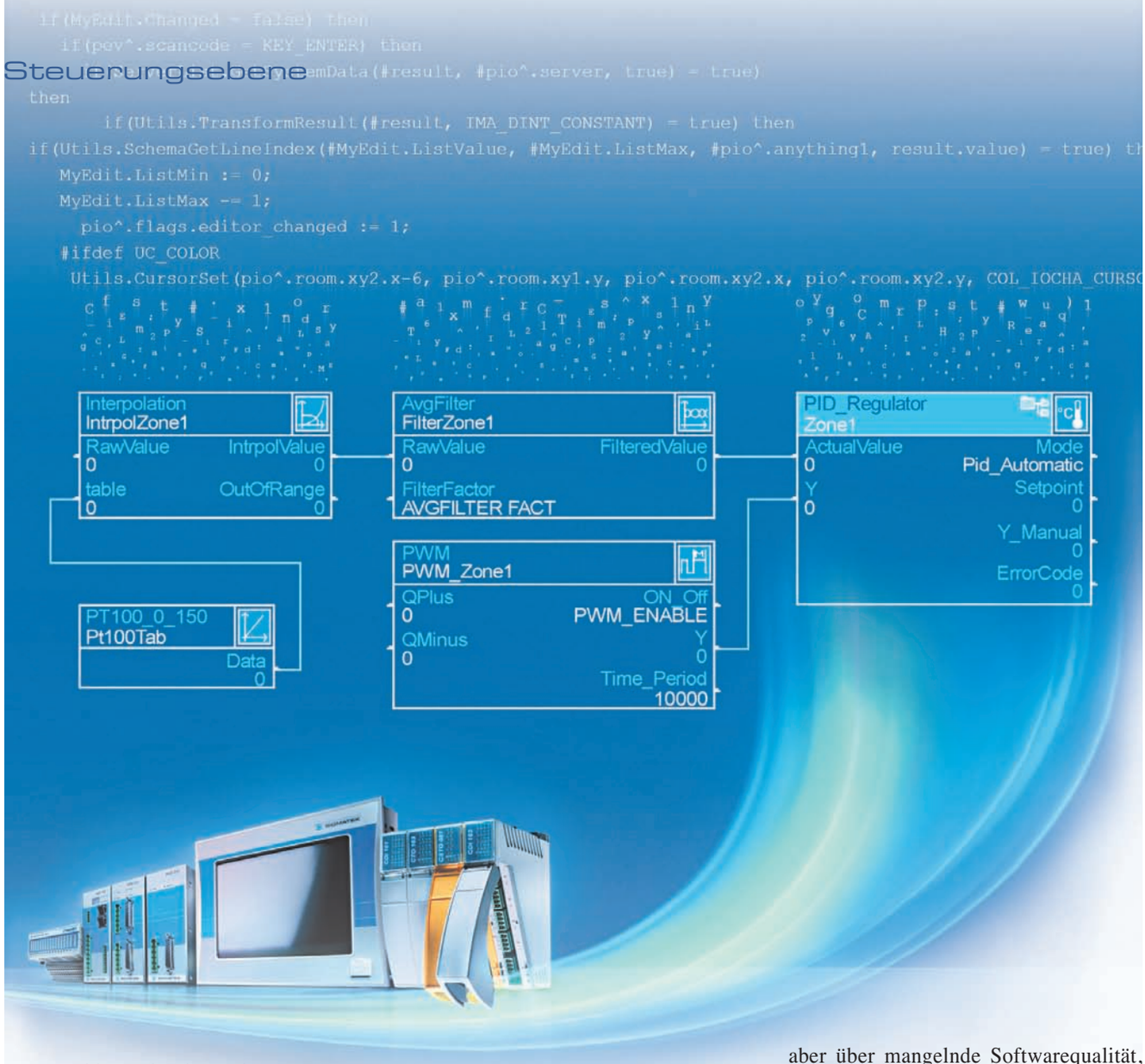

Franz Aschl, Bernhard Gangl

Spaghetticode versus Objektorientierung

Halle A2 Stand 510

Trotz ihrer offensichtlichen Vorteile herrscht im Maschinen- und Anlagenbau noch immer eine gewisse Scheu vor der objektorientierten Programmierung. Zu Unrecht, denn: Mit den passenden Software-Tools bleibt dem Anwender ein Großteil der damit einhergehenden Komplexität erspart.

Objektorientierte Programmierung – kurz OOP – gibt es nunmehr seit mehr als 30 Jahren. In den Schulen und Universitäten wird kaum mehr etwas anderes unterrichtet. Nur bei den SPS-Programmierern setzt sich der objektorientierte Ansatz nur zögerlich durch. Eigentlich unverständlich – gibt es doch kaum ein passenderes Anwendungsfeld dafür als den Maschinenbau. Denn hier wird genau wie in der OOP in Komponenten gedacht – beispielsweise in Motoren, Getrieben und daraus folgenden Antriebssträngen. Ein Grund für die zögerliche Akzeptanz ist sicherlich die uns Menschen eigene Mentalität „das haben wir immer schon so gemacht, warum jetzt alles anders machen?“. Im gleichen Zuge jammern wir aber über mangelnde Softwarequalität, aufwendige Pflege und Wartbarkeit sowie langwierige Entwicklungszeiten.

Zugegeben: Die OOP stellt einen neuen Ansatz in der Automatisierung dar und enthält viele neue Begriffe wie Klassen, Objekte, Instanzen, Enheritance oder Vererbung. Für dieselben Dinge werden dabei oft unterschiedliche Begriffe eingesetzt, was auf den ersten Blick hohe Komplexität impliziert. Bei näherer Betrachtung stellen sich die Dinge aber wesentlich einfacher dar, als sie scheinen. Darüber hinaus gibt es heute Software-Werkzeuge, die komplexe und immer wiederkehrende Funktionen für den Anwender übernehmen. So sind beispielsweise im Engineering-Tool „Lasal“ von Sigmatek die objektorientierten Merkmale wie Konstruktoren oder This-Pointer bereits im System „versteckt“ und werden vollautomatisch erzeugt. Der Anwender wird mit diesen objektorientierten Elementen nicht direkt konfrontiert. Der Code liest sich für ihn wie gewöhnlicher Structured Text und der Entwickler kann sich somit auf seine ei-

(Bilder: Sigmatek)

gentliche Problemstellung konzentrieren. Fakt ist: In Zukunft wird auch in der Steuerungstechnik kein Weg an der OOP vorbeiführen. Wichtig dabei ist nur, dass sich der Anwender für eine Umgebung entscheidet, die von Grund auf den objektorientierten Ansatz verfolgt. Aktuell gibt es zahlreiche Hersteller, die entsprechende Methoden im Nachhinein in ihre Entwicklungsumgebung integrieren. Dabei besteht die Gefahr, dass die Unterstützung der Objektorientierung aufgesetzt wirkt, was letztlich wieder deren Akzeptanz und auch die dahinterstehenden Möglichkeiten der Entwicklungsumgebung schmälert.

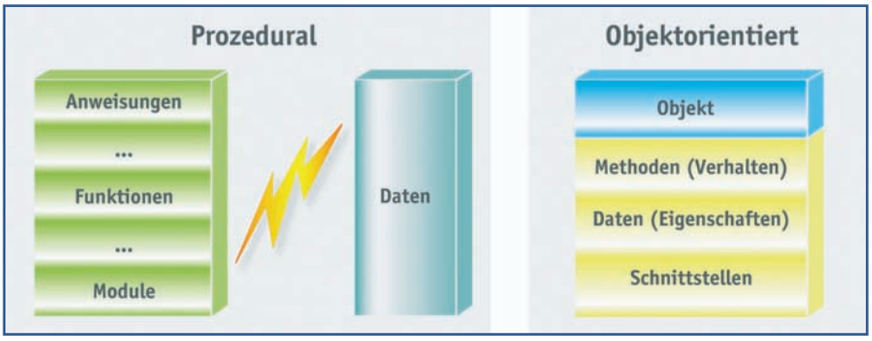

Bild 1. Bei der rein prozeduralen Programmierung werden die Daten beziehungsweise Variablen getrennt vom Code verwaltet. Die fehlende Festlegung, wie die Interaktion zwischen Code und Daten abläuft, kann dabei zu fehlerträchtigen Programmen führen.

Die Maschine – das Paradebeispiel für OOP

Was macht nun die Objektorientierung im Detail aus? Der objektorientierte Ansatz erweitert zunächst die prozedurale Programmierung. Die Grundidee dabei ist, Code und Daten in logische Einheiten zusammenzufassen. Dies macht schon aus dem Grund Sinn, weil zumeist ein bestimmter Code-Teil für bestimmte Daten verantwortlich ist. Code und Daten werden also in einem Objekt zusammengefasst und nach außen hin abgeschottet, sodass fremde Code-Teile die Daten eines Objekts nicht ohne weiteres verändern können.

Hinter einem Objekt steht jeweils eine Klasse, die den Programmcode und die dazugehörenden Datenelemente kapselt.

Warum ist das so wichtig? Jeder Programmierer kennt das Problem von unsauber implementierten Programmen, wo Variablen kreuz und quer über das Projekt verteilt beschrieben werden und die Auswirkungen einer Programm-Änderung praktisch nicht vorhersehbar sind. Anders bei der OOP: Hier ist diese Variable nur über zugehörige Methoden manipulierbar. Somit sind klare Schnittstellen vorgegeben, die dann auch verwendet werden müssen.

Der eigentliche Programmcode einer Klasse wird – um beim Beispiel Lasal zu bleiben – in den gebräuchlichen Sprachen der IEC 61131-3, wie Strukturierter Text, AWL und Kontaktplan, implementiert. Dies ist ein wesentlicher Akzeptanzfaktor, da so die Methoden der objektorientierten Programmierung als durchgängige Erweiterung der vertrauten und bewährten Sprachen zur Verfügung stehen. Die einzelnen Software-Module (Objekte) lassen sich dabei wie in einem Baukastensystem zusammensetzen, und über die modulare Struktur sind bereits erstellte Applikationsteile einfach wieder verwendbar.

Durch Vererbung lässt sich eine Klasse verfeinern beziehungsweise spezialisieren, indem der Basisklasse zusätzliche Informationen und Programmcode hinzugefügt werden. Durch das Aggregieren werden einzelne Klassen zu einer komplexen Klasse zusammengefasst. Mit diesen Techniken ist es möglich, neue Ausprägungen von Maschinenteilen mit minimalem Programmieraufwand umzusetzen. Dies reduziert die Engineering-Zeiten und -Kosten erheblich.

Letztlich kann in der OOP genauso programmiert werden, wie im Maschinenbau gedacht wird – und zwar in Komponenten. Die Schnittstelle zwischen Maschinenbauer und Software-Entwickler wird demzufolge viel einfacher, da sich die verschiedenen Maschinenteile und deren Interaktion miteinander in der Software widerspiegeln.

Zusammenhänge erkennen

Jeder Applikateur, der eine Maschinensoftware entwickelt, muss sich im Vorfeld detailliert mit der Maschine auseinandersetzen. Welche Teile gibt es auf der Maschine, wie hängen die verschiedenen Maschinenteile zusammen, wie beeinflussen sie sich gegenseitig und wie interagieren sie miteinander? Durch Analyse der Maschine und Definition der einzelnen Maschinenteile ergeben sich in der OOP die Klassen mit ihren Eigenschaften und Schnittstellen zu den anderen Maschinenteilen von selbst. Auch Maschinenteile, die sich gleich oder ähnlich verhalten, werden so analytisch extrahiert und sind in allgemeinen Klassen beschreibbar.

Nehmen wir als Beispiel eine einfache Transportstrecke für Stückgut. Die Maschine besteht aus drei aneinander gereihten Förderbändern; jedes einzelne angetrieben von einem Motor mit einer Start- und Stoppbedingung und einem Zylinder am Ende, der das Stückgut auf das nächste Förderband beziehungsweise am Ende auf die Palette schiebt.

Versucht man hier nun entsprechende Software-Klassen zu finden, fällt auf den ersten Blick auf, dass die drei einzelnen Förderbänder ähnliches Verhalten aufweisen. Der objektorientierte Entwurf könnte also folgendermaßen aussehen: Es werden drei Klassen modelliert – eine Klasse kümmert sich um die Motoransteuerung, eine zweite steuert den Zylinder und die dritte ist die Hauptklasse für das Förderband selbst, die die Start- und Stoppbedingungen erkennt und dementsprechend Motor und Zylinder startet und stoppt. Per Aggregation lässt sich aus diesem Klassenverbund eine komplexe Klasse „Förderband-Ein-

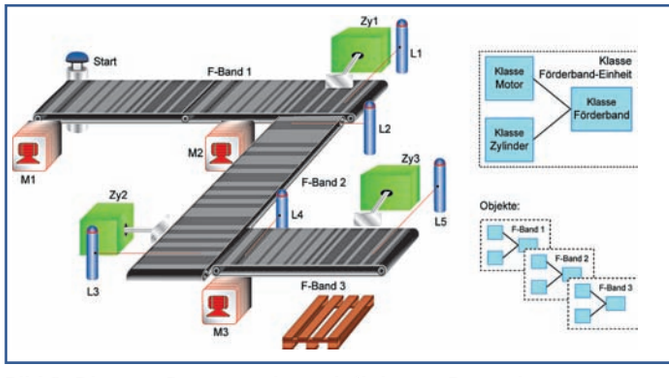

Bild 2. Die vom Programmierer definierten Bausteine werden in sogenannten Klassenbibliotheken abgelegt und sind dann in unterschiedlichen Projekten oder Systemteilen einfach einsetzbar. Standardbibliotheken, wie sie etwa in Lasal vorhanden sind, stellen bereits eine Vielzahl von Funktionsklassen für verschiedene Anwendungskategorien zur Verfügung.

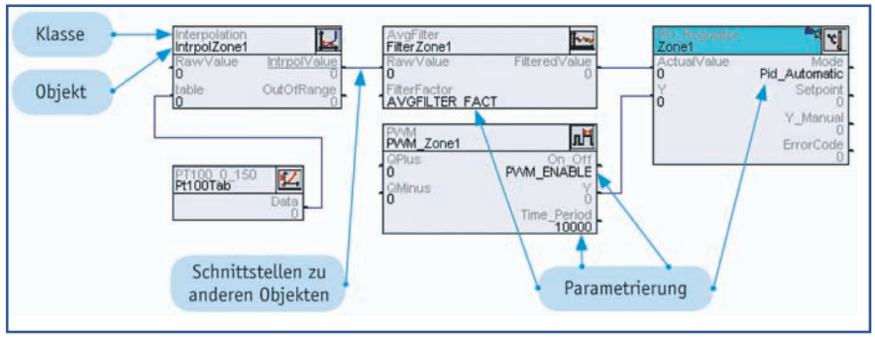

Bild 3. Ein Rechteck auf diesem Netzwerk stellt ein Objekt dar, oben der Klassenname und darunter der Name des daraus erzeugten Objekts. Die Objekte haben Schnittstellen zur Außenwelt, um mit anderen Objekten zu kommunizieren. Parametrierung: Diese Schnittstellen sind zugleich Datenelemente, die für jedes Objekt unterschiedlich initialisierbar sind.

heit“ kreieren. Die fertige Steuerung der Maschine würde dann aus drei Objekten für jeweils ein Förderband bestehen. Wenn es sich hier auch um ein sehr einfaches Beispiel handelt, so ist das Vorgehen auch bei komplexen Maschinen immer dasselbe.

Grafische Darstellung sorgt für Übersicht

Bei der objektorientierten Programmierung hat also das Design von Software mindestens den gleichen Stellenwert wie die Implementierung. Der Fokus liegt dabei auf den abgeschotteten Objekten, die über Schnittstellen mit der Außenwelt kommunizieren. Wenn man sich diese Schnittstellen vorab genau überlegt, sind Objekte später sehr einfach gegen andere austauschbar und die Software lässt sich flexibel, wartungsfreundlich und leicht testbar gestalten – auch noch nach Jahren.

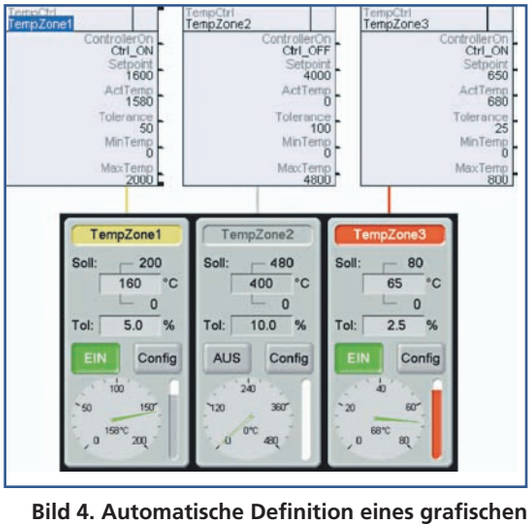

Bild 4. Automatische Definition eines grafischen Objektes zur Visualisierung einer Temperaturregelzone: Zuerst wird eine Klasse definiert, dann das dazugehörige grafische Objekt – jetzt lassen sich beliebig viele Instanzen (Varianten) erzeugen.

Diese Strukturiertheit der OOP kommt dem Maschinenbauer vor allem bei komplexen Projekten zugute. Je umfangreicher das Projekt ist und je mehr Personen über den Produktlebenszyklus daran beteiligt sind – sprich umso mehr das Maschinenprogramm im Laufe der Zeit wächst –, desto gewichtiger ist der Vorteil, der sich durch die Übersichtlichkeit dank klarer Kapselung ergibt.

Noch übersichtlicher wird die Sache, wenn das verwendete Engineeringtool die Möglichkeiten der grafischen Darstellung bietet. Beim grafischen Ansatz, wie er in Lasal von jeher verfolgt wird, werden die von Klassen erzeugten Objekte in so genannten Netzwerken dargestellt. Der große Vorteil dabei ist, dass die Maschine in der Software grafisch nachgebildet wird. Mit anderen Worten: Der Entwickler sieht auf den ersten Blick die Eigenschaften eines Maschinenteils sowie die Kommunikation mit anderen Objekten, sprich Maschinenteilen.

Eine echte Stärke der grafischen Darstellung zeigt sich bei der Online-Diagnose. Sobald eine Online-Verbindung zur SPS hergestellt ist, werden im Netzwerk die „lebenden Werte“ der Schnittstellenvariablen angezeigt. Das heißt: Es lässt sich von sämtlichen Objekten der aktuelle Status bestimmen, ohne mit dem Sourcecode des Objekts konfrontiert zu sein. Service- und Inbetriebnahme-Techniker haben durch die grafische Ansicht einerseits die Möglichkeit, die Softwarestruktur leichter zu verstehen. Andererseits vereinfacht sich die Diagnose extrem, weil der Sourcecode grafisch gekapselt ist.

Apropos Programmiersprachen: Strukturierter Code ist in einer OOP-Sprache einfacher lesbar als Codes für nicht objektorientierte Sprachen, da die mehrfache Anzahl an Aufrufen eines Code-Teils mit unterschiedlichen Datenbereichen nicht im Code berücksichtigt werden muss. Dies macht sich insbesondere bei großen Projekten bemerkbar: Lesbarkeit und Stabilität nehmen enorm zu. Damit nicht genug: Nach Änderung eines objektorientierten Codes reicht es in den meisten Fällen aufgrund der definierten Schnittstellen aus, lediglich die geänderten Klassen zu testen anstatt große Teile des Projekts.

Objektorientierung auch bei der Visualisierung

Die Objektorientierung eröffnet nicht nur im Bereich der Ablaufprogrammierung von Maschinen neue Möglichkeiten, sondern auch hinsichtlich der Visualisierung. Mit „Lasal Screen“ zum Beispiel lassen sich grafische Objekte definieren. Dabei verwendet der Anwender Datenpunkte einer Klasse aus „Lasal Class“, um das visuelle Erscheinungsbild dieser Klasse zu definieren.

Gibt es beispielsweise im Steuerungsprogramm drei Objekte einer Temperaturregelzone (siehe *Bild 4*), so sind diese auf einer entsprechenden Bildschirmseite als grafische Objekte platzierbar. Anschließend genügt das Einstellen des Objektnamens, um in diesem grafischen Objekt die komplette Zone 2 zu visualisieren. Die einzelnen Datenelemente einer Zone bleiben unberührt. Dieser Mechanismus steht programmtechnisch auch zur Laufzeit zur Verfügung. Auf diese Weise kann der Bediener per Tastendruck bei einem platzierten Objekt zwischen den Werten von Zone 1 und Zone 2 umschalten. *gh*

Franz Aschl

ist im Bereich Sales Management bei Sigmatek tätig.

Bernhard Gangl

ist Abteilungsleiter Software bei Sigmatek.