Gekapselter Code erhöht Softwarequalität und Flexibilität

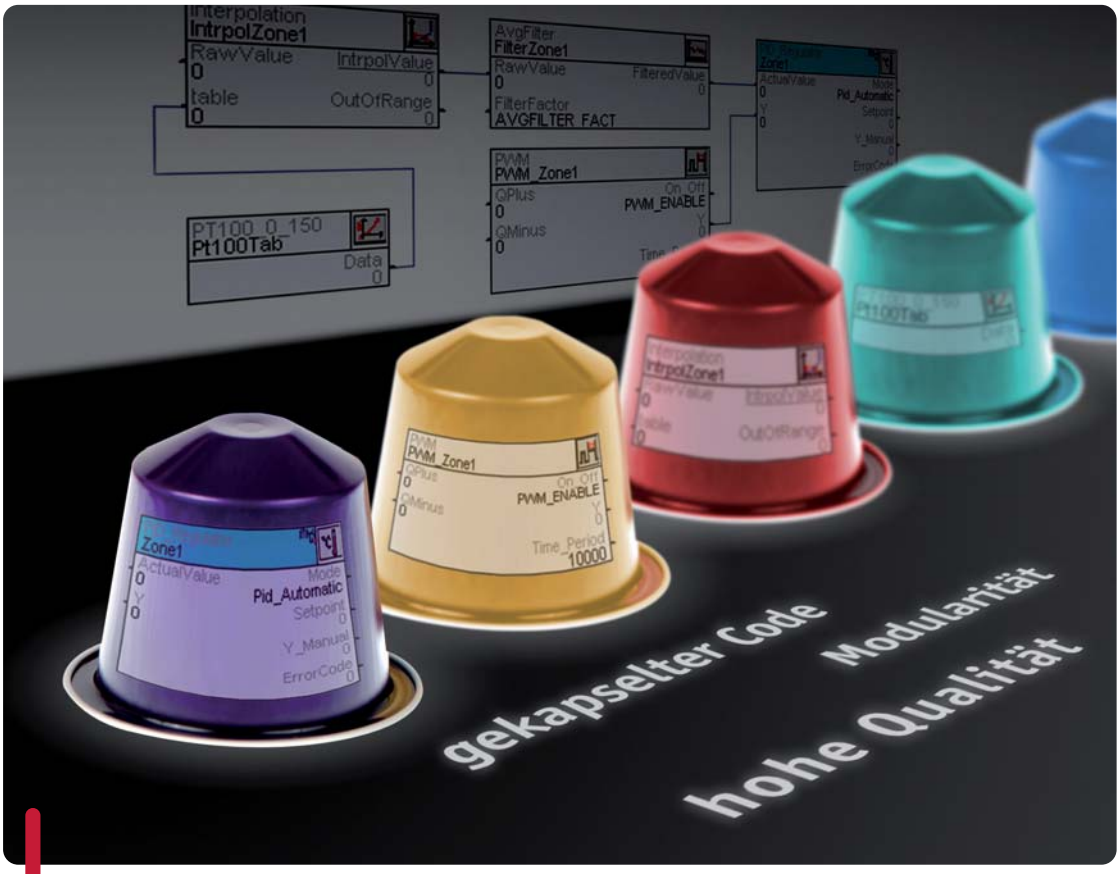

Bild 1: Das objektorientierte Programmieren setzt sich mehr und mehr in der gesamten Steuerungstechnik durch.

Die Applikationssoftware wird immer mehr zu einem entscheidenden Kostenfaktor im Maschinenbau. Die Anwender benötigen ein effizientes Engineering Tool mit komfortablen Möglichkeiten zur Simulation und Visualisierung aller Prozesse, mit einer vollständigen Testumgebung bis hin zum Debugger samt zentraler Verwaltung von Projekten und Versionen. So lässt sich die Transparenz und die Qualität der Software steigern.

Qualität gibt an, in welchem Maß ein Produkt den allgemein bestehenden Anforderungen entspricht und ist gleichzeitig ein Synonym für Güte. (siehe Wikipedia, 'Qualität', 10.3.2012) Im Bereich Software-Engineering steht 'Qualität' für: Komplexes einfach und strukturiert umetzen. Die Minimierung der Entwicklungszeiten und somit der Kosten ist dabei nur ein wichtiger Aspekt. Ebenso wichtig sind Faktoren wie Modularität, Wiederverwendbarkeit, einfache Wartbarkeit und somit eine 'höhere Lebenserwartung' der Software. Diesen Qualitätsmerkmalen kann mit objektorientierten Konzepten entsprochen werden. Mit objektorientierten Engineering Tools wie Lasal lassen sich die Engineeringzeiten erheblich reduzieren und neue Ausprägungen von Maschinenteilen können mit minimalem Programmier- und Testaufwand umgesetzt werden. Bei der objektorientierten Programmierung (=OOP) werden Code und Daten in logische Einheiten, in 'Objekte' zusammengefasst und in einem Block gekapselt. Dies macht schon aus dem Grund Sinn, weil zumeist ein bestimmter Codeteil für bestimmte Daten verantwortlich ist. Die Objekte werden nach außen hin abgeschottet, sodass eine ungewollte Beeinflussung ausgeschlossen werden kann. Hinter einem Objekt steht jeweils eine Klasse. Sie ist der 'Bauplan' für ein Objekt und definiert den Programmcode und die Datenelemente. Sobald diese Funktionsblöcke mit Parametern und Schnittstellen versehen sind, bezeichnet man sie als Objekte.

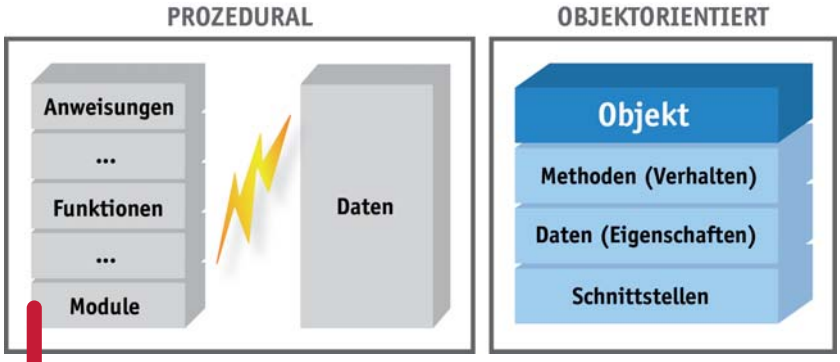

Bild 2: Bei der rein prozeduralen Programmierung werden die Daten beziehungsweise Variablen getrennt vom Code verwaltet. Die fehlende Festlegung, wie die Interaktion zwischen Code und Daten abläuft, kann dabei zu fehlerträchtigen Programmen führen. Im Gegensatz dazu setzen objektorientierte Konzepte darauf, Daten und zugehörige Programmteile zu einer Einheit zusammenzufassen.

Welche Vorteile bietet die Kapselung? Jeder Programmierer kennt das Problem von unstrukturiertem Code, wo Variablen kreuz und quer im Projekt verteilt beschrieben werden, sodass die Auswirkungen einer Programmänderung praktisch nicht vorhersehbar sind. Anders bei der OOP: Hier ist diese Variable nur über zugehörige Methoden manipulierbar. Somit sind klare Schnittstellen vorgegeben, die es dann auch zu verwenden gilt. Durch Client/Server-Technik müssen nicht für jeden Zugriff auf Variablen eigene Methoden erzeugt werden. Ein Server bringt per Definition bereits Write und Read-Methoden mit, der Programmieraufwand verringert sich. Moderne Engineering Tools wie Lasal von Sigmatek übernehmen immer wiederkehrende Funktionen für den Anwender. So wird beispielsweise der Code für die Deklarationen von Klassen, Variablen, Schnittstellen etc. automatisch im Hintergrund erstellt. Dem Anwender bleibt nur mehr die Implementierung der Methoden in der Klasse, fehlerhafte Deklarationen werden minimiert. Dazu kann er auf bewährte Programmiersprachen wie Structured Text (ST), Anweisungsliste (AWL), Kontaktplan (KOP) nach IEC 61131-3 oder ANSI-C. zurückgreifen.

Baukastensystem

Die einzelnen Software-Module (Objekte) lassen sich wie in einem Baukastensystem zusammensetzen. Der Vorteil dieser Entwicklungsmethode ist ihre durchgängige Modularität von der untersten Ebene der einzelnen Funktion bis hinauf zum Gesamtprojekt. Funktionen können einzeln oder in Gruppen entwickelt, getestet, hinzugefügt, ausgetauscht oder bei Nicht-Verwendung einer Option ausgeblendet werden. Wie in der Mechanik, wo eine erprobte Konstruktion wiederverwendet wird, können auch bei der OOP dank der modularen Struktur einmal erstellte und getestete Applikationsteile einfach wieder verwendet werden, ohne diese noch einmal überprüfen zu müssen. Software wird somit 'nachhaltig', das ist gerade bei komplexen Applikationen ein wichtiger Faktor. Denn wenn Code über längere Zeit verwendet wird – was ja prinzipiell erstrebenswert ist – entsteht bei bisherigen Programmiermethoden oft ein undurchschaubares Softwaregebilde. Durch die Strukturiertheit der OOP ist es hingegen möglich, bestehende Klassen zu erweitern beziehungsweise zu verändern (Ableitung) – und zwar ohne dazu in die ursprüngliche Klasse eingreifen zu müssen. Die Änderungen sind somit immer nachvollziehbar und einfach zu erkennen. Mit der OOP wird Code also einfach lesbar, testbar und wiederverwendbar. So lässt sich die Qualität der Software erheblich steigern und der Programmieraufwand minimieren. Diese Strukturiertheit der OOP kommt dem Maschinenbauer vor allem bei komplexen Projekten zu Gute. 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.

Mehr Transparenz durch grafische Darstellung

Die grafische Darstellung trägt ebenfalls zur Vereinfachung bei. Beim grafischen Ansatz werden die von Klassen erzeugten Objekte (Maschinenmodule) in sogenannten Netzwerken dargestellt. Mit anderen Worten: Der Entwickler sieht auf den ersten Blick die Eigenschaften eines Maschinenteils sowie die Kommunikation mit anderen Objekten also anderen Maschinenteilen. Das Ändern einer Klasse über die Bedienoberfläche wirkt sich sofort auf die grafische Darstellung aus. Die Maschine wird somit in der Software grafisch nachgebildet.

Vererbung erhöht Modularität

Stichwort Vererbung: Bei vielen Projekten existieren ähnliche Funktionalitäten in unterschiedlichen Ausprägungen. Eine Klasse lässt sich durch Vererbung verfeinern, indem die Basisklasse um zusätzliche Informationen und Programmcode erweitert und angepasst wird. Nehmen wir als Beispiel die Ansteuerung von Antrieben. Zum einen gibt es hier eine Klasse, die die Bewegungsabläufe des Antriebs je nach Maschinenzustand implementiert. Diese Klasse setzt Kommandos wie `Enable`, `FahreReferenz` oder `MoveToPosition` an den Antrieb ab. Je nach Maschinenvariante ist es oft nötig, unterschiedliche Antriebsarten anzusteuern, wie etwa einen Servoantrieb, einen Schrittmotor oder eine reine Simulation der Verfahrbewegung für Präsentations- und Testzwecke. Softwaretechnisch am unsinnigsten wäre es dabei, die verschiedenen Antriebsarten schon in der Klasse für den Bewegungsablauf zu berücksichtigen, da die Software dadurch komplex und schlechter test- beziehungsweise wartbar wird. Die Objektorientierung ermöglicht hier einfache Abhilfe: Es wird eine Schnittstellenklasse definiert, in der alle Kommandos eingefügt werden, die die Handling-Klasse benötigt. Diese Schnittstellenklasse kann als Basisklasse für die unterschiedlichen Antriebsvarianten dienen. Die Implementierung der einzelnen Methoden wird dann in der Ableitung ausprogrammiert, da die `Enable`-Funktion für einen Servoantrieb sicher anders aussieht, als für die Simulation. Das Wesentliche an diesem Beispiel jedoch ist, dass die Handling-Klasse immer nur auf die Basisklasse referenziert. Somit wird der

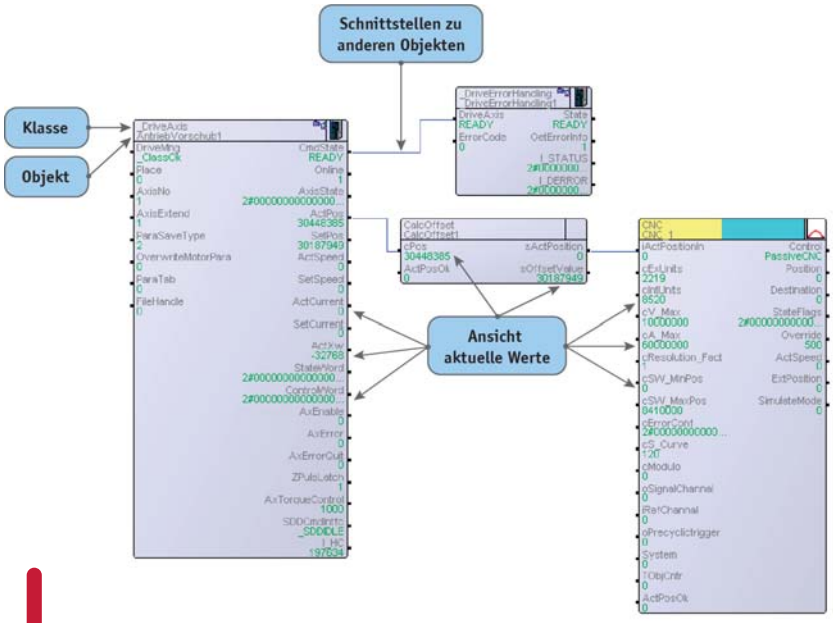

Bild 3: Ein Rechteck auf diesem Netzwerk stellt ein Objekt dar, oben der Klassenname und darunter der Name des daraus erzeugten Objekts. Die Objekte haben Schnittstellen zur Außenwelt, um mit anderen Objekten zu kommunizieren. Eine echte Stärke der grafischen Darstellung zeigt sich bei der Online-Diagnose. Sobald eine Onlineverbindung zur SPS hergestellt ist, werden im Netzwerk die 'lebenden Werte' der Schnittstellenvariablen angezeigt.

tatsächlich angeschlossene Antrieb softwaretechnisch einfach austauschbar. Die einfache Wiederverwendbarkeit, die Eigenschaftsvererbung und die umfangreichen Bibliotheken mit vorgefertigten Softwarebausteinen vereinfachen die Entwicklung, das größte Potential zur Einsparung von Programmierzeiten liegt aber darin, dass durch die frühzeitige Überprüfbarkeit nur einmal programmiert werden muss.

Simulation spart Zeit und Kosten

Mit dem Simulationstool Lars kann der Code ohne angeschlossene Steuerung auf einem Windows-Rechner simuliert und getestet werden. Das gilt für komplette Applikationsprogramme, aber auch für noch nicht fertiggestellte Systemteile. So ist es möglich, die Funktionalität der Maschine zu überprüfen und den Quellcode zu debuggen, noch bevor die mechanischen Komponenten fertig sind. Eventuelle Designfehler lassen sich frühzeitig erkennen. Lars kann vielfältig genutzt werden: zum Entwickeln und Testen von Programmen auf dem PC, wenn die eigentliche Zielhardware nicht zur Verfügung steht, für Visualisierungen auf Windows-Rechnern oder für Demo-Applikationen zu Präsentationszwecken. Im Projekt können sogar die Verknüpfungen zu den I/Os bereits bestehen. Diese lassen sich einfach simulieren, auch wenn Hardware physikalisch nicht vorhanden ist. All dies beschleunigt die Entwicklung.

Flexible Updates per Kapselung

Die gekapselten Objekte kommunizieren über Schnittstellen mit der 'Außenwelt'. 'Unsauberkeiten' wie Zugriffe auf Daten an x Stellen im Projekt sind somit erst gar nicht möglich. Dank dieser klar definierten Schnittstellen sind Objekte später mit Leichtigkeit gegen andere austauschbar. Software wird so wartungsfreundlich und leicht testbar – und das noch nach Jahren. Mit der Kapselung bei der OOP ergeben sich somit neue und höchst flexible Möglichkeiten bei Programm-Updates oder Fehlerkorrekturen in der Software. Ein Beispiel: Angenommen ein Softwarebaustein hat die Aufgabe eine Temperatur zu regeln. Es sind tausende Maschinen in unzähligen Maschinenvarianten am Markt und dann stellt man fest, dass der Temperaturregler einen schwerwiegenden Fehler enthält. Klassischerweise müsste der Hersteller nun für jede Maschine den spezifischen Softwarestand korrigieren und an der Maschine updaten. Lasal Class erstellt in diesem Fall auf Knopfdruck einen USB-Stick mit der korrigierten Klasse, in unserem Beispiel dem korrekten Reglerbaustein. Dieser kann überall per Stick oder Online-Fernwartung eingespielt werden, ohne dafür die tatsächlichen Programme auf den Maschinen kennen zu müssen.

Komfortables Debugging

Eine zügige Entwicklung und umfassende Analyse von Programmen oder auch eine eventuelle Fehlersuchte wird mit einer Vielzahl an integrierten Debugging-Tools erleichtert: Beispiele dafür sind das Setzen von Breakpoints, eine Online-Diagnose im Netzwerk, bei welcher der aktueller Status der einzelnen Objekte ersichtlich ist, eine Wertänderung direkt im Netzwerk und das Absetzen von ganzen Befehlsketten mit Parametern. Mit dem PlcTraceView lassen sich die Task-Abarbeitungen anzeigen. Der Echtzeit Data Analyzer ermöglicht eine Samplerate bis zu 50µs und Realtime Trendaufzeichnungen. Auch ein umfassender Projektvergleich ist integriert.

Programme automatisch erstellen

Gekapselte Bausteine eröffnen die Möglichkeit, Projekte mit Hilfe der standardisierten Script-Sprache Python vollkommen automatisch zu erstellen oder abzuändern. So können abgestimmte Software-Versionen für unterschiedliche Ausstattungsvarianten oder Ausprägungen generiert werden. Oft gibt es für eine bestimmte Maschinen-Baureihe ein einziges Basisprojekt. Von diesem können dann automatisch verschiedene Varianten für die kundenspezifischen Maschinen derselben Baureihe abgeleitet werden. ■

www.sigmatek-automation.com

Autor: Ing. Bernhard Gangl ist Leiter der Softwareabteilung bei der Sigmatek GmbH & Co KG in Lamprechtshausen/Salzburg.