung

In der allgemeinen Softwareentwicklung ist die Objektorientierung eine inzwischen bewährte Methodik, um die steigende Komplexität der Software zu beherrschen. Der Objektorientierung liegt eine Aufteilung der zu beschreibenden Welt in Objekte mit ihren Eigenschaften und Operationen zugrunde. Dies wird durch das Konzept der Klasse, bei dem Objekte aufgrund ähnlicher Eigenschaften zusammengefasst werden, ergänzt. Die Struktur eines Objekts wird durch die Attribute (Eigenschaften) seiner Klassendefinition festgelegt. Das Verhalten des Objekts wird von den Methoden der Klasse bestimmt.

In der Automatisierungstechnik zeichnet sich der Trend ab, dass auch hier aufgrund steigender Kundenanforderungen die Komplexität der Programme zunimmt. Objektorientierung

könnte hier helfen, die Komplexität zu beherrschen, die Lesbarkeit zu erhöhen und die Wartbarkeit zu verbessern.

Im Rahmen dieser Diplomarbeit soll eine im Labor der Automatisierungstechnik an der Hochschule für Angewandte Wissenschaften Hamburg vorhandene Modellanlage objektorientiert automatisiert und visualisiert werden. Dazu ist das Softwaretool *CoDeSys V3* einzusetzen, welches eine objektorientierte Softwareerstellung ermöglicht. Die Anbindung der Modellanlage an den Automatisierungsrechner, der aus einem PC mit einer Realtime-Soft-SPS besteht, erfolgt über PROFIBUS-DP und die serielle Schnittstelle. Nach der Automatisierung der Modellanlage sind die Vor- und Nachteile des objektorientierten Ansatzes bei Einsatz des Werkzeugs *CoDeSys* herauszuarbeiten. Zusätzlich soll eine Bewertung des Werkzeugs *CoDeSys* im Vergleich zu dem bisher in der Ausbildung eingesetzten Tool *Step 7* der Firma *Siemens* abgegeben werden. Die Bewertung ist beispielsweise bezogen auf die Kriterien Erlernbarkeit, Bedienbarkeit, Leistungsfähigkeit, Stabilität etc.

## 1.3 Gliederung der Diplomarbeit

Die vorliegende Diplomarbeit hat zum Ziel, eine verständliche Einführung in das IEC 61131-Norm konforme SPS-Programmiersystem *CoDeSys V3* sowie die damit ermöglichten Objektorientierungseigenschaften zur SPS-Steuerungsprogrammierung zu geben. Dabei wird das Projekt zur Automatisierung der zur Verfügung gestellten Sortieranlage verwendet, um den objektorientierten Ansatz näher zu bringen.

Diese Arbeit besteht insgesamt aus zehn Kapiteln, die in drei große Teile gegliedert sind:

- Technische Grundlagen (Kapitel 1 – 3)
- Praktische Umsetzung (Kapitel 4 – 9)
- Schluss (Kapitel 10)

**Kapitel 2** stellt die SPS-Programmiernorm IEC 61131-3 vor, sowie eine kurze Historie über deren Entwicklung. Anschließend wird das Programmiersystem *CoDeSys* mit seinen Eigenschaften und Editoren vorgestellt. Darüber hinaus wird gezeigt, wie ein einfaches Beispiel-Projekt in *CoDeSys V3* angelegt, auf die Steuerung geladen und gestartet wird.

Im **Kapitel 3** wird auf die Prinzipien der objektorientierten Programmierung eingegangen. Es werden dort einige Grundbegriffe aus der Welt des objektorientierten Programmierens erläutert, die für die Realisierung des Steuerprogramms von erheblicher Bedeutung sind.

Nach den einleitenden technischen Grundlagen wird im **Kapitel 4** die Modellanlage mit ihren einzelnen Komponenten vorgestellt. Dort wird u. a. das Prinzip der analogen Messwertverarbeitung in der Analogbaugruppe der SPS geschildert.

In dem **Kapitel 5** werden die wesentlichen Kommunikationsbeziehungen zum Informationsaustausch zwischen den Anlagekomponenten aufgeführt.

Anschließend wird im **Kapitel 6** Leitfäden für den objektorientierten Softwareentwurf und zur strukturierten Vorgehensweise bei der Programmierung vorgestellt.

**Kapitel 7** ist der Realisierung der identifizierten Anlagenmodule gewidmet. Hier werden die einzeln hergeleiteten Funktionsbausteine sowie deren Methoden und Aktionen ausführlich behandelt.

Im **Kapitel 8** werden die Adressierungsarten in *CoDeSys* aufgezählt und anhand von Beispielen erläutert.

Nach der Implementierung der Module wird im **Kapitel 9** die Visualisierungsmöglichkeit mittels *CoDeSys*-HMI besprochen.

Das abschließende **Kapitel 10** enthält einen Rückblick über die Umsetzung der Steuerungssoftware sowie eine Gegenüberstellung der Programmiersysteme *CoDeSys* und *Step 7*.

An dieser Stelle ist anzumerken, dass die Funktionsbausteine, Methoden, IEC-Aktionen etc. in Anlehnung an das Applikationsbeispiel *Sortieranlage* aus dem 2009 veröffentlichten Buch „Modulares Engineering und Wiederverwendung mit CoDeSys V3“ von Vogel-Heuser und Wannagat zur Automatisierung der vorhandenen Sortieranlage entwickelt worden sind.

# 2 SPS-Programmierung nach IEC 61131-3

Die mehrteilige Norm IEC 61131 umfasst die Anforderungsdefinitionen für SPS-Systeme. Sie ist ein internationaler Standard für Programmiersprachen von speicherprogrammierbaren Steuerungen. Die Anforderungen der Norm betreffen sowohl die SPS-Hardware als auch das Programmiersystem.

## 2.1 Historische Entwicklung der Steuerungstechnik

Die Entwicklung speicherprogrammierbarer Steuerungen reicht bis in die 60er Jahre des 20. Jahrhunderts zurück, als noch verbindungsprogrammierte Relais-Schaltungen die Anlagen steuerten. Hierbei ist das Steuerungsprogramm durch feste Draht- oder Leiterplattenverbindungen aufgebaut und dementsprechend unflexibel. In den 1970er Jahren wurden dann speicherprogrammierbare Logik-Steuerungen entwickelt, die mit einer maschinennahen Programmiersprache, ähnlich *Assembler*, programmiert wurden. Der Vorteil ist, dass das Programm im Speicher der SPS hinterlegt ist und somit jederzeit verändert werden kann. (Seitz, 2008)

Im *Kontaktplan* wurden die ersten speicherprogrammierbaren Steuerungen programmiert, welche die bis dahin durch Verdrahtung von Relaiskontakten realisierten Steuerungen ablösten. Damit bekamen die Steuerungstechniker die Möglichkeit, weiterhin in Stromablaufplänen zu denken und diese am Rechner einzugeben. Die Hersteller von SPSen entwickelten danach die Programmiersprachen *Anweisungsliste* und *Funktionsplan*. Diese beiden Sprachen arbeiteten mit direkten Ein- und Ausgängen sowie mit Merkern und waren ausschließlich für die Verarbeitung von booleschen Signalen konzipiert. Diese Programmiersprachen unterschieden sich von Hersteller zu Hersteller zum Teil beträchtlich voneinander und waren sehr stark maschinenorientiert.

Bereits im Jahr 1970 unternahm die *National Electrical Manufacturers Association* in den USA erste ernsthafte Versuche zur Vereinheitlichung der Programmierung von Industriesteuerungen. Ab dem Jahr 1977 wurde mit GRAFCET[1] in Frankreich und den DIN[2]

[1] **Gra**phe **F**onctionnel de **C**ommande **E**tapes/**T**ransitions oder EN 60848 ist eine Spezifikationssprache für die Darstellung von Ablaufbeschreibungen. (Wikipedia, 2010, GRAFCET, Zugriff: 18.07.2010)

[2] **D**eutsches **I**nstitut für **N**ormung

19239 und 40719 Teil 6 in Deutschland eine Vereinheitlichung der Programmierung von Verknüpfungs- und Ablaufsteuerungen beabsichtigt. Seit 1983 hat die IEC Arbeitsgruppe (WG6) verschiedene Versuche zu einer Vereinheitlichung der SPS-Programmierung unternommen, die dann schließlich 1992 in den internationalen Standard IEC 1131 eingeflossen sind.

Dieser Standard traf auf breite Zustimmung. Die Unzulänglichkeiten, die in der ersten Fassung noch enthalten waren, wurden bei einer Neufassung, die im Jahre 2003 erschienen ist, behoben. Nachdem 1996 eine neue Nummerierung der IEC-Standards vorgenommen wurde, heißt diese Norm heute IEC 61131 oder als europäische- bzw. deutsche Norm DIN EN 61131. (Lepers, 2005)

## 2.2 Programmiersystem CoDeSys

*CoDeSys*[3] ist ein geräteunabhängiges Steuerungsprogrammiersystem der Softwarehersteller *3S – Smart Software Solutions*. *CoDeSys* ist eine Entwicklungsumgebung für speicherprogrammierbare Steuerungen (SPS) nach dem IEC 61131-3 Standard für die Applikationsentwicklung in der Industrieautomation.

Das Programmiersystem *CoDeSys* hat viele Fähigkeiten. Es erlaubt u. a. das Einbinden von *C*-Routinen und unterstützt die objektorientierte Programmierung. Zusammen mit dem *CoDeSys-SP-Laufzeitsystem* erlaubt es auch die *Multi-Device*- und *Multi-Application*-Programmierung. Unter dem Begriff Laufzeitprogramm versteht man ein Gesamtprogramm mit Laufzeit-Eigenschaften, bestehend aus sämtlichen benötigten *POU*s[4] bzw. *POE*s[5] und allen *Tasks*. Mit dieser Definition ist ein Laufzeitprogramm eine in sich geschlossene Programmeinheit, die selbstständig in einer CPU ablaufen kann. Die komponentenbasierte Struktur der Programme in *CoDeSys* macht eine kundenspezifische Konfiguration und Erweiterung der Benutzeroberfläche möglich. Über die Grenzen von Ein-Prozessor-Programmen hinaus ermöglicht ein Softwaremodell in *CoDeSys* nach der IEC 61131-3-Norm das *Multi-Tasking* einer CPU[6], die Zusammenarbeit mehrerer CPUs oder die Zusammenarbeit vernetzter Systeme. (3S, 2009; Becker, 2007)

Ein Steuerungsprogramm besteht nach dem IEC 61131-Standard aus folgenden Elementen:

- **Strukturen:** Aufzählungen (`ENUM`[7]), Benutzerdefinierte Datentypen (`UDT`[8]).

[3] **Co**ntroller **De**velopment **Sys**tem (Entwicklungssystem für Automatisierungssysteme)
[4] **P**rogram **O**rganisation **U**nit
[5] **P**rogramm**o**rganisations**e**inheit
[6] **C**entral **P**rocessing **U**nit
[7] **Enum**arationen
[8] **U**ser **D**efined **T**ype

- **Bausteine:** Programme (PRG), Funktionen (FC), Funktionsbausteine (FB).
- **Globale Variablen**.

Alle fünf der von der IEC 61131-3 spezifizierten Programmiersprachen, die in *CoDeSys* realisiert sind, sind konform zu den Anforderungen der Norm und stehen in *CoDeSys* zur Verfügung. Die textuellen Sprachen:

- **Anweisungsliste** (AWL),
- **Strukturierter Text** (ST)

und die grafischen Sprachen:

- **Ablaufsprache** (AS),
- **Kontaktplan** (KOP),
- **Funktionsplan** (FBS) sowie **Continuous Function Chart** (CFC),

wobei CFC nicht Teil der IEC 61131-3 ist. Die Benennung der Programmiersprachen ist in den verschiedenen Systemen und im deutsch- und englischsprachigen Raum unterschiedlich (Tab. 2.1).

| Deutsch | Englisch | Siemens |
|---|---|---|
| Kontaktplan (KOP) | Ladder Diagram (LD) | Kontaktplan (KOP) |
| Funktionsbausteinsprache (FBS) | Function Block Diagram (FBD) | Funktionsplan (FUP) |
| Anweisungsliste (AWL) | Instruction List (IL) | Anweisungsliste (AWL) |
| Strukturierter Text (ST) | Structured Text (ST) | Strukturierter Text (ST) |
| Ablaufsprache (AS) | Sequential Function Chart (SFC) | Ablaufsprache (AS) |

Tabelle 2.1: Vergleich der Benennung der Programmiersprachen

Ein wesentliches Leistungsmerkmal der IEC 61131-3-Sprachfamilie ist die Umschaltbarkeit zwischen den Programmiersprachen. (Vogel-Heuser und Wannagat, 2009)

### 2.2.1 Erste Schritte

Im Folgenden wird beschrieben, wie ein einfaches Projekt, das ein Steuerungsprogramm enthält, in *CoDeSys V3* angelegt, erstellt, über den Gateway-Server auf das Zielsystem (Steuerung) geladen und dort gestartet und überwacht werden kann.

Um ein neues Projekt in *CoDeSys V3* anzulegen, wählt man den Befehl **Neues Projekt** im **Datei**-Menü (Abb. 2.1).

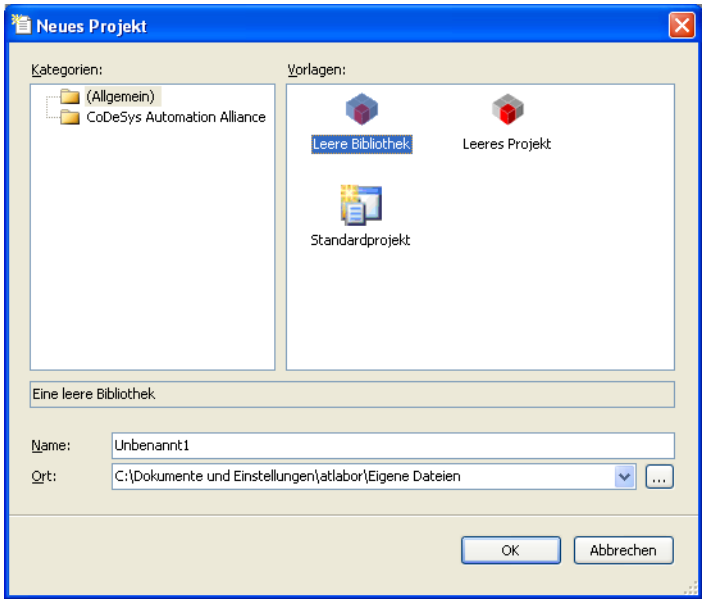



Abbildung 2.1: Neues Projekt anlegen

Im Dialog **Neues Projekt** wählt man im Feld **Vorlagen** die Vorlage **Standardprojekt**. Nach Eingabe des Projektnamens und dessen Speicherorts für die Projektdatei und anschließender Bestätigung der Eingabe durch **OK** erscheint ein Dialog-Fenster zur Auswahl des programmierbaren Gerätes (Abb. 2.2).

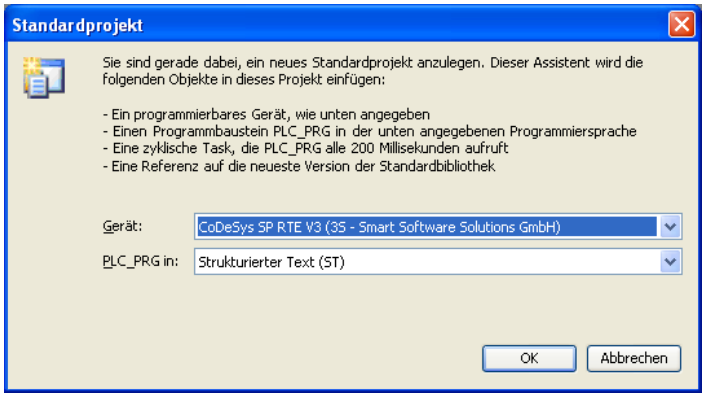



Abbildung 2.2: Geräteauswahl

Nach Auswahl des programmierbaren Gerätes und der Programmiersprache (hier Strukturierter Text `ST`) des Programmbausteins `PLC_PRG` erscheint die *CoDeSys*-Benutzeroberfläche (Abb. 2.3) mit den zugehörigen POUs- und Geräte-Fenstern.

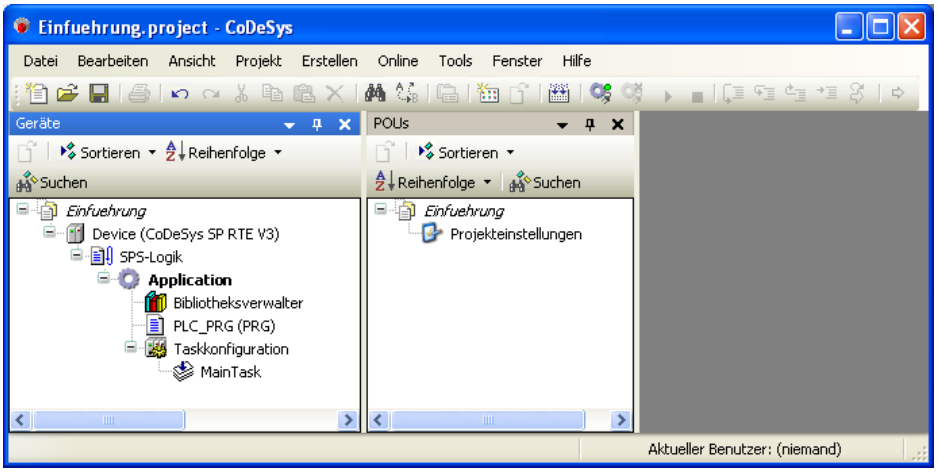



Abbildung 2.3: CoDeSys-Benutzeroberfläche

Das *POUs-Fenster* enthält die `Projekteinstellungen`. Mit POU bzw. POE werden Programme, Funktionen und Funktionsbausteine bezeichnet, die das Anwenderprogramm, das zur Automatisierung einer Anlage dient, zusammenstellen.
Das *Geräte-Fenster* weist eine Baumstruktur auf, welche das Gerät `Device (CoDeSys SP RTE V3)` vom Typ `CoDeSys SP RTE V3` mit einer unterhalb eingefügten `Application` zeigt. Letztere beinhaltet das Programm `PLC_PRG` und die obligatorische `Taskkonfiguration` mit einer `MainTask` zur Steuerung von `PLC_PRG`.

Die Abarbeitung eines *CoDeSys*-Programms beginnt zwingend mit dem speziellen Baustein `PLC_PRG`, welcher andere Bausteine aufrufen kann. Weiterhin ist bereits ein `Bibliotheksverwalter` eingehängt, der automatisch die Bibliothek `IoStandard.library` enthält, die für E/A-Konfigurationen benötigt wird, sowie die `Standard.library`, die alle Funktionen und Funktionsbausteine bereitstellt, die gemäß der Norm IEC 61131-3 als Standardbausteine für ein IEC-Programmiersystem benötigt werden. Der zusätzliche Knoten `SPS-Logik` unterhalb des Knotens `Device (CoDeSys SP RTE V3)` ist nur ein symbolischer Knoten, der anzeigt, dass das Gerät „programmierbar“ ist.

### 2.2.2 Steuerungsprogramm schreiben

Durch einen Doppelklick auf den Eintrag `PLC_PRG` im Geräte-Fenster öffnet sich das `ST`-Editorfenster (Abb. 2.4). Der Editor besteht aus einem oberen *Deklarationsteil* und einem unteren *Implementierungsteil*. Im Deklarationsteil werden zwischen den Schlüsselwörtern `VAR` und `END_VAR` die Variablen deklariert, d. h. ihre Namen, Datentypen und Initialwerte. Für Ein- und Ausgabevariablen wird u. a. von den Schlüsselwörtern `VAR_INPUT` für die

Formalparameter, VAR_OUTPUT für Rückgabewerte und VAR_IN_OUT für les- und beschreibbare Variablen Gebrauch gemacht. Der Implementierungsteil beinhaltet den in ST geschriebenen Programmcode.

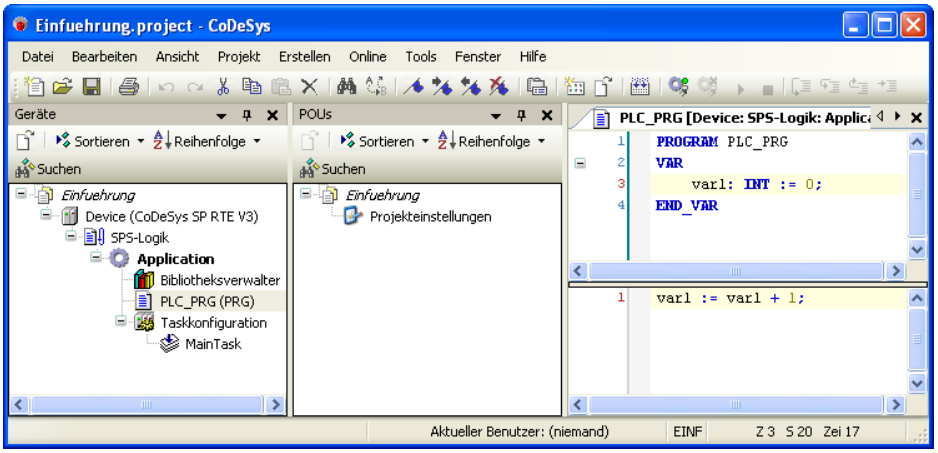



Abbildung 2.4: ST-Editor

Alternativ kann die **Autodeklarations**-Funktion verwendet werden, um während des Programmierens Variable zu deklarieren. Für jede noch nicht deklarierte Variable der aktuellen Programmzeile wird automatisch der Dialog **Variable deklarieren** geöffnet, wo die Deklaration vorgenommen werden kann (Abb. 2.5).

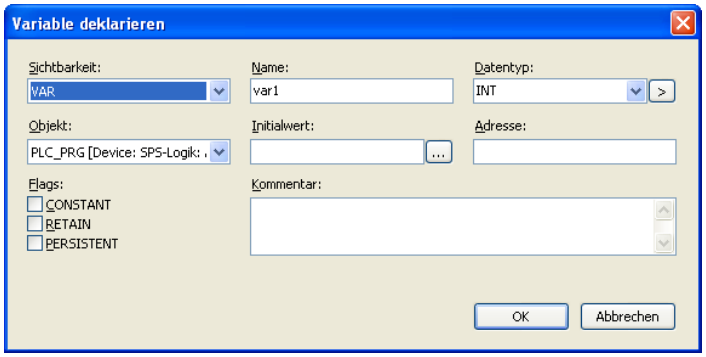



Abbildung 2.5: Autodeklarations-Funktion

Um weitere POUs anzulegen, wählt man den Befehl **Objekt hinzufügen** im **Projekt**-Menü (Abb. 2.6). Zusätzlich stehen im geöffneten Dialog-Fenster alle in *CoDeSys V3* einfügbaren Objekte mit ihren Eigenschaften aufgelistet zur Verfügung.

Nach Angabe des POU-Namens, des Typs und der Implementationssprache werden mit dem Bestätigen der Schaltfläche **Öffnen** die Objekteinstellungen vorgenommen und es öffnet sich automatisch das Editor-Fenster des eingefügten Objektes, in welchem sich dieses bearbeiten bzw. programmieren lässt.

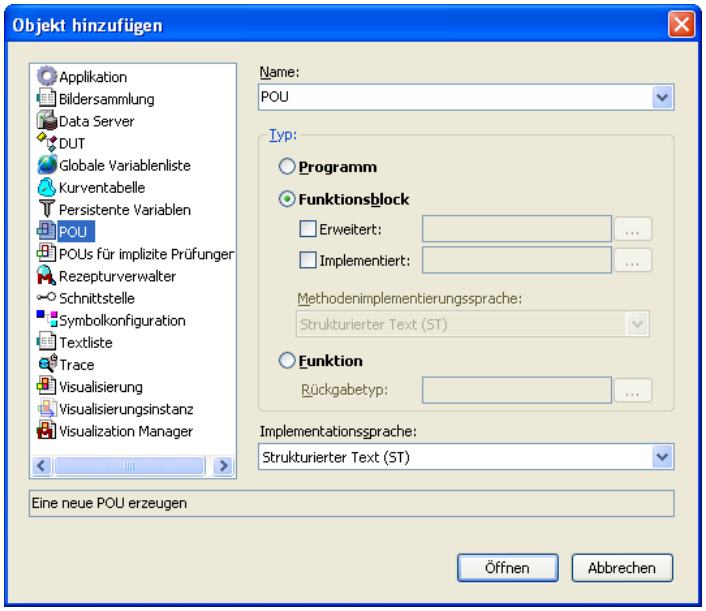



Abbildung 2.6: Objekt hinzufügen

Das Zielsystem `CoDeSys SP RTE V3` wird beim Systemstart automatisch als Dienst verfügbar. Bevor ein Programm auf die Steuerung geladen werden kann, muss sicher gestellt werden, ob das **CoDeSys SP RTE V3**-Symbol den Status `running` aufweist. Das Symbol befindet sich in der Systemleiste am unteren Rand des Bildschirms (Abb. 2.7).


Abbildung 2.7: CoDeSys SP RTE V3-Symbol

Um eine Applikation als „aktive“ Applikation zu setzen wählt man im Geräte-Fenster den Eintrag `Application`. Bei einem anschließenden Klick auf die rechte Maustaste erscheint ein Kontextmenü, das den Befehl **Aktive Applikation setzen** enthält. Damit bezieht sich alle Kommunikation mit der Steuerung auf diese Applikation.

### 2.2.3 Taskkonfiguration

Die Hierarchie des Aufrufs der Programmbausteine wird im Geräte-Fenster mit der Ressource `Taskkonfiguration` organisiert.

In einer SPS verläuft die Informationsverarbeitung zyklisch. Die Verarbeitungsschritte lassen sich wie folgt ordnen:

- Einlesen der Sensordaten,
- Verarbeiten der Informationen im SPS-Programm,
- Ausgeben der Stellsignale an die Aktoren.

Eine oder mehrere *Tasks* können in einer CPU ablaufen. Sie organisiert den zeitlichen Ablauf der Programme. Außerdem können einer Task mehrere Programme zugeordnet sein, die alle mit derselben Zykluszeit abgearbeitet werden. Erst nach Abarbeitung aller zu einer Task gehörenden Programme werden die Stellwerte an den Prozess ausgegeben.Trotz der schrittweisen Abarbeitung der Programme erreichen die Ausgangsdaten aller Programme erst am Ende des Bearbeitungszyklus und somit quasi gleichzeitig den Prozess. (Seitz, 2008)
Abbildung 2.8 zeigt die Konfigurationsdialog-Fenster für den Hauptprogramm-Task. Hier können Definitionen für die folgenden Parameter festgelegt werden:

- **Priorität** des Tasks,
- **Typ:** – Zyklisch mit Intervallangabe
  – Ereignisgesteuert mit Angabe der Ereignisvariable
  – Freilaufend,
- **Watchdog** zur Überwachung der Task aktivieren/deaktivieren, Auslösezeit und dessen Empfindlichkeit,
- Aufzurufende **POUs**.

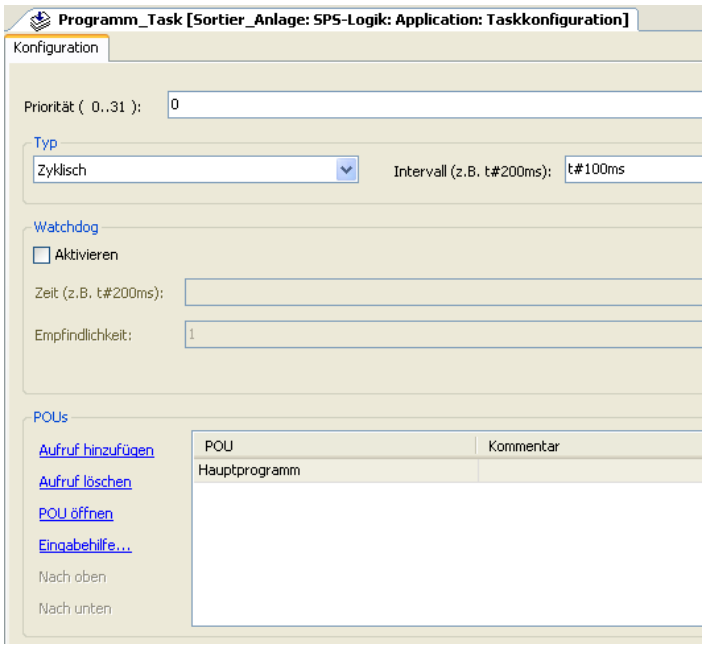



Abbildung 2.8: Taskkonfigurationsdialog

### 2.2.4 Bibliotheken

Bibliotheken werden benutzt, um Bausteine, die in verschiedenen Applikationen zum Einsatz kommen, zu verwalten. Typische Funktionsbausteine wie Zähler (*Counter*), Zeitgeber (*Timer*), bistabile Elemente (*RS-Flip-Flop*), Flankenauswertung (*Trigger*) etc. stehen als Standard-FBs in der Bibliothek `Standard.library` zur Verfügung.
Um den Quellcode der einzelnen Bausteine vor dem Einblick Unbefugter zu verbergen, besteht die Möglichkeit, die vorkompilierten Bausteine in einer Bibliothek abzulegen.

Für weitere Informationen über die Verwaltung, Einbindung und Bereitstellung von Bibliotheken wird auf die *CoDeSys*-Online-Hilfe verwiesen.

### 2.2.5 Starten einer Applikation

Um lediglich eine syntaktische Prüfung der „aktiven“ Applikation durchzuführen, verwendet man den Befehl **Application [· · ·] übersetzen** aus dem Kontextmenü oder dem **Übersetzen**-Menü. In diesem Fall wird noch kein Code generiert. Die Ergebnisse der Prüfung werden im *Meldungsfenster* ausgegeben, das standardmäßig im unteren Drittel der Benutzeroberfläche platziert ist. Bei fehlerfreier Übersetzung des Quellcodes kann die Software dann auf die Steuerung geladen werden und anschließend in die Steuerung eingeloggt werden.

Wählt man den Befehl **Start Application** im **Online**-Menü, so läuft das Programm. Die Einträge für Steuerung und Applikation im Geräte-Fenster werden grün hinterlegt und hinter `Application` steht der Status `run` (Abb. 2.9).

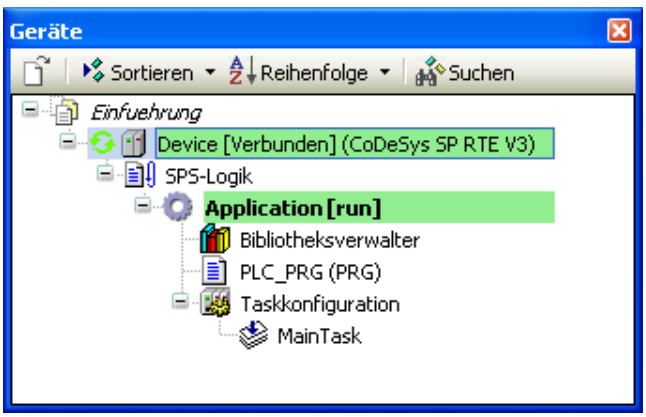



Abbildung 2.9: Application-Status RUN

#### Online-Funktionen

Nach dem Einloggen in die Steuerung stehen in *CoDeSys* Befehle für Online-Funktionen zur Verfügung. Sie sind im **Online**-Menü aufgelistet. Die Ausführung einiger Online-Funktionen ist abhängig vom aktiven Editor.

Eines der wichtigsten Online-Befehle ist der **Reset**-Befehl, der ein Rücksetzen der Steuerung bewirkt. Es wird zwischen drei Reset-Arten unterschieden:

**Reset (warm):** Dieser Befehl setzt mit Ausnahme der remanenten[9] Variablen alle Variablen auf ihren Initialisierungswert zurück. Variablen, die nicht explizit mit einem Initialisierungswert versehen wurden, werden auf die Standardinitialwerte gesetzt. Integer-Zahlen werden beispielsweise auf 0 gesetzt. Bevor alle Variablen überschrieben werden, erfolgt eine Sicherheitsabfrage durch *CoDeSys*. Dies entspricht der Situation bei einem Stromausfall oder beim Aus- bzw. Einschalten der Steuerung während des laufenden Programms.

**Reset (kalt):** Dieser Befehl setzt mit Ausnahme der persistenten[10] Variablen und Retain-Variablen alle Variablen auf den Initialisierungswert zurück. Dies entspricht einem Neuladen des Steuerungsprogramm auf die Steuerung.

**Reset (Ursprung):** Dieser Befehl setzt alle Variablen, auch die remanenten Variablen, auf ihren Initialisierungswert zurück und löscht das Anwenderprogramm auf der Steuerung, d. h. die Steuerung wird in den Urzustand zurückversetzt. (3S, 2009)

### 2.2.6 Fehlersuche und -behebung

Die Fehlersuche wird mit den **Debugging**-Funktionen von *CoDeSys* erleichtert. Sogenannte Haltepunkte (engl. *Breakpoints*) können beispielsweise im Falle eines Programmierfehlers gesetzt werden. Stoppt die Ausführung des Programms in einem solchen Haltepunkt, so können die Werte sämtlicher Projektvariablen zu diesem Zeitpunkt eingesehen werden. Durch schrittweises Abarbeiten (Einzelschritt) kann die logische Korrektheit des Programms überprüft werden. (3S, 2009)

Für weitere Einzelheiten der Debugging-Funktionen wird auf die *CoDeSys*-Online-Hilfe verwiesen.

[9] Remanente Variablen können ihren Wert über die übliche Programmlaufzeit hinaus behalten. Sie werden als *Retain*-Variablen oder noch strenger als *Persistente* Variablen deklariert.

[10] Persistente Variablen werden mit dem Schlüsselwort `PERSISTENT` gekennzeichnet. Sie werden nur bei **Reset (Ursprung)** der Steuerung neu initialisiert.

### 2.2.7 Online-Hilfe

Eine Online-Hilfe wird über das **Hilfe**-Menü angeboten. Dieses enthält Befehle zum Öffnen des Inhaltsverzeichnisses, des Indexverzeichnisses zum Suchen nach Indexbegriffen oder eines Suchen-Dialogs für eine Volltextsuche.

# 3 Objektorientierte Programmierung

> „*[Eine] Philosophie des Programmierens, die von einer Welt ausgeht, die aus gleichberechtigten und einheitlich erscheinenden Objekten besteht.*“ (Engesser u. a., 1993)

Die objektorientierte Programmierung ist, wie bereits erwähnt, eine mittlerweile bewährte Methodik, um die steigende Komplexität von Softwaresystemen in den Griff zu bekommen. Objektorientierung ist also im Allgemeinen eine spezielle Herangehensweise an die Zusammenhänge, Probleme und Lösungsmöglichkeiten im Bereich der Informationswissenschaften.

Aus diesen Gründen werden in diesem Kapitel die gängigsten Begriffe der objektorientierten Programmierung, die vor allem für die Umsetzung in das SPS-Programmiersystem *CoDeSys* von Relevanz sind, erläutert.

Die objektorientierte Programmierung baut nicht auf Prozeduren und Daten auf, sondern auf Zuständen, Aktivitäten und Kommunikation:

- Es gibt nur Objekte, die sich nach außen alle einheitlich verhalten und gleichberechtigt sind.
- Objekte haben bestimmte Zustände, führen Operationen aus und können Informationen sowohl empfangen als auch verschicken.
- Um Objektfähigkeiten zu nutzen, schicken sich Objekte mit Hilfe ihrer Methoden gegenseitig Nachrichten zur Erledigung bestimmter Aufgaben. Das angefragte Objekt liefert dann die Ergebnisse der Verarbeitung an das fragende Objekt zurück.
- Objekte erledigen ihre Aufgaben prinzipiell in eigener Verantwortung.
- Objektfähigkeiten und -zustände können an andere Objekte vererbt werden. (Engesser u. a., 1993)

In den folgenden Abschnitten werden zuerst die allgemein aus der Objektorientierungswelt bekannten Begriffe wie Klassen, Objekte, Vererbung etc. und anschließend die objektorientierte Erweiterung der Norm IEC 61131-3 aufgeführt.

## 3.1 Klassen und Objekte

Unter einer *Klasse* versteht man einen selbst definierten Datentyp, der dazu dienen kann, neue Strukturen zu modellieren. Auch als eine Sammlung von Variablen verschiedener Datentypen kann eine Klasse verstanden werden. Klassen können neben Variablen auch Methoden zur Variablenmanipulation enthalten. Die konkrete Realisierung einer Klasse wird als *Instanz* der Klasse oder als *Objekt* bezeichnet.

Klassen, von denen keine konkreten Exemplare erzeugt werden können, d. h. von denen es grundsätzlich keine Objekte geben wird, bezeichnet man als „abstrakte“ Klassen.

Die objektorientierte Analyse einer Programmieraufgabe betrachtet eine Software als ein System von kooperierenden Objekten. (Ratz u. a., 2001)

## 3.2 Methoden

Durch Methoden wird der ausführbare Code unter einem Namen zusammengefasst. Dieser Code kann unter Verwendung von Parametern so formuliert sein, dass ihm später beim Aufruf der Methode Werte übergeben werden. Methoden gehören in der Regel neben den Variablen zum festen Bestandteil von Klassen.

Zur Definition einer Methode müssen folgende Komponenten eindeutig sein:

**Name:** Der Bezeichner der Methode, unter dem die Methode erkannt wird. Der Methodenname darf kein reserviertes Schlüsselwort sein.

**Rückgabetyp:** Der Datentyp des Ergebnisses, das die Methode zurückliefern soll.

**Parameterliste:** Eine Liste von Variablendeklarationen. Die darin aufgeführten Variablen werden als „formale“ Parameter bezeichnet.

## 3.3 Instantiierung

Als Instantiierung wird in der objektorientierten Programmierung das Erzeugen eines Objekts einer bestimmten Klasse bezeichnet. Das erzeugte Objekt wird *Instanz* einer Klasse genannt. Die Instantiierung erfordert die Angabe von objektspezifischen Parametern. Sie umfasst das Anlegen eines Speicherbereichs für die Instanz sowie die Belegung von Variablen und Datenstrukturen mit Default- oder Initialwerten.

Also ist die Instanz einer Klasse ein fest umrissenes Exemplar mit konkreten Ausprägungen, mit dem bis zu deren Zerstörung gearbeitet werden kann. (Wikipedia, 2010, Instantiierung, Zugriff: 08.07.2010)

## 3.4 Interface

Schnittstellen (engl. *Interface*) beschreiben mit ihren Methoden und Attributen (Eigenschaften) einen ausgewählten Teil des extern sichtbaren Verhaltens und der extern sichtbaren Struktur von Klassen. Interfaces werden verwendet, um Methoden zu verwalten, die von Funktionsblöcken implementiert werden. Eine Schnittstelle ist also eine abstrakte Klasse, die eine Sammlung von *Methoden-Prototypen* beschreibt. Prototyp heißt, dass nur Deklarationen, aber keine Implementierungen enthalten sind.

Eine Klasse kann eine oder mehrere Schnittstellen implementieren, was bedeutet, dass grundsätzlich alle in der Schnittstelle angegebenen Methoden in der Klasse verfügbar sein müssen. Außerdem können Schnittstellen andere Schnittstellen erweitern, d. h. Vererbungen zwischen Schnittstellen sind möglich. Zusätzlich können auch eigene Methoden und Eigenschaften zu den „vererbten“ eingefügt werden.

## 3.5 Vererbung

Die Vererbung (engl. *Inheritance*) ist eines der grundlegenden Konzepte der Objektorientierung und hat eine große Bedeutung in der Softwareentwicklung. Klassen können auch wie bei den Schnittstellen ihre Methoden und Eigenschaften an andere Klassen weitergeben. Die Klasse, die Methoden und Eigenschaften weitergibt, wird *Oberklasse* bzw. *Superklasse* genannt und die, die etwas erbt, heißt *Unterklasse* bzw. *Subklasse*. Die Vererbung bezeichnet den Vorgang, bei dem die Beziehung zwischen ursprünglicher und neuer Klasse dauerhaft definiert ist. Eine Subklasse (Kindklasse) kann dabei eine Erweiterung oder eine Einschränkung der Superklasse (Elternklasse) sein. Den Vorgang des Erbens nennt man in der Objektorientierung meist *Ableitung* oder *Spezialisierung*. Die Umkehrung von der Spezialisierung ist als *Generalisierung* bekannt. (Ratz u. a., 2001)

## 3.6 Objektorientierte Erweiterung der IEC 61131-3

In der Anwendungsentwicklung für PCs und in der Universitätsausbildung ist objektorientiertes Programmieren in den Sprachen *C++* und *Java* weit verbreitet. Die objektorientierte Programmierung hat das Potential bewiesen, Softwareentwicklungsprobleme auf elegante Weise zu lösen und flexible, wiederverwendbare Softwarekomponenten zu produzieren.
Im Gegensatz dazu werden Industriesteuerungen (SPSen) meistens immer noch in prozeduralen Sprachen wie *C* oder den Sprachen der IEC 61131-3-Norm programmiert. Während manche SPS-Entwickler nur darauf warten, die objektorientierte Programmierung einsetzen

zu können, mögen andere skeptisch sein, ob objektorientiertes Programmieren für den Bereich der Steuerungsprogrammierung überhaupt angemessen ist. Um beide Gruppen anzusprechen, sollte ein Programmierwerkzeug für die objektorientierte SPS-Programmierung folgende Anforderungen erfüllen:

**Integration in eine SPS-Entwicklungsumgebung:** Die Erstellung des objektorientierten Programms und die Konfiguration der E/As sollten im Programmiersystem integriert sein; das objektorientierte Programm sollte direkten Zugriff auf E/A-Signale haben; Download und Online-Debugging auf der Steuerung sollten weiterhin möglich sein.

**Multi-paradigmatisch:** Die Objektorientierung sollte optional sein. Klassisch, d. h. prozedural programmierter Code sollte mit dem neuen, objektorientierten Code mischbar sein. Vorteil hierdurch ist, dass die Skeptiker schrittweise umsteigen können und dabei die Option behalten, zum prozeduralen Stil zurückzukehren.

**OOP basierend auf der IEC-Norm:** Objektorientierte SPS-Programmierung sollte nicht durch die Anpassung einer objektorientierten Programmiersprache an die SPS-Programmierung erfolgen, sondern durch eine Erweiterung der IEC-Programmierung um einen kleinen Satz an etablierten objektorientierten Fähigkeiten. Vorteilhaft dabei ist, dass die SPS-Programmierer nicht eine völlig neue Sprache lernen müssen.

**Mehrsprachigkeit:** Die Objektorientierung sollte in allen Sprachen der IEC 61131-3 verwendbar sein (textuell und grafisch), nicht nur in der textuellen Sprache `ST`, welche *C*++ und anderen bekannten OOP-Sprachen am nächsten kommt. Vorteil: Manche wichtige Aspekte von SPS-Applikationen - wie Zustandsmaschinen und komplexe boolesche Verknüpfungen - lassen sich textuell nicht übersichtlich darstellen. (Vogel-Heuser, 2008)

Durch Erweiterung der IEC 61131-3 um neue Schlüsselwörter kann der Steuerungsentwickler seine *CoDeSys*-Applikation objektorientiert in den Sprachen der Norm programmieren. Allerdings ist die objektorientierte Programmierung eine Option, d. h. der Anwender kann weiterhin funktional programmieren oder funktionale und objektorientierte Programmierungen kombinieren. Dabei ist sprachwissenschaftlich bemerkenswert, dass das objektorientierte Klassenkonzept in *CoDeSys* nicht wie üblich durch den Ausbau eines prozeduralen Datenstrukturkonzepts entsteht, sondern durch den Ausbau eines Prozedurmodulkonzepts. Beispielsweise erweiterte *C*++ die `STRUCTs` von *C*; und *Simula* erweiterte die `RECORDs` von *Pascal* um Methoden und Vererbung. Im Gegensatz dazu erweitert *CoDeSys* nicht die `STRUCTs` der IEC, sondern deren `Funktionsblöcke`:

- *CoDeSys* baut POEs vom Typ `FUNCTION_BLOCK` zu Klassendefinitionen aus durch das Hinzufügen untergeordneter Methoden-POEs und Eigenschaften.
- Analog zu diesen objektorientierten Funktionsbausteinen gibt es das neue Konstrukt `INTERFACE` zur Deklaration abstrakter Klassen (Objekttypen). Die darunter eingefügten Methoden und Eigenschaften haben wohlgemerkt keinen Implementierungsteil.

- Subklassenbeziehungen zwischen den Funktionsbausteinen und den Schnittstellen werden mittels `EXTENDS`- und `IMPLEMENTS`-Deklarationen hergestellt.

Bei der Implementierung von Objektorientierung zur SPS-Programmierung in *CoDeSys* wurden folgende Anforderungen berücksichtigt:

- Die funktionale, d. h. nicht-objektorientierte Programmierung muss weiterhin möglich sein.
- Die funktionale und objektorientierte Programmierung muss innerhalb eines einzigen Projekts gemischt werden können.
- Bestehende Applikationen müssen mit sinnvollen Mitteln ganz oder teilweise auf ein objektorientiertes Design umgestellt werden können.
- Die objektorientierte Programmierung muss in allen Sprachen der IEC 61131-3 möglich sein.
- Die Komplexität der Sprachdefinition soll den Anwendungsprogrammierer nicht sonderlich belasten. (Vogel-Heuser, 2008)

In der Automatisierungstechnik wird die objektorientierte Programmierung verwendet, um die Komplexität der entstehenden Programme zu verringern. Typischen Anwendungsszenarien sind beispielsweise die Programmierung von unterschiedlichen Antrieben, die über eine identische Funktionalität verfügen (z. B. Positionieren) oder auch die Programmierung unterschiedlicher Maschinenmodule mit einheitlicher oder ähnlicher Funktionalität (z. B. Initialisierungsbetrieb, Automatikbetrieb).

### 3.6.1 Schlüsselwörter in CoDeSys

In *CoDeSys* wurde das Konzept des IEC 61131-3 Funktionsblocks zur Klasse erweitert. Demnach kann ein Funktionsbaustein von anderen Funktionsbausteinen erben, Methoden enthalten sowie Interfaces implementieren. Die Instanzen der Funktionsblöcke sind die sogenannten Objekte.

Die objektorientierte Spracherweiterung in *CoDeSys* wurde mit folgenden neuen Schlüsselwörtern erzielt:

**`METHOD:`** Eine Methode ist eine Funktion, die einer FB-Definition zugeordnet ist. Für die Implementierung der Methoden-Funktionen sind die formalen und lokalen Parameter sowie die Variablen des übergeordneten FBs sichtbar.

**`PROPERTY:`** Eine solche „Eigenschaft“ enthält zwei spezielle Methoden, welche automatisch im Objektbaum unterhalb des Eigenschaften-Objekts eingehängt werden:

**`Set`-Methode:** Wird aufgerufen, wenn auf die Eigenschaft *schreibend* zugegriffen wird, d. h. der Name der Eigenschaft wird als *Eingabeparameter* verwendet.

**`Get`-Methode:** Wird aufgerufen, wenn auf die Eigenschaft *lesend* zugegriffen wird, d. h. der Name der Eigenschaft wird als *Ausgabeparameter* verwendet.

**`THIS`:** Dieser Bezeichner kann verwendet werden, um direkt auf die implizite Instanz des betreffenden Funktionsbausteins zu zeigen, die automatisch verfügbar ist.

**`EXTENDS`:** Leitet einen Funktionsbaustein (Kindklasse) von einem anderen Funktionsbaustein (Elternklasse) ab. Der abgeleitete FB erhält dabei die Methoden und Eigenschaften des „vererbenden“ FBs zusätzlich zu seinen eigenen.

**`INTERFACE`:** Ein Interface ist wie eine leere Hülle eines Funktionsblocks und legt die Schnittstellen fest, ohne die konkrete Umsetzung der eigentlichen Funktion. Die Interface-Methoden und Properties haben keinen Inhalt, also keinen Implementierungsteil und keine lokalen Variablen. Der Rumpf der Methode wird erst in der Klasse ausprogrammiert, in der das Interface implementiert wird.

**`IMPLEMENTS`:** Ein Funktionsbaustein, der ein Interface implementiert, muss alle Methoden, die in dieser Schnittstelle definiert sind, enthalten, d. h. der Name, die Eingänge und der Ausgang der Methoden müssen jeweils genau identisch sein. (Vogel-Heuser und Wannagat, 2009)

Unter anderem bietet *CoDeSys* die Constructor- und Destructor-Methoden `FB_Init` und `FB_Exit` zum Initialisieren bzw. Beenden von Objekten an.

### 3.6.2 Vorteile der objektorientierten SPS-Programmierung

- Der erzeugte Programmcode ist übersichlich und leichter zu verändern bzw. zu erweitern und damit wartungsfreundlicher.
- Die Wiederverwendung von Codes ist wesentlich einfacher und die Kapselung[1] der Daten deutlich verbessert.
- Die Performance der Steuerungsapplikation ist - außer bei sehr kleinen Applikationen - verbessert.
- Objektorientierte Programmierung ist eine gängige Lehrmethode in der Hochsprachenprogrammierung. (3S, 2009)

[1] Das Kapselungsprinzip in der Objektorientierung besagt, dass Klassen die Attribute und Operationen zu einer Einheit zusammenfassen und diese Attribute nur indirekt über die Operationen der Klasse zugänglich sind. (Oestereich, 2006)

Nach einer Einführung in das Programmiersystem *CoDeSys* und anschließender Einleitung in die Prinzipien der objektorientierten Programmierung wird in den nachfolgenden Kapiteln ein großes Augenmerk auf die Modellierung und Realisierung des objektorientierten SPS-Steuerungsprogramms für ein Applikationsmodell mit *CoDeSys V3* gelegt. Als Applikationsmodell dient die dankend zur Verfügung gestellte Sortieranlage von Herrn Dipl.-Ing. Krebbel, Assistent des Labors für Automatisierungstechnik der Hochschule für Angewandte Wissenschaften Hamburg.

# 4 Applikationsmodell: Sortieranlage

Als Anwendungsbeispiel für die Einführung von *CoDeSys V3* und den modularen Entwurf von SPS-Programmen wird eine anschauliche Sortieranlage (Abb. 4.1) als Modell einer einfachen fertigungstechnischen Aufgabe benutzt.

**Bemerkung!**
In Anlehnung an das Applikationsbeispiel *Sortieranlage* aus dem 2009 veröffentlichten Buch „Modulares Engineering und Wiederverwendung mit CoDeSys V3“ von Vogel-Heuser und Wannagat werden die Funktionsbausteine, Methoden, IEC-Aktionen etc. zur Automatisierung der vorhandenen Sortieranlage entwickelt.

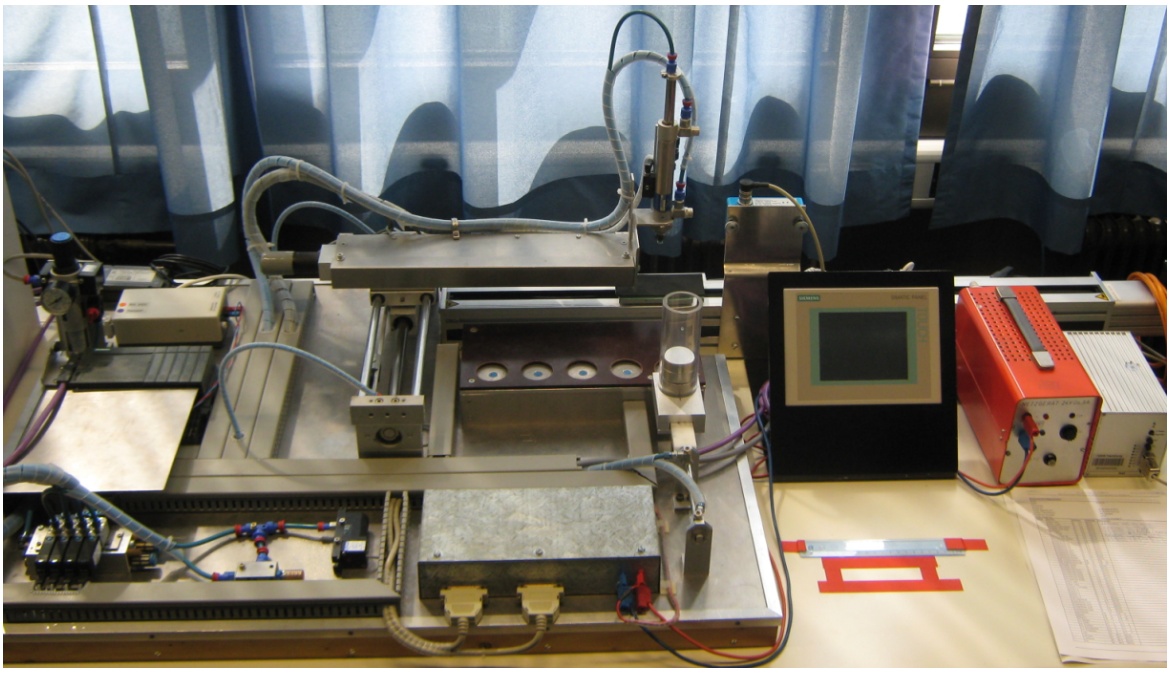


Abbildung 4.1: Applikationsbeispiel: Sortieranlage

Zunächst wird in den folgenden Abschnitten der dynamische Ablauf des Sortierprozesses beschrieben. Da es für die Analyse der Anlage und die objektorientierte Softwarestrukturierung sehr bedeutend ist, dass man sich mit den Bestandteilen (*Gewerken* bzw. *Aggregate*) der Anlage und deren Funktionalität vertraut macht, wird auf die Gewerke im Einzelnen eingegangen.

Anschließend werden in dem nächsten Kapitel die unterschiedlichen Kommunikationsarten aufgeführt. Diese beinhalten:

- Die Kommunikation des Programmierrechners mit der Steuerung.
- Die serielle Kommunikation der Linearachse mittels RS-232-Schnittstelle.
- PROFIBUS-DP zur Ansteuerung von Sensoren und Aktoren durch die Siemens-Dezentrale Peripherie ET 200S.

Im Anschluss wird im Kapitel 6 die Vorgehensweise sowohl zur strukturierten Programmierung als auch zur Identifikation der wiederverwendbaren Module vorgestellt.
Das darauffolgende Kapitel 7 ist der Realisierung der Module und deren Implementierung gewidmet.

## 4.1 Dynamischer Ablauf des Sortierprozesses

Wie einleitend erwähnt stellt der zugrundeliegende Prozess einen einfachen diskreten fertigungstechnischen Prozess dar. Aus einem Materialvorratslager (*Ausschieber*) werden zylinderförmige Teile (*Werkstücke*) auf eine Abgabeposition hinausgeschoben. Sobald ein Werkstück zur Weiterverarbeitung in der Abgabeposition vor dem Ausschieber bereit steht, befördert ein zweidimensionaler Greifer (*XY-Schieber*) es auf eine mit einem Schrittmotor betriebene Antriebseinheit (*Linearachse*). Die Linearachse hat die Möglichkeit nach links und rechts zu fahren, um das Objekt unter einem Abstandsmessgerät (*Triangulationslaser*) zwecks seiner „Höhenprofil“-Vermessung zu führen.
Das zuallererst vermessene Werkstück dient als ein Referenzstück, mit dessen Höhenprofildaten die weiteren Werkstücke verglichen werden. Nach Erfassung der Profildaten des Werkstückes wird dies zur Aufnahme- bzw. Abnahmestelle der Linearachse gebracht. Von dieser Stelle aus wird das Werkstück mit Hilfe des XY-Schiebers in die Endstation (*Sortierstation*), die die Senke des Prozesses darstellt, sortiert.

## 4.2 Struktur und Bestandteile der Anlage

Die grobe Beschreibung der Funktionalität und der Struktur der Sortieranlage vermittelt einen guten ersten Eindruck über die Anforderungen an die Anlage, der aber längst nicht ausreichend ist, um die Automatisierungsaufgabe zu meistern. Dazu wird in den folgenden Abschnitten jedes Gewerk einzelnd und samt der dort verfügbaren Sensoren und Aktoren vorgestellt.

### 4.2.1 Ausschieber

Der *Ausschieber* besteht aus einem Materialspeicher und einem Ausstoßer, der durch einen pneumatischen Zylinder die zylinderförmigen Werkstücke auf die Abholposition vereinzeln kann. Die Werkstücke werden in den Materialspeicher von oben eingebracht und dort gestapelt. Der unterste Teil wird vom Ausstoßer hinausgeschoben und somit funktioniert der Ausschieber nach dem FIFO[1]-Prinzip.

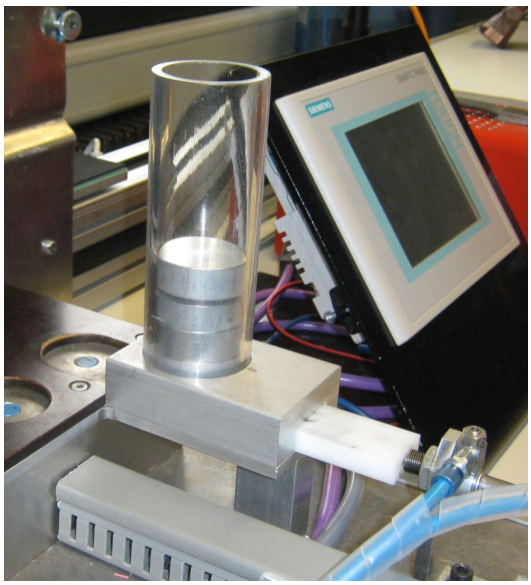


Abbildung 4.2: Der Ausschieber

Der pneumatische Zylinder ist eine Lineareinheit, d. h. er beschreibt mit seinem Verhalten (Ausstoßen) eine geradlinige Bewegung. Er ist über zwei elektrisch pneumatische Ventile mit einer Schalterstellung (boolesches Signal) steuerbar, welche den Kolben des Zylinders in zwei entgegengesetzte Richtungen treiben. Da der Zylinderkolben keinen Sensor für die Endlage besitzt, wird mit Hilfe des *Position 1*-Sensors der Sortierstation festgestellt, ob ein Werkstück vom Ausschieber bereits ausgeschoben wurde.

### 4.2.2 XY-Schieber

Der *XY-Schieber* stellt das Bindeglied zwischen den Aggregaten Ausschieber, Linearachse und Sortierstation dar. Seine Freiheitsebene ermöglicht ihm, die Transportaufgabe der Werkstücke vom Ausschieber zur Linearachse bzw. von der Linearachse zur Sortierstation zu übernehmen. Der zweidimensionale XY-Schieber wird in Y-Richtung mittels eines Pneumatikzylinders bewegt. Ebenfalls pneumatisch erfolgt die Objekt-Greifbewegung (hoch bzw.

[1] **F**irst **I**n **F**irst **O**ut

runter) sowie seine Ansaug-funktion mit Hilfe des *Vakuumsaugers*. Der Vakuumsauger lässt sich durch Zwei-Wege-Ventile aktivieren und deaktivieren. Durch einen Spindelantrieb wird der XY-Schieber entlang der X-Achse gefahren.

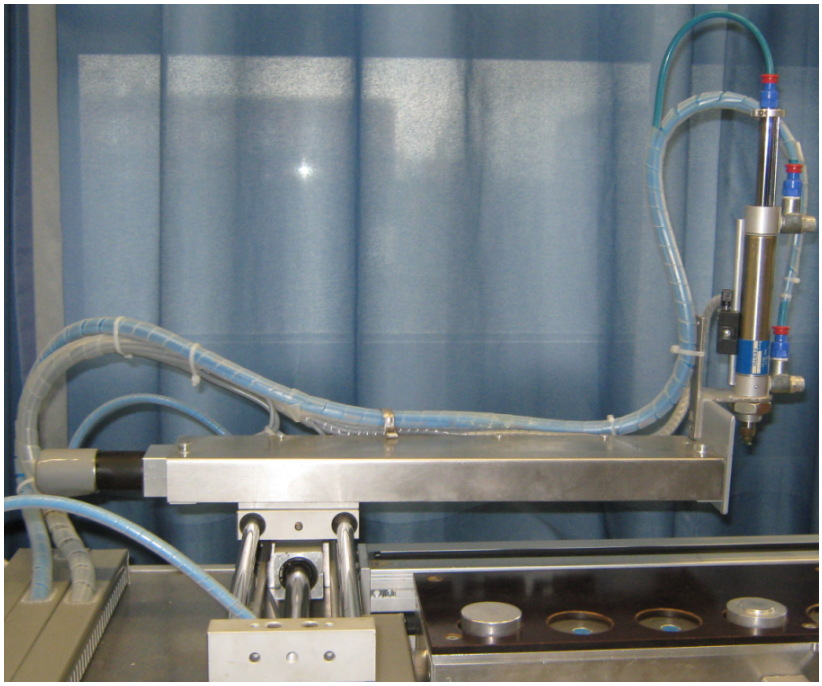


Abbildung 4.3: Der XY-Schieber

Die Bewegungen des XY-Schiebers in X- und Y-Richtung werden mit Sensoren detektiert. In X-Richtung sind fünf Sensoren, die mit denen der Sortierstation korrespondieren, zur Feststellung der Positionslage des XY-Schiebers angebracht. In Y-Richtung kann er nur zwei mögliche Positionen annehmen, nämlich die Seite der Linearachse oder die des Ausschiebers bzw. der Sortierstation.

### 4.2.3 Linearachse

Die *Linearachse* ist eine Lineareinheit mit Spindelantrieb, die durch eine Schrittmotor-Antriebseinheit angesteuert wird. Beim vorhandenen Sortierprozess hat die Linearachse die Aufgabe, die Werkstücke, die unter dem Triangulationslaser vermessen werden, von und zur Auf-/Abnahmestelle zu transportieren.

Die Antriebseinheit stellt dem Anwender einen intelligenten und umfangreichen Befehlsumfang zur Antriebssteuerung zur Verfügung. Für eine ausführliche Beschreibung der Betriebsarten und den Befehlsumfang der Schrittmotor-Antriebseinheit wird auf die Dokumentationen der Firma *isel Germany AG* verwiesen. (isel, 2010)


Abbildung 4.4: Linearachse und ihre Antriebseinheit

Zur Datenübertragung zwischen der Linearachse und dem Steuer-PC wird die serielle Schnittstelle (RS-232) eingesetzt. Die Verbindung ist über eine 3-Draht-Leitung[2] realisiert. Ein Software-Protokoll ermöglicht die fehlerfreie Übertragung der ASCII-Zeichen. Dabei ist es notwendig, dass sich beide Systeme an das im Folgenden beschriebene Übertragungsprotokoll halten:

- Der angeschlossene Steuer-PC sendet einen Befehl, der mit einem Zeilenendezeichen (`LF+CR`[3] bzw. `char(11)+char(13)`) abgeschlossen ist.
- Die Schrittmotor-Antriebseinheit quittiert die Ausführung bzw. Speicherung des Befehls durch das Quittierungssignal `0` bzw. `char(48)` oder meldet einen aufgetretenen Fehler mit einem ASCII-Zeichen ungleich `0`.

Auf der Schrittmotor-Antriebssteuerungseinheit sind als Datenübertragungsparameter für die serielle Schnittstelle folgende Werte festgelegt:

- 9600 Baudrate
- 8 Daten-Bits
- 1 Stopp-Bit
- keine Parität

Wie die Übertragungsparameter in *CoDeSys V3* eingestellt werden und welche Bibliothek dafür eingebunden werden soll, wird im Abschnitt 5.4 beantwortet.

[2] Es werden für die Datenübertragung lediglich die drei Leitungen RxD, TxD und GND benötigt.
[3] **L**ine **F**eed **C**arriage **R**eturn

Das Betriebssystem der Antriebsteuerungseinheit ermöglicht sowohl eine Programmierung im *DNC*[4]*-Modus* (direkte Ausführung der übergebenen Befehle) als auch im *CNC*[5]*-Modus*, d. h. auszuführende Programme werden im internen Datenspeicher abgelegt und später durch ein Start-Signal gestartet. (isel, 2010)

Hier wird die Antriebsteuerungseinheit im DNC-Modus betrieben. Im Gegensatz zum CNC-Modus werden im DNC-Modus der Steuerungseinheit die Bearbeitungsparameter einzeln übergeben und von ihr direkt ausgeführt. Durch Auswertung der Quittierungssignale der Einheit ist der übergeordnete Steuer-PC in der Lage, kontinuierlich und ohne Begrenzung Daten zu übergeben.

Für den Sortierprozess sind folgende Positionen von Relevanz, die von der Linearachse befahren werden sollen:

**Referenzpunkt:** Die anzufahrende Positionen-Bezugsstelle nach dem „kalten“ Start der Anlage.

**Auf-/Abnahmestelle:** Die Position, an der die Werkstücke vom XY-Schieber aufgenommen bzw. an ihn abgegeben werden.

**Vor-Mess-Position:** Die Stelle, wo das Werkstück unmittelbar vor dem Laserstrahl zur Vermessung bereit steht.

**Nach-Mess-Position:** Die begrenzende Stelle der Messfahrt. Dahin gelangt das Werkstück, nach dem es komplett vermessen wurde.

#### Befehlsaufbau der DNC-Modus-Befehle

Im DNC-Modus werden die von dem Steuer-PC übergebenen Datensätze bzw. Befehle direkt ausgewertet und ausgeführt. Dafür ist zu Beginn der Datenkommunikation eine sogenannte *Initialisierung* notwendig. Sie besteht aus dem Dateneröffnungszeichen `@`, der Gerätenummer (Standard = 0) und der Anzahl der zu verfahrenen Achsen. Anschließend werden der Prozessorkarte der Antriebseinheit die „Programmschritte“ einzeln übergeben und von ihr direkt ausgeführt.

Zur Überprüfung der Datenübertragung bzw. Meldung von aufgetretenen Fehlern werden über die serielle Schnittstelle entsprechende ASCII-Zeichen an den Steuer-PC zurückgesendet. Dieses sogenannte *Software-Handshake-Verfahren* kann zu zwei unterschiedlichen Zeitpunkten realisiert werden:

1. Die Prozessorkarte setzt direkt nach Empfang des abzuarbeitenden Befehls das Quittierungs- bzw. Fehlerzeichen ab.

[4] **D**istributed **N**umerical **C**ontrol
[5] **C**omputerized **N**umerical **C**ontrol

2. Die Prozessorkarte arbeitet den übersendeten Befehlssatz ab und meldet anschließend das Quittierungs- bzw. Fehlerzeichen zurück.

Der gewünschte Modus wird durch Groß- und Kleinschreibung des Befehlszeichens unterschieden. Großbuchstaben führen zur Rückmeldung nach Abarbeitung der Befehle, Kleinbuchstaben zur direkten Rückmeldung an den Steuer-PC. (isel, 2010)

Im Folgenden sind die für die Positionierung der Linearachse benötigten Befehle aufgeführt. Hierbei werden die Lage der absoluten Position und die angepasste Verfahrgeschwindigkeit, die variabel einstellbar ist, mit Hilfe der Software *HTerm 0.8.1beta*[6] empirisch ermittelt:

**Achsenanzahl setzen:** ”`@01 LF+CR`”

**Referenzgeschwindigkeit setzen:** ”`@0d6000 LF+CR`”

**Referenzfahrt:** ”`@0R1 LF+CR`”

**Bewegung zur Auf-/Abnahmestelle:** ”`@0M 47600,7500 LF+CR`”

**Bewegung zur Vor-Mess-Position:** ”`@0M 31000,7500 LF+CR`”

**Bewegung zur Nach-Mess-Position:** ”`@0M 34400,250 LF+CR`”

Auf die konkrete Implementierung der Linearachse-Befehle in *CoDeSys V3* wird im Abschnitt 7.2 eingegangen.

### 4.2.4 Sortierstation

Die einfach gestaltete *Sortierstation* ist das Endziel eines vermessenen Werkstücks. An der Sortierstation sind fünf metalldetektierende Sensoren (Position 1 – 5) angebracht, die nach dem Prinzip der induktiven Näherung arbeiten und erfassen, ob sich dort ein metallisches Objekt befindet.

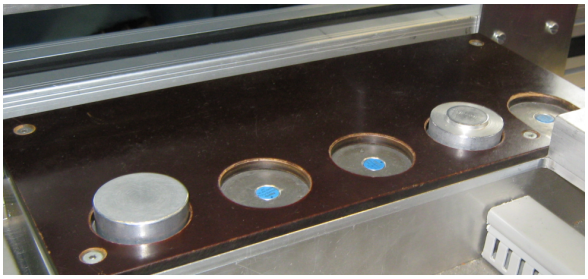


Abbildung 4.5: Die Sortierstation

[6](c) Tobias Hammer 2006. URL: (`http://www.der-hammer.info/terminal/`)

### 4.2.5 Triangulationslaser

Das zur Höhenprofil-Datenerfassung eingesetzte Laser-Abstandsmessgerät (LT 100/70 Laser-Distanz-Sensor, Abb. 4.6) der Firma *Eltrotec Sensor GmbH* zählt zu der Klasse der Laser-Sensoren. Unter Laser-Sensoren versteht man Präzisions-Sensorsysteme auf optoelektronischer Basis, die als Lichtquelle keine Lampe oder Lumineszenzdiode[7] enthalten, sondern einen Halbleiterlaser.

In der industriellen Automatisierung werden Laser-Lichtquellen u. a. wegen ihrer guten Fokussierbarkeit vorwiegend in Sensoren verwendet, die für Messaufgaben unterschiedlichster Art zum Einsatz kommen. (Hannemann, 2000)

Das vorhandene Laser-Abstandsmessgerät (*Triangulationslaser*) zeichnet sich nicht nur dadurch aus, dass es eine hohe Linearität durch interne digitale Signalverarbeitung enthält, sondern auch, dass es gegen Kontrast- und Farbübergänge durch einen speziellen Auswertalgorithmus unempfindlich ist. (Eltrotec, 2010)

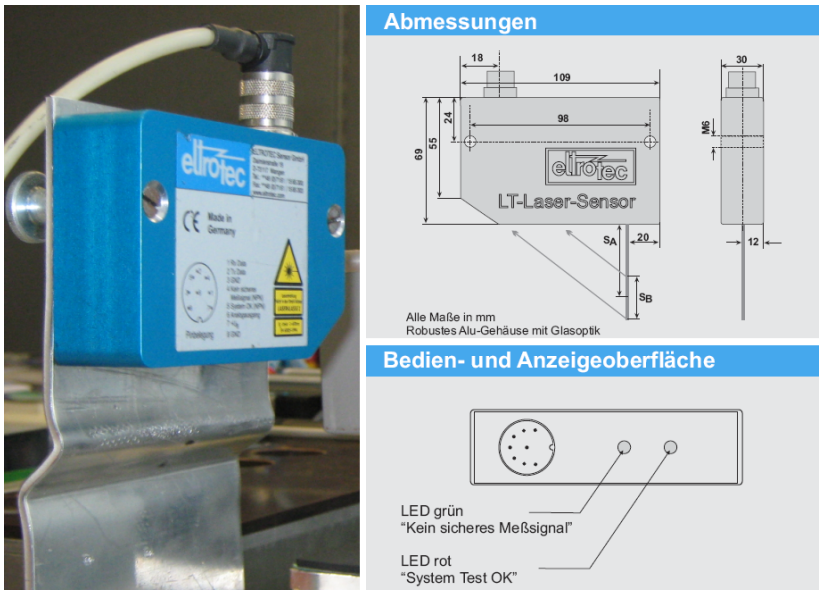



Abbildung 4.6: Laser-Distanz-Sensor der Firma Eltrotec (Eltrotec, 2010)

#### 4.2.5.1 Funktionsprinzip des Laser-Sensors

Das von einer Lichtquelle in einem Sensorsystem emittierte Licht wird von einem zu erfassenden Objekt rückgestreut und von einem Lichtempfänger im Sensorsystem wieder aufgefangen.

[7] Halbleiterdiode, die aufgrund eines elektrischen Stroms Lichtstrahlung aussendet

Für die Auswertung des Sensorsignals wird bei Laser-Abstandsensoren als Messprinzip das sogenannte *Triangulationsverfahren* angewandt. Dieses Verfahren hat seinen Ursprung in der Landvermessung und ist eine Methode der optischen Abstandsmessung mit Hilfe von trigonometrischen Funktionen.

Bei der Lasertriangulation (Abb. 4.7) wird auf das Messobjekt ein Laserstrahl fokussiert und ein Teil der vom Objekt rückgestreuten Strahlung mit einer im Sensorsystem empfindlichen Kamera, einer ortsauflösenden Fotodiode (PSD[8]) oder einer CCD[9]-Zeile beobachtet. Ändert sich die Entfernung des Messobjekts vom Sensor, ändert sich auch der Winkel, unter dem der Lichtpunkt beobachtet wird und damit auch die Position seines Abbilds auf dem Lichtempfänger. Aus der Positionsänderung wird mit Hilfe von trigonometrischen Funktionen die Entfernung des Objekts vom Laserprojektor berechnet. Demzufolge erzeugt der „positionsempfindliche“ Lichtempfänger in Abhängigkeit der bestrahlten Stelle eine unterschiedliche Messspannung. (Hannemann, 2000)

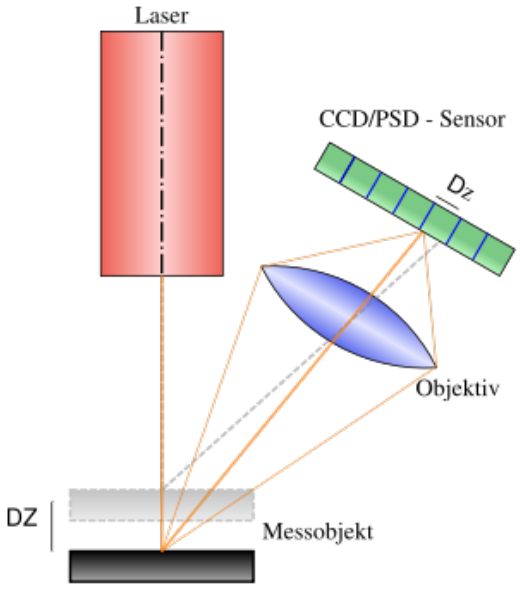



Abbildung 4.7: Prinzip der Lasertriangulation (Wikipedia, 2010, Abstandsmessung_(optisch), Zugriff: 02.07.2010)

Der folgende Abschnitt beschäftigt sich mit der Verarbeitung des vom Lichtempfänger erzeugten Messsignals und ihre Bereitstellung als Peripherie-Eingangswort (`PEW`) von der Analogeingangsbaugruppe im SPS-Programm.

[8] **P**osition **S**ensitive **D**evice

[9] **C**harge **C**oupled **D**evice: Lichtempfindliches elektronisches Bauelement, das auf dem inneren Photoeffekt beruht.

#### 4.2.5.2 Analogwertverarbeitung

Die Aufgabe einer Analogeingangsbaugruppe ist es, ein analoges Prozesssignal in einem digitalen Messwert im Format eines 16-Bit-Datenworts umzusetzen und in einem Peripherie-Eingangswort zur Weiterverarbeitung im SPS-Programm bereitzustellen. Der Umsetzungsvorgang verläuft in den aus der Digitaltechnik bekannten Schritten:

- Abtastung
- Quantisierung
- Codierung

Die *Abtastung* bezeichnet den Messvorgang, bei dem über einen festgelegten Zeitraum ein Prozesssignal durch Aufladung eines Kondensators aufwärts integrierend gemessen wird. Das Messergebnis ist ein interner Spannungswert, der proportional dem arithmetischen Mittelwert des Prozesssignals ist. Die dem Messsignal überlagerte Netzspannungsstörung von 50 Hz wird dabei vollständig unterdrückt.

Die nachfolgende *Quantisierung* ist der Vorgang, bei dem für den internen Spannungswert ein proportionaler Zahlenwert ermittelt wird. Das geschieht durch die abwärts integrierende Entladung des Kondensators mit Hilfe einer Referenzspannung und gleichzeitigem Zählen der Taktimpulse in einem Dualzähler.

Bei der abschließenden *Codierung* werden die Zahlenwerte als vorzeichenbehaftete 16-Bit-Ganzzahlen (Integer) entsprechend dem Bitmuster nach Tabelle 4.1 gebildet. Die negativen Zahlenwerte werden in Zweierkomplimentform dargestellt.

| **Auflösung** | **Analogwert** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bitnummer | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Wertigkeit der Bits | VZ | $2^{14}$ | $2^{13}$ | $2^{12}$ | $2^{11}$ | $2^{10}$ | $2^{9}$ | $2^{8}$ | $2^{7}$ | $2^{6}$ | $2^{5}$ | $2^{4}$ | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ |

Tabelle 4.1: Analogwertdarstellung (Siemens, 2010)

Der digitalisierte Analogwert wird als Eingangswort (`EW` oder `%IW`) bzw. Peripherie-Eingangswort (`PEW`) der Analogbaugruppe zur Übernahme in das SPS-Programm bereitgestellt. Die Eingangsvariable eines Bausteins zur Übernahme des digitalisierten Analogwertes wird demnach mit dem Datentyp `Integer` deklariert. (Wellenreuther und Zastrow, 2008)

Üblicherweise hat eine Analogeingangsbaugruppe mehrere Analogkanäle, die zyklisch bearbeitet werden. Dabei ist jedem Analogeingang ein Peripherie-Eingangswort mit einer entsprechenden Adresse zum Ablegen des digitalen Messwerts zugeordnet. Der Anschluss und die Konfiguration der Analogkanäle der Analogeingangsbaugruppe sind im Abschnitt 5.2.2 beschrieben.

Beim Eichvorgang des Laser-Sensors wird der gesamte Messbereich (85 mm) in 2048 ($2^{11Bit}$) Intervalle unterteilt, da die Messdaten über einen Wert-Umsetzer von seriellen Daten zu 0 – 10 V Analogsignale umgewandelt werden. Daraus folgt:

$$1\,Bit\,(Inkrement) \;\hat{=}\; 0,0415\,mm\,(85\,mm \times 1/2048)$$

Dabei entspricht die 0 der größten messbaren Entfernung vom Sensor und 2048 der kleinsten messbaren Entfernung.

Der Triangulationslaser ist an die Analogeingangsbaugruppe (Analoges Elektronik Modul [2AI U ST]) der SPS angeschlossen und liefert Spannungswerte von 0 – 10 V, die den Abstandsmesswerten proportional sind. Die nachfolgende Tabelle in der Abbildung 4.8 zeigt für ausgewählte Bereiche den Zusammenhang zwischen den Analogwerten der einzelnen Messbereiche und den zugehörigen Digitalwerten für die Spannungssignalgeber.

| Messbereich ± 5 V | Messbereich ± 10 V | Einheiten | | Bereich |
|---|---|---|---|---|
| | | dezimal | hexadezimal | |
| > 5,8794 | > 11,7589 | 32767 | $7FFF_H$ | Überlauf |
| 5,8794 | 11,7589 | 32511 | $7EFF_H$ | Übersteuerungsbereich |
| : | : | : | : | |
| 5,0002 | 10,0004 | 27649 | $6C01_H$ | |
| 5,00 | 10,00 | 27648 | $6C00_H$ | Nennbereich |
| 3,75 | 7,50 | 20736 | $5100_H$ | |
| : | : | : | : | |
| -3,75 | -7,50 | -20736 | $AF00_H$ | |
| -5,00 | -10,00 | -27648 | $9400_H$ | |
| -5,0002 | -10,0004 | -27649 | $93FF_H$ | Untersteuerungs-bereich |
| : | : | : | : | |
| -5,8796 | -11,759 | -32512 | $8100_H$ | |
| < -5,8796 | < -11,759 | -32768 | $8000_H$ | Unterlauf |

Abbildung 4.8: Messbereiche für Spannung: $\pm$ 5 V, $\pm$ 10 V (Siemens, 2010)

Laut Betriebsanleitung des Triangulationslasers ist beim Anschließen des Sensors für den Analogausgang Folgendes zu beachten:

$$\Delta\,85\,mm \;\hat{=}\; \Delta\,10\,V(theoretisch)$$

$$\Delta\,70\,mm \;\hat{=}\; \Delta\,8,23\,V$$

$$\Delta\,8,5\,mm \;\hat{=}\; \Delta\,1\,V$$

$$\Delta\,1\,mm \;\hat{=}\; \Delta\,0,1176\,V$$

Am Ausgang des Sensors entspricht 0 V dem größten und 10 V dem kleinsten messbaren Abstand vom Sensor. (Eltrotec, 2010)

Analog-Digital-Umsetzer müssen den kontinuierlichen Wertebereich eines Analogsignals auf den diskreten Zahlenbereich eines Digitalwortes abbilden. Bei einer n-Bits-Darstellung ergeben sich also genau $2^n$ Zahlenwerte (Stufen) im Dualcode. Je mehr Bits zur Zahlendarstellung zur Verfügung stehen, um so feinstufiger ist die Aufteilung des analogen Messbereiches. Die Anzahl der Bits entscheidet über die *Auflösung* im Sinne von Unterscheidbarkeit kleinster Werteänderung im Analogsignal. Der Absolutwert der kleinsten noch unterscheidbaren Spannungsänderung im Analogsignal lässt sich aus dem Messbereich-Endwert (engl. *Full Scale*) errechnen:

$$
\begin{aligned}
U_{LSB} &= \frac{Endwert(FS)}{2^n} \\
U_{LSB} &\overset{hier}{=} \frac{10\,V}{2^{11\,Bits}} \\
&= \frac{10\,V}{2048\,Bits} \\
&= 4,883\ mV/Bit
\end{aligned}
$$

Daraus folgt:

$$\Delta\ 4,883\ mV \mathrel{\hat{=}} \Delta\ 0,0415\ mm \mathrel{\hat{=}} \Delta\ 1\ Bit$$

Die Abkürzung LSB[10] bezeichnet das gerigwertigste Bit, das für die kleinste Änderung im Zahlenwert des Digitalwortes steht.

Die konkrete Implementierung der Berechnungen von Spannungswerten in [V] zu Längenangaben in [mm] wird im Abschnitt 7.2 ausführlich behandelt.

[10] **L**east **S**ignificant **B**it

# 5 Kommunikation

Das SPS-Programmiersystem *CoDeSys* kann als Soft-SPS im PC fungieren. Um Sensoren und Aktoren mit dieser Soft-SPS ansteuern zu können, muss im PC eine Soft-SPS-PROFIBUS-Karte eingebaut sein. Am PROFIBUS ist dann eine dezentrale Peripherie mit E/A-Karten angeschlossen, mit denen die Sensoren und Aktoren verbunden sind.

## 5.1 Soft-SPS-PROFIBUS-Karte CIF50

Als CPU verwendet die Soft-SPS die CPU des Host-PCs, was unter *Zielsystemeinstellungen* durch `CoDeSys SP RTE V3` festgelegt wird. Danach sind im Gerätebaum unterhalb der Knote `CoDeSys SP RTE V3` folgende Ressourcen einzutragen:

- Soft-SPS-PROFIBUS-Karte `CIF50_PB` der Firma *Hilscher* (Abb. 5.1) mit Angabe der Stationsadresse und der Baudrate.
- Dezentrale Peripherie `ET_200S_Compact_16DI_16DO` der Firma *Siemens* mit Angabe der Slave-Adresse.
- E/A-Karten, z. B. `6ES7_131_4BD01_0AB0_4DI_DC24V` und `6ES7_134_4FB01_0AB0_2AI_U`.

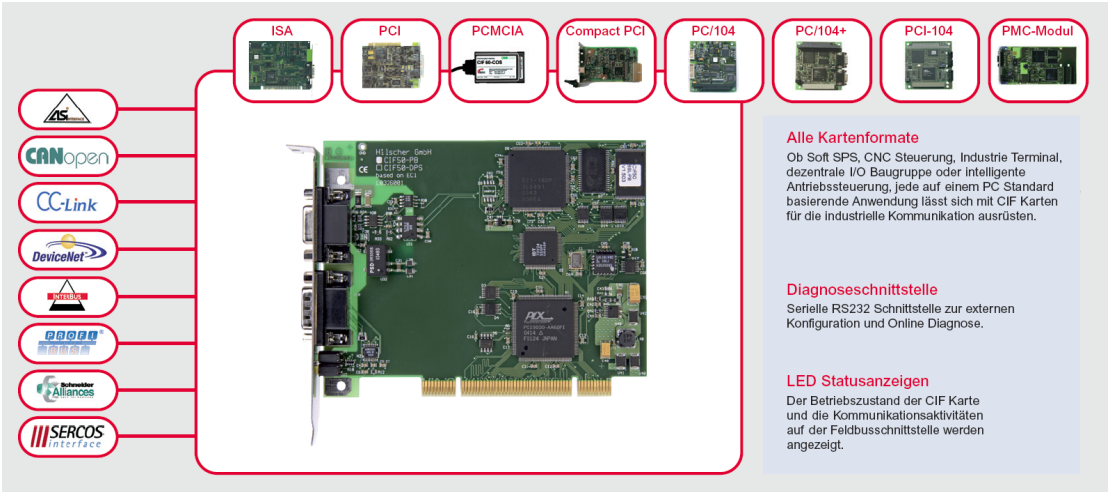



Abbildung 5.1: CIF50_PB-Karte der Firma Hilscher (Hilscher, 2009)

Damit *CoDeSys* diese Ressourcen kennt, müssen die Gerätestammdateien (GSD- bzw. GSG-Dateien) der Soft-SPS-PROFIBUS-Karte im voreingestellten Konfigurationsverzeichnis („···\Programme\Hilscher\SyCon\Fieldbus\PROFIBUS\GSD\“) abgelegt werden.

Abbildung 5.2 zeigt, wie hardewaremäßig in *CoDeSys* im Gerätebaum die Soft-SPS-PROFIBUS-Karte CIF50_PB, die Dezentrale Periphie ET_200S_Compact_16DI_16DO und die E/A-Karten eingebunden werden.

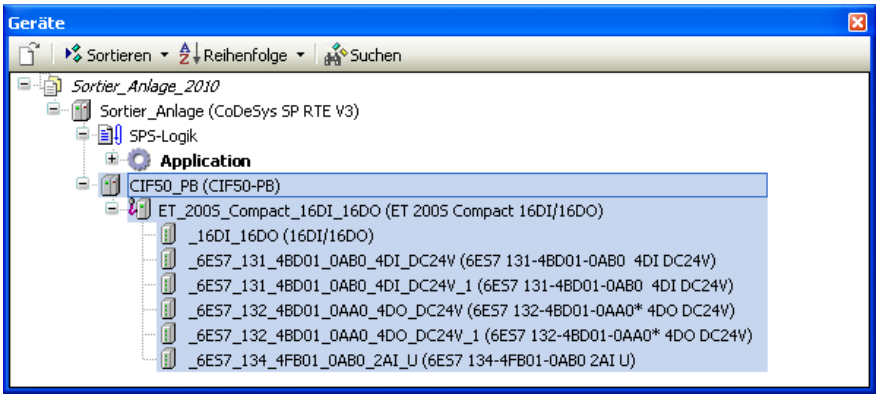



Abbildung 5.2: Festlegung der Steuerungskonfiguration in CoDeSys

Der Systemkonfigurator *SyCon* (Abb. 5.3) ermöglicht die einheitliche Konfiguration der Kommunikationsbeziehungen zwischen Feldbus, Slave Geräten und dem Master. Die Konfiguration der Kommunikationsbeziehungen zu den einzelnen Slave Geräten und dem jeweiligen Feldbus Master werden dabei dauerhaft im Konfigurationsspeicher der CIF-Karte abgelegt. (Hilscher, 2009)

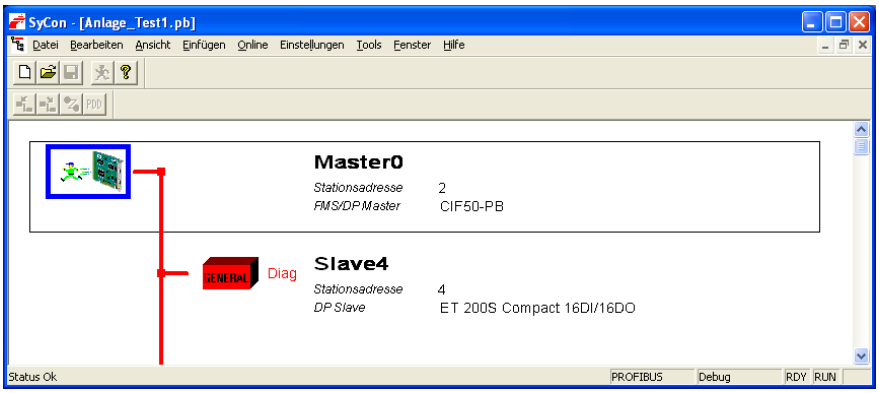



Abbildung 5.3: Systemkonfigurator SyCon

Um die Funktionen der Hischer-Soft-SPS-PROFIBUS-Karte im SPS-Programm in *CoDeSys* zu nutzen, muss die Bibliothek IoDrvHilscher.library im Projekt eingebunden wer-

den. Sie ist nach Auswahl des Befehls **Bibliotheks-Repository** im **Tools**-Menü unter den installierten Bibliotheken zu finden.

## 5.2 Siemens-Dezentrale Peripherie ET 200S

Die *SIMATIC ET 200S* der Firma *Siemens* ist ein multifunktionales und feinmodulares Peripheriesystem in Schutzart IP20[1]. Die „ET 200S“ verfügt über eine Aufbautechnik mit stehender Verdrahtung. In einem Steuerungsschaltschrank können die Terminalmodule auf einer *Hutprofilschiene* aufgerastet werden und können ohne Peripheriemodule vorverdrahtet und geprüft werden. Zudem ermöglicht diese Aufbautechnik das Ziehen und Stecken von Peripheriemodulen auch unter Spannung zum Modultausch während des laufenden Betriebs bzw. zur schnellen Fehlerlokalisierung des betreffenden Moduls ohne Notwendigkeit der Spannungsabschaltung (*Hot Swapping*). (Siemens, 2010, Zugriff: 01.07.2010)

### 5.2.1 COMPACT-Module (IM151-1)

In einem *COMPACT-Modul* sind die Funktionen eines Interfacemoduls und eines digitalen Elektronikmoduls vereinigt. Das „IM151-1 COMPACT“ ist u. a. zusammen mit einem Abschlussmodul ein kompletter DP-Slave. Tabelle 5.1 zeigt die Konfigurationsmöglichkeiten der COMPACT-Module. (Siemens, 2010, Zugriff: 01.07.2010)

| COMPACT-Modul | Anwendungen | |
|---|---|---|
| IM151-1 COMPACT | • Anschließen des PROFIBUS DP über RS485-Schnittstelle<br>• Betrieb als DPV0-Slave<br>• Direkter Datenaustausch<br>• Integrierte Peripherie:<br>– 32DI: digitale Eingänge<br>– 16DI/16DO: digitale Ein-/Ausgänge<br>• Anzahl zusätzlicher Module: max. 12<br>• Modultypen: alle außer Fehlersichere Module | Übertragungsraten: 9,6; 19,2; 45,45; 93,75; 187,5; 500 kBaud, 1,5; 3; 6; 12 Mbit/s |

Tabelle 5.1: Zuordnung COMPACT-Modul und Anwendung (Siemens, 2010)

[1] **I**nternational **P**rotection. Die Schutzart gibt die Eignung von elektrischen Betriebsmitteln (z. B. Geräte, Leuchten und Installationsmaterial) für verschiedene Umgebungsbedingungen an, zusätzlich den Schutz von Menschen gegen potentielle Gefährdung bei deren Benutzung. (Wikipedia, 2010, Schutzart, Zugriff: 16.08.2010)

In der Abbildung 5.4 ist die Anschlussbelegung des „IM151-1 COMPACT (16DI/16DO DC24V/0,5A“ für PROFIBUS-DP dargestellt.

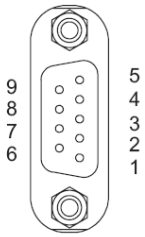
| Ansicht | | Signalname | Bezeichnung |
|---|---|---|---|
|  | 1 | - | - |
| | 2 | - | - |
| | 3 | RxD/TxD-P | Datenleitung-B |
| | 4 | RTS | Request To Send |
| | 5 | M5V2 | Datenbezugspotenzial (von Station) |
| | 6 | P5V2 | Versorgungs-Plus (von Station) |
| | 7 | - | - |
| | 8 | RxD/TxD-N | Datenleitung-A |
| | 9 | | |

Abbildung 5.4: Anschlussbelegung des IM151-1 COMPACT (16DI/16DO DC24V/0,5A) für PROFIBUS-DP (Siemens, 2010)

## 5.2.2 Analoges Elektronikmodul (2AI U ST)

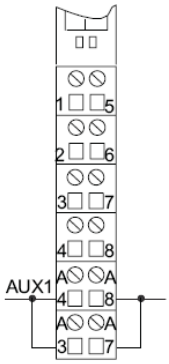
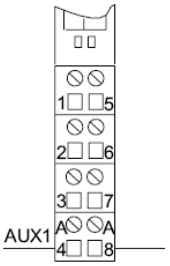
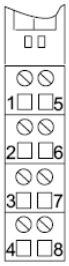
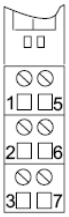
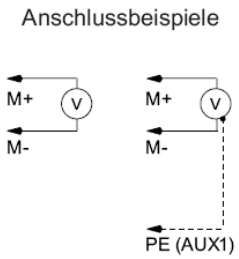
Der *analoge Elektronikmodul (2AI U ST)* besitzt folgende Eigenschaften:

- 2 Eingänge für die Spannungsmessung
- Eingangsbereiche:
  $\pm$ 10 V, Auflösung 13 Bit + Vorzeichen
  $\pm$ 5 V, Auflösung 13 Bit + Vorzeichen
  1 bis 5 V, Auflösung 13 Bit (Siemens, 2010)

Für den Elektronikmodulanschluss ist das Messsignal (analoges Spannungssignal vom Triangulationslaser) zwischen den Klemmen M+ und M- anzuschließen. Die Anschlussbelegung sowie die Hardware-Beschreibung sind den Abbildungen 5.5 und 5.6 zu entnehmen.

| Anschlussbelegung für 2AI U ST (6ES7134-4FB01-0AB0) | | | | |
|---|---|---|---|---|
| Klemme | Belegung | Klemme | Belegung | Erläuterungen |
| 1 | $M_{0+}$ | 5 | $M_{1+}$ | • $M_{n+}$: Eingangssignal "+", Kanal n<br>• $M_{n-}$: Eingangssignal "-", Kanal n<br>• $M_{ana}$: Masse des Moduls<br>• n.c.: Not connected (max. DC 30 V anschließbar)<br>• AUX1: Schutzleiteranschluss oder Potenzialschiene (frei verwendbar bis AC 230 V) |
| 2 | $M_{0-}$ | 6 | $M_{1-}$ | |
| 3 | $M_{ana}$ | 7 | $M_{ana}$ | |
| 4 | n.c. | 8 | n.c. | |
| A4 | AUX1 | A8 | AUX1 | |
| A3 | AUX1 | A7 | AUX1 | |

Abbildung 5.5: Anschlussbelegung für das analoge Elektronikmodul (2AI U ST) (Siemens, 2010)

| Verwendbare Terminalmodule für 2AI U ST (6ES7134-4FB01-0AB0) | | | | |
|---|---|---|---|---|
| TM-E15C26-A1 (6ES7193-4CA50-0AA0) | TM-E15C24-A1 (6ES7193-4CA30-0AA0) | TM-E15C24-01 (6ES7193-4CB30-0AA0) | TM-E15C23-01 (6ES7193-4CB10-0AA0) | ← Federklemme |
| TM-E15S26-A1 (6ES7193-4CA40-0AA0) | TM-E15S24-A1 (6ES7193-4CA20-0AA0) | TM-E15S24-01 (6ES7193-4CB20-0AA0) | TM-E15S23-01 (6ES7193-4CB00-0AA0) | ← Schraubklemme |
| TM-E15N26-A1 (6ES7193-4CA80-0AA0) | TM-E15N24-A1 (6ES7193-4CA70-0AA0) | TM-E15N24-01 (6ES7193-4CB70-0AA0) | TM-E15N23-01 (6ES7193-4CB60-0AA0) | ← Fast Connect |
|   |   |  |  |   |

**Prinzipschaltbild**

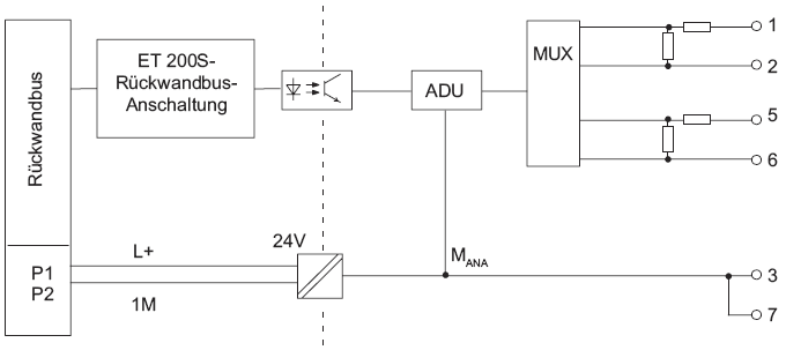



Abbildung 5.6: Hardware und Prinzipschaltbild (Siemens, 2010)

In der folgenden Abbildung 5.7 sind die vom Hersteller beschriebenen Parameter abgebildet.

| 2AI U ST | Wertebereich | Voreinstellung | Wirkungsbereich |
|---|---|---|---|
| Sammeldiagnose (Parametrierfehler, interner Fehler) | • sperren<br>• freigeben | sperren | Modul |
| Diagnose: Überlauf / Unterlauf | • sperren<br>• freigeben | sperren | Modul |
| Diagnose: Drahtbruch * | • sperren<br>• freigeben | sperren | Kanal |
| Glättung | • keine<br>• schwach<br>• mittel<br>• stark | keine | Kanal |
| Messart/ -bereich | • deaktiviert<br>• ± 5 V<br>• ± 10 V<br>• 1 bis 5 V | ± 10 V | Kanal |
| * nur im Messbereich 1 bis 5 V | | | |

Abbildung 5.7: Parameter für analogen Eingabemodul (Siemens, 2010)

Konkret sieht die Konfiguration des analogen Elektronikmoduls in *CoDeSys* so aus, wie in Abbildung 5.8 dargestellt:

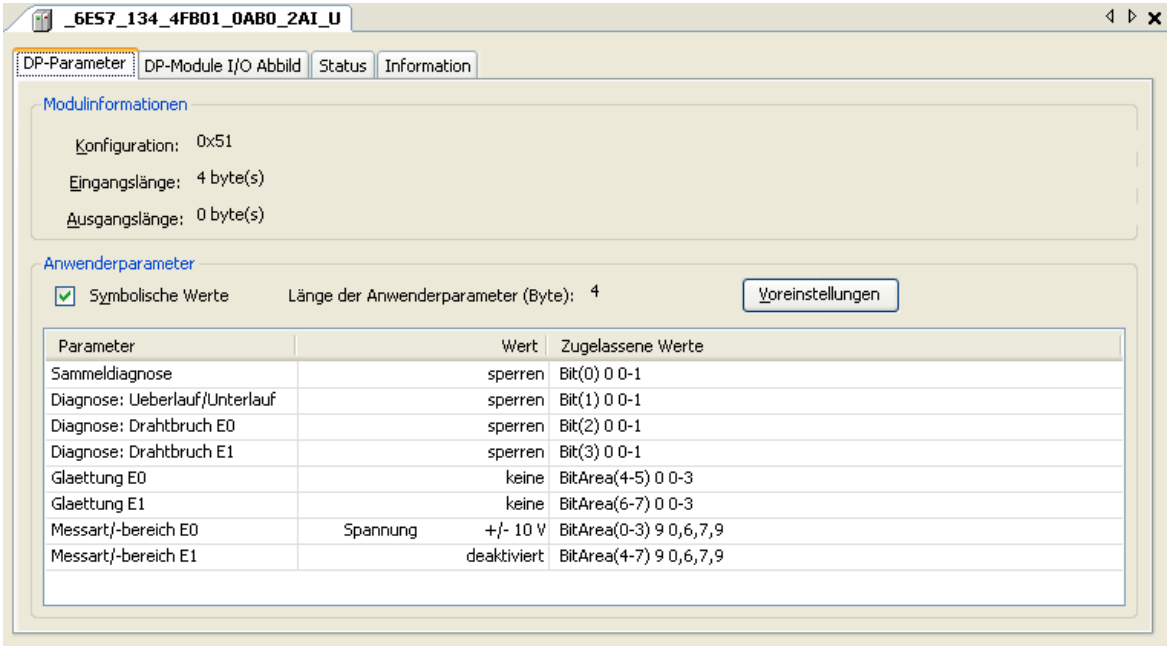



Abbildung 5.8: Konfiguration des analogen Elektronikmoduls in CoDeSys

## 5.3 PROFIBUS-DP

Die Architektur des PROFIBUS[2]-Protokolls orientiert sich an den bereits bestehenden nationalen und internationalen Normen. Die Protokollarchitektur basiert auf dem OSI[3]-Referenzmodell, entsprechend der internationalen Norm ISO[4]. Das in Abbildung 5.9 dargestellte ISO/OSI-Modell für Kommunikationsstandards besteht aus 7 Layern (Schichten) und lässt sich in zwei Klassen einteilen:

- Die netzorientierten Layer 1 bis 4
- Die anwenderorientierten Layer 5 bis 7

Die Layer 1 bis 4 beschreiben den Transport der Daten von einem Ort zum anderen, während die Layer 5 bis 7 dem Anwender den Zugriff auf das Netz- bzw. Bussystem in entsprechender Form zur Verfügung stellen.

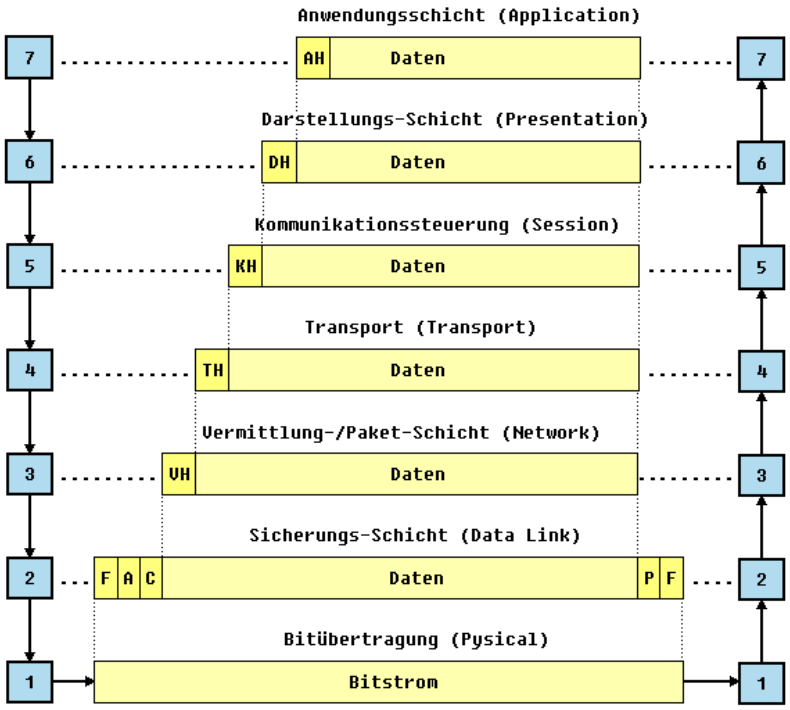



Abbildung 5.9: Das ISO/OSI-Modell für Kommunikationsstandards (Plate, 2010, Zugriff: 08.07.2010)

[2] **Pro**cess **Fi**eld **Bus**
[3] **O**pen **S**ystems **I**nterconnection
[4] **I**nternational **S**tandard **O**rganisation

PROFIBUS-DP[5] verwendet nur die Layer 1 und 2 sowie das User-Interface[6]. Durch diese schlanke Architektur wird eine relativ schnelle Datenübertragung erreicht. Diese für den schnellen Nutzdatenaustausch optimierte PROFIBUS-Variante ist speziell für die Kommunikation zwischen Automatisierungssystemen und den dezentralen Peripheriegeräten in der Feldebene von Vorteil. (Weigmann und Kilian, 2000)

PROFIBUS-DP wird zur Ansteuerung von Sensoren und Aktoren durch eine zentrale Steuerung in der Fertigungstechnik eingesetzt. Weitere Einsatzgebiete sind die Verbindung von *verteilter Intelligenz*, also die Vernetzung von mehreren Steuerungen untereinander. Datenraten sind bis zu 12 Mbit/s auf verdrillten Zweidrahtleitungen bzw. Lichtwellenleitern möglich. (Wikipedia, 2010, Profibus, Zugriff: 16.06.2010)

Als Standard für die Verbindung der Busteilnehmer über die Busleitung wird in der PROFIBUS-Norm EN 50170 ein 9-poliger D-Sub-Stecker (Abb. 5.10 und 5.11) vorgesehen. Der D-Sub-Stecker mit den Buchsen-Kontakten befindet sich hierbei am Busteilnehmer und der mit dem Stift-Kontakten am Buskabel.

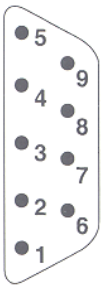
| Ansicht | Pin-Nr. | Signalname | Bezeichnung |
|---|---|---|---|
|  | 1 | SHIELD | Schirm bzw. Funktionserde |
| | 2 | M24 | Masse der 24 V-Ausgangsspannung (Hilfsenergie) |
| | **3** | **R×D/T×D-P** | **Empfangs-/Sendedaten-Plus B-Leitung** |
| | 4 | CNTR-P | Signal für Richtungssteuerung-P |
| | **5** | **DGND** | **Datenbezugspotential (Ground)** |
| | **6** | **VP** | **Versorgungsspannung-Plus** |
| | 7 | P24 | 24V-Plus der Ausgangsspannung (Hilfsenergie) |
| | **8** | **R×D/T×D-N** | **Empfangs-/Sendedaten-Minus A-Leitung** |
| | 9 | CNTR-N | Signal für Richtungssteuerung-N |



Abbildung 5.10: Kontaktbelegung des 9-poligen D-Sub-Steckers (Weigmann und Kilian, 2000)

In Ergänzung zum beidseitigen Busleitungabschluss der Datenleitungen des EIA-485-Standards[7] besteht der PROFIBUS-Leitungsabschluss zusätzlich aus einem Pulldown-

[5] **D**ezentrale **P**eripherie

[6] Durch das User Interface sind die nutzbaren Anwendungsfunktionen sowie das System- und Geräteverhalten der verschiedenen PROFIBUS-DP-Gerätetypen festgelegt.

[7] Auch als RS-485 bekannt. Es ist ein Schnittstellen-Standard für digitale leitungsgebundene, differentielle, serielle Datenübertragung. Aufgrund der symmetrischen Signalübertragung ist EIA-485 durch eine hohe Toleranz gegenüber elektromagnetischen Störungen gekennzeichnet. (Wikipedia, 2010, EIA-485, Zugriff: 01.07.2010)

Abbildung 5.11: Offener D-Sub-Stecker

Widerstand gegen das Datenbezugspotential (DGND) und einem Pullup-Widerstand gegen den Versorgungsspannungs-Pluspol (VP). Durch diese beiden Widerstände wird ein definiertes Ruhepotential auf der Busleitung sichergestellt, wenn kein Busteilnehmer sendet. Die benötigten Busleitungsabschlusswiderstände sind in fast allen Standard-PROFIBUS-Busanschlusssteckern integriert und müssen nur durch Einlegen von Schaltern aktiviert werden.

### Master-Slave-Verfahren

Das Master-Slave-Verfahren ermöglicht es dem Master („aktiver“ Busteilnehmer), der die Sendeberechtigung hat, die ihm zugeordneten Slaves („passiver“ Busteilnehmer) anzusprechen. Demnach hat der Master die Möglichkeit, Nachrichten an die Slaves zu übermitteln bzw. von ihnen abzuholen. Dabei tauscht der „aktive“ DP-Master in zyklischer Reihenfolge Informationspakete mit den „passiven“ DP-Slaves aus.

## 5.4 Serielle Schnittstelle RS-232

Die serielle Schnittstelle dient dem Datenaustausch zwischen Rechnern und Peripheriegeräten. Bei der seriellen Datenübertragung werden die Bits seriell (nacheinander) über eine Verbindungsleitung übertragen. In der Regel spricht man von der EIA[8]-RS[9]-232-Schnittstelle, wenn ohne nähere Kennzeichnung von einer seriellen Schnittstelle gesprochen wird.
Die Bedeutung und Anordnung der Bits wird bei der RS-232 mit einem sogenannte UART[10]

[8] **E**lectronic **I**ndustries **A**lliance
[9] **R**ecommended **S**tandard
[10] **U**niversal **A**synchronous **R**eceiver **T**ransmitter

(*Motorola*-Bezeichnung) oder USART[11] (*Intel*) vorgegeben. Beispiele für moderne serielle Schnittstellen sind Ethernet, USB[12], Firewire, CAN-BUS[13] oder RS-485 (siehe Abschnitt 5.3), allerdings werden diese nicht als serielle Schnittstellen bezeichnet. (Wikipedia, 2010, Serielle_Schnittstelle, Zugriff: 10.07.2010)

In der vorhandenen Sortieranlage geschieht die Kommunikation zwischen der Linearachse und dem Steuer-PC über die serielle Schnittstelle RS-232. Dafür ist die Schrittmotor-Antriebseinheit direkt über eine 3-Draht-Leitung mit dem seriellen Port (COM 1) des Steuer-PCs verbunden.

Für die Unterstützung der synchronen, seriellen Kommunikation in *CoDeSys V3* wird die Bibliothek `SysCom.library`[14] benötigt. Sie ist nach Auswahl des Befehls **Bibliotheks-Repository** im **Tools**-Menü unter den standardmäßig installierten Bibliotheken zu finden.

### 5.4.1 SysCom-Funktionen

Um einen seriellen Port zu öffnen und zu schließen, Daten über diesen Port zu schreiben oder zu lesen und seine Einstellungen zu lesen bzw. zu verändern, bietet die `SysCom.library`-Bibliothek u. a. folgende Funktionen:

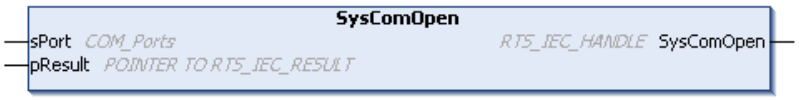



Abbildung 5.12: Funktion SysComOpen

**`SysComOpen:`** Diese Funktion öffnet einen seriellen Port.
Der Rückgabewert vom Typ `RTS_IEC_HANDLE` liefert ein *Handle* auf den Port, das beim Aufruf der anderen Bibliotheksfunktionen übergeben wird. Wenn der Port nicht geöffnet werden kann, wird `0xFFFFFFFF` als Handle zurückgegeben. Zusätzlich wird ein Fehler-Code ausgegeben, der über den Erfolg der Operation Auskunft gibt.

[11] **U**niversal **S**ynchronous/**A**synchronous **R**eceiver **T**ransmitter
[12] **U**niversal **S**erial **B**us
[13] **C**ontroller **A**rea **N**etwork - **Bus**. Der CAN-Bus ist ein asynchrones, serielles Bussystem und gehört zu den industriellen Feldbussen. Um die Kabelbäume (bis zu 2 km in Fahrzeugen) zu reduzieren und dadurch Gewicht zu sparen, wurde der CAN-Bus 1983 von der Firma *Bosch* für die Vernetzung von Steuergeräten in Automobilen entwickelt und 1987 zusammen mit Intel vorgestellt. (Wikipedia, 2010, Controller Area Network, Zugriff: 10.07.2010)
[14] Für eine detaillierte Beschreibung der `SysCom.library`-*Enumerationen* und -*Strukuren* siehe 3S-Dokumentation: `SysCom_V3x_D.pdf`

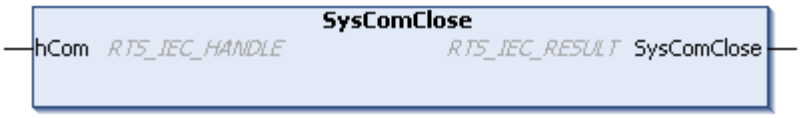



Abbildung 5.13: Funktion SysComClose

**`SysComClose:`** Diese Funktion schließt einen geöffneten COM-Port.
Der Rückgabewert vom Typ `RTS_IEC_RESULT` gibt Auskunft über den Erfolg der Operation.


Abbildung 5.14: Funktion SysComSetSettings

**`SysComSetSettings:`** Diese Funktion setzt für einen seriellen Port die *Standardparameter* (Baudrate, Stoppbits, Parität, Funktions-Timeout, Buffer-Größe etc.). Dies geschieht über die Strukturvariable `COMSettings`.


Abbildung 5.15: Funktion SysComPurge

**`SysComPurge:`** Diese Funktion leert den Ein- und Ausgangspuffer der seriellen Schnittstelle.

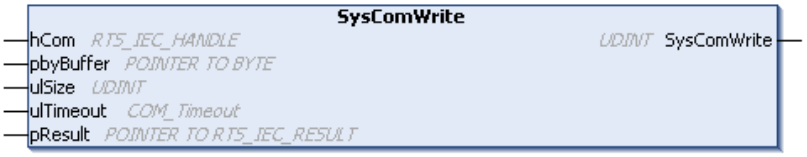



Abbildung 5.16: Funktion SysComWrite

**`SysComWrite:`** Diese Funktion schreibt Daten an den angegebenen Port. Dazu werden die Port-Nummer, die Adresse, von der die Daten kopiert werden sollen, die gewünschte Größe der zu schreibenden Daten und ein Timeout für die Funktion angegeben.


Abbildung 5.17: Funktion SysComRead

**SysComRead:** Diese Funktion liest Daten vom COM-Port. Dazu werden das Handle der seriellen Schnittstelle, die Adresse, an die die gelesenen Daten kopiert werden sollen, die gewünschte Größe der zu lesenden Daten und ein Timeout für die Funktion angegeben. Der Rückgabewert vom Typ `UDINT` liefert die Anzahl der tatsächlich gelesenen Bytes.

### 5.4.2 SysCom-Strukturen und -Enumerationen

Die Funktionen der `SysCom.library`-Bibliothek verwenden u. a. folgende Strukturen und Enumerationen:

**ComSettings:** Diese Struktur enthält die Standardparameter eines COM-Ports. Diese können über die Funktionen `SysComSetSettings` bzw. `SysComGetSettings` gesetzt bzw. gelesen werden.

**COM_Ports:** Diese Enumeration verwaltet den in der Struktur `ComSettings` verwendeten Parameter `sPort` für die Port-Nummern, die als Eingabe für die Funktionen der `SysCom.library` benötigt werden.

**COM_Baudrate:** Diese Enumeration verwaltet die Werte für den in der Struktur `ComSettings` verwendeten Parameter `ulBaudRate`.

**COM_StopBits:** Diese Enumeration enthält Werte für den in der Struktur `ComSettings` verwendeten Parameter `byStopBits` zur Definition, wie viele Stoppbits im Anschluss an jedes übertragene Daten-Byte gesendet werden.

**COM_Parity:** Diese Enumeration verwaltet die Werte für den in der Struktur `ComSettings` verwendeten Paritäts-Parameter `byParity`.

**COM_Timeout:** Diese Enumeration enthält Werte für den in der Struktur `ComSettings` verwendeten Parameter `ulTimeout`. (3S, 2009)

In dem nachfolgenden Kapitel wird die Zerlegung des Applikationsbeispiels für den objektorientierten Entwurf geschildert.

# 6 Wiederverwendung und Modularität

> „*Je höher die Variabilität, desto größer ist die Wiederverwendbarkeit der Komponente. [···] Zwei Ziele sind für die Realisierung von Variabilität wichtig: Änderungseffizienz und Verständlichkeit*“ (Vogel-Heuser und Wannagat, 2009)

Für die Wiederverwendbarkeit der Software ist sowohl die Verständlichkeit der Programme entscheidend als auch der mit der Software-Neuanpassung verbundene Aufwand. Die Nutzung von Modulen fördert beide Aspekte: die Änderungseffizienz und die Verständlichkeit. Ein System ist *modular* aufgebaut, wenn es aus abgrenzbaren Einheiten (Bausteinen) zusammengesetzt ist und wenn diese Einheiten einzelnd ausgetauscht, verändert oder hinzugefügt werden können, ohne dass andere Teile des Systems hierdurch beeinflusst werden oder das System arbeitsunfähig wird. Die Beziehungen zwischen Modulen werden durch ihre Schnittstellen festgelegt. Solche universell einsetzbaren Module reduzieren die Komplexität in der Handhabung der Software und erhöhen die Verständlichkeit.

## 6.1 Objektorientierte Softwarestrukturierung

In der Prozessautomatisierung sind in der Regel eine große Anzahl an Feldgeräten (Sensoren und Aktoren) anzusteuern. Um die beträchtliche Größe der Steuerungssoftware effektiv zu gestalten, ist es sinnvoll, diese Software mit Hilfe des objektorientierten Ansatzes zu strukturieren.

Objektorientiertheit bedeutet, die Objekte der realen Welt in der Software nachzubilden. In der Fabrik- und Prozessautomatisierung sind die Objekte die Feldgeräte in der Anlage. Jedem Feldgerät (Objekt) wird ein Feldgerätetyp (Klasse) zugeordnet, der als Funktionsbaustein zu programmieren ist. Die Feldgeräte unterscheiden sich in der Regel nur durch relativ wenige Feldgerätetypen voneinander. Diese Anlagenstruktur wird also durch folgende Modularisierung in der Anwendersoftware nachgebildet:

- Für jedes Feldgerät in der Anlage, z. B. den Heber des XY-Schiebers (siehe Abschnitt 6.3), wird ein eigenes Ansteuerprogramm in der SPS erstellt.
- Für die Daten jedes Feldgeräts wird eine globale Strukturvariable angelegt.
- Für jeden Feldgerätetyp, z. B. den Pneumatikzylinder (siehe Abschnitt 6.3), wird mindestens ein Anwender-Funktionsbaustein entwickelt.

- Für die Daten jedes Feldgerätetyps wird ein Anwender-Datentyp angelegt.

Durch die Vielzahl der Feldgeräte entstehen so zwar viele Programme und globale Variablen, aber die Liste der erstellten Programme und die der globalen Variablen entspricht dabei exakt der Feldgeräteliste des Anlagenplaners. (Seitz, 2008)

Somit erreicht man eine eindeutige Abbildung zwischen der realen Welt in der Anlage und der Steuerungssoftware, was sehr zur Verständlichkeit der SPS-Programme beiträgt. Ziel ist es jedoch, mit möglichst wenigen Funktionsbausteinen (Klassen) eine Vielzahl von Geräten (Objekten) ansteuern zu können.

## 6.2 Strukturierte SPS-Programmierung

Auf Basis der objektorientierten Softwarestrukturierung hat sich nach dem 2008 veröffentlichten Buch „Speicherprogrammierbare Steuerungen“ von Seitz bei der Entwicklung der Automatisierungssoftware folgendes Vorgehen als zweckmäßig erwiesen:

1. Softwarestrukturierung:
   a) Pro Feldgerätetyp werden ein Funktionsbaustein und ein Anwender-Datentyp definiert.
   b) Pro Feldgerät sind ein Ansteuerprogramm und eine globale Strukturvariable festzulegen.
2. Entwurf der Funktionsbausteine und Funktionen:
   a) Deklaration der Ein- und Ausgangsvariablen.
   b) Konzeption der Schaltungslogik.
3. Entwurf der Programme:
   a) Deklaration der Prozessregelvariablen (Messsignale $x_e$ bzw. Stellsignale $y_a$) und ihrer E/A-Zuordnung.
   b) Deklaration der globalen Variablen für Steuer- und Statussignale ($u$ bzw. $v$) möglichst für jeden Gerätetyp gesammelt als globale Strukturvariable.
   c) Instantiierung der „entworfenen“ Funktionsbausteine.
   d) Entwurf der Zusatzlogik, die nicht im Funktionsbaustein vorhanden, aber zur Ansteuerung erforderlich ist.
4. Implementierung der Software:
   a) Implementierung der entworfenen Programme, FCs, FBs, Datentypen und Variablen.

b) Steuerungskonfiguration festlegen.

c) Taskzuordnung der implementierten Programme.

5. Test und Inbetriebnahme:

a) Test der implementierten Software in der Simulation.

b) Laden der Software in die SPS.

c) Inbetriebnahme von Software und SPS im Zusammenspiel mit der Anlage.

d) Protokollierung des Tests.

Diese Vorgehensweise der strukturierten Programmierung wird anhand der zu automatisierenden Sortieranlage veranschaulicht.

## 6.3 Identifikation der Module

> *„Bei der Identifikation der Module ist darauf zu achten, dass diese möglichst eine abgrenzbare Funktionalität aufweisen, die in Methoden einer Klasse zusammengefasst werden können.“* (Vogel-Heuser und Wannagat, 2009)

Angesichts der Anforderungen zur objektorientierten Softwarestrukturierung ist es sinnvoll, die Struktur der Anlage zu modularisieren.

Die Sortieranlage lässt sich in drei grundlegende Aggregate (Module) unterteilen:

- Ausschieber
- XY-Schieber
- Linearachse,

wobei die Sortierstation als Endziel des vermessenen Werkstückes dient, d. h. sie ist ein „passives“ Aggregat, das keine weiteren Funktionen bereitstellt. Im Gegensatz dazu ist der Triangulationslaser ein Funktionsbaustein, welcher zur Laufzeit von der Linearachse aufgerufen wird und einige Aufgaben erledigt.

In Hinsicht auf ihre Funktionalität haben die Aggregate der Anlage die Betriebsarten *Initialisierung*-, *Automatik*- und *Stopp*-Betrieb anzubieten, das ein Kriterium für die Wahl von geeigneten Modulen ist.

Innerhalb des XY-Schiebers fällt der häufige Einsatz von „gleichartigen“ Pneumatikzylindern in unterschiedlichen Funktionen, nämlich die der *Schieber* und die der *Heber*. Mit dem Schieber wird der XY-Schieber in die Y-Richtung bewegt. Der Heber ist für die Greifbewegung

zuständig. Die zwei Pneumatikzylinder der XY-Schieber eignen sich also für die Betrachtung als ein Feldgerätetyp bzw. Elementarmodul. Um die Pneumatikzylinder zu einem Modul bzw. zu einer Klasse zusammenfassen zu können, muss gewährleistet sein, dass genügend Gemeinsamkeiten vorhanden sind.

Der Pneumatikzylinder besteht aus einem zylindrischen Körper und einem Kolben, der durch Druckluft bewegt wird und dessen Endlagen durch induktive Sensoren erkannt werden. Die Pneumatikzylinder mit zwei Ventilen besitzen zwei Druckkammern, welche den Kolben entlang des Druckgefälles treiben.
Diese Kombination aus Aktoren und Sensoren wird zu einem Modul Pneumatikzylinder zusammengefasst und bietet die Funktionalitäten *Kolben einfahren* und *Kolben ausfahren*. Mit ihnen lassen sich die Werkstücke sowohl von dem Ausschieber zur Linearachse transportieren und von der Linearachse zur Sortierstation bringen als auch hochheben und runtersetzen.

Der Ausschieber hat einen pneumatischen Zylinder in der Rolle eines Vereinzlers sowie mehr oder minder einen induktiven Sensor, um das Vorhandensein eines Werkstückes festzustellen. Bemerkenswert dabei ist allerdings, dass er nur einen „einzigen“ Sensor bzw. Aktor besitzt.

Damit und mit den Beschreibungen der einzelnen Gewerke aus Abschnitt 4.2 ist die Sortieranlage vollständig beschrieben. Bei größeren Anlagen können durchaus weitere Abstraktionen der Module sinnvoll sein. Die Vorgehensweise ändert sich aber nicht.

Eine ausführliche Herleitung aller implementierten Module, ihrer Schnittstellen, Methoden und Aktionen würde den Rahmen dieser Diplomarbeit sprengen. Daher wird nur auf ausgewählte Teile der Implementierung eingegangen, aus denen die Bedeutung der objektorientierten SPS-Programmierung und ihre Merkmale hervorgehen.

## 6.4 Schnittstellen der Module

Auf Basis der bisherigen groben Modulstruktur werden nun die benötigten Schnittstellen der Sortieranlage ermittelt. Mit der Definition der Schnittstellen (siehe Abschnitt 3.4) wird festgelegt, auf welche Weise die einzelnen Module interagieren werden. Ein Modul kann auch mehrere Schnittstellen implementieren. Die mechanischen und elektrischen Schnittstellen sind bereits vorgegeben. So besitzen alle Aktoren und Sensoren der Anlage eine *Spannungsschnittstelle*, die für analoge Signale von 0 – 10 V und für digitale Signale von 0 – 24 V ausgelegt sind. In der Steuerung sind sie dann als Variablen eines bestimmten Datentyps verfügbar. Sie bilden damit die Schnittstelle zum technischen System für das Steuerungsprogramm.

Die Operationen innerhalb der Betriebsarten und die darin definierten Zustände werden als Methoden bzw. Eigenschaften in einer Schnittstelle zusammengefasst. Die Implementierung der Methoden, die in der Schnittstelle definiert werden (z. B. `METH_Init()`, `METH_Automatik()`, `METH_Stop()`) muss für jedes Betriebsartenmodul individuell ausprogrammiert werden.
Um entsprechend in die unterschiedlichen Betriebsarten schalten zu können, muss jedes Aggregat der Anlage Methoden für den Initialisierung-, Automatik- und Stopp-Betrieb implementieren. Die Schnittstelle beschreibt also nur die von den Modulen bereitzustellenden Methoden. Erst in den Modulen, die die Schnittstelle implementieren, werden die Methoden individuell ausprogrammiert und es wird festgelegt, welche *Aktionen* auszuführen sind, wenn beispielsweise der Anlagenteil gestartet oder gestoppt wird.

Die geeignete Koordination der Module führt zum Ablauf des Gesamtprozesses. Der Prozess besteht im Wesentlichen daraus das Werkstück in der korrekten Reihenfolge zu der Linearachse zu befördern, dort mit Hilfe des Triangulationslasers vermessen zu lassen und letztendlich in der Sortierstation auszusortieren. Die Koordination der Module untereinander wird daher maßgeblich durch die Übergabe des Werkstückes bestimmt. Es bietet sich also an eine einheitliche *Werkstückschnittstelle* zu definieren, die jede Station implementiert, die ein Werkstück aufnehmen oder abgeben kann.
Trotz der individuellen Abläufe bei der Übergabe des Werkstückes ist die Schnittstelle für alle Module identisch.

In der Abbildung 6.1 sind die zu implementierenden Schnittstellen abgebildet, wobei `IAggregat`, `IWerkstueck` und `IZylinder` von dem jeweils implementierenden Modul individuell ausprogrammiert werden muss.

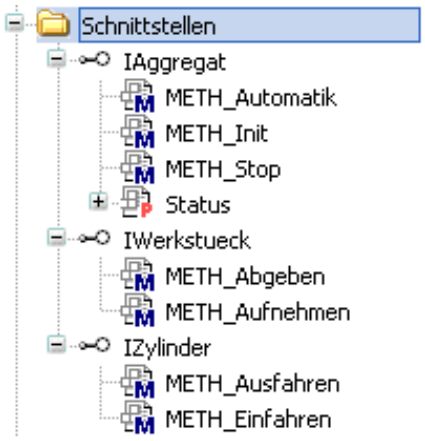



Abbildung 6.1: Schnittstellen der Module

# 7 Realisierung der Module

Die bisher bekannten „einfachen“ Funktionsbausteine sind deutlich in *CoDeSys V3* erweitert worden, sodass sie nahezu den Klassen in der objektorientierten Programmierung entsprechen (siehe Abschnitt 3.6). Methoden haben ein ähnliches Verhalten wie Funktionen. Es lassen sich Eingangsparameter übergeben und ein Ausgabewert zurückliefern. Sie werden im Gegensatz zu den Aktionen innerhalb eines SPS-Zyklus berechnet. Es ist daher in *CoDeSys* auch nicht möglich, die Ablaufsprache (`AS`) für Methoden-Programmierung zu verwenden.

Methoden eignen sich aufgrund ihrer Charakteristik als Schnittstelle zu anderen Funktionsbausteinen. Am Beispiel des zweifach eingesetzten Pneumatikzylinders (Abb. 7.1) legen die Aktionen `act_Ausfahren` bzw. `act_Einfahren` den Vorgang fest, dass der Kolben aus- bzw. eingefahren wird. Die Methoden `METH_Ausfahren` bzw. `METH_Einfahren`, die von der Schnittstelle `IZylinder` „geerbt“ wurden, werden eingesetzt, um diese Aktionen auszulösen. Der entsprechende Befehl für den Funktionsblock `FB_2VentilZylinder` sieht folgendermaßen aus:

```
FUNCTION_BLOCK FB_2VentilZylinder IMPLEMENTS IZylinder
```

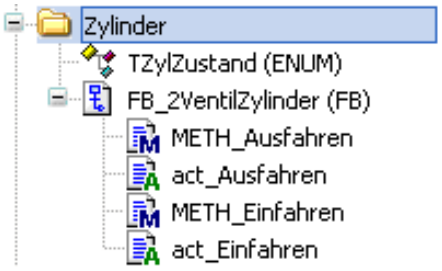



Abbildung 7.1: FB_2VentilZylinder und zugehörige Methoden und Aktionen

Die benutzerdefinierte Enumeration `TZylZustand` gibt den aktuellen Zustand des 2-Ventile-Pneumatikzylinders an und enthält die Zylinderzustände `stehend`, `einfahren` und `ausfahren`.

In diesem Zusammenhang ist es für die aufrufende Klasse wichtig, eine Rückmeldung über den Stand der eingeleiteten Aktion zu erhalten. Diese Statusmeldung kann über den Rückgabeparameter der Methode erfolgen.

Hierfür wird die benutzerdefinierte Enumeration `TJobZustand` aus dem 2009 veröffentlichten Buch „Modulares Engineering und Wiederverwendung mit CoDeSys V3“ von Vogel-Heuser und Wannagat in das Sortieranlage-Projekt übernommen. Dies enthält die drei Zustände `bereit`, `belegt` und `erledigt`. Ist im Falle des Pneumatikzylinders der Kolben ausgefahren und wird die Methode `METH_Einfahren` aufgerufen, so gibt die Methode den Wert `bereit` zurück. Wäre der Zylinder bereits vorher eingefahren, würde die Methode den Wert `erledigt` zurückliefern. In dem Fall, dass der Zylinder noch beim Verfahren ist, also der Kolben gerade bewegt wird, gibt die Methode `belegt` zurück.
Auf diese Weise kann die aufrufende Klasse durch wiederholte Methodenaufrufe den aktuellen Stand der eingeleiteten Aktion überprüfen.

## 7.1 Definition der Klassen

Im Folgenden wird die bereits identifizierte Modulstruktur der Sortieranlage durch *Klassen* nachgebildet, welche in *CoDeSys* durch die speziell erweiterten Funktionsbausteine repräsentiert werden.

**Aggregate**
Alle drei Aggregate *Ausschieber*, *XY-Schieber* und *Linearachse* sind Erben einer gemeinsamen Klasse `FB_Aggregat`. In dieser Klasse werden die gemeinsamen Eigenschaften und Funktionalitäten zusammengefasst. Jedes Aggregat wird innerhalb der einzelnen Betriebszustände der Sortieranlage aufgerufen. So wird beispielsweise im Anlagenzustand `aggStatus.Init` die Methode `METH_Init` aller drei Aggregate aufgerufen, die die Initialisierungs-Aktion auslöst (Abb. 7.2).

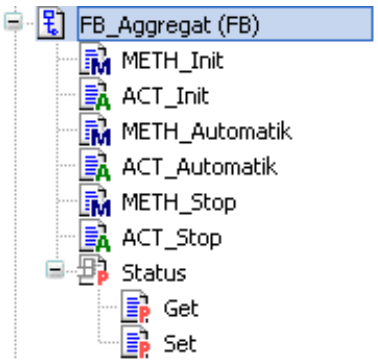



Abbildung 7.2: FB_Aggregat und zugehörige Methoden, Aktionen und Eigenschaften

In `FB_Aggregat` wird zusätzlich die Schnittstelle `IAggregat` implementiert, d. h. die Methoden, die dort definiert sind, sind hier vorhanden. Der Code dafür sieht wie folgt aus:

```
FUNCTION_BLOCK FB_Aggregat IMPLEMENTS IAggregat
```

**Ausschieber**

Der Ausschieber hat die einzige Aufgabe, die zylinderförmigen Werkstücke aus dem Materialspeicher herauszuschieben. Dafür bietet er die Methode `METH_Ausstossen`, die eine entsprechende Rückmeldung an das fragende Aggregat zurückliefert. Zusätzlich implementiert er die im Abschnitt 6.4 vorgestellte Werkstückschnittstelle `IWerkstueck` (Abb. 7.3). Der entsprechende Befehl zur Erweiterung der Klasse `FB_Aggregat` und zur Implementierung der Schnittstelle lautet:

```
FUNCTION_BLOCK FB_Ausschieber EXTENDS FB_Aggregat
IMPLEMENTS IWerkstueck
```

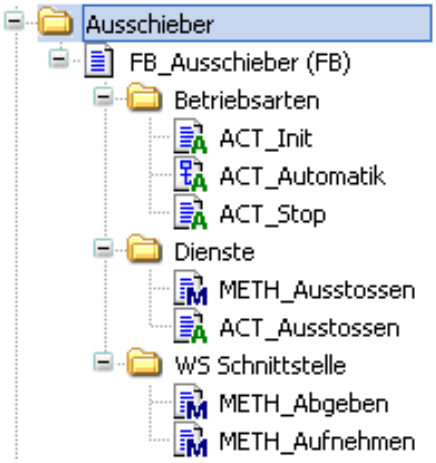



Abbildung 7.3: FB_Ausschieber und zugehörige Methoden und Aktionen

Das Schlüsselwort `EXTENDS` zeigt die Vererbung an. Im Implementierungsteil des Funktionsbausteins `FB_Ausschieber` verweist der Bezeichner `SUPER^` auf den Programmtext des „Eltern“-Bausteins `FB_Aggregat`.

An dieser Stelle ist bei dem Ausschieber anzumerken, dass sein Ausstoßer eine vereinfachte Variante der Klasse `FB_2VentilZylinder` implementieren könnte, die an Stelle der zwei Sensoren und zwei Aktoren nur einen einzigen Sensor sowie einen einzigen Aktor besitzt. Diese einfache Klasse könnte dann die „Elternklasse“ für die Klasse `FB_2VentilZylinder` sein.

**XY-Schieber**

Der XY-Schieber enthält zwei „zylinderartige“ Komponenten, der Schieber und der Heber, die es erlauben, den XY-Schieber in Y-Richtung zu verfahren bzw. den Werkstücksgreifer (Heber) hoch und runter zu setzen. Der Schieber und der Heber sind also Feldgeräte vom Feldgerätetyp `FB_2VentilZylinder`. Hinzu kommen noch:

- ein Ventil, welches das Ansaugen von Werkstücken erlaubt,
- ein Sensor, der das angesaugte Teil detektiert,

- fünf Sensoren, die aktiviert werden, sobald der XY-Schieber eine der Ablagepositionen erreicht
- und die zwei Sensoren, die die Lage des XY-Schiebers überprüfen, ob dieser sich an der Seite der Linearachse oder an der des Ausschiebers befindet.

Der XY-Schieber ist mehr oder weniger ein Koordinator des Prozessablaufs. Er interagiert mit den anderen Modulen bezüglich der Bereitschaft Werkstücke abzugeben bzw. aufzunehmen.
Die Werkstückschnittstelle `IWerkstueck` vereinheitlicht die Übergabe des Werkstücks zwischen dem XY-Schieber und den Aggregaten. Sie enthält die Methoden `METH_Abgeben` und `METH_Aufnehmen` (siehe Abb. 6.1 auf Seite 52), die als Rückgabewert `bereit`, `belegt` oder `erledigt` vom Datentyp `TJobZustand` liefert.
Die Methoden, Aktionen und Eigenschaften des XY-Schiebers sind in Abbildung 7.4 dargestellt.

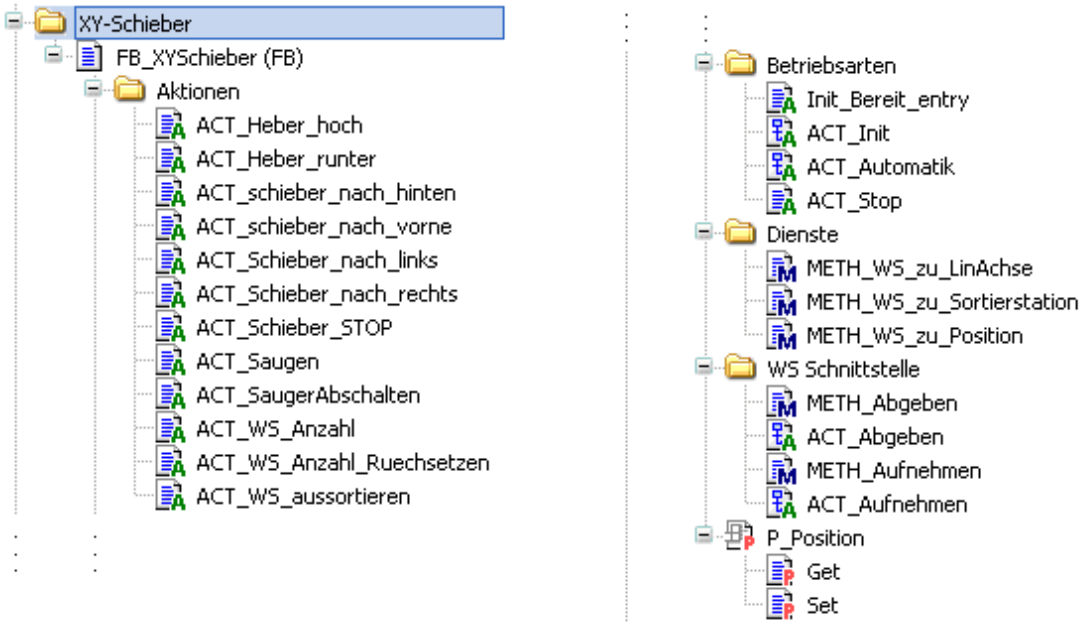



Abbildung 7.4: FB_XYSchieber und zugehörige Methoden, Aktionen und Eigenschaften

**Linearachse**
Die Linearachse muss, wie im Abschnitt 4.2.3 erläutert, einige Methoden und Aktionen implementieren (Abb. 7.5), um das gewünschte Verhalten zu erzielen. Beispielsweise muss sie beim Starten der Anlage zuerst den Referenzpunkt anfahren, der als Bezugspunkt zur Positionierung des Schlittens dient. Dafür muss aber zuvor Folgendes durchgeführt sein:

- Datenübertragungsparameter für die serielle Kommunikation festlegen.
- Seriellen COM-Port öffnen.

- Achsenanzahl setzen.
- Referenzgeschwindigkeit setzen.

Danach kann erst der Befehl "`@0R1 LF+CR`" zur Durchführung der Referenzfahrt erfolgen. Wenn der COM-Port geöffnet und die Referenzfahrt beendet ist, können dann die eigentlichen Verfahrbefehle von bzw. zur Linearachse geschickt werden (siehe Abschnitt 4.2.3).

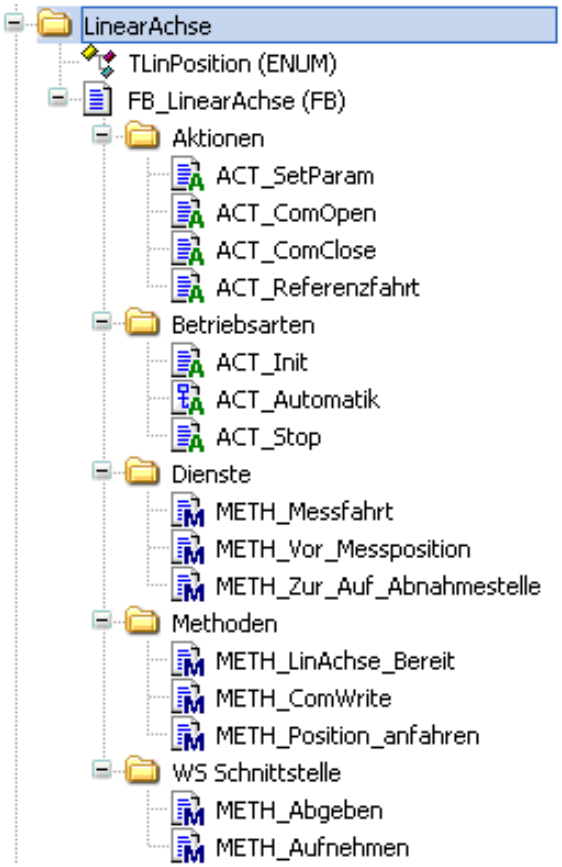



Abbildung 7.5: FB_LinearAchse und zugehörige Methoden und Aktionen

Die Linearachse ist das einzige Aggregat in der Sortieranlage, welches über eine serielle Schnittstelle mit dem Steuer-PC verbunden ist. Deshalb müssen die Enumerationen und Strukturen der Bibliothek `SysCom.library` innerhalb des `FB_Linearachse`-Funktionsbausteins deklariert sein, um die Funktionen der Bibliothek benutzen zu können (siehe Abschnitt 5.4). Im Folgenden ist ein Ausschnitt aus dem Deklarationsteil des FBs `FB_Linearachse` angezeigt:

```
// RS232 Variablen: // Zeiger auf Fehler-Code
com_result : DWORD;
p_com_result : POINTER TO RTS_IEC_RESULT :=
    ADR(com_result);
```

```
// Aus SysComOpen erhaltenes Handle für den aktuell
   geöffneten Port
h_com : RTS_IEC_HANDLE := RTS_INVALID_HANDLE;

// struktur, die die Standardparameter eines COM-Ports
   enthält
com_settings : SysCom.ComSettings;

// Nummer des Ports, für den die Parameter gelten
com_port : SysCom.COM_Ports;

// Speicheradresse, aus der die zu schreibenden Daten
   geholt werden sollen
com_byBuffer : STRING;
p_com_byBuffer : POINTER TO BYTE;

// Anzahl der zu schreibenden Bytes
com_ulSize : UDINT;

// Zeit in ms, nach der die Funktion spätestens zurückkehrt
com_ulTimeout : SysCom.COM_Timeout := UDINT#500;

// SysComPurge leert den Ein- und und Ausgangspuffer der
   seriellen Schnittstelle
com_purge : DWORD;
p_com_purge : POINTER TO RTS_IEC_RESULT := ADR(com_purge);
```

**Sotrtierstation**

Die simple Sortierstation (Abb. 7.6) enthält fünf metalldetektierende Sensoren, die feststellen, ob sich in den entsprechenden Ablagepositionen (Position 1 – 5) ein metallisches Objekt befindet. Der Funktionsbaustein `FB_Sortierstation` implementiert die Werkstückschnittstelle `IWerkstueck`:

```
FUNCTION_BLOCK FB_Sortierstation IMPLEMENTS IWerkstueck
```

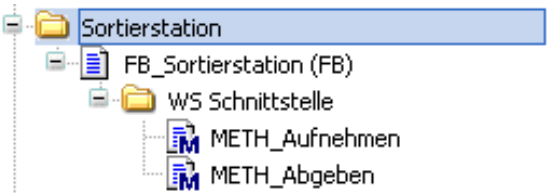



Abbildung 7.6: FB_Sortierstation mit Methoden der Schnittstelle

**Triangulationslaser**

Die Hauptfunktionen des Triangulationslasers sind die Erfassung der analogen Messwerte vom Sensorsystem, die Bereitstellung der Messwerte zur Verarbeitung im SPS-Programm (siehe Abschnitt 4.2.5) als auch die Archivierung der erfassten Höhenprofildaten der Werkstücke.

Der einzulesende Messwert steht in einem Peripherie-Eingangswort von der Analogeingangsbaugruppe (siehe Abschnitt 4.2.5.2) bereit. Dieser Wert sagt aber noch nichts über die Höhe des gemessenen Werkstücks. Um den digitalen Eingangswert in einem äquivalenten Spannungswert für den Messbereich $\pm$ 10 V (siehe Tabelle in Abb. 4.8 auf Seite 34) anzugeben, muss der eingelesene Wert mittels folgender Vorschrift umgerechnet werden:

$$U_{AE}[V] \quad = \quad AE[Bit] \cdot \frac{10\,V}{27648\;Bits}$$

Dabei bezeichnet AE den Digitalwert des Analogeingangs. Der berechnete Spannungswert kann dann mit

$$h[mm] \quad = \quad U_{AE}[V] \cdot \frac{1\;mm}{0,1176\,V}$$

in einem Höhenwert in [mm] angegeben werden.

Dieser vom Triangulationslaser erfasste und umgerechnete Wert gibt den Werkstückabstand vom Lasersensor an. Um diesen zu korrigieren, wird ein *Offset*-Wert des leeren Schlittens der Linearachse (ermittelt: `schlitten_L_offset` = 67 mm) davon subtrahiert. Damit erhält man einen exakten Wert des Werkstückprofils in [mm].

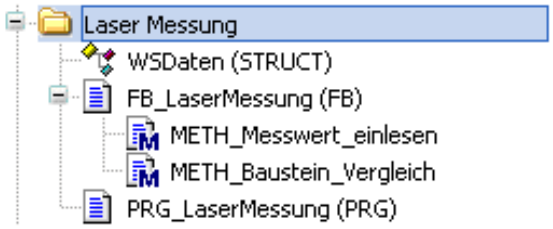



Abbildung 7.7: FB_LaserMessung und zugehörige Methoden

Der Funktionsblock `FB_LaserMessung` enthält zwei Methoden (Abb. 7.7) zur Analogwertverarbeitung:

**`METH_Messwert_einlesen`:** Diese Methode enthält die Berechnungsvorschriften zur Ermittlung der Höhenprofildaten und speichert sie in die entsprechende Werkstück-Strukturvariable `WSDaten` ab.

**`METH_Baustein_Vergleich`:** Mit Hilfe einer *Toleranz*-Variablen werden bei dieser Methode die Werkstücke mit dem Referenzwerkstück verglichen. Dazu wird hier ent-

schieden, ob das aktuelle Werkstück im Vergleich zum Referenzwerkstück gut oder schlecht ist.

Außerdem enthält der Funktionsbaustein `FB_LaserMessung` ein kleines Programm `PRG_LaserMessung`, das in Abhängigkeit von der Werkstücksanzahl entscheidet, um welches Werkstück es sich bei der aktuellen Messung handelt.

Alle bisher vorgestellten Module und Aggregate werden über den Aufruf der Methoden gesteuert, welche alle den aktuellen Zustand (`bereit`, `belegt` oder `erledigt`) des angeforderten Dienstes zurückliefern. Mit den drei grundlegenden Zuständen lässt sich die gesamte Anlage koordinieren, ohne direkt die einzelnen Sensoren oder Aktoren der beteiligten Module abzufragen bzw. zu setzen. Diese Beschreibung ist sehr abstrakt und lässt sich in einem anderen Kontext durch die Unabhängigkeit von der konkreten Realisierung leicht wieder verwenden.

## 7.2 Implementierung der Module

In den IEC-Programmiersprachen strukturierter Text (`ST`) und Ablaufsprache (`AS`) werden die realisierten Programme, Funktionsbausteine, Methoden und IEC-Aktionen implementiert. Die Programmiersprache `ST` zeichnet sich dadurch aus, dass die Syntax ihrer Sprachelemente denen der Hochsprache $C$ sehr ähnelt. Außerdem bietet sie mehr Strukturierungsmöglichkeiten als die SPS-Programmiersprache `AWL`. Sie ist die mächtigste Programmiersprache hinsichtlich des Speicherbedarfs. Die wesentlichen Sprachelemente des strukturierten Textes sind:

- Logische oder arithmetische Ausdrücke,
- Zuweisungen,
- Aufrufe von Programmen, Funktionsbausteinen, Funktionen oder Aktionen.

Auf der anderen Seite dient die Ablaufsprache `AS` zur Programmierung in Form eines Petri-Netzes (Zustandsautomaten). Vorteilhaft ist bei der Ablaufsprache, dass durch die Aufeinanderfolge von Schritten eindeutige Abläufe erzwungen werden. Eine Ablaufsteuerung ist eine geschlossene Kette von

- Steuerungsschritten mit angehängten Aktionen,
- Transitionen,
- Gerichtete Verbindungen.

Bevor ein Funktionsbaustein in AS erstellt werden kann, muss die entsprechende Bibliothek IecSfc.library im *CoDeSys*-Projekt eingebunden werden.

AS-Schritte (Stellen) können die Zustände *aktiv* oder *inaktiv* annehmen. Ein aktiver Schritt wird im Online-Betrieb durch dunkle Einfärbung markiert. Zu AS-Schritten können Aktionen hinzugefügt werden. Man unterscheidet bei der Ablaufsteuerungs-Programmiersprache drei Arten von Aktionen:

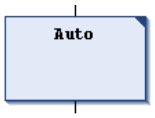



Abbildung 7.8: Schritt-Aktion

**Schritt-Aktion:** Solange der zugehörige Schritt aktiv ist, wird die Aktion in jedem SPS-Zyklus einmal ausgeführt. Ein AS-Schritt mit einer Aktion ist durch ein kleines Dreieck in der oberen rechten Ecke gekennzeichnet.

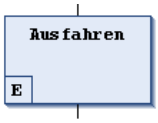



Abbildung 7.9: Eingangs-Aktion

**Eingangs-Aktion:** In dem auf die Aktivierung des AS-Schrittes folgenden SPS-Zyklus wird sie eimalig ausgeführt und zwar vor der Ausführung der Schrittaktion. Eine Eingangsaktion ist durch den Buchstaben E in der linken unteren Ecke angezeigt.


Abbildung 7.10: Ausgangs-Aktion

**Ausgangs-Aktion:** In dem auf die Deaktivierung des AS-Schrittes folgenden SPS-Zyklus wird sie einmalig ausgeführt und zwar vor der Ausführung der Eingangsaktion der neu aktivierten Schritte. Sie ist durch den Buchstaben X in der rechten unteren Ecke angezeigt.

Die vorgestellten Aktionen sind Erweiterungen des von der IEC 61131-3 vorgesehenen Aktionsbegriffs. IEC-konforme Aktionen können unabhängig vom übergeordneten Schritt aktiv oder inaktiv sein. Sie verfügen über einen sogenannten *Qualifizierer* (engl. *Qualifier*), der

über ihren Status entscheidet und darüber, in welcher Art sie ausgeführt werden sollen. (Vogel-Heuser und Wannagat, 2009)

Nachdem alle Klassen, Methoden und Aktionen in den vorangegangenen Abschnitten angelegt worden sind und damit die Struktur der gesamten Anlage feststeht, werden die entsprechenden POEs programmiert.
Da eine ausführliche Erläuterung aller programmierten POEs nicht in den Rahmen dieser Diplomarbeit passen würde, wird deshalb nur auf ausgewählte Teile der Programmierung eingegangen.

**FB_2VentilZylinder**

Der Grundablauf des Pneumatikzylinders repräsentiert die Zustände und die Zustandsänderungen. Die Grundzustände sind `Eingefahren` und `Ausgefahren`. Die Zustandsänderungen werden durch die Aktionen `act_Einfahren` und `act_Ausfahren` beschrieben. Eingeleitet werden die Aktionen durch die beschriebenen Methoden (siehe Anfang dieses Kapitels).
Der Funktionsbaustein `FB_2VentilZylinder` wird in `AS` programmiert (Abb. 7.11).

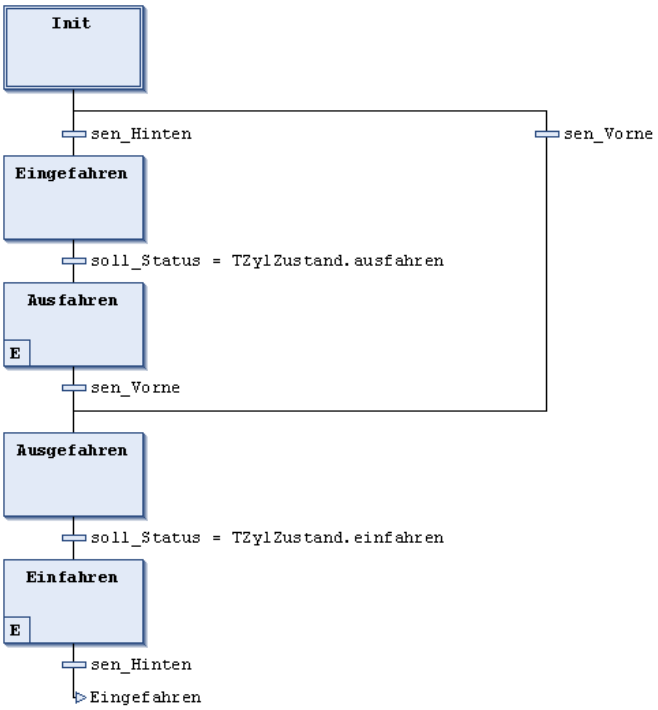



Abbildung 7.11: Zustände des Pneumatikzylinders in der Ablaufsprache AS

Um ein Beispiel für Methoden-Programmierung zu bringen, wird der Programmcode der Methode `METH_Ausfahren` gezeigt:

```
METH_Ausfahren := TJobZustand.belegt;

IF Eingefahren.x THEN
   METH_Ausfahren := TJobZustand.bereit;

   IF Ausfuehren THEN
      soll_Status := TZylZustand.ausfahren;
   END_IF
END_IF

IF Ausgefahren.x THEN
   METH_Ausfahren := TJobZustand.erledigt;
END_IF
```

Die Methode METH_Ausfahren dient dazu, den „eingefahrenen“ Zylinder auszufahren. Sobald die Methode METH_Ausfahren aufgerufen wird, liefert sie zuerst die Rückmeldung belegt. Dann wird mit der Bedingung Eingefahren.x[1] geprüft, ob der Schritt Eingefahren aktiv ist. Wenn ja, dann liefert die Methode den Status bereit zurück und setzt den soll_Status des Zylinders auf ausfahren, welches als Transitionsbedingung ausreichend ist, um den folgenden Schritt zu aktivieren und damit die Statusänderung Ausfahren zu markieren. Wenn nein, wird geprüft, ob der Schritt Eingefahren aktiv ist. Ist er aktiv, wird der Status erledigt an den aufrufenden Baustein zurückgemeldet.

In den Aktionen act_ Einfahren und act_Ausfahren, die dem Funktionsbaustein FB_2VentilZylinder gehören, werden die lokalen Variablen gesetzt, die den entsprechenden Ausgängen im Hauptprogramm in der Aktion ACT_EAs zugewiesen sind.

**Hauptprogramm**

Das Hauptprogramm ist das Kernstück der Anlagesteuerung. Es koordiniert die Aufrufe der Funktionsbausteine sowie deren Methoden. Es gilt auch als Schnittstelle zwischen der Steuerungssoftware und den Ein- und Ausgängen der Sortieranlage oder der externen Bedienung. Im Hauptprogramm kann zu jedem Zeitpunkt festgestellt werden, in welcher Betriebsart sich die Anlage gerade befindet (Abb. 7.12).

Die Methoden METH_Init, METH_Automatik und METH_Stop, die von der einheitlichen Schnittstelle IAggregat an die Aggregate weitergegeben wurden, werden genutzt, um die Anlage zu initialisieren, in den Automatik-Betrieb zu versetzen oder zu stoppen.

[1] Grundsätzlich wird für jeden AS-Schritt und jede IEC-Aktion eine *implizite* Variable angelegt. Eine Strukturinstanz - genauso benannt wie das AS-Element, z. B. Eingefahren für einen Schritt mit Schrittnamen Eingefahren - vom Datentyp SFCStepType bzw. SFCActionType wird erzeugt. Die Strukturkomponenten (engl. *Flags*) beschreiben u. a. den Status eines Schrittes bzw. einer Aktion. Hier zeigt das boolesche Flag x den Aktivationsstatus des Schrittes Eingefahren im aktuellen Zyklus.

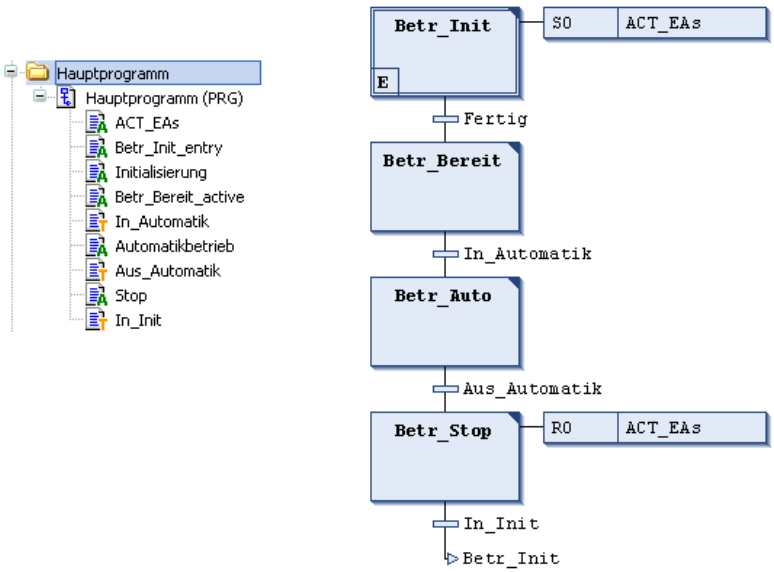



Abbildung 7.12: Hauptprogramm und zugehörige Aktionen / Implementierung in AS

Innerhalb eines bestimmten Anlagenzustands wird in einer `for`-Schleife von allen Aggregaten die entsprechende Methode aufgerufen. Der Zustand wird dann verlassen, sobald die Schaltbedingung erfüllt ist. Beispielsweise wird der Automatik-Betrieb verlassen, sobald die externe Variable `in_Start` auf `FALSE` gesetzt wird. Damit wird die Anlage in den Status `Stop` versetzt. Hier ist ein kleiner Ausschnitt aus dem Quellcode der Aktion `Automatikbetrieb`:

```
FOR i := 1 TO MAXAGGREGATE DO
    Aggregate[i].METH_Automatik();
END_FOR

IF not in_Start THEN
    Z_Anlage := TAggZustand.Stop;
END_IF
```

Der *Initial*-Schritt `Betr_Init` enthält zwei am Anfang dieses Abschnitts erläuterten Aktionen, nämlich die Schritt- und Eingangs-Aktion. Hinzu kommt die IEC-konforme Aktion `ACT_EAs`, die durch den Qualifizierer `S0` gekennzeichnet ist. Diese Aktion wird ausgeführt, sobald der Schritt `Betr_Init` aktiv ist und wird weiter ausgeführt, auch wenn der übergeordnete Schritt schon deaktiviert wurde, bis sie einen *Reset*-Befehl erhält. Dabei ist zu beachten, dass eine IEC-Aktion ein weiteres Mal ausgeführt wird, wenn sie deaktiviert wird. Dies bedeutet, dass eine solche Aktion mindestens zweimal ausgeführt wird.
Im *Eigenschaften*-Fenster können die Eigenschaften eines `AS`-Elements angezeigt und bearbeitet werden (Abb. 7.13).

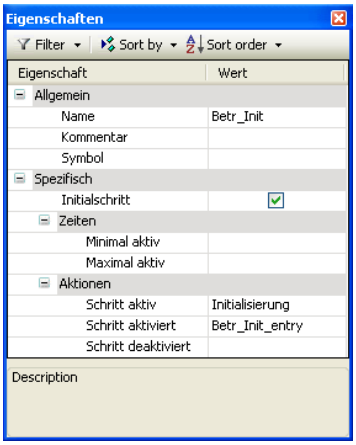



Abbildung 7.13: Eigenschaften des AS-Schritts Hauptprogramm.Betr_Init

In der Aktion `ACT_EAs` werden die Instanzen der Funktionsbausteine zur Laufzeit erzeugt und die symbolischen Adressen der Prozessvariablen mit den Applikationsvariablen verbunden (siehe Abschnitt 8.2).

**GVL**

Eine globale Variablenliste (GVL) wird verwendet, um globale Variablen im Projekt zu deklarieren. Wenn eine GVL im POUs-Fenster vorliegt, sind die darin enthaltenen Variablen im gesamten Projekt verfügbar. Falls eine GVL einer bestimmten Applikation zugeordnet ist, sind die Variablen innerhalb dieser Applikation gültig.

Im Sortieranlage-Projekt werden drei GVLs angelegt:

**`GVLInputs:`** In dieser Liste werden alle Eingänge der Sortieranlage deklariert.

**`GVLOutputs:`** In dieser Liste werden alle Ausgänge deklariert.

**`GVLAggregate:`** Alle anderen globalen Variablen werden in dieser Liste definiert. Diese beinhaltet beispielsweise die konstanten Variablen, die Instanzen der programmierten Funktionsbausteine etc.

**FB_Ausschieber**

Der Funktionsbaustein `FB_Ausschieber` bildet mit seinen Methoden und Aktionen eine geschlossene Einheit, die das Verhalten der Ausschieber bestimmt. Die Methode `METH_Ausstossen` mit der zugehörigen Aktion `ACT_Ausstossen` dient dazu, ein Werkstück aus dem Materialspeicher hinauszuschieben. Sobald ein Werkstück in der Abgabeposition vor dem Ausschieber zur Abnahme vom XY-Schieber bereitsteht, meldet die Methode den Status `erledigt` zurück.

In den Betriebsarten Initialisierungs- und Stopp-Betrieb wird sichergestellt, dass der Kolben des Ausschiebers eingefahren ist. Dafür melden seine „vererbten“ Initialisierungs- und Stopp-Methoden den Aggregatstatus `bereit` zurück. Im Automatik-Betrieb hat der Ausschieber den in Abbildung 7.14 dargestellten Ablauf.

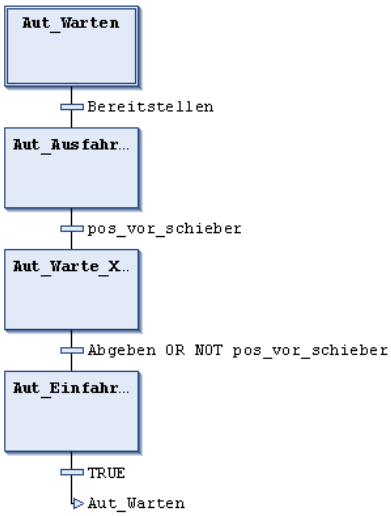



Abbildung 7.14: Ausschieber - Ablauf im Automatik-Betrieb

Bemerkenswert beim Funktionsbaustein `FB_Ausschieber` ist, dass die aus der Schnittstelle `IWerkstueck` implementierte Methode `METH_Aufnehmen` nur den Status `belegt` zurückliefert, da der Ausschieber kein Werkstück von anderen Aggregaten aufnehmen kann.

### FB_Linearachse

Mit einem relativ großen Satz an Methoden und Aktionen ist der Funktionsbaustein `FB_LinearAchse` in der Lage, über den seriellen COM-Port Befehle an die Schrittmotor-Antriebseinheit zu schicken bzw. entsprechende Rückmeldungen von ihr zu empfangen.

Wie alle anderen Aggregate lässt sich das Verhalten der Linearachse in den drei Betriebsarten (Initialisierungs-, Automatik- und Stopp-Betrieb) einordnen.
Zunächst werden im Initialisierungs-Betrieb die Datenübertragungsprotokoll-Parameter in der deklarierten `SysCom`-Struktur `ComSettings` festgelegt (siehe Abschnitt 4.2.3):

```
// ACT_SetParam()
// setzen der Standardparameter eines COM-Ports
com_settings.sPort := SysCom.COM_Ports.SYS_COMPORT1;
com_settings.ulBaudrate := SysCom.COM_Baudrate.SYS_BR_9600;
```

```
com_settings.byParity := SysCom.COM_Parity.SYS_NOPARITY;
com_settings.byStopBits := SysCom.COM_StopBits.
SYS_ONESTOPBIT;
```

Danach wird der COM-Port, über den die Daten übertragen werden, geöffnet:

```
// ACT_ComOpen()
// COM-Port oeffnen
h_com := SysComOpen(sPort := com_settings.sPort, pResult :=
   ADR(com_result));

// Setzen der Standardparameter des seriellen Ports
SysComSetSettings(hCom := h_com, pSettings :=
   ADR(com_settings), pSettingsEx := 0);
```

Anschließend werden die Befehle zur Durchführung der Referenzfahrt hintereinander an die Steuereinheit gesendet[2]:

```
// ACT_Referenzfahrt()
(* Befehle zur Referenzfahrt-Durchfuehrung *)
METH_ComWrite(com_WString := '@01$R$L');
METH_ComWrite(com_WString := '@0d6000$R$L');

IF METH_ComWrite(com_WString := '@0R1$R$L') = TRUE THEN
   ist_Position := TLinPosition.referenz_OK;
END_IF
```

Abschließend wird mit der Rückmeldung der Linearachse, dass die Referenzfahrt beendet ist, die `ist_Position` auf den Wert `referenz_OK` gesetzt. All das geschieht mit dem Aufruf der Aktion `ACT_Init`[3]:

```
//ACT_Init()
// RS232 Initialisierung
IF h_com = 16#FFFFFFFF THEN
   ACT_SetParam();
   ACT_ComOpen();
END_IF
```

[2] Mit der Methode `METH_ComWrite` werden die Befehl-*Strings* an den geöffneten COM-Port geschrieben (siehe Abschnitt 5.4). Die Zeichenkombination `$R$L` entspricht dem Zeilenendezeichen `LF+CR`.

[3] Der Wert `16#FFFFFFFF` wird als Handle vom Typ `RTS_IEC_HANDLE` zurückgegeben, wenn der zu öffnende Port noch nicht geöffnet ist.

```
IF ist_Position = unbekannt THEN
// LinearAchse-Initialisierung
   ACT_Referenzfahrt();
END_IF
IF ist_Position = referenz_OK THEN
   aggStatus := TAggZustand.Bereit;
END_IF
```

Im Automatikbetrieb, welcher in der Programmiersprache AS programmiert wird, wird u. a. von den im strukturierten Text ST implementierten Methoden METH_Zur_Auf_Abnahmestelle, METH_Vor_Messposition und METH_Messfahrt Gebrauch gemacht. Diese Methoden werden im Automatikbetrieb der Linearachse als Transitionsbedingungen aufgerufen und erlauben mit ihren entsprechenden Rückmeldungen das Weiterschalten der AS-Schritte und somit einem koordinierten Ablauf des „Transport"-Prozesses.
Diese Methoden benutzen die Methode METH_Position_anfahren, um den Schlitten mit vorgegebener Position und Geschwindigkeit zu verfahren. Innerhalb METH_Position_anfahren werden mit Hilfe der CONCAT[4]-Funktion der Standard.library die eingegebenen Parameter (position und Geschw) aneinandergehängt, um einen kompletten Verfahrbefehl zusammenzustellen, der anschließend an den COM-Port geschrieben wird (siehe Abschnitt 4.2.3).

Eine besonders wichtige Methode des Funktionsbausteins FB_Linearachse ist die Methode METH_LinAchse_Bereit. Sie ist konstruiert, um zu prüfen, ob der an die Linearachse gesendete Befehl abgearbeitet wurde oder nicht. An dieser Stelle ist dies sehr relevant, da sonst eine korrekte Wertzuweisung der Variable ist_Position vom Typ TLinPosition[5] und somit eine fehlerfreie Abarbeitung des Transportprozesses nicht gewährleistet werden kann.
Das Ablaufdiagramm (Abb. 7.15) der Methode METH_LinAchse_Bereit zeigt, dass diese Methode nur zwei mögliche Rückmeldungen liefert und zwar TRUE oder FALSE. Wenn die Rückmeldung der Methode TRUE ist, bedeutet das, dass der zuletzt an die Linearachse gesendete Befehl abgearbeitet ist. Solange dies nicht der Fall ist, d. h. die Rückmeldung hat den Wert FASLE, kann kein neuer Befehl ausgeführt werden.
Wenn ein Befehl zum Verfahren der Linearachse gesendet wird, wird in einer while-Schleife der Lesepuffer ausgelesen, solange die Rückmeldung nicht gleich 0 ist. Wird das Zeichen 0 vom Lesepuffer gelesen, so wird darauf der Ein- und Ausgangspuffer der seriellen Schnittstelle gelöscht. Als weitere Sicherheitsmaßnahme wird noch einmal getestet, ob tatsächlich der Puffer geleert wurde. Wenn ja, dann ist damit garantiert, dass der Befehl vollständig abgearbeitet wurde und erst dann ein neuer Positionierbefehl erfolgen kann.

[4] To concatenate (aneinanderhängen bzw. verketten)

[5] Ein benutzerdefinierter Datentyp bzw. eine Aufzählungsvariable (engl. *Enumeration*), die zur Beschreibung der Position der Linearachse dient. Diese Enumeration enthält den Initialwert unbekannt, die auch zugewiesen wird, solange der Schlitten beim Verfahren ist, sowie die vier Positionen, die von der Linearachse angefahren werden: referenz_OK, lade_pos, vormess_pos, messfahrt_Ende.

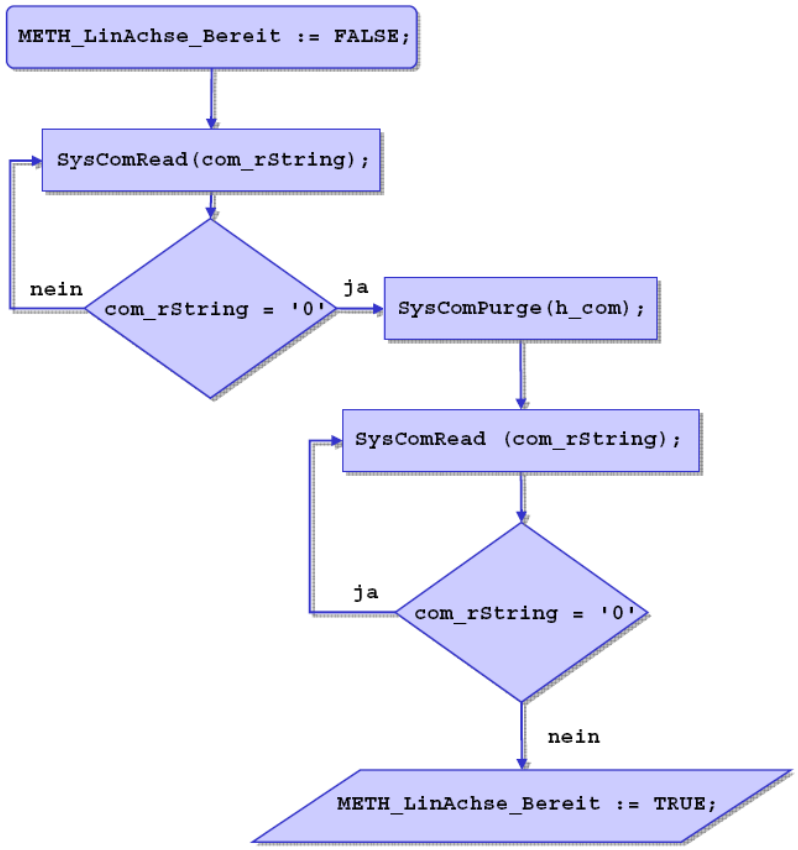



Abbildung 7.15: Ablaufdiagramm der Methode METH_LinAchse_Bereit

Ein kleiner Auszug des umgesetzten Quellcodes der Methode `METH_LinAchse_Bereit` sieht folgendermaßen aus:

```
// Einlesen des Eingangspuffers der seriellen Schnittstelle
com_rueckmeldung := SysComRead(hCom := h_com, pbyBuffer :=
    ADR(com_rString), ulSize := 1, ulTimeout :=
    com_ulTimeout, pResult := ADR(com_result));

// Warten, solange der Rückmeldung != 0 ist
WHILE com_rString <> '0' DO
    com_rueckmeldung := SysComRead(hCom := h_com,
       pbyBuffer := ADR(com_rString), ulSize := 1,
       ulTimeout := com_ulTimeout, pResult :=
       ADR(com_result));
END_WHILE
```

```
// Wenn Rückmeldung = 0 => Ein- und und Ausgangspuffer der
   seriellen Schnittstelle leeren
IF com_rString = '0' THEN
   com_purge := SysComPurge(h_com);

   com_rueckmeldung := SysComRead(hCom := h_com,
      pbyBuffer := ADR(com_rString), ulSize := 1,
      ulTimeout := com_ulTimeout, pResult :=
      ADR(com_result));

   IF com_rueckmeldung = 0 THEN
      METH_LinAchse_Bereit := TRUE;
   END_IF
END_IF
```

Eine geeignete Stelle, um die Methode `METH_LinAchse_Bereit` aufzurufen, ist unmittelbar nach dem Senden einer Zeichenkette (String) eines Vefahrbefehls, also innerhalb der Methode `METH_ComWrite`, die für das Schreiben eines Befehls an den COM-Port verantwortlich ist. Grund hierfür ist die darin enthaltene `while`-Schleife, die solange läuft, bis ein bestimmtes Ereignis eintritt und zwar Rückmeldung = 0. Dies führt beim häufigen Aufruf der Methode `METH_LinAchse_Bereit` durch die unterschiedlichen Aggregate zu extrem langer Laufzeit des Steuerungsprogramms.
Da eine Rückmeldung der Linearachse bzgl. ihrer Bereitschaft, neue Befehle auszuführen, nicht festgestellt werden kann, solange der Schlitten beim Verfahren ist, wird die Messfahrt so angepasst, dass die Fahrt der Linearachse in hundert Schritten unterteilt wird, um es zu ermöglichen, genauso viele Messwerte vom Laser-Messgerät aufzunehmen bzw. abzuspeichern.
Zum Abtasten aller Höhenprofil-Werte des Laser-Messgerätes wird die empirisch ermittelte Messstrecke eines Werkstücks in einhundert Schritten unterteilt:

$$\begin{aligned}
\frac{Messstrecke}{Anzahl\ der\ Messwerte} &= \frac{Nachmessposition - Vormessposition}{100} \\
&= \frac{34400 - 31000}{100} \\
&= \frac{3400\ Schritte}{100\ Messwerte} \\
&= 34\ Schritte/Messwert
\end{aligned}$$

Dabei stellen die Werte der Vor- und Nachmessposition die Anzahl der Schritte des Spindelmotors bezogen auf den Referenzpunkt (0-Punkt) der Linearachse dar. Das heißt, dass nach jedem 34. Schritt des Antriebsmotors das Programm `PRG_LaserMessung` zum Einlesen

und anschließenden Vergleich der Werkstücke mit dem Referenzteil aufgerufen wird.

**FB_LaserMessung**

In den Abschnitten 4.2.5 und 4.2.5.2 wurden allgemein das Funktionsprinzip der Laser-Sensoren sowie die analoge Messwertverarbeitung erläutert. Im Funktionsbaustein `FB_LaserMessung` werden Methoden zum Einlesen des Triangulationslaser-Messwerts (`METH_Messwert_einlesen`) und zum Vergleich der Werkstücksdaten mit denen des Referenzwerkstücks (`METH_Baustein_Vergleich`) implementiert.

Das Programm `PRG_LaserMessung`, das in Abhängigkeit der Werkstückanzahl entscheidet, um was für ein Werkstück es sich bei der aktuellen Messung handelt, wird in der Methode `METH_Messfahrt` des Funktionsbausteins `FB_LinearAchse` aufgerufen. Dieses Programm ruft den `FB_LaserMessung` mit entsprechender Funktionsnummer auf, um die Messdaten der Bausteine einzulesen, in Höhenprofildaten umzurechnen, mit dem Referenzstück zu vergleichen und anschließend in die dem Baustein zugeordnete Struktur `WSDaten` abzuspeichern. Die Funktionsnummer ist auf die Reihenfolge der vermessenen Werkstücke bezogen, d. h. das erste Werkstück (Referenzteil) bekommt die Nummer 1, das zweite die 2 usw. Diese Unterscheidung dient dazu, dass beispielsweise beim Referenzstück kein Vergleich durchgezogen wird.

Das Einlesen der Messwerte von der Analogeingangsbaugruppe der SPS wird - wie im vorangegangenen Abschnitt 7.1 (unter Triangulationslaser) erklärt - wie folgt in der `METH_Messwert_einlesen` ausgeführt:

```
// Einlesen des Messwertes von Eingangswort IW2
laser_Messwert := WORD_TO_INT(in_laser_messwert);

// Umrechnung in Volt: U_AE = AE * (10 V / 27648 Bits)
messung_U := (INT_TO_REAL(laser_Messwert) *
   (10.0/27648.0));

// Umrechnung in mm: h_mm = U_AE * (1mm / 0.1176V)
messung_mm := messung_U *(1.0/0.1176);
```

Dieser vom Triangulationslaser erfasste Wert `messung_mm` gibt den Abstandswert des Werkstücks vom Lasersensor an. Um diesen zu korrigieren, wird der ermittelte *Offset*-Wert des leeren Schlittens der Linearachse (`schlitten_L_offset` = 67 mm) davon subtrahiert. Somit erhält man einen exakten Wert des Werkstückprofils in [mm]:

```
// Berechnung der tatsächlichen Höhe des Bausteins
profil_daten := messung_mm - schlitten_L_offset;
```

Bevor es zum Vergleich bzw. Speichern der Höhenprofildaten kommt, wird mittels folgender Bedingung geprüft, ob der gemessene Wert tatsächlich größer als der Schlitten-Offset ist:

```
IF (messung_mm > schlitten_L_offset) AND
   (schlitten_L_offset > offset) THEN
   ...
```

Dabei bezeichnet `offset` den umgerechneten Wert des gemessenen Abstands ohne Anwesenheit des Schlittens.
Falls diese Bedingung erfüllt ist, wird der aktuelle Wert in der dem Werkstück zugehörigen Variable bzw. Struktur `bausteindaten_X.ws_daten` abgespeichert und anschließend die Methode `METH_Baustein_Vergleich` aufgerufen, um im Vergleich zum Referenzstück festzustellen, ob der Teil gut oder schlecht ist. Wird das Werkstück mit „gut“ bewertet, so wird die Variable `bausteindaten_X.ws_gut` auf `TRUE` gesetzt. Andernfalls bekommt sie den Wert `FALSE` zugewiesen.

Die Unterscheidung zwischen den „guten“ und den „schlechten“ Bauteilen wird innerhalb der Methode `METH_Baustein_Vergleich` vorgenommen. Diese sieht vor, dass die gebildete Differenz zwischen dem aktuell gemessenen i-ten Wert und dem entsprechenden des Referenzstücks einen vorgegebenen Toleranz-Wert nicht überschreitet. Wird dieser Schwellwert nicht erreicht, wird in einer lokalen Hilfsvariable `gut` vom Datentyp `Integer` additiv eine 1 dazu gerechnet. Ist der Toleranz-Wert doch überschritten, wird in einer anderen Hilfsvariable `schlecht` die 1 dazu addiert. Dieser Vorgang wird bei jedem der hundert Messwerte durchgeführt. Für den Fall, dass die „schlechten“ Werte mehr als 17% der insgesamt aufgenommenen Werte sind, wird das Werkstück als „schlecht“ markiert und in seine Strukturvariable `bausteindaten_X.ws_gut` der Wert `FALSE` geschrieben. Ansonsten ist das aktuelle Werkstück als „gut“ zu bewerten.

#### FB_XY-Schieber

Der Funktionsbaustein `FB_XYSchieber`, der die Handlungsweise des Aggregats XY-Schieber bestimmt, besitzt eine umfangreiche Ansammlung von Methoden und Aktionen.

Beim Neustart der Anlage wird der XY-Schieber in seine anfängliche Stellung gebracht. Die Initial-Stellung wird in drei Schritten erzielt:

- Sauger abschalten,
- Heber hochfahren,
- Schieber vor dem Ausschieber stellen.

Erst nach Beendigung der Referenzfahrt der Linearachse und des Positionierens des XY-Schiebers kann die Sortieranlage in den Automatik-Betrieb umschalten.

Im Automatik-Betrieb wird die Aktion `ACT_Automatik` als Dienst vom XY-Schieber ausgeführt. Diese Aktion wird in der Programmiersprache `ST` implementiert und wird aufgerufen, sobald der XY-Schieber den Status `Automatik` vom Typ `TAggZustand` bekommt.

Diese Aktion regelt den Ablauf des Sortierprozesses. Sie verwendet dafür die unterhalb des Funktionsbausteins `FB_XYSchieber` zur Verfügung gestellten Aktionen und Dienste (siehe Abb. 7.4 auf Seite 56). Diese erlauben u. a. das Verfahren des XY-Schiebers zum Ausschieber oder zur Linearachse, das Ansaugen der Messobjekte, das Zählen der verarbeiteten Werkstücke sowie das Anfahren vorgegebener Ablagepositionen der Sortierstation. Die Aufrufe der Methoden anderer Funktionsbausteine (Module) mit ihren Rückmeldungen werden als Transitionsbedingungen zum Weiterschalten genutzt.
Zunächst fordert der XY-Schieber vom Ausschieber die Bereitstellung eines Werkstückes in der Abgabeposition. Wenn dies erledigt ist, dann wird in den nächsten `AS`-Schritt weitergeschaltet:


Abbildung 7.16: XY-Schieber erster Schritt

Danach wird geprüft, ob der XY-Schieber bei der Abgabeposition des Ausschiebers angekommen ist. Dann ruft der XY-Schieber die Methode `METH_Abgeben` des Ausschiebers auf, wenn er bereit ist, die Übergabe des Werkstücks einzuleiten und erhält die Information, ob der Ausschieber ebenfalls bereit ist für die Übergabe. Ist dies der Fall, wird die Übergabe eingeleitet und der XY-Schieber nimmt das Werkstück auf. Der Ausschieber liefert den Methodenstatus `erledigt`, sobald die Übergabe abgeschlossen ist.
Es ist hier zu beachten, dass die Übergabe des Werkstücks individuell in den verschiedenen Modulen der Anlage realisiert wird.
So werden die Prozessschritte nach und nach abgearbeitet. Auf diese Weise kann sichergestellt werden, dass das Werkstück vermessen, von der Linearachse abgeholt und vom XY-Schiebers in die Ablageposition abgegeben wird.

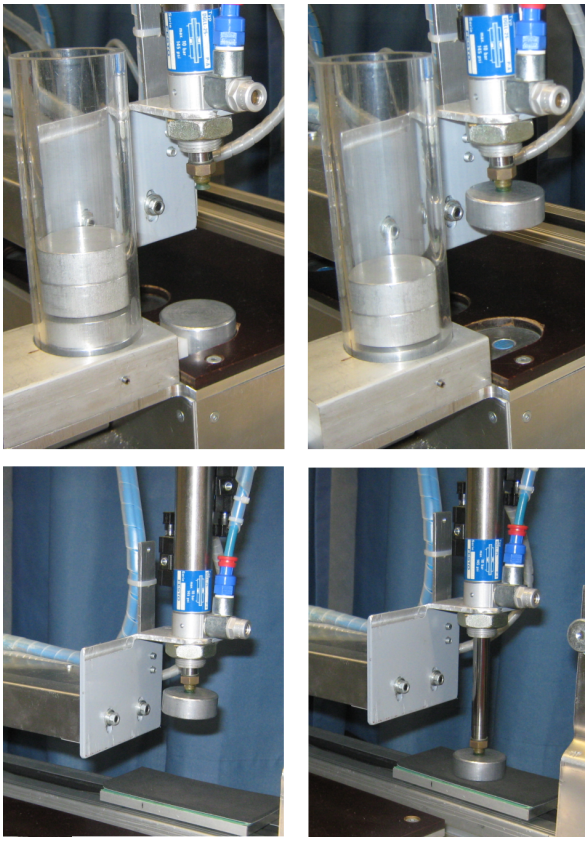


Abbildung 7.17: XY-Schieber Transport-Job

**FB_Sortierstation**

Im Funktionsbaustein `FB_Sortierstation` wird das Verhalten der Sortierstation festgelegt. Diese hat als „passives“ Aggregat keine Funktionen bereitzustellen außer der von der Schnittstelle `IWerkstuek` geerbten Methode `METH_Aufnehmen`. Mit dieser Methode kann die Sortierstation die Rückmeldung `bereit` vom Datentyp `TJobZustand` an den XY-Schieber zurückliefern, wenn er ein Werkstück ablegen muss. Außerdem wird bei voller Belegung der Ablagepositionen der Sortierprozess vorübergehend gestoppt. Der Prozess wird dann fortgeführt, sobald alle Ablagestellen geräumt sind. In diesem Fall wird das zuletzt vermessene Objekt als Referenzteil identifiziert und dessen Daten dann mit denen der weiteren Werkstücke verglichen.

Eine komplette Übersicht über die Schnittstellen sowie die realisierten Module der Sortieranlage inklusive deren Methoden und Aktionen wird in einer UML[6]-ähnlichen Darstellung in Abbildung 7.18 gezeigt.

[6] **U**nified **M**odeling **L**anguage (Vereinheitlichte Modellierungssprache)

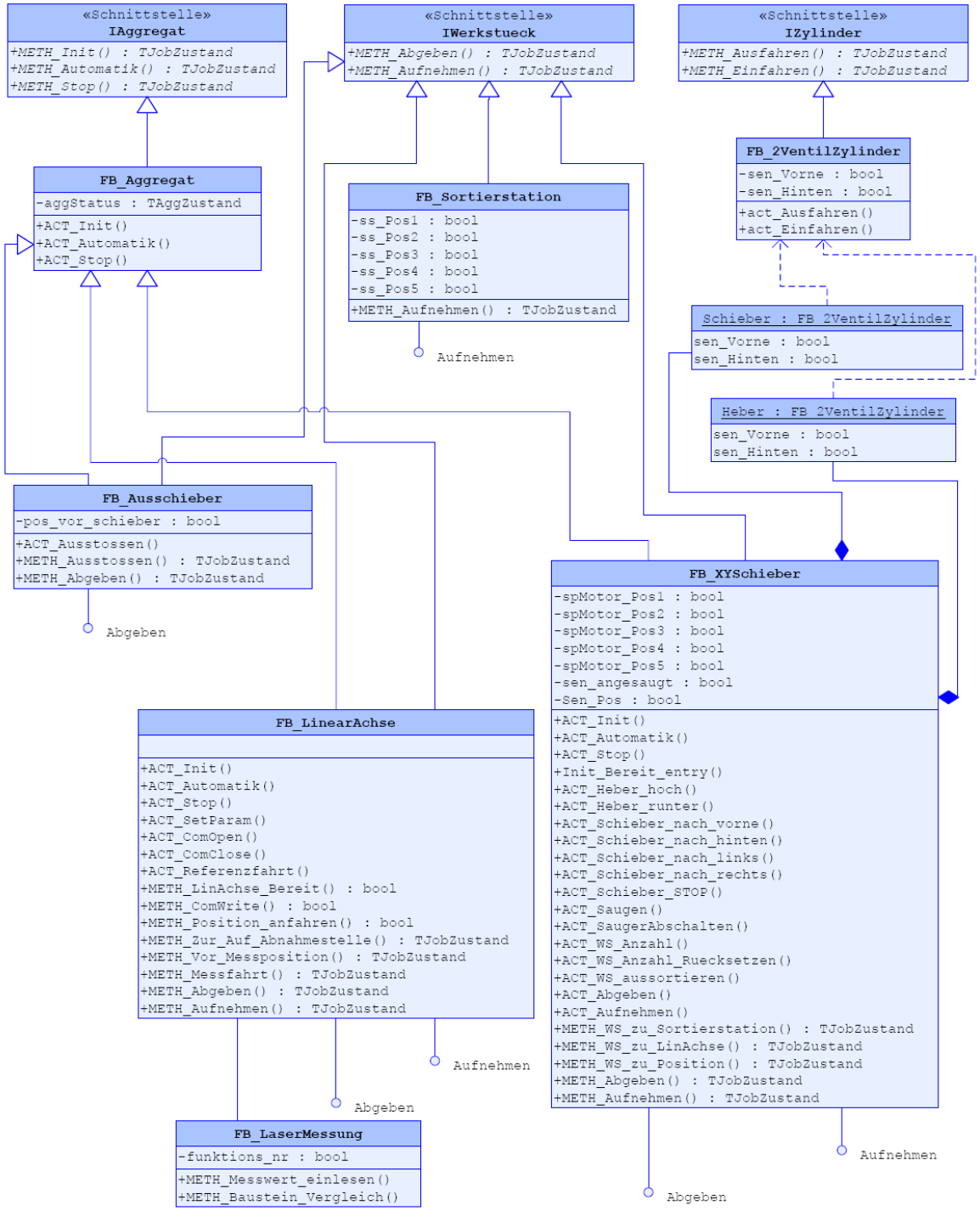



Abbildung 7.18: Übersicht der Funktionsbausteine und Schnittstellen der Sortieranlage

# 8 Prozessanbindung in CoDeSys

Nachdem das Projekt samt seiner Funktionsbausteine und Methoden realisiert wurde, müssen die Variablen der Applikation mit den Prozessvariablen verknüpft werden. Dies geschieht meist nicht durch eine direkte Zuordnung der Adressen zur Hardwarestruktur. Grund hierfür ist, dass diese Art der Zuordnung nicht für die verschiedenen E/A-Systeme einheitlich möglich ist. Deshalb können diese sogenannten direkten Adressen zwar im Projekt genutzt werden, sie verweisen aber nicht direkt auf die physikalischen Ein- und Ausgänge eines Hardewarekanals, sondern auf eine Stelle im Prozessabbild, die das E/A-System im Speicher repräsentiert. (Vogel-Heuser und Wannagat, 2009)

Die Tabelle 8.1 enthält eine Übersicht über die Ein- und Ausgänge der Sortieranlage und deren symbolischen Namen.

Eine Möglichkeit zur Zuordnung der Applikationsvariablen ist, die Variablen innerhalb der Programmtexte durch die Verwendung von Adressen der Prozessvariablen zu verbinden. Diese Adressierung lässt sich in drei Arten unterteilen:

- Direkte Adressierung
- Symbolische Adressierung
- Instanzbezogene Adressierung

## 8.1 Direkte Adressierung

Die Verwendung von direkten Adressen im Programmtext ist die einfachste Art der Verknüpfung der Applikationsvariablen. In *CoDeSys* werden im *Gerätemanager* bei korrekter Verknüpfung der direkten Adressen mit den Hardwarekanälen die entsprechenden Prozessvariablen beim Aufruf eines Funktionsbausteins direkt übergeben bzw. nach der Bearbeitung des Bausteins auf die Ausgangsadressen geschrieben. Ein Beispiel für direkte Adressierung sieht folgendermaßen aus:

```
Ausschieber(pos_vor_schieber:= %IX1.1,
   ausschieben=> %QX0.5);
```

Hiermit können individuelle Ein- und Ausgänge allen Instanzen eines Funktionsbausteins zugewiesen werden. (Vogel-Heuser und Wannagat, 2009)

| Symbol | Adresse | Datentyp |
|---|---|---|
| Eingänge | | |
| in_schieber_hinten | %IX0.0 | BOOL |
| in_schieber_vorne | %IX0.1 | BOOL |
| in_spMotor_Pos1 | %IX0.2 | BOOL |
| in_spMotor_Pos2 | %IX0.3 | BOOL |
| in_spMotor_Pos3 | %IX0.4 | BOOL |
| in_spMotor_Pos4 | %IX0.5 | BOOL |
| in_spMotor_Pos5 | %IX0.6 | BOOL |
| in_heber_oben | %IX0.7 | BOOL |
| in_heber_unten | %IX1.0 | BOOL |
| in_ss_Pos1 | %IX1.1 | BOOL |
| in_ss_Pos2 | %IX1.2 | BOOL |
| in_ss_Pos3 | %IX1.3 | BOOL |
| in_ss_Pos4 | %IX1.4 | BOOL |
| in_ss_Pos5 | %IX1.5 | BOOL |
| in_teil_angesaugt | %IX2.3 | BOOL |
| in_laser_messwert | %IW2 | BOOL |
| Ausgänge | | |
| out_heber_hoch | %QX0.0 | BOOL |
| out_heber_runter | %QX0.1 | BOOL |
| out_schieber_nach_hinten | %QX0.2 | BOOL |
| out_schieber_nach_vorne | %QX0.3 | BOOL |
| out_ansaugen | %QX0.4 | BOOL |
| out_ausstossen | %QX0.5 | BOOL |
| out_spMotor_rechts | %QX0.6 | BOOL |
| out_spMotor_links | %QX0.7 | BOOL |

Tabelle 8.1: Symboltabelle der Ein-und Ausgänge der Sortieranlage

## 8.2 Symbolische Adressierung

Bei dieser Art der Adressierung werden die Adressen (wie `%IX2.3`) mit einem Namen versehen, um die Lesbarkeit der Applikation zu verbessern.

Die Zuordnung der Namen zu den Adressen wird im Projekt der Sortieranlage in der globalen Variablenlisten `GVLInputs` und `GVLOutputs` vorgenommen. Dies wird mit Hilfe des Schlüsselworts `AT` erzielt. Damit lassen sich die Adressen wie globale Variablen verwenden.

```
GVLInputs
// XY-Schieber Sensoren
in_schieber_hinten AT %IX0.0 : BOOL;
in_schieber_vorne AT %IX0.1 : BOOL;
```

Wie bei der direkten Adressierung müssen die „globalen“ symbolischen Adressen an die Funktionsbausteine übergeben werden. Es folgt ein kleiner Ausschnitt vom Quellcode der Aktion `ACT_EAs` des `Hauptprogramms`, der diesen Sachverhalt zeigt:

```
Hauptprogramm.ACT_EAs()
XYSchieber.schieber(sen_Vorne:= in_schieber_vorne,
   sen_Hinten:= in_schieber_hinten,
   akt_Ausfahren=> out_schieber_nach_vorne,
   akt_Einfahren=> out_schieber_nach_hinten);
```

## 8.3 Instanzbezogene Adressierung

In *CoDeSys* besteht weiterhin die Möglichkeit, die lokalen Applikationsvariablen ohne die exakte Angabe der Adressen bei der Variablendeklaration im Funktionsbaustein zu definieren. Die Zuweisung der Adressen erfolgt dann für alle FBs-Instanzen der Applikation in einer getrennten globalen Variablen-Konfigurationsliste `VAR_CONFIG`. Dazu können den Funktionsbaustein-Variablen im Deklarationsteil zwischen den Schlüsselwörtern `VAR` und `END_VAR` „unvollständige“ Adressen zugewiesen werden. Diese *Adressplatzhalter* haben die Syntax `AT %I*` für einen Prozesseingabewert oder `AT %Q*` für einen Prozessausgabewert. Dabei hat jede Adresszuordnung das Format:

```
<Instanzpfad>.<Bausteinvariable> AT <I/O-Adresse>:<Typ>;
```

Die Deklaration entspricht der üblichen Variablendeklaration mit Ausnahme der Verwendung eines vollständigen Instanzpfades anstelle eines einfachen Variablennamens.
Konfigurationsvariablen, deren Instanzpfad nicht richtig oder nicht gültig ist, weil die Instanz nicht existiert, werden als Fehler vom Kompiler gemeldet. Umgekehrt wird auch ein Fehler ausgegeben, wenn für eine Instanzvariable, die mit einer unvollständigen Adresse deklariert ist, keine Adresskonfiguration vorliegt.

Vorteilhaft ist der Einsatz der instanzbezogenen Adressierung, da eine Übergabe von absoluten Adressen beim Aufruf eines Funktionsbausteins völlig entfällt. Weitere Vorteile der instanzbezogenen Adressierung sind:

- Alle Adresszuordnungen befinden sich an einer zentralen Stelle im Projekt.
- Auch bei geschachtelten Instanzen und Feldern von Instanzen möglich.
- Keine Übergabe der Adressen an Bausteine. Dadurch wird die Applikation einfacher und schneller.
- Der Entwickler eines Funktionsbausteins muss nicht die genauen Adressen kennen, d. h. er kann exakter die Verwendung des Funktionsbausteins festlegen. (Vogel-Heuser und Wannagat, 2009)

# 9 Visualisierung der Anlage

*CoDeSys* stellt zur Veranschaulichung der Projektvariablen die Möglichkeit einer *Visualisierung* zur Verfügung. Mit Hilfe der Visualisierung können im Offline-Modus geometrische Elemente zur Anlagendarstellung gezeichnet werden. Diese können dann im Online-Modus in Abhängigkeit von bestimmten Variablenwerten ihre Form, Farbe oder Textausgabe verändern.

Eine Visualisierung kann auch als ausschließliche Bedienoberfläche eines Projekts mit *CoDeSys*-HMI[1] oder zielsystemabhängig auch als Web- oder Target-Visualisierung über Internet bzw. auf der Steuerung vor Ort genutzt werden.
Das *CoDeSys*-HMI ist ein Laufzeitsystem zur Ausführung von Visualisierungen, die mit dem Programmiersystem *CoDeSys* erstellt werden. Wenn ein Steuerungsprogramm entsprechende Visualisierungen enthält, werden diese nach dem Start von *CoDeSys*-HMI im Vollbildmodus dargestellt und der Anwender kann darüber per Mausklick oder Tastatur die in dem zugrundeliegenden Projekt enthaltenen Steuer- und Überwachungsfunktionen bedienen. (3S, 2009)

Die hier aufgeführten Erläuterungen zum Visualisierungs-Editor und zur Trace-Aufzeichnung dienen dem prinzipiellen Verständnis ihrer Funktionalität sowie deren Einsatz im vorhandenen Sortieranlage-Projekt. Für ausführliche Erklärungen bezüglich ihrer Möglichkeiten und ihrem Befehlsumfang wird auf entsprechende Literaturen verwiesen.

## 9.1 Visualisierungs-Editor

Der Visualisierungs-Editor besteht aus einer Zeichenfläche , einem *Werkzeuge*-Fenster (engl. *Toolbox*) und einem *Eigenschaften*-Fenster (engl. *Properties*). Die Toolbox enthält alle verfügbaren Visualisierungselemente, die zum Einfügen im Visualisierungs-Editor bereitgestellt sind. In dem Eigenschaften-Fenster werden die Attribute des gerade im Visualisierungs-Editor markierten Elementes angezeigt und können dort bearbeitet werden. Weitere Konfigurationsmöglichkeiten eines Elements hängen vom Elementtyp ab. Die Handhabung des Editors ist sehr ähnlich zu anderen bekannten Zeichenwerkzeugen.
Nach dem Einfügen eines Elements in den Visualisierungs-Editor kann er selektiert und

[1] **H**uman **M**achine **I**nterface (**M**ensch-**M**aschine-**S**chnittstelle (MMS))

seine Position und Größe direkt über „Drag&Drop“-Mausaktionen verändert bzw. seine Ausrichtung und Reihenfolge (Hintergrund, Vordergrund etc.) mit Hilfe der entsprechenden Visualisierungsbefehle festgelegt werden.

Die objektorientierten Erweiterungen in *CoDeSys* sind auch für die Visualisierung umgesetzt worden. Genau wie die Klassen der Applikation lassen sich Visualisierungselemente instantiieren oder mit anderen Elementen zu neuen Elementen kombinieren.

Zunächst werden die Visualisierungsobjekte der einzelnen Module der Sortieranlage angelegt. Die Objekte sollen die realen Komponenten der Anlage widerspiegeln. Sie werden im Geräte-Fenster eingefügt (Abb. 9.1).

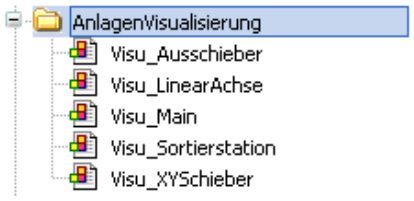



Abbildung 9.1: Visualisierungsobjekte

Danach werden die einzelnen Komponenten gezeichnet. Die Zeichnungen geben alle von einem Objekt annehmbaren Stellungen wieder (Abb. 9.2). Zum Beispiel werden bei der Zeichnung der Linearachse alle Positionen inklusive des darauf gelegten Werkstücks gezeichnet.

Zusätzlich ist ein simples Bedienpanel eingezeichnet, mit dem die Anlage in den bzw. vom Automatik-Betrieb umgeschaltet werden kann. Dies wird mit Hilfe der globalen Variable `in_Start` aus der globalen Variablenliste `GVLInputs` realisiert.

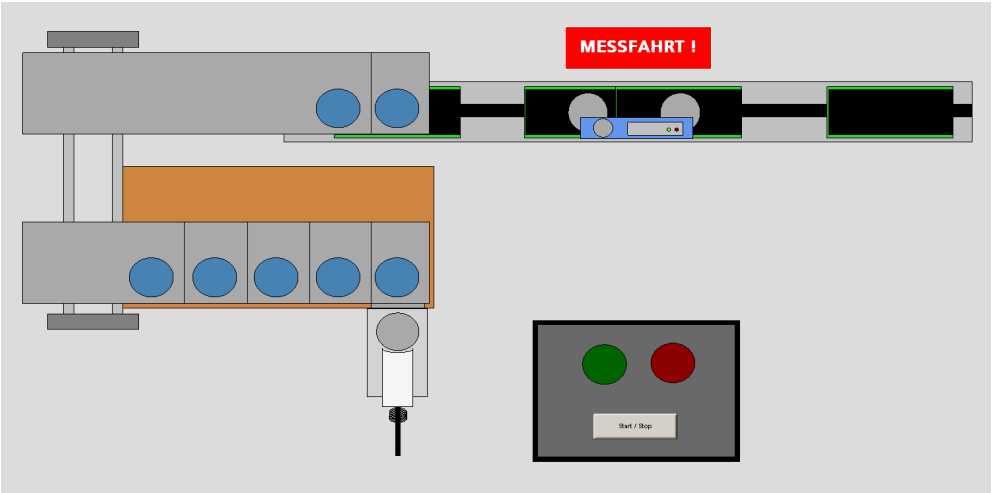



Abbildung 9.2: Alle Visualisierungskomponenten der Sortieranlage

Um ein Visualisierungsobjekt in einer übergeordneten Visualisierung zu nutzen, muss dort ein Objekt vom Typ `Frame` eingefügt werden. Dies kann dann mit dem Befehl **Frame Auswahl** mit dem Visualisierungsobjekt verknüpft werden.

## 9.2 Simulation der Visualisierung

Nachdem die Objekte angelegt werden, müssen sie für die Prozessvisualisierung „dynamisiert“ werden, d. h. sie sollen im Online-Modus in Abhängigkeit von bestimmten Variablenwerten ihre visuellen Eigenschaften verändern.
Dafür hat man in *CoDeSys* die Möglichkeit, Simulationsprogramme zu schreiben, die Veränderungen an den Visualisierungsobjekten hervorrufen können. Zum Beispiel kann das Gebilde in `Visu_Ausschieber` seine vertikale Position abhängig von der Variablen `schieben`, die im Simulationsprogramm `PRG_Ausschieber` definiert ist, ändern. Hierfür steht in der `Movement`-Option unter `Absolute movement` in dem Eigenschaften-Fenster des Ausschieber-Objekts die Zuweisung

```
Y := -PRG_Ausschieber.schieben
```

Seine Simulationsprogramm-Struktur ist in der Abbildung 9.3 dargestellt.

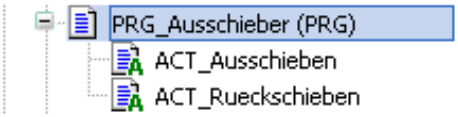



Abbildung 9.3: Simulationsprogramm PRG_Ausschieber

Dabei werden die Aktionen `ACT_Ausschieben` und `ACT_Rueckschieben` im `PRG_Ausschieber` in Abhängigkeit der global im Projekt bekannten Variable `Ausschieber.pos_vor_schieber` aufgerufen. So wird in der Visualisierung der tatsächliche Zustand des Ausschiebers in der Anlage entsprechend dargestellt. Abbildung 9.4 zeigt die realisierte Visualisierung im Online-Modus.

Die Visualisierung der Anlage wird mit dem *CoDeSys*-HMI im Vollbildmodus ausgeführt, sobald das Steuerungprogramm auf die Steuerung geladen ist und gestartet wurde. Bei der Online-Visualisierung (Abb. 9.4) sind alle Komponenten der Sortieranlage zu sehen. Nach der Initialisierung kann die Anlage mit Betätigen des `Start/Stop`-Knopfes in den Automatik-Betrieb umschalten.
Die Visualisierung ist sehr einfach gestaltet. Die Bewegung der Objekte wird mittels boolescher Abfragen bestimmter Prozesssignale erzielt. In Abhängigkeit dieser Signale werden die Objekte entsprechend ein- bzw. ausgeblendet, entlang ihrer Achse bewegt oder sogar in

ihrer Größe geändern. Darüber hinaus besteht bei der *CoDeSys*-Visualisierung die Möglichkeit, die Visualisierungsojekte mit anderen Formen der Animation zu versehen[2].

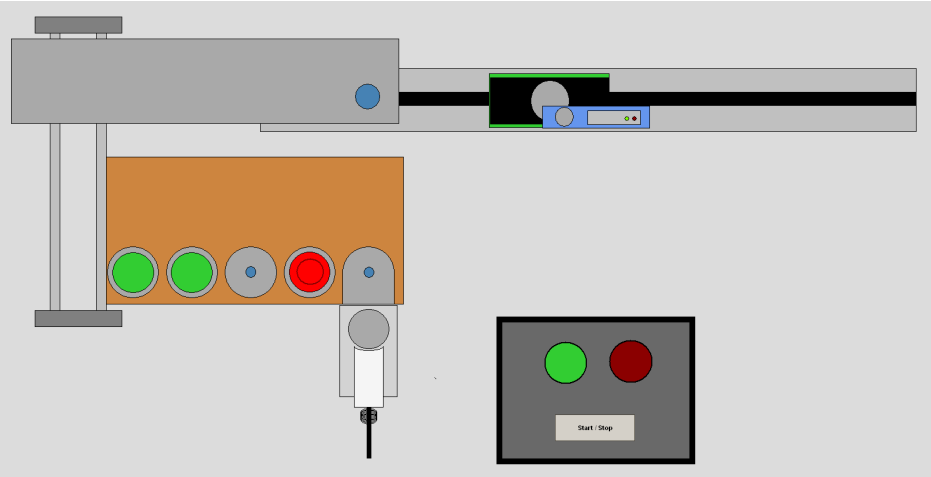



Abbildung 9.4: Online-Visualisierung der Sortieranlage

## 9.3 Trace-Aufzeichnung

Die *Trace*-Funktionalität erlaubt das Aufzeichnen von Variablenwerten aus der Steuerung über einen bestimmten Zeitraum. Zu diesem Zweck werden die Werte definierter Trace-Variablen im Laufzeitsystem kontinuierlich in einen Puffer bestimmter Größe geschrieben und können dann in Form einer Kurve über einer Zeitachse in *CoDeSys* angezeigt werden. Nach dem Anlegen einer Trace-Variable (hier `Trace-TEST`) erfolgen die Konfiguration und die Darstellung der Trace-Aufzeichnung in den Dialogen und Fenstern des Trace-Editors (Abb. 9.5).

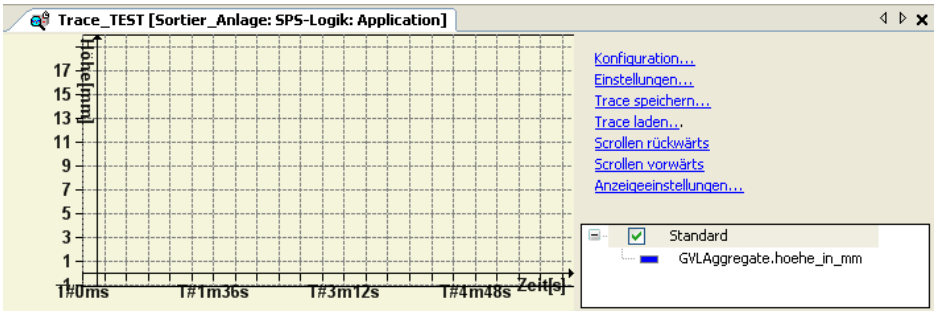



Abbildung 9.5: Trace-Editor

[2]Für eine detaillierte Beschreibung der Visualisierungskonfigurationen siehe 3S-Dokumentation: CodeSys_Visualisierung.pdf

Diese Eigenschaften der Trace-Aufzeichnung werden genutzt, um die vom Laser-Messgerät gelieferten Höhenprofildaten der vermessenen Werkstücke grafisch darzustellen.
Im **Trace-Konfigurations**-Fenster (Abb. 9.6) kann die Prozessvariable angegeben werden, für die eine Aufzeichnung erfolgen soll.

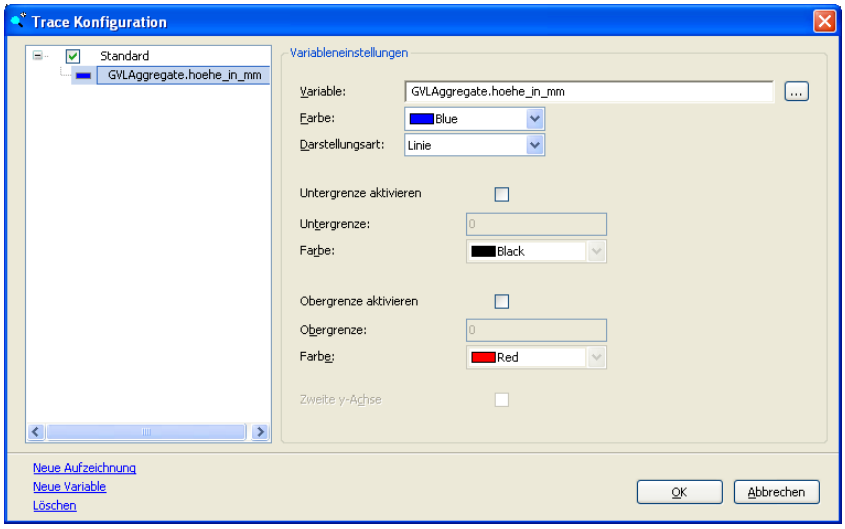



Abbildung 9.6: Trace-Konfigurations-Fenster

Die Variable `hoehe_in_mm` bezeichnet den globalen Höhenprofil-Messwert in [mm]. Die Umrechnung ihres Werts wird zur Laufzeit innerhalb der Aktion `ACT_EAs` vorgenommen. Sobald der Schlitten der Linearachse sich in der Vormess-Position befindet, startet die Messung des Höhenprofils. Die Höhe des Werkstückes wird aus der Differenz zwischen dem in [mm] umgerechneten Messwert und dem Schlitten-Offset gebildet. Solange der aufgenommene Wert größer als der Schlitten-Offset ist, wird der Wert gespeichert. Ist dies nicht der Fall, wird eine `0` geschrieben. Und so erhält man einen genauen Wert zur Aufzeichnung des Profils über die Zeitachse (Abb. 9.7).

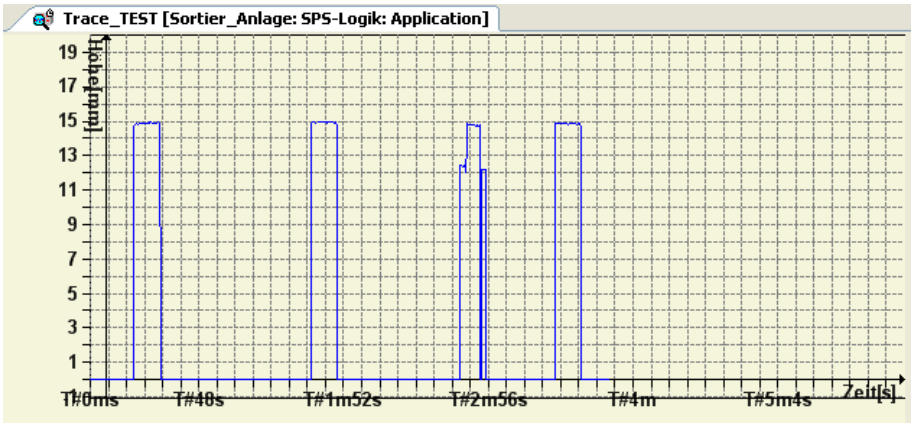



Abbildung 9.7: Trace-Aufzeichnung der vermessenen Werkstücke

Die Profile der vermessenen Werkstücke sind aus der obigen Abbildung deutlich zu erkennen. Der dritte von links aufgezeichnete Verlauf beschreibt das Profil des „schlechten“ Werkstückes. Die „Zacken“ an den Profil-Oberflächen sind auf die Tatsachen zurückzufüren, dass die Messung zum Einen beim bewegten Schlitten und zum Anderen auf die raue Oberfläche der vermessenen Werkstücke durchgeführt wird.

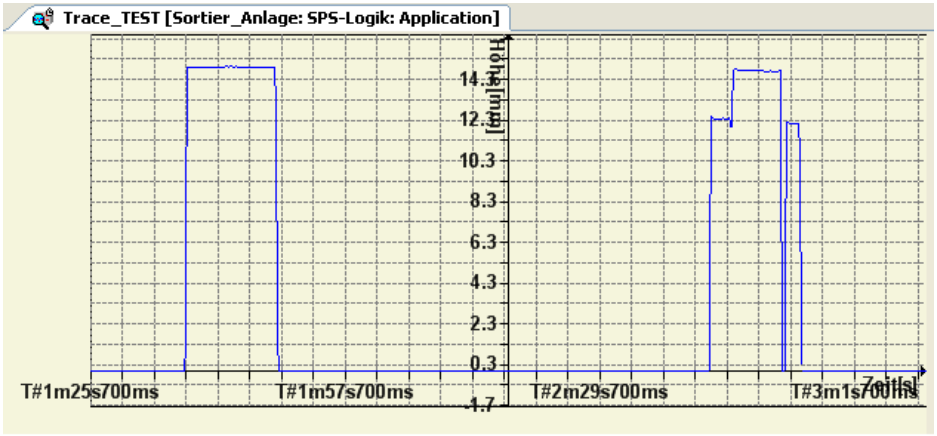



Abbildung 9.8: Gutes und schlechtes Werkstück im Vergleich

Zum Vergleich sind die Profile eines „guten“ und eines „schlechten“ Werkstückes in Abbildung 9.8 dargestellt. Es ist dabei zu erkennen, dass das auszusortierende Objekt eine deutliche Abweichung seiner Form aufweist. Der Sortierprozess bildet im Grunde genommen die Grundlage zahlreicher Prozesse in der Produktions- und Fertigungstechnik. Die Sortierung kann alternativ auf andere Merkmale (wie Gewicht, Farbe, magnetische Eigenschaften etc.) eines Fertigungsprodukts bezogen sein, falls die entsprechenden Sensorik-Systeme vorhanden sind.

# 10 Résumé

## 10.1 Über die Umsetzung

Trotz der umfassenden Themen dieser Diplomarbeit und der damit verbundenen hohen Erwartungen konnten viele positive Erfahrungen gesammelt und vorhandenes technisches Wissen eingesetzt und weiterentwickelt werden.

Der Einstieg in das Programmiersystem *CoDeSys* war mit einer gewissen Umstellung von den in der Ausbildung vermittelten Methoden und Verfahren verbunden. Trotzdem wurden alle Anforderung an die Kommunikation und an SPS erfüllt.

Der Sortierprozess wurde bei der Realisierung sehr einfach gehalten, da dies nicht im Mittelpunkt der Arbeit steht. Vielmehr liegt die Herausforderung bei der Analyse der Anlage und die Erkennung ihrer modularen Struktur für den objektorientierten Softwareentwurf.

Zur Prozessverbesserung bzw. -optimierung können weitere Aspekte im Sinne der Steuerungs- und Automatisierungstechnik herangezogen werden, speziell bei den Handlungen der Aggregate der Sortieranlage. Hinzu kann die Auswahl an Betriebsarten mit den Modi Manuell und Halbautomatik erweitert werden, um eine professionelle Handhabung der Anlage zu ermöglichen.

## 10.2 Programmiersystem CoDeSys VS. Step 7

Das Automatisierungssystem *Simatic S7* der Firma *Siemens* mit dem Programmiersystem *Step 7* darf sich hinsichtlich Verbreitung und Bekanntheitsgrad durchaus als Weltmarktführer bezeichnen und so wurden viele andere SPS-Programmiersysteme am Markt mehr oder minder verdrängt. Außerdem hat das Automatisierungssystem *S7* u. a. folgende Vorteile:

- einsetzbar in vielen Teilgebieten der Automatisierungstechnik,
- zusammenhängende, d. h. durchgehende Datenhaltung,
- integrierte Diagnose- und Fernwartungssysteme.

Die meisten Bildungseinrichtungen und Studierenden der Automatisierungstechnik setzen das Programmiersystem *Step 7* für die Ausbildung ein, da es zusätzlich zu seiner Verbreitung und Bekanntheit quasi ein „Standard“ in der Industrie ist.
Dennoch ist der Einsatz von *Step 7* nachteilig, da er mit relativ hohen Kosten für Hardware-Komponenten verbunden ist, insbesondere dann, wenn die Leistungsmerkmale hochwertiger Hardware nicht genutzt werden. Die hohen Kosten für die Lizenzierung der Software *Step 7* sprechen auch dagegen. Noch dazu ist die Simulation der entworfenen Programme auf dem PC wenig komfortabel. Für Visualisierungen braucht man zusätzliche Tools, da keine integrierte Visualisierung vorhanden ist.

Im Gegensatz dazu hat eine Entscheidung für ein IEC-konformes Programmiersystem wie *CoDeSys* viele Vorteile. Einige davon sind:

- große Auswahl an unterschiedlicher und preisgünstiger Hardware, die feinmodular gestaltet werden kann,
- kostenloses Programmiersystem,
- komfortable integrierte Simulation und Visualisierung,
- erlaubt standardisierte und den Ansprüchen der aktuellen Informatik genügende Programmierung.

Die Umstellung vom Programmiersystem *Step 7* zu *CoDeSys* erfordert, dass bestimmte Gewohnheiten, die beim System *Step 7* erlernt sind, überwunden und u. a. folgende Bedingungen eingehalten werden:

- Programme sind strikt als strukturierte Programme aufzubauen, d. h. die Programme sollten in universelle und mehrfach nutzbare Einheiten unterteilt werden.
- Bausteine sind hardwareunabhängig zu schreiben. Deshalb werden Programmbausteine grundsätzlich mit lokalen Variablen geschrieben.
- Grundsätzlich ist bei den POEs auf Merker und Datenbausteine zu verzichten und stattdessen mit lokalen Variablen zu arbeiten. (Becker, 2007)

## 10.3 Ausblick

Durch die Aktivitäten leistungsstarker und kostengünstiger PC-basierter Steuerungen zielen die aktuellen Trends der Steuerungstechnik in Richtung einer hardwareunabhängigen Programmierung. Unter dem Dach der IEC-Norm wachsen klassische Automatisierungsaufgaben der numerischen und binären Steuerung und Regelung sowie die Prozessbedienung und -beobachtung zusammen, das bisher in getrennten Systemen programmiert und implementiert wurde. In der Zukunft sollen auch Funktionen der industriellen Bildverarbeitung in

einem einheitlichen Automatisierungssystem PAC[1] zusammengefasst werden. Im Zuge dieser Möglichkeiten steigt der Planungs- und Koordinationsbedarf immens.

Auf der anderen Seite wird durch die Objektorientierung in der SPS-Programmierung eine gewisse Transparenz geschaffen. Der Anwender erkennt die Anlagenstruktur in der Softwarestruktur wieder. Dadurch wird er in die Lage versetzt, Änderungen und Ergänzungen, die im Lauf der Zeit infolge von Wartungs- und Instandhaltungsarbeiten in einer Anlage anfallen, selbst durchführen zu können. (Seitz, 2008)

Zukünftig sollen noch offenere Systemstrukturen entstehen, damit der Datenaustausch zwischen allen Werkzeugen vereinfacht wird, die für die Planung, den Entwurf, die Programmierung, die Visualisierung, die Inbetriebnahme und die Instandhaltung von Steuerungen zum Einsatz kommen.

[1] **P**rogrammable **A**utomation **C**ontroller

# Literaturverzeichnis


[3S 2009] 3S: *Smart Software Solutions GmbH, Technische Dokumente und Broschüre*. 2009. – URL http://www.3s-software.com

[Becker 2007] BECKER, Dr. U.: *Vorlesungsskript: Grundlagen der Automatisierungstechnik I: 3. Programmiersysteme*. Fachzentrum Automatisierungstechnik und vernetzte Systeme im BTZ Rohr-Kloster. 2007

[Berger 2009] BERGER, Hans: *Automatisieren mit STEP 7 in AWL und SCL. Speicherprogrammierbare Steuerungen SIMATIC S7-300/400*. Erlangen. 6. Auflage : Publicis Publishing, 2009. – ISBN 978-3-89578-324-1

[Bernstein 2007] BERNSTEIN, Herbert: *SPS-Workshop mit Programmierung nach IEC 61131-3*. Berlin : VDE Verlag GmbH, 2007. – ISBN 978-3-8007-2997-5

[Eltrotec 2010] ELTROTEC: *Eltrotec Sensor GmbH, Technische Dokumente und Broschüre: LT45D4.pdf*. 2010. – URL http://www.eltrotec.com/

[Engesser u. a. 1993] ENGESSER, Hermann ; CLAUS, Prof. Dr. V. ; SCHWILL, Dr. A.: *DUDEN-Informatik. Ein Sachlexikon für Studium und Praxis*. Mannheim. 2. Auflage : Bibliographisches Institut & F. A. Brockhaus AG, 1993. – ISBN 3-411-05232-5

[Hannemann 2000] HANNEMANN, Peter: *Steuerung von Industrieanlagen. Moderne Steuerungs- und Antriebstechnik, Projektierung von Anlagen, Abnahmekriterien, praktische Beispiele*. Poing : Franzis Verlag GmbH, 2000. – ISBN 3-7723-5074-7

[Hilscher 2009] HILSCHER: *Hilscher Ges. für Systemautomation mbH, Technische Dokumente und Broschüre: HilscherCIFE_Communication_Interface.pdf*. 2009. – URL http://de.hilscher.com

[isel 2010] ISEL: *isel Germany AG, Technische Dokumente und Broschüre: IT 116.pdf*. 2010. – URL http://www.isel-germany.de/

[John und Tiegelkamp 2009] JOHN, Karl H. ; TIEGELKAMP, Michael: *SPS-Programmierung mit IEC 61131-3. Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen*. Berlin Heidelberg. 4. Auflage : Springer-Verlag, 2009. – ISBN 978-3-642-00268-7

[Langmann 2010] LANGMANN, Rienhard: *Taschenbuch der Automatisierung*. München. 2. Auflage : Carl Hanser Verlag, 2010. – ISBN 978-3-446-42112-7

[Lepers 2005] LEPERS, Heinrich: *SPS-Programmierung nach IEC 61131-3*. Poing : Franzis Verlag GmbH, 2005. – ISBN 3-7723-5801-2

[Meiners 2000] MEINERS, Prof. Dr.-Ing. U.: *Vorlesungsskripte: Steuerungstechnik, Prozessautomatisierung und Betriebsysteme*. Hochschule für Angewandte Wissenschaften Hamburg, Department Informations- und Elektrotechnik. 2000. – URL `http://users.etech.haw-hamburg.de/users/Meiners`

[Neumann u. a. 1995] NEUMANN, Peter ; GRÖTSCH, Ebehard E. ; LUBKOLL, Cristoph ; SIMON, René: *SPS-Standard: IEC 1131. Programmierung in verteilten Automatisierungssystemen*. München : R. Oldenbourg Verlag GmbH, 1995. – ISBN 3-486-23348-3

[Oestereich 2006] OESTEREICH, Bernd: *Analyse und Design mit UML 2.1. Objektorientierte Softwareentwicklung*. München. 8. Auflage : Oldenbourg Wissenschaftsverlag GmbH, 2006. – ISBN 3-486-57926-6

[Plate 2010] PLATE, Prof. J.: *Grundlagen Computernetze*. 2010. – URL `http://www.netzmafia.de/skripten/netze/netz0.html#0.1`

[Ratz u. a. 2001] RATZ, Dietmar ; SCHEFFLER, Jens ; SEESE, Detlef: *Grundkurs Programmieren in Java. Band 1: Der Einstieg in Programmierung und Objektorientierung*. München Wien : Carl Hanser Verlag, 2001. – ISBN 3-446-21813-0

[Schnell und Wiedemann 2006] SCHNELL, Gerhard ; WIEDEMANN, Bernhard: *Bussysteme in der Automatisierungs- und Prozesstechnik. Grundlagen, Systeme und Trends der industriellen Kommunikation*. Wiesbaden. 6. Auflage : Vieweg & Sohn, 2006. – ISBN 3-8348-0045-7

[Seitz 2008] SEITZ, Matthias: *Speicherprogrammierbare Steuerungen. System- und Programmentwurf für die Fabrik- und Prozessautomatisierung, vertikale Integration*. München. 2. Auflage : Carl Hanser Verlag, 2008. – ISBN 978-3-446-41431-0

[Siemens 2010] SIEMENS: *Siemens AG - Automatisierungstechnik*. 2010. – URL `http://www.automation.siemens.com`

[Vogel-Heuser 2008] VOGEL-HEUSER, Birgit: *Automation & Embedded Systems*. München : Oldenburger Industrieverlag, 2008. – ISBN 978-3-8356-3150-2

[Vogel-Heuser und Wannagat 2009] VOGEL-HEUSER, Birgit ; WANNAGAT, Andreas: *Modulares Engineering und Wiederverwendung mit CoDeSys V3.Für Automatisierungslösungen mit objektorientiertem Ansatz*. München : Oldenburger Industrieverlag, 2009. – ISBN 978-3-8356-3105-2

[Weigmann und Kilian 2000] WEIGMANN, Josef ; KILIAN, Gerhard: *Dezentralisieren mit PROFIBUS-DP. Aufbau, Projektierung und Einsatz des PROFIBUS-DP mit SIMATIC S7*. Erlangen und München. 2. Auflage : Publicis MCD Verlag, 2000. – ISBN 3-89578-123-1

[Wellenreuther und Zastrow 2008] WELLENREUTHER, Günter ; ZASTROW, Dieter: *Automatisieren mit SPS - Theorie und Praxis*. Wiesbaden. 4. Auflage : Vieweg + Teubner, 2008. – ISBN 978-3-8348-0231-6

[Wikipedia 2010] WIKIPEDIA: *Wikipedia - Die freie Enzyklopädie*. 2010. – URL `http://de.wikipedia.org`

[Wratil 1996] WRATIL, Peter: *Moderne Programmiertechnik für Automatisierungssysteme. EN 61131 (IEC 1131) verstehen und anwenden*. Würzburg. 1. Auflage : Vogel Verlag und Druck GmbH & Co. KG, 1996. – ISBN 3-8023-1575-8

# Hinweise

Diese Diplomarbeit ist auf der zugehörigen CD im PDF-Format zu finden.
(siehe „`D:\Diplomarbeit_MYMMohamed.pdf`“).

Zusatzmaterialien (PDFs) und die *CoDeSys*-Projektdatei sind in einer Zusatz-CD enthalten, die bei Herrn Prof. Dr.-Ing. Ulfert Meiners hinterlegt ist.

# Versicherung über Selbstständigkeit

Hiermit versichere ich, Murwan Yousif Merghani Mohamed, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach § 25 (4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtliche oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

| Hamburg, 23. September 2010 | |
|---|---|
| Ort, Datum | Unterschrift |