SIEMENS

Michael Braun | Wolfgang Horn

Objektorientiertes Programmieren mit SIMOTION

Grundlagen, Programmbeispiele und Softwarekonzepte nach IEC 61131-3

Braun / Horn
Objektorientiertes Programmieren mit SIMOTION

Objektorientiertes Programmieren mit SIMOTION

Grundlagen, Programmbeispiele
und Softwarekonzepte
nach IEC 61131-3

Von Michael Braun und Wolfgang Horn

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in
der Deutschen Nationalbibliografie; detaillierte bibliografische Daten
sind im Internet über http://dnb.d-nb.de abrufbar.

Autoren und Verlag haben alle Texte in diesem Buch mit großer Sorgfalt
erarbeitet. Dennoch können Fehler nicht ausgeschlossen werden. Eine
Haftung des Verlags oder der Autoren, gleich aus welchem Rechtsgrund,
ist ausgeschlossen. Die in diesem Buch wiedergegebenen Bezeichnungen
können Warenzeichen sein, deren Benutzung durch Dritte für deren
Zwecke die Rechte der Inhaber verletzen kann.

www.publicis-books.de

Print ISBN 978-3-89578-455-2
ePDF ISBN 978-3-89578-946-5

Herausgeber: Siemens Aktiengesellschaft, Berlin und München
Verlag: Publicis Publishing, Erlangen
© 2016 by Publicis Pixelpark Erlangen – eine Zweigniederlassung
der Publicis Pixelpark GmbH

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt.
Jede Verwendung außerhalb der engen Grenzen des Urheberrechtsgesetzes
ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt
insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen,
Bearbeitungen sonstiger Art sowie für die Einspeicherung und Verarbeitung
in elektronischen Systemen. Dies gilt auch für die Entnahme von einzelnen
Abbildungen und bei auszugsweiser Verwendung von Texten.

Printed in Germany

Inhaltsverzeichnis

Informationen für Leser 13

1 Die Entwicklung der Steuerungstechnik 18

1.1 Die Anfänge der speicherprogrammierbaren Steuerung 19
1.2 Die SPS lernt zu kommunizieren 22
1.3 Entwicklung von Feldbussystemen 24
1.4 Integration von Anzeige-Systemen in die SPS 25
1.5 Integration von Motion Control in die SPS 27
1.6 Antriebe werden zu vollwertigen Bus-Teilnehmern 30
1.7 PLC und PAC – der Unterschied 31
1.8 Fazit zur bisherigen Entwicklung 32

2 Grundsätzliches zur objektorientierten Programmierung 33

2.1 Basis der objektorientierten Programmierung 33
2.1.1 Geschichte 33
2.1.2 Was ist anders? 34
2.1.3 Was bedeutet Objektorientierung? 35
2.1.4 Objekte und deren Interaktionen 36

2.2 Allgemeine Prinzipien von OOP 38
2.2.1 Objekte 38
2.2.2 Klassen 39
2.2.3 Vererbung 40
2.2.4 Überschreibung 41
2.2.5 Interfaces zur Objekt-Interaktion 42
2.2.6 Zusammenfassung 44
2.2.7 Vorteile der Nutzung von OOP 45
2.2.8 Nachteile der OOP 46

2.3 Tipps zur Klassenbildung 46

3 Objektorientierte Programmierung 49

3.1 Umsetzung von OOP mit SIMOTION 49
3.2 Funktionsbausteine mit Methoden 51
3.2.1 Modularisierung ohne OOP-Erweiterungen 51
3.2.2 Programm und Daten sind getrennt 53
3.2.3 Weiterentwicklung im Softwarelebenszyklus 56
3.2.4 Nachteile der Programmierung ohne OOP-Erweiterungen 57
3.2.5 Erweiterungen zu FBs und deren Zugriffsspezifikation 58
3.2.6 Nutzung von Methoden zur übersichtlichen Programmierung 59
3.2.6.1 Beispiel FB mit Methoden 60

3.2.6.2 Aufrufbeispiel des Funktionsbausteins ... 62
3.2.7 Funktionsbaustein mit Methoden zur Kommandierung ... 63
3.2.7.1 Beispiel des FB mit Kommando-Methoden ... 64
3.2.7.2 Aufrufbeispiel zum FB mit Kommando-Methoden ... 65

3.3 Klassen (CLASS) ... 67
3.3.1 Unterstützte Schlüsselworte für eine Klasse ... 68
3.3.2 Methoden (METHOD) ... 70
3.3.3 Methoden und deren Zugriffspezifikation ... 71
3.3.4 Deklaration von Klasseninstanzen ... 72
3.3.5 Regeln für die Bezeichner in einer Klasse ... 73
3.3.6 Verwendung von Klassen-Methoden ... 73
3.3.6.1 Beispiel CLASS Counter ... 74
3.3.6.2 Nutzung der Methode von CLASS COUNTER ... 75
3.3.6.3 Erweiterung von CLASS COUNTER und Nutzung von THIS ... 76
3.3.6.4 Nutzung der Methoden UP und DOWN ... 77
3.3.7 Klassen und Vererbung ... 78
3.3.7.1 Beispiel Ableitung einer Klasse ... 79
3.3.7.2 Beispiel zur Nutzung der Basis- und abgeleiteten Klasse ... 81
3.3.7.3 Weitere Aspekte zum Methodenaufruf ... 82
3.3.7.4 Beispiel von Basis- und abgeleiteter Klasse in einer Funktion ... 83
3.3.8 Abstrakte Klassen ... 84

3.4 Beispiele zur Nutzung von Ventilen mit OOP ... 85
3.4.1 Beispiel mit 4/3-Wegeventil ... 86
3.4.1.1 Beispiel für eine Klasse für 4/3-Wegeventile ... 87
3.4.1.2 Beispiel für den Aufruf der Ventile ... 88
3.4.1.3 Beispiel mit 4/3-Wegeventil mit Eilgang/Schleichgang ... 90
3.4.1.4 Beispiel einer abgeleiteten Klasse ValveControl43FS ... 91
3.4.1.5 Beispiel für Aufruf der Basisklasse und der erweiterten Klasse ... 92
3.4.1.6 Beispiel Aufruf der erweiterten Klasse mit Basisfunktion ... 93

3.5 Interfaces ... 94
3.5.1 Unterstützte Features ... 95
3.5.2 Prinzipien von Interfaces ... 96
3.5.2.1 Beispiel zur Deklaration von Interfaces ... 97
3.5.3 Repräsentanz der Interfaces im PNV von SCOUT ... 99
3.5.4 Nutzen der Interfaces ... 100
3.5.5 Interfaces als Verweis auf Klassen ... 101
3.5.6 Ventil-Klassen mit Interfaces ... 105
3.5.7 Vereinbarung des Interfaces für Ventil ... 106
3.5.7.1 Beispiel ValveControl43 mit Endschalterüberwachung ... 107
3.5.7.2 Beispiel ValveControl43 mit Error Reporting ... 109
3.5.7.3 Beispiel ValveControl43 mit Test-Error-Reporting ... 114
3.5.7.4 Beispiel Klasse HMIReporting ... 115
3.5.7.5 Beispiel ValveControl43 mit Error Reporting ... 116
3.5.8 Interface für die Neutralisierung von Peripherie-Komponenten ... 118
3.5.8.1 Anbindung von Kameras an die Steuerung ... 118
3.5.8.2 Interface-Definition für eine Kamera-Anbindung ... 124

3.5.9 Interface für neutrale Peripherie-Anbindung (reduziert) ... 125
3.5.9.1 Interface-Definition für neutrale Peripherie-Anbindung ... 127
3.5.9.2 Implementierung in Klassen ... 127
3.5.9.3 Programm der Interface-Definition und Mapping-Tabelle ... 128
3.5.9.4 Programm der Klassenimplementierung und Verwendung ... 129
3.5.9.5 Interface für Eilgang-Schleichgang-Umschaltung ... 131
3.5.9.6 Klassenimplementierung für Eilgang-Schleichgang ... 132

3.6 Weitere Optimierungen in der Klasse Ventil ... 133
3.6.1 Bisherige Realisierung von ValveControl ... 133
3.6.2 Aufbau eines Zustandsautomaten ... 134
3.6.2.1 Beispiel ValveControl43ST – Zustandsautomat über Case ... 136
3.6.2.2 Beispiel ValveControl43ST – Zustandsautomat mit Klassen ... 142

3.7 Abstrakte Klasse für unterschiedliche Antriebe ... 145
3.7.1 Funktionelle Unterschiede der jeweiligen Antriebslösungen ... 146
3.7.2 Klassenmodell zur Anbindung unterschiedlicher Antriebe ... 148
3.7.2.1 Beispiel Abstrakte Klasse „CDrive“ ... 149
3.7.2.2 Beispiel Klasse der direkt einschaltbaren Antriebe ... 150
3.7.2.3 Beispiel Klasse der Stern-Dreieck-geschalteten Antriebe ... 152
3.7.2.4 Beispiel Klasse der drehzahlgeregelten Antriebe ... 153
3.7.2.5 Beispiel Ansteuerung der unterschiedlichen Antriebe ... 158

3.8 Abstrakte Klasse versus Interface ... 160

3.9 OOP öffnet die Welt der Entwurfsmuster ... 161

4 OOP unterstützt modulare Software-Konzepte ... 163

4.1 Zusammenbau von Projekten für reale Maschinen ... 164
4.1.1 Modulgestaltung ... 165
4.1.2 Die Rolle der Software-Entwickler ... 165
4.1.3 Software modularisieren ... 167
4.1.3.1 Equipment-Module erstellen ... 168
4.1.3.2 Software-Design des Equipment-Moduls ... 170
4.1.3.3 Beispiel für die Klasse „CEMPusher“ ... 171
4.1.3.4 Beispiel für den Aufruf des Equipment-Moduls ... 176
4.1.4 Vorbereitung zur mehrfachen Wiederverwendung ... 178
4.1.4.1 Beispiel des neutralisierten Equipment-Moduls ... 178

4.2 Projektgenerator SIMOTION easyProject ... 180
4.2.1 Erweiterung des Projektgenerators um eigene Module ... 184
4.2.2 Bedienung für den Projektgenerator erstellen ... 185
4.2.3 XML-Beschreibung des Equipment-Moduls ... 187

5 Hinweise zum Design und der Entwicklung von Software ... 191

5.1 Ermittlung der Anforderungen ... 191
5.1.1 Aufsetzpunkt Bedienoberflächen ... 192
5.1.2 Aufsetzpunkt Prozessabläufe ... 192
5.1.3 Aufsetzpunkt Elemente des Maschinenbaus ... 193
5.1.4 Bestehende Lösungen ... 194

5.2 Objektorientiertes Design ... 195
5.2.1 Kapselung ... 195
5.2.2 Verantwortung einer Klasse ... 196
5.2.3 Gemeinsamkeiten und Unterschiede von Objekten ... 197
5.2.4 Prinzip der Ersetzbarkeit bei Ableitungen ... 198
5.2.5 Ermittlung der Beziehungen ... 198
5.2.6 SOLID-Prinzipien ... 200

5.3 Wiederverwendbare und leicht wartbare Software ... 200
5.3.1 Wie ist Wiederverwendbarkeit erreichbar? ... 200
5.3.2 Bibliotheken helfen ... 201
5.3.3 Wie sind Module zu entwickeln? ... 202

5.4 Organisatorische und rechtliche Aspekte ... 205
5.4.1 Umstellung muss geplant werden ... 205
5.4.2 Software muss geplant werden ... 206
5.4.2.1 Analyse der bestehenden Programme ... 206
5.4.2.2 Wiederverwendung von Software ... 207
5.4.3 Wiederverwendung und Eigentum der Software ... 209
5.4.3.1 Weitergabe von Software ... 210
5.4.3.2 Übernahme von Software ... 211
5.4.4 „Gute Software“ und objektorientiertes Design ... 212

5.5 Softwaretests sind ein Muss! ... 215
5.5.1 Modultest ... 217
5.5.2 Integrationstest ... 218
5.5.3 Systemtest ... 219
5.5.4 Abnahmetest ... 220

6 Ergänzende Themen zur Softwarestrukturierung ... 222

6.1 I/O-Referenzen ... 222
6.1.1 Deklaration ... 223
6.1.2 Bindung der Referenzen an I/O-Variablen ... 223

6.2 Namensräume ... 225

6.3 Allgemeine Referenzen ... 228
6.3.1 Deklaration und Initialisierung ... 228
6.3.2 Arbeiten mit Referenzen ... 229

7 Referenz zu Erweiterungen von SIMOTION ... 233

7.1 Allgemeine Erweiterungen des Programmiermodells ... 233

7.2 Klassen in SIMOTION ... 234
7.2.1 Konstanten und anwenderdefinierte Datentypen in Klassen ... 234
7.2.2 Benennung von Variablen in Klassen und Methoden ... 235
7.2.3 Methodenaufrufe ... 236
7.2.4 FINAL bei Methoden und Klassen ... 237
7.2.5 Deklaration von abstrakten Klassen und Methoden ... 237
7.2.6 Interface-Implementierung und Klassen-Ableitungen ... 238
7.2.7 Typumwandlungen bei Klassen und Interfaces ... 239

7.3 Instanziierung von Klassen und Funktionsbausteinen ... 241
7.3.1 Anwenderdefinierte Initialisierung von Instanzen ... 241
7.3.2 Vorbelegung von Interface-Variablen ... 243
7.3.3 Anlegen von Klassen- und Funktionsbausteininstanzen ... 243
7.3.4 RETAIN-Daten in Klassen und Funktionsbausteinen ... 244
7.3.5 Arrays variabler Länge ... 245

7.4 Tipps für kompatible und effiziente Software ... 245
7.4.1 Methoden- und Funktionsaufrufe ... 245
7.4.2 Verwendung von Enum-Werten und Konstanten ... 246
7.4.3 Nutzung vordefinierter Namensräume ... 247
7.4.4 Deklaration von Datentypen, Variablen und Methoden ... 248
7.4.5 Aufbereitung von strukturierten Daten zur Datenübertragung ... 249

8 Einführung in SIMOTION ... 252

8.1 Klassische Entwicklung von Steuerungssystemen ... 252
8.2 Neue Steuerungskonzepte erforderlich ... 253
8.3 Technologische Objekte in SIMOTION ... 254
8.4 Drei Hardware-Plattformen ... 256
8.5 Anbindung von Antrieben und Peripherie an SIMOTION ... 257
8.6 Handling-Kinematiken in SIMOTION ... 258

8.7 Das Programmiermodell von SIMOTION ... 258
8.7.1 Die Units von SIMOTION ... 259
8.7.2 Variablenmodell in SIMOTION ... 261
8.7.3 Bibliotheken in SIMOTION ... 264

8.8 Das Engineeringsystem SIMOTION SCOUT ... 265
8.9 Komponenten von SCOUT ... 266
8.9.1 Der Projektnavigator von SCOUT ... 267
8.9.2 Neues Projekt anlegen ... 268
8.9.3 Neues Gerät anlegen ... 270
8.9.4 Hardware-Konfiguration ... 273
8.9.5 Adressliste von SIMOTION ... 275
8.9.6 Achsen anlegen ... 276
8.9.7 Antriebe anlegen ... 281
8.9.8 Bahnobjekte anlegen ... 283
8.9.9 Spracheditoren in SCOUT ... 285
8.9.10 Unterstützung der Programmiersprachen ... 287
8.9.11 Programmqellen (Units) einfügen ... 288
8.9.12 Programme eingeben ... 290
8.9.13 Programme dem Ablaufsystem zuordnen ... 292
8.9.14 Integrierte Testfunktionen ... 293
8.9.15 Testen mit Status Programm ... 294

Hinweis zur Nutzung der Beispielprogramme ... 302
Stichwortverzeichnis ... 303

Abbildungs- und Tabellenverzeichnis

Abbildung 1 Beispiel eines KOP-Programms ... 20
Abbildung 2 Beispiel AWL von SIEMENS ... 20
Abbildung 3 SIMATIC S5-150K ... 22
Abbildung 4 WF470 mit Kompaktbedienfeld ... 26
Abbildung 5 S5-150K mit WF625 und WS600G ... 28
Abbildung 6 Kommunikation zwischen Objekten – objektorientiert ... 34
Abbildung 7 Kommunikation zwischen Objekten – prozedural ... 35
Abbildung 8 Hydraulikaggregat ... 38
Abbildung 9 Klasse und Objekt ... 39
Abbildung 10 Vererbung bei Klassen ... 40
Abbildung 11 Hydraulik mit HMI-Anzeige ... 43
Abbildung 12 Ventil-Zylinder-Kombination ... 52
Abbildung 13 FB_Valve ... 53
Abbildung 14 Beim Funktionsbaustein sind Programm und Daten getrennt ... 55
Abbildung 15 Funktionsbausteine müssen kopiert und geändert werden ... 56
Abbildung 16 Programmierung des FB Valve43 mit Methoden ... 60
Abbildung 17 Weiterentwicklung FB Valve43 extended ... 63
Abbildung 18 CLASS im PNV ... 67
Abbildung 19 Zugriffsdefinition bei Methoden (Quelle: IEC 61131-3 3ED) ... 72
Abbildung 20 Klassen und deren Ableitungen ... 78
Abbildung 21 Ableitung und Aufrufprinzip des Zählers ... 80
Abbildung 22 Anlage mit 4/3-Wegeventil ... 86
Abbildung 23 4/3-Wegeventil mit Eilgang/Schleichgang ... 90
Abbildung 24 Interfaces (Quelle: IEC 61131-3 ED3) ... 96
Abbildung 25 Interfacedarstellung im PNV ... 99
Abbildung 26 Interfaces in Klassen ... 100
Abbildung 27 Übersicht Ventil und HMI ... 105
Abbildung 28 Interface für Error Reporting ... 106
Abbildung 29 Delta-Picker mit zwei Bändern ... 119
Abbildung 30 Förderband mit Teilen ... 121
Abbildung 31 Produktregister SIMOTION Handling ... 122
Abbildung 32 Vorschlag eines Standard-Telegramms für Kameras ... 123
Abbildung 33 Interface für Kamera ... 124
Abbildung 34 Prinzip der Signalweitergabe in Schichten ... 126
Abbildung 35 Interface für Neutralisierung ... 127
Abbildung 36 Ventil mit neutraler Peripherie-Anbindung ... 128
Abbildung 37 Ventil mit Signalverschaltung ... 134
Abbildung 38 Zustandsautomat Ventil ... 135
Abbildung 39 Unterschiedliche Antriebe in einer Anlage ... 146

Abbildung 40 Klassenmodell CDrive ... 149
Abbildung 41 Technologieobjekte in SIMOTION ... 154
Abbildung 42 Hierarchie nach ISA-88-01 ... 168
Abbildung 43 Equipment-Modul Förderband mit Ausschieber ... 169
Abbildung 44 Software-Design des Equipment-Moduls ... 170
Abbildung 45 Zustände des Equipment-Moduls ... 172
Abbildung 46 Funktionen des Projektgenerators easyProject ... 180
Abbildung 47 Projektgenerator „easyProject“ ... 181
Abbildung 48 Oberfläche des Projektgenerators ... 182
Abbildung 49 Equipment-Module des Projektgenerators ... 183
Abbildung 50 Projekt generieren ... 183
Abbildung 51 Struktur der Daten des Projektgenerators ... 184
Abbildung 52 Equipment-Modul PusherX ... 185
Abbildung 53 Bedienung des Equipment-Moduls ... 186
Abbildung 54 Klassen- und Objektdarstellung in UML ... 199
Abbildung 55 Zusammenspiel SPS, Technologiebaugruppen und Motion Control ... 253
Abbildung 56 Integration SPS, Motion und Technologie ... 254
Abbildung 57 Technologische Objekte in SIMOTION ... 255
Abbildung 58 Die 3 Hardware-Plattformen von SIMOTION ... 256
Abbildung 59 SIMOTION mit Antrieben und Peripherie ... 257
Abbildung 60 Unterstützte Kinematiken von SIMOTION ... 258
Abbildung 61 Programme und Daten sind in Units organisiert ... 260
Abbildung 62 Variablenmodell von SIMOTION ... 262
Abbildung 63 Bibliotheken im SIMOTION-Projekt ... 265
Abbildung 64 Engineeringsystem SCOUT ... 266
Abbildung 65 SCOUT Workbench ... 267
Abbildung 66 Projektnavigator ... 268
Abbildung 67 Neues Projekt anlegen – Ergebnis ... 269
Abbildung 68 Gerät einfügen ... 270
Abbildung 69 Eigenschaften – Ethernet Schnittstelle PNxIO ... 271
Abbildung 70 PG/PC-Kommunikation einrichten ... 272
Abbildung 71 SIMOTION-Gerät einfügen mit HW-Konfig öffnen ... 273
Abbildung 72 HW-Konfig mit dem SIMOTION-Gerät ... 274
Abbildung 73 Adressliste ... 275
Abbildung 74 SCOUT mit eingefügter D435-2 ... 276
Abbildung 75 Drehzahlachse anlegen ... 277
Abbildung 76 Achskonfiguration Achstyp ... 279
Abbildung 77 Achskonfiguration – Zusammenfassung ... 281
Abbildung 78 Antriebszuordnung zur Achse ... 282
Abbildung 79 Achsassistent für Antrieb zuordnen ... 283
Abbildung 80 Bahnobjekt einfügen ... 284
Abbildung 81 Bahnobjekt im PNV ... 284
Abbildung 82 Delta-Picker 3D ... 285

Abbildung 83 Programmiersprachen in SCOUT ... 286
Abbildung 84 Vergleichsfunktion in SCOUT ... 287
Abbildung 85 Programmquellen einfügen ... 288
Abbildung 86 ST-Quelle einfügen ... 289
Abbildung 87 ST-Editor zur Programmierung ... 290
Abbildung 88 Programm für Ablaufsystem ... 291
Abbildung 89 Ablaufsystem von SIMOTION ... 292
Abbildung 90 Background-Task: Programme zuordnen ... 293
Abbildung 91 Status-Anzeigen in SCOUT ... 294
Abbildung 92 Status Programm ermöglichen ... 295
Abbildung 93 Status-Anzeige des Programms ... 296
Abbildung 94 Aufrufkette Methoden ... 297
Abbildung 95 Aufrufpfad/Taskauswahl einstellen ... 298
Abbildung 96 Prinzip Ablauf Status Programm ... 300

Tabellenverzeichnis

Tabelle 1 Schlüsselworte für Klassen ... 68
Tabelle 2 Deklaration von Instanzen einer Klasse ... 72
Tabelle 3 Schlüsselworte für Interfaces ... 95
Tabelle 4 Gegenüberstellung abstrakte Klasse und Interface ... 160
Tabelle 5 Vordefinierte Namensräume (Scopes) ... 247

Informationen für Leser

Die steigenden Anforderungen im Maschinenbau zu immer flexibleren Lösungen führen auch zu Veränderungen bei der Programmierung der Steuerungssysteme. Da der Weg zu mechatronischen Systemen bereits beschritten wird, erfolgen daraus starke Anforderungen an die Modularität der Software und die Art der Programmierung. Die zunehmende Bildung von modularen Funktionseinheiten in Maschinen bringt es mit sich, dass sich diese Modularität auch in der Software widerspiegeln muss. Die in der IEC 61131-3 ED3 definierten Erweiterungen zur objektorientierten Programmierung unterstützen in hohem Maße diese gewünschte Modularisierung in der Software. Somit wird in der Automatisierungstechnik derselbe Wandel bei der Software-Erstellung stattfinden, wie er bereits ab Mitte der 80er Jahre bei der PC-Programmierung erfolgt ist.

Wenn die Applikationssoftware von Automatisierungssystemen viel besser konzipiert, klarer strukturiert, leichter änderbar und modularer aufgebaut sein soll, führt kein Weg an der objektorientierten Programmierung vorbei. SIMOTION stellt ab Softwarestand 4.5 die Möglichkeit der Nutzung von objektorientierter Programmierung nach der IEC 61131-3 ED3 zur Verfügung. Dieses Buch unterstützt den Umgang mit dieser Denk- und Programmierweise. Um die Programmiermethodik leichter zu erlernen, helfen die zum jeweiligen Thema passenden Beispiele. Diese Beispiele sind über die einzelnen Themen aufeinander aufbauend gestaltet. Am Ende entsteht daraus ein komplett in OOP programmiertes, wiederverwendbares Maschinenmodul.

Dieses Buch eignet sich für *alle*, die sich mit der objektorientierten Programmierung in der Automatisierungstechnik vertraut machen möchten. Der erste Teil des Buchs beschäftigt sich mit den Grundlagen zur objektorientierten Programmierung; Basis ist die Realisierung in SIMOTION nach der IEC 61131-3 ED3 (Kapitel 1 bis 6). Der zweite Teil stellt eine allgemeine Einführung in das SIMOTION-System selbst zur Verfügung (Kapitel 7 und 8).

Für Anwender, die das Steuerungssystem SIMOTION noch nicht kennen, sei hier empfohlen, zuerst den zweiten Teil „Einführung in SIMOTION" zu lesen. Dort sind die Grundlagen des Steuerungssystems SIMOTION und das dazu gehörige Engineering-System SIMOTION SCOUT erläutert.

Für das Aneignen der Inhalte zur objektorientierten Programmierung werden Kenntnisse über Hochsprachenprogrammierung wie Structured Text oder Pascal vorausgesetzt. Des Weiteren sind für das Verständnis Grundlagen über speicherprogrammierbare Steuerungen und deren Systemverhalten erforderlich.

Sie werden feststellen, dass in verschiedenen Kapiteln bestimmte Sachverhalte wiederholend beschrieben sind. Grund dafür war unser Wunsch, mit möglichst wenigen Querverweisen auf andere Kapitel auszukommen. Damit können Sie zwischen den Kapiteln springen, ohne den Faden zu verlieren.

Wir haben eine Reihe von aufeinander aufbauenden Beispielen entwickelt und diese auch im Buch dargestellt. Die Beispiele sind bewusst einfach gehalten und zielen darauf, die Nutzung der objektorientierten Mechanismen transparent dar-

zustellen. Aus diesem Ansinnen heraus ergab es sich, dass einige Beispiele doch einen gewissen Umfang erreicht haben. Uns ist natürlich klar, dass niemand sich die Arbeit machen möchte, den abgedruckten Programmcode abzutippen. Deshalb sind die enthaltenen Beispiele für unsere Leser im Internet zum Download verfügbar. Sie finden unter www.siemens.com/simotion entsprechende Links zu den Beispielen. Zu beachten sind die aufgeführten Bedingungen zur Nutzung der Beispiele.

Persönliche Anmerkungen der Autoren

Michael Braun

Über diese Anmerkungen habe ich mir viele Gedanken gemacht und war mir lange Zeit sicher, sie nicht zu schreiben. Vielleicht auch aus dem Grund, weil ich selbst derartige Kapitel in Büchern oft überspringe. Aber man lernt im Leben immer wieder etwas dazu – hier bietet sich die gute Gelegenheit, unseren Lesern etwas von mir und der Motivation zur Buchentstehung zu vermitteln.

Die Entwicklungen in der Automatisierungstechnik habe ich seit Anfang meines Berufslebens als spannendes Betätigungsfeld erlebt. Vor allem stand dabei die Erstellung von Software im Vordergrund. Programme zu entwickeln, die am Ende eine Produktionsanlage dazu befähigen, ihre Aufgabe zu erledigen, war und ist für mich bis heute eine absolut interessante Beschäftigung. Diese Tätigkeit ist geprägt durch stetiges Lernen von neuen Dingen. Nur leider fällt die Fähigkeit, gute Software zu schreiben, für einen Programmierer (so auch für mich) nicht vom Himmel. Meinem Erleben nach durchläuft ein Programmierer drei verschiedene Phasen.

Die erste Phase ist geprägt vom Einstieg in ein neues System oder eine Programmiersprache. Bestimmte Zusammenhänge versteht man noch nicht und macht die ersten Gehversuche. Dabei lernt man die Grundzüge kennen und schreibt die ersten Programme. In der Regel würde man diese später in den Abfalleimer werfen, weil sie ineffektiv und vielleicht umständlich erstellt wurden. Aber weil sie trotzdem funktioniert haben, gleitet man in die zweite Phase.

In der zweiten Phase nähert man sich dem Punkt, bei dem einem alle Elemente der Sprache bekannt sind, und man wendet „geniale“ Tricks an, um eine Lösung zu formulieren. Die Tatsache, dass die Lösung niemand nachvollziehen oder verstehen kann, ignoriert man geflissentlich. Der Stolz über die eigene Genialität und wie toll man seine Programme findet, überdeckt das eventuell vorhandene schlechte Gewissen. Dies ist die gefährlichste Zeit eines Programmierers, denn er schreibt unleserlichen Code. Nun ist es an der Zeit, dass ein Kollege einem deutlich sagt, was für einen Käse man da programmiert hat (ist mir selbst passiert). Eine weitere Heilungschance ergibt sich daraus, dass man den „genialen“ Code nicht wieder los wird und man ihn ständig selbst pflegen muss (was eine gute Möglichkeit ist, sich vermeintlich unersetzbar zu machen). Diese Schocks helfen, die zweite Phase zu überwinden.

Hat man die dritte Phase erreicht, ist man über den Berg und in der Lage, gut leserliche, einfach pflegbare und damit auch wiederverwendbare Software zu schreiben. Jeder Programmierer sollte danach streben, möglichst schnell diese Phase zu erreichen.

Im Maschinenbau ist die Software-Erstellung heute eine Teamaufgabe. Die Zeit, in der ein Entwickler im stillen Kämmerlein seine Software ausbrütet, ist längst vorbei. Der Erstellprozess lebt von einem ständig stattfindenden Informationsaustausch zwischen den verschiedenen Teammitgliedern, aber auch vom Austausch zwischen den Konstruktionsdisziplinen. Je besser die Kommunikation zwischen den Beteiligten läuft, umso mehr kann jeder einzelne Entwickler seine Teilaufgabe effizient umsetzen. Software zu entwickeln ist ein iterativer Prozess, der immer wieder Schleifen der Verbesserung durchläuft. Mit jeder Verbesserung nähert man sich einem imaginären Optimum. Imaginär deshalb, weil sich auch die Definition des Optimums mit der Zeit verändert.

Somit ist es eine Tatsache, dass die Software eigentlich nie fertig wird. Selbstverständlich erreicht man mit dem Entwicklungsfortschritt früher oder später einen lieferfähigen Stand. Aber dieser Lieferstand bildet gleichzeitig die Basis für die nächste Entwicklungsstufe.

Wie die Anwendersoftware ist auch das Entwicklungssystem selbst eine Software, die Veränderungen unterworfen ist. Objektorientierte Programmierung stellt eine Systemerweiterung in der Entwicklungsumgebung für Automatisierungstechnik zur Verfügung, die den Entwickler von Applikations-Software wesentlich besser unterstützen soll und kann als klassische Entwicklungsumgebungen. Allerdings muss man die dazu notwendigen Mechanismen kennenlernen und verstehen.

Dazu wurden Unterlagen für die SIMOTION-Dokumentation mit den Erweiterungen zur objektorientierten Programmierung erstellt. Diese Unterlagen erreichten irgendwann einen Umfang, der zu der Idee eines Buches über OOP geführt hat. Dieses Buch halten Sie nun in Händen – verbunden mit meinem Wunsch, dass beim Lesen viele neue Ideen zur Verbesserung Ihrer eigenen Software entstehen. Lernen und umsetzen müssen Sie selbst, aber vergessen Sie vor allem den Spaß dabei nicht!

Danke

Die Einführung von objektorientierter Programmierung in Automatisierungssystemen ist noch nicht so weit fortgeschritten wie bei der Programmierung von PC-Systemen. Dieses Buch soll dabei unterstützen, den Umgang mit dieser Methodik zu erlernen. Entstehen konnte es nur, weil viele Kollegen mitgeholfen haben. Die Diskussionen mit verschiedenen dieser Kollegen haben mir genau dann am meisten weitergeholfen, wenn ich gedanklich in einer Sackgasse steckte. Mein Dank richtet sich an alle Helfer und Unterstützer.

Zu Anfang musste ich meine prozedurale Denkweise aufgeben. Dabei haben mir vor allem unser Software-Architekt Dr. Michael Schlereth, mein Kollege und Mitautor Dr. Wolfgang Horn und Thomas Hennefelder aus unserem Applikationszentrum geholfen. Mit ihnen konnte ich meine Programme und Grundgedanken jederzeit besprechen. Wenn manchmal der Satz fiel „So macht man das aber nicht!“, war ich gezwungen, meinen eingeschlagen (Irr-)Weg zu verwerfen. Genau das hat mich vorwärts gebracht. Weitere Diskussionspartner und Hilfe aus der SIMOTION-Entwicklung waren Rumwald Hermann, Klaus Lummer und Nils Focke.

Viel Unterstützung habe ich durch meine Kollegen aus Produktmanagement und Systemmanagement erfahren. Sie waren immer bereitwillig beim Korrekturlesen. Deshalb danke ich den Kollegen Benno Bruss, Jürgen Büssert, Kai Flucke, Alexander Heider, Manfred Popp und Wolfgang Wiedemann vom Integrationstest. Nicht

zu vergessen sind meine Chefs Erwin Neis, Josef Hammer und Rudolf Teplitzky. Sie haben mich stets bestärkt und mir den notwendigen Freiraum verschafft.

Für die Hilfe bei der Erstellung und Tests der Beispielprogramme danke ich besonders meinem Kollegen Frank Becker vom APC Köln.

Wolfgang Horn

Als Michael Braun mit der Idee auf mich zukam, ein Buch über objektorientierte Programmierung zu schreiben, dachte ich zunächst: Wieso sollte man zu dieser Thematik noch ein Buch schreiben? Es gibt ja bereits unzählige Veröffentlichungen aus Sicht der unterschiedlichsten Programmiersprachen und Anwendungsgebiete zu diesem Thema. Bei näherer Betrachtung wurde allerdings schnell klar, dass es im Umfeld der Automatisierung nicht allzu viel Literatur gibt, die den Einstieg in die objektorientierte Steuerungsprogrammierung angemessen unterstützt.

Aus meiner eigenen beruflichen Praxis sind mir die Möglichkeiten und das Potenzial sehr gut bekannt, das sich erschließt, wenn man OOP anwendet. Selbstverständlich setzt dies die direkte Unterstützung durch die Programmierumgebung und die verwendete Programmiersprache voraus. Mit der Umsetzung der 3rd Edition der IEC in SIMOTION steht unseren Anwendern nun solch ein direkter Zugang zu dieser Welt zur Verfügung. Damit allein ist es jedoch nicht getan.

In vielen Diskussionen mit Michael Braun und anderen Kollegen, die sich mit Begeisterung dieser Thematik stellen, wurde mir klar, dass es nicht ausreicht, sich nur die neuen Sprachkonstrukte anzueignen. Um einen durchschlagenden Effekt zu erzielen muss man zusätzlich wissen, zu welchem Zweck man welches Sprachmittel bzw. welche Technologie einsetzen kann und sollte. Aus den Fragestellungen der Kollegen wurde deutlich, dass es neben einfachen Beispielen für Programmierkonstrukte auch einer Anleitung bedarf, welche neuen Lösungsmöglichkeiten für die Aufgabenstellungen der Automatisierungstechnik die objektorientierte Programmierung bietet.

Auf viele dieser Aspekte gehen wir in diesem Buch ein. Dem Leser soll eine Hilfe geboten werden, sich an Beispielen aus der Steuerungstechnik in die Thematik der Objektorientierung einzuarbeiten. Den Anwendern, die bereits mit anderen objektorientierten Programmiersprachen Erfahrungen gesammelt haben, werden wir versuchen, die speziellen Belange der Steuerungstechnik näher zu bringen. Steuerungsspezifische Sprachen sind darauf zugeschnitten, bereits bei der Programmierung sicherzustellen, dass bestimmte Programmlaufzeiten eingehalten werden. Es sollen möglichst wenig Laufzeitfehler bei der Programmbearbeitung auftreten. Folglich wird man Konstrukte zur dynamischen Objekterzeugung und -zerstörung in diesem Buch vergeblich suchen. Dass diese im Umfeld der Steuerungsprogrammierung nicht angeboten werden, hat den einfachen Grund, dass dadurch die Echtzeitfähigkeit der Applikation erheblich beeinträchtigt werden könnte.

Zu Beginn meiner beruflichen Laufbahn gehörte die Programmiersprache Structured Text (ST) durchaus noch zu den Außenseitern in der Steuerungstechnik. Die zunehmende Komplexität der Programme, wie wir sie insbesondere im Umfeld von Motion Control erleben, erzwingt jedoch förmlich den Übergang zu höheren Programmiersprachen wie ST. Eine logische Fortsetzung dieses Weges ist in der Unterstützung der Objektorientierung durch Steuerungssysteme zu sehen. Ich bin

der Ansicht, dass in nur wenigen Jahren diese Art der Steuerungsprogrammierung zum Standard bei Automatisierungslösungen gehören wird.

Mit diesem Buch möchten wir einen Beitrag dazu leisten, dass sich die objektorientierte Programmierung im Umfeld der Steuerungstechnik weiter durchsetzt. Im Hinblick auf Modularität und Kombinierbarkeit von Softwaremodulen lassen sich gerade bei komplexeren Applikationen enorme Effekte erzielen. Auf dem Weg dahin wünsche ich, wie schon mein Kollege Michael Braun, unseren Lesern viel Freude und Geduld beim Lernen und Ausprobieren.

1 Die Entwicklung der Steuerungstechnik

Eine der wesentlichen Erweiterungen der IEC 61131-3 ED3 beschreibt die Mechanismen zur objektorientierten Programmierung von Steuerungssystemen in der Automatisierungstechnik. Damit ist ein weiterführender Weg zur Standardisierung von Programmen in den Automatisierungssystemen möglich, die mit den bisherigen prozeduralen Programmiermethoden an verschiedene Grenzen stößt.

Die zunehmende Flexibilisierung der Anlagen im Maschinenbau erzwingt eine Veränderung der bestehenden Programme dahingehend, dass modulare Maschinenkonzepte sich auch in der Software widerspiegeln. Die Programmgestaltung ist damit einer zunehmenden Modularisierung unterworfen, mit eigenständig funktionsfähigen und ausgetesteten Softwaremodulen. Innerhalb der unterschiedlichen Maschinen können die Module zu einem funktionsfähigen Ganzen kombiniert werden.

Wer am internationalen Markt wettbewerbsfähig sein – und bleiben – will, muss die Inbetriebnahmezeiten der Maschinen minimieren können. Dies ist nur möglich, wenn standardisierte Programmmodule in ihrer Kombination fehlerfrei funktionieren und eine Nachbesserung in der Inbetriebnahmephase entfällt bzw. so gering wie möglich gehalten wird.

Somit ergeben sich an die Architektur der Anwendungs-Software und an die Automatisierungs-Systeme folgende Anforderungen:

- Die Software muss modular aufgebaut sein. Die Module sind insgesamt gekapselt und für sich eigenständig funktionsfähig.
- Die Eigenständigkeit der Module führt dazu, dass auch die Daten zu einem Modul gehören, also Bestandteil des Moduls sind. Damit müssen die Daten an das Modul gebunden werden können und dürfen nicht außerhalb des Moduls änderbar sein.
- Die Kombinationsfähigkeit der Module und Zusammenführung zu einem funktionsfähigen Ganzen setzt voraus, dass die Module in sich getestet sind. Somit müssen die Module so gestaltet sein, dass sie in einer Testumgebung testbar sind.
- Die Kombinierbarkeit von unterschiedlichen Modulen in einer Umgebung erfordert nur sehr geringe bzw. möglichst keine Softwareanpassungen. Um dies zu erreichen, wird die Interaktion zwischen den Modulen über neutrale Schnittstellen modelliert.
- Da die Maschinen in ihrer Gesamtheit mit allen Komponenten über ihre Lebenszeit Innovationen unterworfen sind, ist es unabdingbar, dass auch die Software auf neue Gegebenheiten angepasst werden kann. Dabei ist es wünschenswert, dass bereits erprobte und funktionsfähige Module möglichst unangetastet bleiben.
- Eine möglichst unabhängige Entwicklung von Softwaremodulen in verschiedenen Entwicklungsbereichen beim Maschinenhersteller ist unbedingt erforderlich. Dazu bedarf es entsprechender Mechanismen in den Program-

miersprachen, um Abhängigkeiten zwischen Softwaremodulen zu reduzieren. Das bedeutet, dass eine Vereinbarung von Schnittstellen zwischen den verschiedenen Softwaremodulen zum Informationsaustausch definierbar ist.

Um allen diesen Ansprüchen gerecht zu werden, ist ein besseres Programmiermodell notwendig als die bisherige prozedurale Programmierung in der Steuerungstechnik.

Die Automatisierungslösungen haben sich über viele Jahre entwickelt und waren auch schon in der Vergangenheit verschiedenen Wechseln bei der Programmierung unterworfen. Diese Wechsel ergaben sich aufgrund von unterschiedlichen Weiterentwicklungsschritten in der Automatisierungstechnik und haben die Anwender geprägt. Jede erzwungene Änderung löste teilweise auch Hemmnisse bei den Nutzern aus, die der Anwendung von neuen Methodiken in der Programmierung entgegenstanden. Neue Methodiken konnten sich nur dann etablieren, wenn eventuell existierende Hemmnisse beseitigt bzw. ausgeräumt wurden.

Mit der objektorientierten Programmierung steht der Automatisierungstechnik ein weiterer Paradigmenwechsel bevor. Die von den Anwendern gesammelten Erfahrungen mit den erfolgten Entwicklungen in der Automatisierungstechnik haben zu einer bestimmten Programmiermethodik in den jeweiligen Anwendungsgebieten geführt. Diese existierenden Methodiken müssen überprüft, unter Umständen verändert oder gar verworfen werden. Dies kann wiederum Vorbehalte oder Hemmnisse hervorrufen. Wenn vorhandene Vorbehalte nicht ausgeräumt werden können, behindern sie die Umsetzung von neuen Methodiken. Solche Hemmnisse können zum Beispiel in der Prägung der Anwender aus der bisherig erfolgten, vergangenen Entwicklung liegen.

Deswegen sei hier ein kleiner Ausflug in die Entwicklung der Automatisierungstechnik mit dem Fokus auf den Konsequenzen in der Programmierung erlaubt. Die im Folgenden beschriebenen Entwicklungsschritte sind durchaus repräsentativ, aber chronologisch nicht unbedingt in der dargestellten Reihenfolge abgelaufen. Sie haben auch nicht den Anspruch auf Vollständigkeit. Jedoch brachte tatsächlich jeder dieser Entwicklungsschritte Veränderungen in der Programmierung mit sich. Es geht nun darum, über diese Konsequenzen nachzudenken und eventuell verbliebene Hemmnisse aus der Vergangenheit auszuräumen.

1.1 Die Anfänge der speicherprogrammierbaren Steuerung

Die Anwendung der ersten Speicherprogrammierbaren Steuerungen (SPS) war geprägt durch die Tatsache, dass die Nutzer dieser neuen Steuerungsgeneration kaum Programmierkenntnisse besaßen. Die Automatisierungsanlagen wurden bisher durch verdrahtete Relais- und Schützsteuerungen oder durch verdrahtete Elektronikkomponenten realisiert. Nun wurden die Konstrukteure, das Inbetriebsetzungs- und Servicepersonal mit Programmierung statt Verdrahtung konfrontiert. Somit musste die Programmiermethodik auf die vorhandenen Kenntnisse bei den Anwendern Rücksicht nehmen.

Diese Kenntnisse bestanden aus der Fähigkeit, Stromlaufpläne zu lesen und die Funktion in Form von Verdrahtung umzusetzen. Also lag es nahe, dass die Pro-

grammiermethodik diese vorhandenen Fähigkeiten unterstützte. So entstand, angelehnt an eine Stromlaufplan-Darstellung, die Kontaktplan-Programmierung (Abbildung 1) sowie, orientiert an elektronischen Plänen, die Funktionsplan-Programmierung.

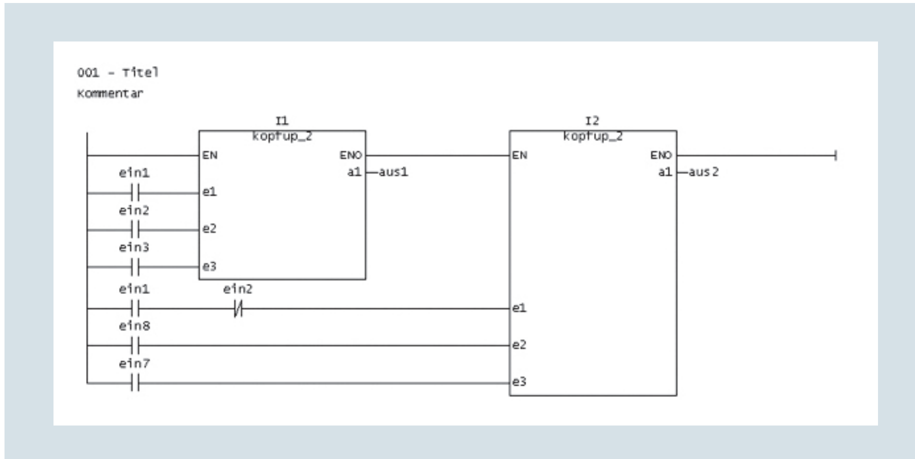

Abbildung 1 Beispiel eines KOP-Programms

Komplexere Funktionselemente des Systems wie beispielsweise Zeitglieder oder Zähler wurden als Box mit entsprechenden Ein- und Ausgängen dargestellt. Damit der Anwender eigene komplexe Funktionsmodule erstellen konnte, wurde in den Systemen die Möglichkeit geschaffen, eigene Funktionsbausteine oder Funktionen zu programmieren, die wiederum als Komplexelemente (Boxen) mit Ein- und Ausgängen im Kontaktplan oder Funktionsplan dargestellt wurden.

Solche komplexeren Elemente benötigten eine andere Programmierung als die funktionell begrenzte Kontakt- oder Funktionsplanprogrammierung. Hierfür wurde den Anwendern die assemblernahe Programmierung in Anweisungsliste (AWL) (Abbildung 2) oder Instruction List (IL) zur Verfügung gestellt.

```
U     #Switch_On             #Switch_On     -- Motor einschalten
UN    "Automatikbetrieb"     A4.2           -- Speicherfunktion
S     #Engine_On             #Engine_On     -- Motor wird eingeschaltet
U(
O     #Switch_Off            #Switch_Off    -- Motor ausschalten
ON    #Failure               #Failure       -- Motorstörung, führt zum Ausschalten
)
R     #Engine_On             #Engine_On     -- Motor wird eingeschaltet
NOP   0
```

Abbildung 2 Beispiel AWL von SIEMENS

Damit konnte der Anwender durchaus komplexere Programme erstellen, die unter anderem auch Berechnungsfunktionen und Sprünge innerhalb des Programms erlaubten. Die zur Verfügung stehenden Befehle waren jedoch sehr stark durch den jeweiligen Steuerungshersteller geprägt. Deshalb war eine Übernahme von Programmen zwischen verschiedenen Steuerungen nur schwer möglich. Somit mussten die Anwender die verschiedenen Dialekte und die Verhaltensweisen individuell für jeden Steuerungshersteller erlernen.

Des Weiteren hatte diese assemblernahe Mnemonik (= ein für den Menschen lesbares Kürzel für einen Assemblersprachenbefehl) den Nachteil, dass die Strukturierungsmöglichkeiten in der Programmierung sehr begrenzt und umständlich waren. Viele Anwender unterlagen darüber hinaus auch noch dem Zwang, sich die Programmierung selbst beibringen zu müssen. So entstanden mehr oder weniger gut lesbare Programme, bis hin zu nicht wartbaren Programmteilen.

Trotz all dem war die schnelle Änderbarkeit der Programme und die damit verbundenen Flexibilität im Vergleich zur Verdrahtung ein immenser Vorteil, welcher der speicherprogrammierbaren Steuerung zum Siegeszug verhalf. Damit hat die Programmierung im Maschinenbau Einzug gehalten, und sie ist heute nicht mehr wegzudenken.

Hemmnisse

Allerdings brachte die leichte Änderbarkeit das Problem mit, dass die Anwender teilweise geneigt waren, die Programme nach der Methode „Trial and Error" zu erstellen. Vor allem wenn die Programmierer unter Zeitdruck die Maschinenfunktion zur Abnahme oder Auslieferung fertigstellen mussten. Dieses Vorgehen, Programme über das Austesten bei der Inbetriebnahme fertigzustellen, führte zu einer unerwünschten Variantenvielfalt der Programme und damit zu einer unstrukturierten Software. Die Wiederverwendbarkeit war nachhaltig negativ beeinflusst.

Oft ist in Firmen immer noch zu beobachten, dass die ursprünglich geplante Zeit zur Erstellung der Software in einem fortschreitenden Maschinenbauprojekt stetig schrumpft. Der Liefertermin ist zugesagt, aber weitere technische Änderungen an der Anlage (auch auf Kundenwunsch) führen zu ungeplanten Mehraufwendungen und verschärfen die Terminsituation. Änderungen an der Anlagenausführung lassen sich wahrscheinlich nie vermeiden, da sie ja meist technisch oder anderweitig begründbar sind. Jedoch müssten größere Mehraufwände logischerweise zu der Konsequenz einer Verschiebung des Liefertermins führen.

Der Endkunde wird eine Terminverschiebung nur akzeptieren, wenn eindeutig dargelegt werden kann, dass die Mehraufwände durch seine Änderungswünsche zustande gekommen sind. Nachzuweisen ist das aber nur, wenn der Lieferumfang eindeutig und klar formuliert wurde. Ist beim Verkauf der Anlage der Lieferumfang nicht eindeutig beschrieben, führt das zwangsläufig zu einer Interpretation des zugesagten Umfangs. In diesem Fall wird in der Regel der Endkunde auf Einhaltung des Liefertermins bestehen. Daraus folgt, dass die Zeit für die Softwareerstellung knapper wird. „Software lässt sich doch einfach ändern, und das geht ja schnell." Aufgrund dieser Einstellung ergibt sich häufig die Schwierigkeit, dass unfertige Software der Inbetriebnahme übergeben und während der verbleibenden Zeit fertiggestellt werden muss. Die festgelegten Designrichtlinien sind nicht mehr einzuhalten und die Wiederverwendbarkeit leidet.

Lösungen

Die exakte Planung der Software ist der Schlüssel zum Erfolg. Um die Softwarefunktionalität eindeutig zu beplanen, ist eine genaue Ermittlung der Anforderungen notwendig. Basierend auf diesen Anforderungen ist die erforderliche Umsetzung für die Software und deren Strukturierung durchführbar. Außerdem können daraus die benötigte Entwicklungszeit und die daraus resultierenden Termine abgeleitet werden.

Eine regelmäßige Abstimmung mit dem Auftraggeber vermeidet Überraschungen auf beiden Seiten. Die geplante Umsetzung ist dazu in Zusammenarbeit mit dem Kunden rechtzeitig schriftlich zu fixieren.

Ebenso ist die Realisierung der Softwareentwicklung über einen vorgegebenen Prozess durchzuführen. Damit ist ein klarer Status des Entwicklungsfortschritts vorhanden und Abweichungen werden frühzeitig sichtbar. Bei frühzeitigem Erkennen von Abweichungen ist somit ein entsprechendes Gegensteuern noch möglich.

1.2 Die SPS lernt zu kommunizieren

Die Ressourcen wie z. B. der Speicher oder die Arbeitsgeschwindigkeit einer SPS waren am Anfang begrenzt. Natürlich war auch die hohe Skalierbarkeit von unterschiedlichen Leistungsklassen der Steuerungen zu Beginn noch nicht weit ausgebaut. Konsequenz dieser Tatsache war, dass häufig mehrere Steuerungen in einer Anlage eingesetzt wurden. War dies der Fall, mussten diese Steuerungen miteinander synchronisiert werden. Die einfachste, aber sehr begrenzte Möglichkeit dazu war die Synchronisation über Ein- und Ausgänge. Mussten mehr Daten ausgetauscht werden, war diese Möglichkeit nicht mehr tragfähig.

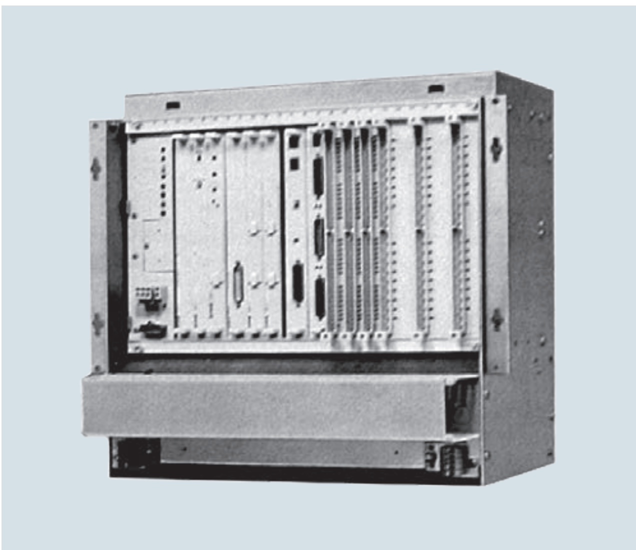

Abbildung 3 SIMATIC S5-150K

Die Lösung des Problems erfolgte über die Nutzung von speziellen Kommunikationsbaugruppen, die in die SPS gesteckt werden konnten. Diese sogenannten Rechnerkopplungen (z. B. RK512) nutzten standardisierte Protokollrahmen (z. B. 3964R-Protokoll) für den Datenaustausch und wurden über Treiberbausteine in der SPS betrieben. Auch wenn mit diesen Baugruppen im Wesentlichen nur Punkt-zu-Punkt-Kopplungen möglich waren, führte der Einsatz dieser Baugruppen zu einer gesteigerten Kommunikationsfähigkeit der SPS. Abbildung 3 zeigt eine Kompaktsteuerung aus der Zeit um 1980.

Ein weiterer Vorteil dieser Rechnerkopplungen war die Fähigkeit zur Leitrechner-Anbindung. Damit konnten mit derselben Technik Kommunikationsbeziehungen zwischen der Steuerungs- und Leitrechnerebene zur Produktionsdatenerfassung aufgebaut werden.

Durch die Kommunikation zwischen Steuerungen zur Synchronisation waren die Programmierer gezwungen, die Programme zusätzlich zur eigentlichen Steuerungsaufgabe um eben die entsprechenden Kommunikationsmechanismen zu erweitern. Das bedeutete die Berücksichtigung folgender Aspekte in der Software:

- Die Verbindung zu den einzelnen Geräten/Steuerungen mussten auf- und ggf. abgebaut werden. Dabei waren die Verhaltensweisen beim Einschalten bzw. Abschalten der einzelnen Komponenten zu berücksichtigen.
- Verbindungen mussten je nach Anzahl der Teilnehmer verwaltet werden. Die Daten waren entsprechend zu strukturieren und für den Datenaustausch an die jeweilige Kommunikationsbaugruppe zu übertragen bzw. zu entsorgen.
- Verbindungsüberwachungen mussten implementiert und eine passende Reaktion im Maschinenablauf eingebaut werden.
- In bestimmten Betriebszuständen konnte es zu Verlusten von Telegrammen kommen. Dies passierte, wenn zum Beispiel eine Steuerung aufgrund ihrer temporär verlängerten Zykluszeit den Telegrammpuffer einer Kommunikationsbaugruppe nicht mehr entsorgen konnte. Auch hierfür bedarf es geeigneter Reaktionen im Programm.
- Produktionsrelevante Daten mussten in der Steuerung gesammelt und für die Übertragung an Leitrechner bereitgestellt werden.

Hemmnisse

Der Umfang der Programme in den Steuerungen wuchs um die Anteile für Kommunikation. Diese größer werdenden Programme mussten beherrschbar bleiben. Eine passende Strukturierung der Software und eine damit verbundene modulare Programmierung waren die logische Konsequenz. Neben der eigentlichen Programmierung von Abläufen mussten sich die Programmierer somit Gedanken über die Datenstrukturen in der Steuerung machen. Zum einen war es sinnvoll, die Programmierung der Kommunikation von den Maschinenabläufen zu trennen. Andererseits beeinflussten die Kommunikationsdaten selbstverständlich die Maschinenabläufe. Ein geeignetes Verbinden von Kommunikation und Maschinenablauf war zu implementieren. Dabei sollten Softwareänderungen in einem Bereich (z. B. Maschinenablauf) nicht unbedingt auch Änderungen im anderen Bereich (z. B. Kommunikation) nach sich ziehen.

Lösungen

Ohne klare Definitionen und Strukturen entstehen Softwareunikate, die die Pflege und Wartung auf Dauer schwieriger oder gar unmöglich machen. Software-Design, Modularisierung und Standardisierung von Software sind bis heute eine klare Aufgabe des Entwicklungsprozesses. Durch wiederverwendbare Softwareanteile und eine durchdachte Struktur sind die Aufwände reduziert und die Termine einfacher zu beplanen.

1.3 Entwicklung von Feldbussystemen

Der zentrale Aufbau der speicherprogrammierbaren Steuerungen mit Zentralbaugruppen (CPU) und im gleichen Baugruppenträger gesteckten Peripheriebaugruppen erzwang einen großen Verdrahtungsaufwand vom Schaltschrank zu den in der Maschine verbauten Aktoren, z. B. Ventilen, oder den entsprechenden Bedienkomponenten wie Schaltern und Tastern zur Steuerung der Anlage. Dieser Aufwand bei der Verkabelung war die Triebkraft zur Veränderung. Die Elemente (Aktoren und Sensoren) befanden sich außerhalb des Schaltschranks in der Maschine und sollten mit reduziertem Verdrahtungsaufwand an die SPS angeschlossen werden können.

Dazu entwickelten die Steuerungshersteller neue Kommunikationsbaugruppen, die eine serielle Buskopplung zwischen der Steuerung und den im Feld befindlichen Geräten ermöglichten. Der Vorteil dieser Feldbusse lag in der reduzierten Verkabelung zu den Aktoren und Sensoren.

Diese Reduzierung der Verkabelung hatte einen weiteren Nebeneffekt: Da nun die Aktoren und Sensoren über ein Bussystem an die SPS angekoppelt wurden, verringerte sich auch die notwendige Anzahl von Peripheriebaugruppen im Baugruppenträger der SPS. Die Klemmleistenumsetzer wanderten direkt in die Schaltkästen vor Ort und der Schaltschrank konnte wesentlich verkleinert werden.

Ziel dieser Entwicklung war, den Verdrahtungsaufwand erheblich zu reduzieren, als Konsequenz ergab sich jedoch zusätzlicher Aufwand in der Software:

- Der ordnungsgemäße Hochlauf beim Einschalten der externen Peripherie am Bus musste überwacht werden.
- Der Ausfall einer Komponente während des Betriebs war durch die Software zu detektieren und in eine geeignete Reaktion im Prozess zu modellieren.
- Für den möglichen Ausfall externer Peripherie musste sich der Software-Entwickler über eine Ersatzwert-Strategie für die nicht mehr vorhandenen Ein- bzw. Ausgänge Gedanken machen. Diese Ersatzwertstrategie musste in entsprechende Programme integriert werden.

Hemmnisse

Die Auslagerung der Peripherie in busgekoppelte Komponenten führte zu weiterer Komplexität der Software, die aber bei rein digitaler oder analoger Peripherie noch relativ einfach beherrschbar war. Da jedoch durch die fortschreitende Entwicklung auch kom-

plexere Peripherien über Busse angekoppelt wurden, musste die Software-Entwicklung darauf besondere Rücksicht nehmen. Die Interaktion mit zum Teil eigenständig agierenden aufwändigen Peripheriekomponenten ist nicht einfach. Vor allem dann, wenn die Komponenten einen eigenen, selbstständigen Ablauf haben und dieser mit dem Maschinenprozess synchronisiert werden muss. Fällt die Verbindung zu einer solchen Komponente aus, ist die eventuell notwendige Stoppreaktion noch leicht beherrschbar. Das erneute Anfahren danach kann aber erheblichen Aufwand in den Programmen bedeuten. Dazu sind weitergehende Informationen im Hauptprozess notwendig, die über einen zusätzlich zu programmierenden Informationsaustausch mit der Peripheriekomponente ermittelt werden müssen.

Lösungen

Nur durch ein durchdachtes Design der Software lassen sich diese Zusammenhänge beherrschbar halten. Ohne ein solches Design bekommt man über die Zeit durch die ausgelieferten Maschinen viele Software-Unikate, die eine Pflege wesentlich erschweren und eine Modularisierung und Standardisierung unmöglich machen.

Somit beeinflussten die Feldbussysteme mit ihren Komponenten die Programmierung sehr stark, wie wir auch später noch sehen werden (siehe Kapitel 1.6 „Antriebe werden zu vollwertigen Bus-Teilnehmern").

1.4 Integration von Anzeige-Systemen in die SPS

Die zunehmende Komplexität und Größe der Anlagen machte es nötig, dem Maschinenbediener das Prozessgeschehen in der Anlage transparent zu machen. Diese Anforderung ging über die bisherige Anzeige des Maschinenstatus per Lampen und Leuchttaster weit hinaus. Die Abläufe in der Maschine wurden komplexer und benötigten deshalb bessere Anzeigesysteme. Aus dieser Notwendigkeit heraus und mit der fortschreitenden Entwicklung von Computersystemen wurde der Einsatz von Visualisierungssystemen in Form steckbarer Baugruppen in der speicherprogrammierbaren Steuerung wirtschaftlich. Die Steuerungshersteller entwickelten in die SPS steckbare Einplatinencomputer, welche den Anschluss von Bildschirmen als Human-Machine-Interface (HMI, Mensch-Maschine-Schnittstelle) erlaubten. Für den Datenaustausch kommunizierten diese Minirechner mit der SPS über den Rückwandbus und entsprechende Treiberbausteine in der SPS-CPU. Zur Projektierung der Bilder stellten die Hersteller den Anwendern eigens entwickelte Projektierungswerkzeuge zur Verfügung. Eines der ersten Bildschirmsysteme dieser Art für SIMATIC-Steuerungen war das Visualisierungssystem WS400 (bestehend aus der Bildschirm-Anzeigebaugruppe WF470 und verschiedenen Bedienfeldern) von Siemens (Abbildung 4).

Mit diesem Visualisierungssystem konnte die Anlage über eine Blockgrafik übersichtlich dargestellt werden, in die sich der Status der einzelnen Maschinenmodule einblenden ließ. Weitere zu projektierende Detailbilder lieferten dem Maschinenbediener genauere Informationen zu den einzelnen Modulen. Die Schrittketten in der Maschine zur Abarbeitung der Prozesse konnten im Visualisierungssystem

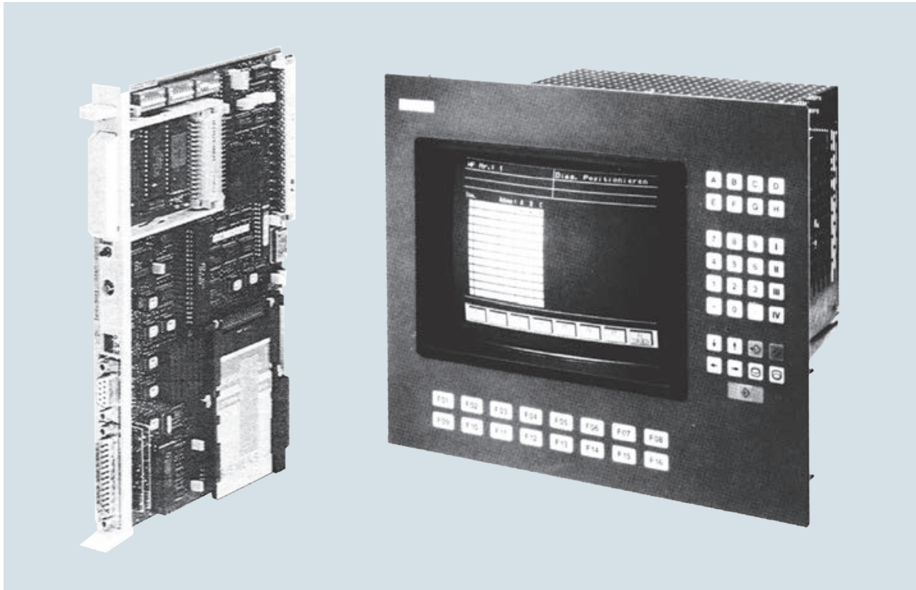

Abbildung 4 WF470 mit Kompaktbedienfeld

dargestellt werden und eine standardisierte Fehlerdiagnose zeigte aufgetretene Probleme an.

Somit mussten die Programmierer die Software- und Datenstrukturen so gestalten, dass eine Übergabe der Informationen an das Visualisierungssystem möglich war:

- Die Datenmodelle mussten zumindest für die Anzeige modularisiert und eine Skalierbarkeit bei unterschiedlichen Anlagengrößen berücksichtigt werden. Nur durch solche Gestaltungen in der Software war eine Wiederverwendbarkeit der Software über verschiedene Anlagen gegeben.
- Traten in der Anlage Fehler auf, musste es möglich sein, dem Bediener Detailinformationen zu liefern. Somit mussten die Programmierer vorsehen, dass die entsprechenden Anlagenfehler über Flags in der Steuerung dem Visualisierungssystem mitgeteilt und die entsprechenden Fehlermeldungen den Flags über Textlisten zugeordnet wurden.
- Statusmeldungen waren zu programmieren und ebenfalls dem HMI-System mitzuteilen.
- Da bei gravierenden Fehlern Anlagenteile oder gar die gesamte Anlage stillzusetzen war, führten die Meldeverfahren oft zur Anzeige vieler weiterer Folgefehler, zusätzlich zur eigentlichen Fehlerursache. Diese Fehlerschauer halfen dem Bediener oder Servicetechniker nicht bei der Problemlösung. Die Konsequenz für die Software-Entwicklung war, dass die Programmierer sich nun auch über die Auswertung von Erst- und Folgefehlern Gedanken machen mussten und dies in die Anlagensoftware entsprechend zu integrieren hatten.

Die Integration von Visualisierungssystemen in die SPS zwang die Programmierer zur Umsetzung von weiteren Modulen in der Anlagensoftware, die für die Bedienbarkeit der Maschine absolut notwendig waren, aber mit der eigentlichen Steuerungsaufgabe weniger zu tun hatten. Damit wuchsen die Programme weiter an. Modularisierung und klare Strukturen wurden verstärkt notwendig, um die Software wartbar und pflegbar zu halten. Programmiert wurden die Systeme aber nach wie vor mit KOP/FUP oder AWL. Für die Schrittkettenprogrammierung diente das eigens dafür entwickelte GRAPH-5.

Heute werden die Visualisierungssysteme über Industrial Ethernet an die Steuerung angebunden. Die Projektierung der HMI-Systeme ist Bestandteil der Engineering-Software, die integriert in entsprechenden Softwaresuiten arbeitet. Die notwendige Modularisierung in der Software und die Entwicklung passender Datenmodelle in der Steuerung ist aber nach wie vor eine wichtige Aufgabe der Software-Entwicklung und des Software-Designs.

Hemmnisse

Moderne Visualisierungssysteme erlauben die direkte Verwendung von Variablen der speicherprogrammierbaren Steuerung in den projektierten Anlagenbildern. Diese Möglichkeiten werden bei der Präsentation von HMI-Systemen häufig in den Vordergrund gestellt. Das sieht einfach aus und impliziert eine leichte Erstellung der gewünschten Anlagenbilder. Allerdings führt diese Vorgehensweise zu nicht entkoppelten Softwareanteilen. Änderungen in der Maschinenprogrammierung führen dann auch zu Änderungen im Variablenhaushalt und damit zu geänderten Adressen der Variablen. Das hat zur Folge, dass die HMI-Bilder zumindest neu übersetzt und erneut geladen werden müssen. Eine Entkopplung und damit eine unabhängige Entwicklung sind nicht mehr gegeben.

Lösungen

Deshalb ist es vorteilhafter, eine durchdachte Schnittstelle zum HMI zu definieren. Das Maschinenprogramm überträgt die notwendigen Daten in die Schnittstelle und holt aus dieser die benötigten Daten für den Ablauf. In den Bildern werden nur Variablen der Schnittstelle verwendet. Bei dieser Vorgehensweise sind die Daten entkoppelt. HMI und Steuerung können unabhängig voneinander geladen werden und eine unabhängige Entwicklung von Steuerungsprogramm und Bildprojektierung ist möglich. Darüber hinaus gewinnt man den weiteren Vorteil, dass die Datenübertragung an das HMI-System in einem Block wesentlich schneller abläuft als beim Sammeln der Daten über das gesamte Programm.

1.5 Integration von Motion Control in die SPS

Die Anforderung an den Maschinenbau, die Maschinen in kürzester Zeit für die Fertigung neuer Produkte umzurüsten, führte zu der Notwendigkeit, die Bewegungen in der Maschine flexibler zu gestalten. Dies bedeutete, dass die Bewegungsachsen zunehmend mit elektrischen Antrieben ausgestattet wurden statt mit mechanischen oder hydraulischen Lösungen. Der Einsatz von elektrischen Antrieben erforderte wiederum die Nutzung von Systemen, die eine flexible Positionierung

des Antriebs erlaubten. Wie bei den HMI-Systemen konnten mit speziellen Mikrocomputersystemen in die SPS steckbare Positionierbaugruppen entwickelt werden, die eine flexible Positionierung der Antriebe in der Maschine ermöglichten. Solche Positionierbaugruppen mussten zum Anschluss an die damalige Antriebstechnik über analoge Sollwertausgaben verfügen, sowie die Position der Achsen über anschließbare Gebersysteme erfassen können.

Eines der ersten Positioniersysteme für die SIMATIC-SPS war das System WS600 (Abbildung 5). Mit diesem System stand dem Maschinenbau die Positionierbaugruppe WF625 mit dem Anzeigesystem WS600G zur Verfügung.

Die Kommunikation zwischen Anwenderprogramm und den Positionierbaugruppen wurde durch ein Standard-Softwarepaket durchgeführt. Als Schnittstelle zum Anwenderprogramm dienten Datenbausteine, die durch das Anwenderprogramm versorgt bzw. entsorgt werden mussten.

Die Programmierung der Verfahr-Programme erfolgte durch die Bedientafeln WS600G und orientierte sich an der von den CNC-Steuerungen bekannten Semantik der DIN66025/ISO6983. Damit konnten die Verfahr-Bewegungen flexibel an die Erfordernisse des Produktionsprozesses angepasst werden.

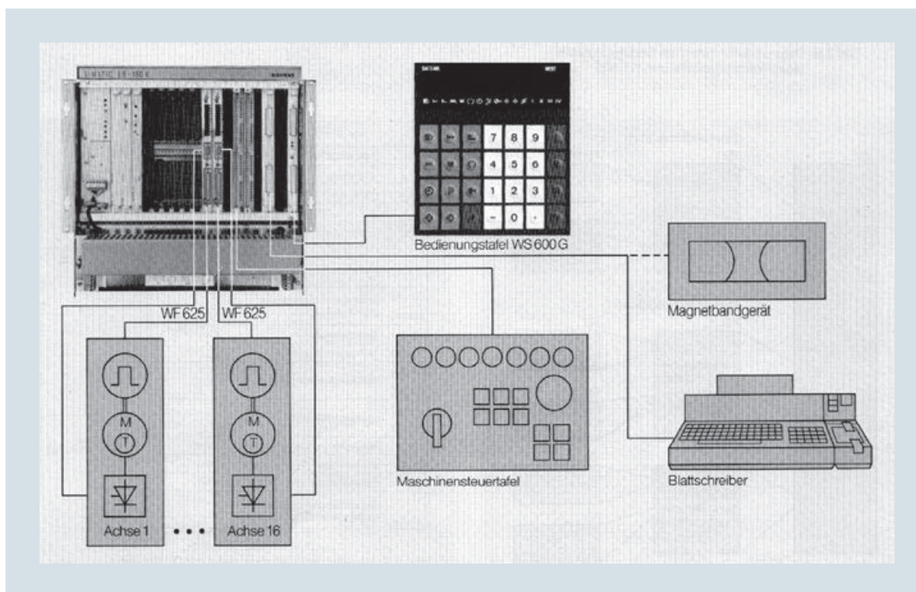

Abbildung 5 S5-150K mit WF625 und WS600G

Diese Integration von Motion Control in die SPS hatte gravierende Einflüsse auf die Steuerungsprogramme in der SPS. Die Verfahr-Bewegungen in Form von CNC-Programmen und die Koordination der verschiedenen Baugruppen wurden Aufgabe des Anwenderprogramms. Da die Positionierbaugruppen einen eigenen Takt besaßen, der in der Regel wesentlich kürzer war als der SPS-Zyklus, musste das SPS-Programm diesem Umstand durch entsprechende Programmierung Rech-

nung tragen.Durch diese unterschiedlichen Taktzeiten musste das SPS-Programm um Synchronisationsroutinen angereichert werden.

War zum Beispiel die Verfahr-Bewegung schneller abgearbeitet als die Umlaufdauer eines SPS-Zyklus, konnten die Signalwechsel der Positionierbaugruppe im SPS-Programm nicht mehr richtig registriert werden. Somit bestand in diesem Fall auf Seite der SPS-Programmierung die Unsicherheit, ob der Positioniervorgang auch wirklich durchgeführt wurde. Diese Unsicherheit konnte man sich bei einem automatisch ablaufenden Prozess nicht leisten. Also musste der SPS-Programmierer im Ablaufprogramm sicherstellen, dass die beauftragten Positioniervorgänge eindeutig beendet wurden. Dies konnte beispielsweise durch Vergleichsfunktionen von der Istposition gegen die Zielposition ermittelt werden. Da jedoch die Positionswerte über Programmierung an der Bedientafel WS600G änderbar waren, musste der Programmierer die für den Vergleich notwendigen Zielpositionen dynamisch ermitteln, um die Flexibilität der Anlage nicht zu beeinträchtigen. Dies war ein nicht zu unterschätzender zusätzlicher Programmieraufwand.

Weiterentwicklungen im Bereich der Positionierbaugruppen lösten diese Problematik durch die Implementierung von geeigneten Handshake-Mechanismen an der Schnittstelle. Damit wurden die notwendigen Programme in der SPS wieder einfacher.

In heutigen Steuerungen ist Motion Control in der SPS komplett integriert. Die Motion-Control-Funktionalität ist Bestandteil des Betriebssystems der speicherprogrammierbaren Steuerung. Aber trotz der Motion-Control-Integration sind die Aufwände in der Programmierung für die Organisation der Motion-Control-Bewegungen nach wie vor notwendig:

- Um die im Motion Control ausgeführten Prozesse zu verstehen, waren die Programmierer gezwungen, ihr Domänenwissen zu erweitern. Die gewonnenen Erkenntnisse führten zwangsläufig zu Anpassungen in der Programmierung des Maschinenablaufs.
- Regelvorgänge der lagegeregelten Positionierbaugruppen und der unterlagerten Antriebsregelung mussten verstanden werden. In diesem Bereich auftretende Fehler waren zu detektieren und über geeignete Aktionen im Maschinenablauf zu berücksichtigen.
- Jedes Positioniermodul arbeitete autark. Dadurch wurde die Synchronisation von Positionierbewegungen über verschiedene Baugruppen hinweg in der SPS notwendig. Dabei kostete es den meisten Aufwand, die jeweiligen Unterbrechungen des Prozesses und das nachfolgende störungsfreie Wiederaufsetzen zu beherrschen.
- Neben der Beherrschung von Motion Control selbst wurden in bestimmten Fällen Routinen zur Verwaltung der CNC-Programme in der SPS notwendig, damit die verschiedenen Umrüstvorgänge automatisch und schnell bewerkstelligt werden konnten.

Die Integration von Motion Control in die speicherprogrammierbare Steuerung war und ist ein großer Schritt zur Anlagenflexibilisierung. Ohne Motion Control sind heutige Anlagen nicht mehr vorstellbar, da der Weg zu mechatronischen Systemen voll im Gange ist.