Objektorientiert programmieren in der SPS?

Bild: HS Mannheim

OOP für Automatisierungstechniker an der Hochschule Mannheim: Prof. Seitz lehrt seit Jahren objektorientierte Programmierung – mittlerweile mit UML-Klassendiagramm

Bereits mit der Einführung der objektorientierten Programmierung (OOP) in Codesys V3 im Jahr 2006 und damit noch vor der Aufnahme in die 3rd Edition der IEC61131-3 gab es sehr kontroverse Reaktionen auf die Frage: Macht es Sinn, SPSen objektorientiert zu programmieren?

Heute werden viele leistungsfähige Maschinen immer noch klassisch programmiert und in Betrieb genommen: Einige Hundert Zeilen Code, z.B. in AWL, von der vorherigen Maschine übernehmen, diejenigen Codestellen suchen, finden und editieren, die gemäß Maschinenänderungen anzupassen sind, eventuell noch neue Applikationsteile hinzufügen und dann die gesamte Applikation auf die Maschine laden. Applikationsprogrammierer, die nach dieser Vorgehensweise arbeiten, fragen zu Recht: „Wofür benötige ich die OOP?“ Schließlich sind sie erfahrene Spezialisten, haben die Maschinen- und Anlagenfunktionen im Griff und lösen mit ihrer vertrauten Programmiermethode alle Aufgaben. Das Paradigma der OOP haben sie weder erlernt noch Erfahrungen damit gemacht. Steigen sie auf OOP um, dann hätte das sofort spürbare Konsequenzen: Längere Inbetriebnahmen, weniger Übersicht über ihren Code, aufwendigere Änderungen und Anpassungen. Der Aufwand für Umstellung und das Erlernen der neuen Programmiermöglichkeiten übersteigt den möglichen Nutzen. Für kleine Projekte und Teams kann die Objektorientierung ihre Vorteile nicht entfalten und bringt kaum Verbesserungen. Mittlerweile fließen objektorientierte Programmiermethoden zunehmend in die Ausbildung von Automatisierungstechnikern ein. Und auch Ingenieure und Informatiker, die bereits im Studium mit der OOP vertraut gemacht wurden, erstellen immer öfter den Applikationscode für modulare oder komplexe Maschinen. Für sie ist eine Zerlegung der Applikation in relevante ‘Objekte’ ganz natürlich, auch deren vereinheitlichte Definition durch Schnittstellen, sowie die Vererbung von Programmcode für andere Bausteine. Früher wurden industrielle Applikationen oft geringschätzig als ‘Klapperlogik’ bezeichnet und waren bei Software-Entwicklern verpönt. Mit der Möglichkeit, Steuerungen objektorientiert zu programmieren, steigt die Attraktivität der Applikationsentwicklung. Insofern garantiert die OOP,

> *„So etwas kommt nicht in unsere Maschinen!*

Auszug aus dem SPS-MAGAZIN, Fachzeitschrift für Automatisierungstechnik, Ausgabe 6/2016.

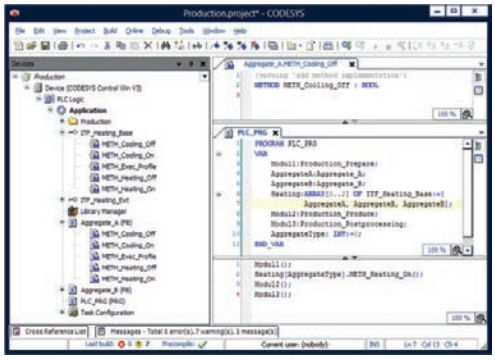

Bild: 3S-Smart Software Solutions GmbH

Schnittstellen in einem SPS-Programm 'zwingen' den Applikationsprogrammierer, die entsprechenden Methoden auszuprogrammieren. Gleichzeitig ermöglichen sie eine zentrale Festlegung, welche konkrete Methode wirklich ausgeführt wird.

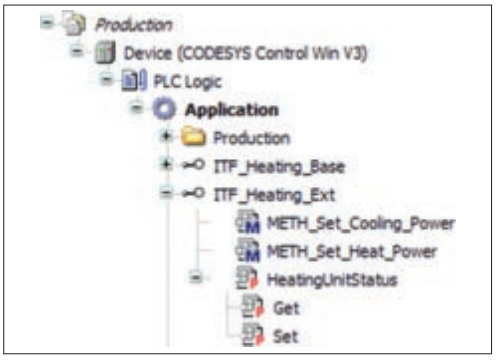

Bild: 3S-Smart Software Solutions GmbH

Mit dem Anlegen von Interface Properties stehen Get- und Set-Methoden zum Zugriff auf die gekapselten Daten zur Verfügung.

dass auch künftig sehr gut ausgebildete Programmierer Spaß daran haben, die technologische Weiterentwicklung des Maschinen- und Anlagenbaus in Applikationscode umzusetzen. Denn es ist kein Geheimnis: Die Software ist verantwortlich für große Teile der Wertschöpfung – dafür sind effiziente Werkzeuge, Methoden und entsprechend ausgebildete Programmierer unerlässlich.

Komplexität beherrschen

„Objektorientiert programmierte Programme sind schwieriger zu verstehen" – diese Aussage trifft sicherlich oft zu – aber nur dann, wenn man nicht an die abstrakte Zerlegung von Aufgaben gewöhnt ist. Entsprechend äußert sich der User 'Structured-Trash' im SPS-Forum: „Ob sich das lohnt, hängt von der Anzahl der FBs ab. Bei meinen Programmen (allgemeiner Maschinenbau) wird die Anzahl der FBs, die ich in einem PRG oder einem anderen FB instanziiere, nur selten zweistellig. Da sehe ich noch keinen Bedarf für Interfaces, sondern schätze beim Debuggen den Vorteil, mit jedem FB 'per Du' zu sein. ..." **(www.sps-magazin.de/?14358)**. Der 'totale Blick' auf das ganze Projekt geht ganz schnell verloren, wenn die Zahl der Bausteine steigt. Komplexere Maschinen führen von selbst zu komplexeren Projekten. Das Zerlegen der Applikation in eine überschaubare Objektstruktur erfordert zwar erst einmal einen nicht unerheblichen Initialaufwand. Dieser Aufwand lohnt sich aber, wenn das Projekt laufend weiterentwickelt wird, vielleicht sogar von verschiedenen Applikateuren. Insbesondere für die Projektierung von Serienmaschinen, die aus ähnlichen Modulen in Varianten gefertigt werden, bietet die OOP unschätzbare Vorteile bei der Beherrschung der Komplexität. So können die Anwender mit dem Sprachmittel der Schnittstelle Methodensätze vordefinieren, die dann in unterschiedlichen Einheiten ähnlichen Typs zum Einsatz kommen, d.h. 'implementiert' werden. Setzt ein Sondermaschinenbauer z.B. Heizaggregate verschiedener Hersteller und Bauart in seiner Maschine ein, so müsste er in der funktionalen Programmierung die Softwarebausteine für jedes einzelne Aggregat separat erstellen und aufrufen. Eine Aggregatsänderung führte damit zwangsläufig zum Editieren des gesamten relevanten Softwareteils. Und das, obwohl alle Aggregate ähnliche oder sogar identische Funktionen bereitstellen, wie z.B. Heizung an/aus, Heizprofil durchfahren, Kühlung ein/aus oder ähnliches. In der OOP definiert er eine einheitliche Schnittstelle für solche Basisfunktionen, die für sich zunächst leere Methoden enthalten. Codiert der Anwender dann Funktionsbausteine für das jeweilige Aggregat und implementiert die Schnittstelle mit den Methoden, so gewinnt er zweierlei: Zum einen kann und wird er nicht vergessen, alle definierten Methoden der Schnittstelle für dieses Aggregat auch auszucodieren – das Tool erinnert ihn daran. Zum anderen muss er lediglich an einer zentralen Stelle festlegen, welches spezielle Aggregat denn an der Maschine wirklich zum Einsatz kommt. Der Aufruf der speziellen Methoden kann direkt über die Schnittstelle erfolgen – gleichgültig, in welchem FB für das eingesetzte Aggregat die entsprechende Methode definiert wurde. Und daraus folgt: Wie komplex die Maschine auch sein mag, welche untergeordneten Einheiten konkret eingesetzt werden, der Aufruf von spezifischen Softwareteilen kann zentral festgelegt werden – er muss nicht jedes Mal neu durchdacht werden.

> *„Endlich kann man auch Industriesteuerungen vernünftig programmieren!*

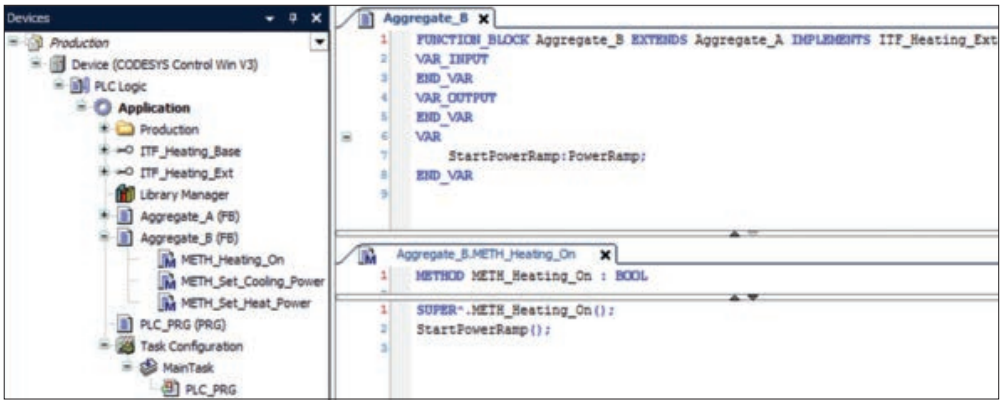

Bild: 3S-Smart Software Solutions GmbH

Aggregate_B erbt mit dem Schlüsselwort EXTENDS alle Methoden von Aggregate_A. Die überladene Methode METH_Heating_On führt durch den Operator Super^ zunächst den Code der ursprünglichen Methode aus.

Kapselung von Daten

Mit den verschiedenen Gültigkeitsbereichen VAR_INPUT, VAR_OUTPUT, VAR_INOUT und VAR für die Deklaration von Steuerungsvariablen haben die Väter der IEC61131-3 Möglichkeiten zur sinnvollen Datenstrukturierung geschaffen. Der Anwender legt damit fest, auf welche dieser Variablen innerhalb eines Bausteines zugegriffen werden darf. Einen versehentlichen Zugriff und potenzielle Fehlerquellen kann er so vermeiden. In der OOP geht die Datenkapselung viel weiter: Daten können einerseits in Methoden angelegt und dann nur von diesen verwendet werden. Andererseits bietet die OOP den neuen Objekttyp 'Eigenschaft' bzw. 'Property' als Kindobjekt eines FBs, in dem Daten sauber gekapselt und lediglich über zugehörige Set- und Get-Methoden verwendet werden können. Referenzen auf bestehende Objekte und eine Datentyp-Abfrage ermöglichen dabei einen definierten Umgang mit diesen Daten – auch für komplexere Variablen, wie z. B. Strukturen oder abgeleitete Datentypen. Der Nutzen solcher Konstrukte steigt mit der Komplexität eines SPS-Programmes.

Wiederverwendung von existierendem Code

In IEC61131-3-Entwicklungssystemen wie Codesys ist eine komfortable Bibliotheksverwaltung seit jeher verfügbar. So kann der Anwender FBs oder Funktionen zur späteren Verwendung in einer Bibliothek abspeichern und diese in anderen Projekten einbinden. Folgende Funktionen ermöglichen dem Anwender beispielsweise eine flexible und einfache Verwendung von solchen Bausteinen in beliebigen Projekten:

- Versionierung von Bibliotheken
- Bindung von dedizierten Versionen auf bestimmte Geräte oder Profile
- Dokumentation der Funktionalität innerhalb des Quellcodes
- Nachladen von fehlenden Bibliotheken von Online-Ablageorten
- Zugriff über Platzhalter

Was ist mit einem Code, der nicht vollständig in einen FB ausgelagert und in eine Bibliothek verpackt werden kann? Zum Beispiel, weil er sich auf spezifische Teile der Applikation bezieht, aber dennoch wiederverwendet werden könnte? Für solche Fälle bietet die OOP wiederum ein mächtiges Mittel: die Vererbung! Im oben genannten Beispiel von den unterschiedlichen Heizaggregaten könnte der Anwender z.B. den 'Prototyp' eines Aggregats in Form eines Funktionsbausteins mit sämtlichen Funktionen bzw. Methoden codieren, die in allen real verwendeten Aggregaten verfügbar sind. Er erstellt damit eine Basisklasse. Ausgehend von solch einer Klasse kann er jetzt mittels des neuen Keywords 'Extends' ein komplexeres Aggregat definieren und mit Methoden ausprogrammieren, die die Basisfunktionalität erweitern. Dabei stehen ihm ohne weiteres Zutun die in der Basisklasse erstellten Methoden zur Verfügung. Reicht die Funktion einzelner Methoden jedoch nicht, so kann er sie 'überladen', d.h. um Funktionen erweitern, die in der Basisklasse so nicht implementiert waren. Dabei ist der Code des Basistyps jedoch nicht verloren, sondern kann über den Super-Operator bequem wieder aufgerufen werden – auch er wird an die erweiterte Klasse vererbt. Anhand von FBs, die auf andere FBs bzw. auf Schnittstellen zugreifen ('Aggregation'), kann der Anwender jetzt die gesamte Applikation hierarchisch aufbauen und dabei bereits codierte Funktionen weiterverwenden.

Effizienz beim Engineering

Wenn einem Anwender die Strukturierung einer gesamten Applikation mit den Mitteln der OOP nicht vertraut ist, so mag ihm schnell der Überblick verloren gehen. Mit ein wenig Übung und Erfahrung gewinnt jedoch eine komplexe Applikation sogar an Übersichtlichkeit und wird deutlich besser beherrschbar. Denn oft kann das gesamte Projekt durch Aufruf weniger Bausteine komplett beschrieben werden. Diese Bausteine verzweigen ihrerseits in die Tiefe und führen die codierten bzw. vererbten Funktionen aus. Die weite Verbreitung der OOP in anderen Bereichen der Software-Entwicklung beweist eindrucksvoll, dass ihre Vorteile überwiegen. Eine Umstellung von Applikationen darf aber nicht 'mit der Brechstange' erfolgen, sondern sollte im Takt der natürlichen Weiterentwicklung von Anwendung und Programmierern erfolgen. Codesys wird dieser Forderung gerecht: Objektorientiert programmierte Bausteine werden bereitgestellt, vom Anwender jedoch funktional aufgerufen, ohne dass er von der OOP etwas merkt. Letztlich ermöglicht erst die OOP eine nahtlose, 'geschmeidige' Integration vieler mächtiger Funktionen wie Visualisierung, Motion Control oder Feldbusunterstützung in die IEC61131-3 Oberfläche. Viele Anwender nutzen selbst die Möglichkeiten der OOP für ein effizienteres Engineering, andere profitieren nur implizit davon. Ob explizite Anwendung oder impliziter Nutzen: Von der OOP profitieren alle Anwender! ■

Autor: ***Dipl.-Ing. (FH) Roland Wagner,***
Leiter Produkt Marketing,
3S-Smart Software Solutions GmbH
www.codesys.com

Auszug aus dem SPS-MAGAZIN, Fachzeitschrift für Automatisierungstechnik, Ausgabe 6/2016.