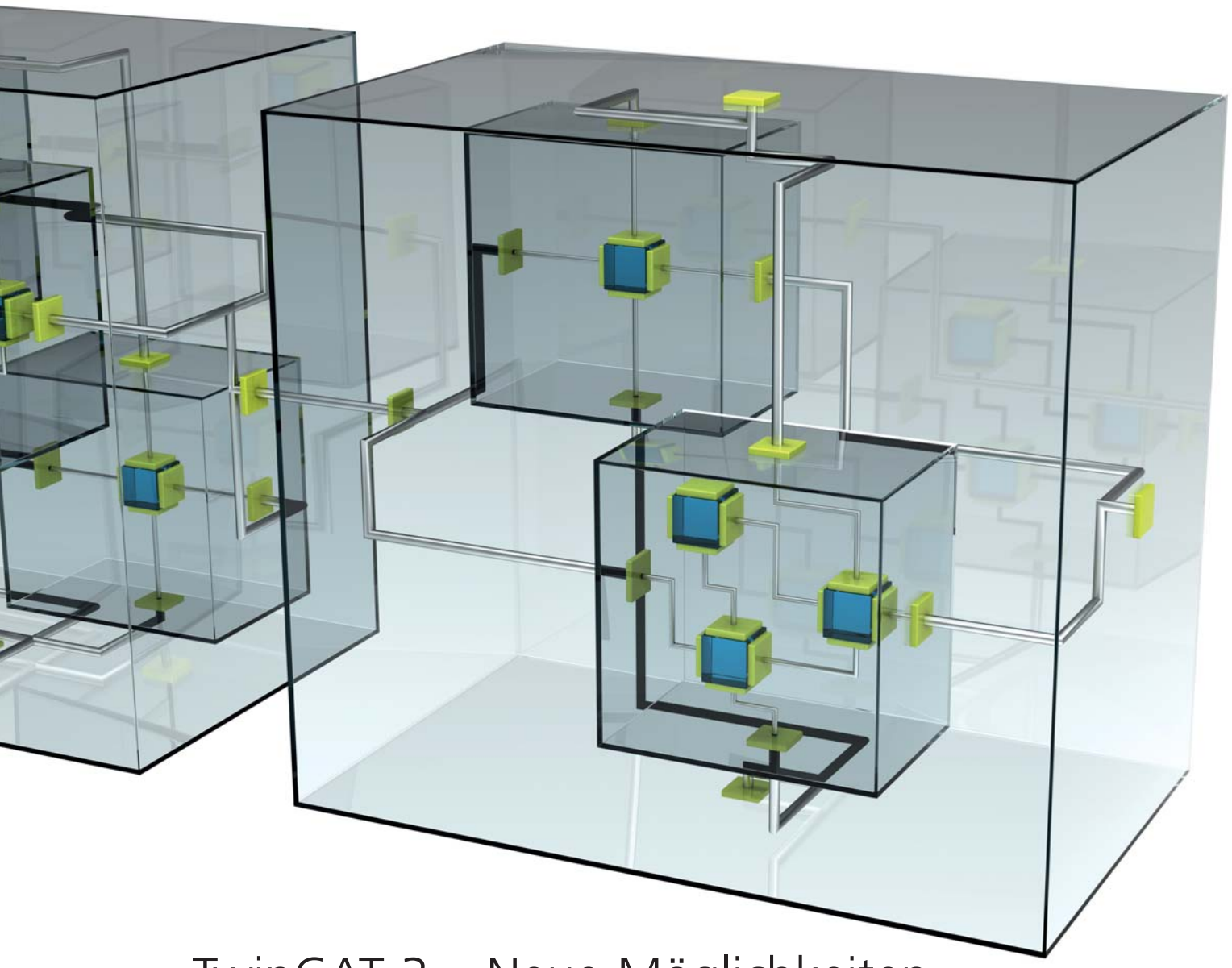
eXtended Automation Engineering (XAE): Modulares Engineering
unter Nutzung der objektorientierten Erweiterungen der IEC 61131-3



TwinCAT 3 – Neue Möglichkeiten durch objektorientierte Erweiterungen

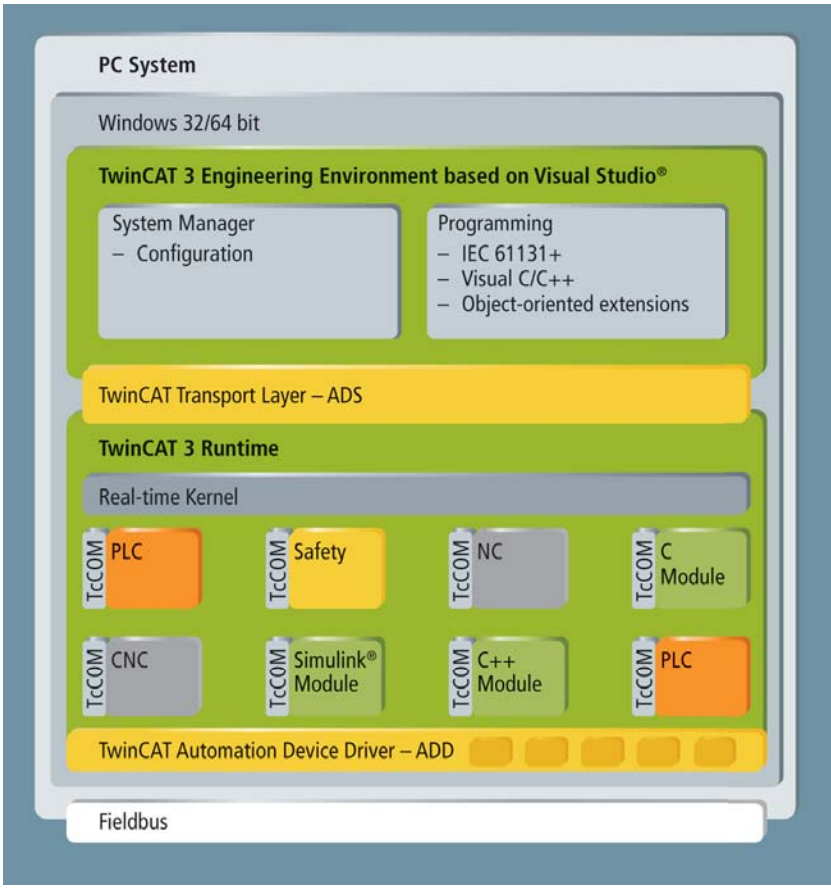
Mit der neuen Softwaregeneration TwinCAT 3 nutzt Beckhoff das Microsoft Visual Studio®. In einer Entwicklungsumgebung sind alle wesentlichen Programmiersprachen, wie C/C++, eine Matlab®-/Simulink®-Anbindung sowie die Sprachen der IEC 61131-3 inklusive der objektorientierten Erweiterungen der IEC 61131-3, integriert. Diese ermöglichen u. a. die Verwendung von objektorientierten Techniken, wie Einfachvererbung, Interfaces, Methoden und Attribute, die sowohl die Wiederverwendbarkeit als auch die Qualität des Steuerungscodes deutlich erhöhen.

Um den ständig steigenden Anforderungen an Produktivität, Flexibilität und Verfügbarkeit einer Anlage Herr zu werden und dennoch die Kosten für den Engineering-Prozess, die Inbetriebnahme und die Wartung einer Anlage zu reduzieren, sind neue Methoden und Werkzeuge erforderlich. Ein möglicher Weg, um dies zu erreichen, ist die Anwendung eines mechatronischen Ansatzes. Dabei wird die Anlage in mechatronische Module (Objekte) zerlegt, welche einzelne Funktionen des Gesamtprozesses einer Anlage abbilden. Sind diese Module nicht nur funktional gekapselt, sondern die Schnittstellen zwischen diesen Modulen definiert, erhält man eine Art Baukasten, aus dem man sich bei der Projektierung neuer Anlagen bedienen kann.

Die einzelnen Module des Baukastens bilden dabei die jeweilige Teilfunktion komplett ab. Mit anderen Worten umfassen sie alle Facetten eines Moduls von der Mechanik über die Elektrotechnik bis hin zu den Steuerungsfunktionen, die nicht zwangsläufig auf einer eigenen Steuerung laufen.

Um diesen objektorientierten Gedanken auch im Bereich der Steuerungstechnik Rechnung zu tragen, müssen die hier verwendeten Sprachmittel um diese Möglichkeiten erweitert werden. Ziel kann es dabei aber nicht sein, eine neue objektorientierte Sprache einzuführen, sondern vielmehr zu ermöglichen, diese objektorientierten Erweiterungen in den Standardsprachen der Steuerungstechnik zur Verfügung zu stellen. An diesem Punkt setzt TwinCAT 3 an.

Die Software TwinCAT 3 ist die konsequente Weiterentwicklung des seit vielen Jahren und durch viele Applikationen bekannten TwinCAT 2. Ziel dieser Weiterentwicklung war, neben anderen wichtigen Punkten, wie der Integration der Beckhoff-Engineering-Werkzeuge PLC-Control und System Manager in eine Umgebung und der Verwendung einer weltweit anerkannten Softwareumgebung, dem Microsoft Visual Studio® 2010, als Framework, auch die Anpassung der TwinCAT-Software-Umgebung an die ständig steigenden Anforderungen im Bereich des Engineerings von Maschinen und Anlagen. Um dabei der neuen modul- bzw. objektorientierten Denkweise Rechnung zu tragen, wurden die Sprachmittel der IEC 61131-3 um zehn neue Schlüsselworte erweitert, deren Bedeutung und Anwendung im Folgenden erläutert werden soll.



TwinCAT-3-Architektur der Entwicklungsumgebung

Neue Schlüsselworte in TwinCAT 3

EXTENDS

Mit dem Schlüsselwort EXTENDS können Funktionsbausteine von anderen Funktionsbausteinen abgeleitet werden. Dabei haben die abgeleiteten Funktionsbausteine dieselben Methoden und Eigenschaften wie die Basisfunktionsbausteine, die allerdings auf den Daten des abgeleiteten Funktionsbausteins arbeiten.

INTERFACE

Bei Interfaces handelt es sich um virtuelle Klassen, die Methoden definieren, die von ihnen abgeleitete Objekte instanziieren müssen, um fehlerfrei kompilierbar zu sein. Sie können somit verwendet werden, um die Struktur bzw. um Schnittstellen von Objekttypen zu definieren. Darüber hinaus ermöglichen sie aber auch durch Pointer auf das Interface, dass alle von ihnen abgeleiteten Funktionsbausteine über ebendiesen Interface-Pointer gleich „angesprochen" werden können.

IMPLEMENTS

Das Schlüsselwort IMPLEMENTS implementiert ein Interface in einem Funktionsbaustein.

METHOD

Methoden sind eine Art Funktion, die einem Funktionsbaustein zugewiesen sind. Mit anderen Worten sind diese keine eigene POUs (Program Organisation Units), sondern sind Bestandteil des Funktionsbausteines und arbeiten auf den Daten der sie beinhaltenden Funktionsbaustein-Instanz.

PROPERTY

Properties sind Eigenschaften, die einem Funktionsbaustein oder einem Programm zugewiesen werden können. Diese Eigenschaften können jeweils eine Get- und eine Set-Methode beinhalten und dienen als eine Art Schreib- bzw. Leseschutz für interne Variablen. Dabei ist es ebenfalls möglich, in diesen Get- bzw. Set-Methoden neben der eigentlichen Zuweisung noch weitere Anweisungen auszuführen, um

den benötigten Wert zur Verfügung zu stellen bzw. den zugewiesenen Wert vorzuverarbeiten.

| **THIS**
Mithilfe des This-Pointers ist es möglich, auf die Funktionsbausteininstanz selbst zu zeigen.

| **SUPER**
Mithilfe des Super-Pointers kann auf Methoden des vererbenden Funktionsbausteines gezeigt werden.

| **FB_init/FB_reinit/FB_exit**
In Anlehnung an Konstruktoren und Destruktoren werden diese Methoden beim Initialisieren, Kopieren und Verlassen der Funktionsbaustein-Instanz aufgerufen.

Diese neuen Schlüsselworte stehen in allen Sprachen der IEC 61131-3 zur Verfügung. Die Anwendung der neuen Schlüsselworte soll nun anhand eines kleinen Beispiels vorgestellt werden.

Als Basis für dieses Beispiel soll uns eine Anlage dienen, in der mehrere Arten von Zylindern eingesetzt werden. Die Aufgabenstellung besteht nun darin, den Steuerungscode für diese Anlage so flexibel zu gestalten, dass der eigentliche Sollablauf der Anlage unabhängig vom eingesetzten Zylindertyp ist. Allen diesen Zylindertypen ist gemein, dass sie über zwei Positionen (links und rechts) verfügen, die mittels Endlagenschalter detektiert werden können.

Als Lösungsansatz für dieses Beispiel wird nun definiert, dass entsprechend dem objektorientierten Gedanken die Funktionalität dieser Zylinder jeweils komplett in einem Funktionsbaustein abgebildet ist. Darüber hinaus wird definiert, dass alle Zylinder eine Zustandsmaschine besitzen, die mittels Ein- bzw. Ausgängen die geforderte Position des Zylinders übergeben bekommen bzw. die aktuelle Position des Zylinders zurückliefern soll.

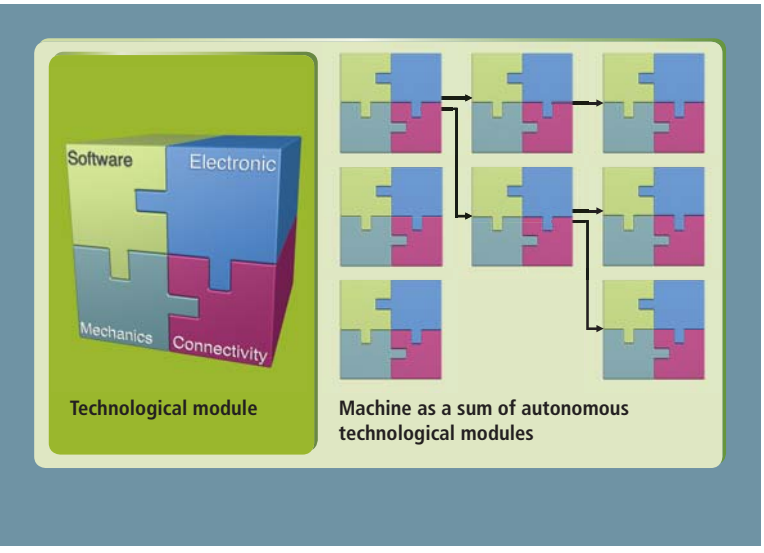


Abbildung 1: Mechatronischer Aufbau des Gesamtsystems

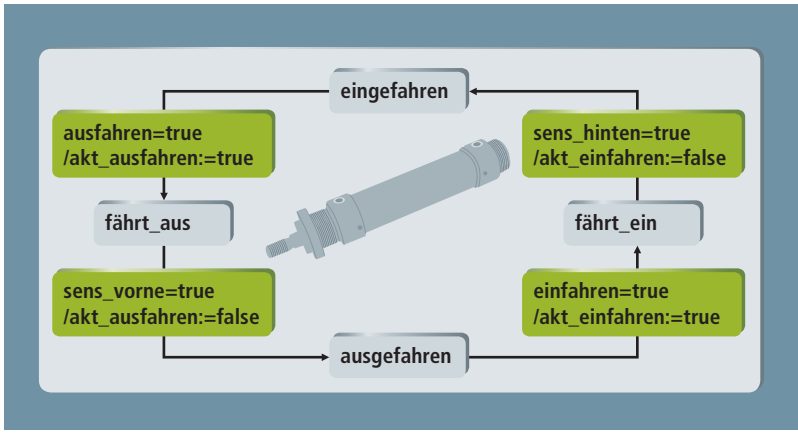


Abbildung 2: Beispiel Zylinder mit Zustandsmaschine

Da laut Aufgabenstellung der jeweils in der Anlage eingesetzte Typ des Zylinders flexibel sein soll, muss das Interface, über das die Zylinder angesprochen werden, gleich sein. Aus diesem wird als erstes ein Interface *iCylinder* definiert, das von allen Zylinder-Funktionsbausteinen implementiert werden muss. Dieses Interface besitzt eine Methode *mStateMachine*, welche die Zustandsmaschine der Zylinder abbildet. Diese Methode hat die Eingänge *bPosLeftReq* und *bPosRightReq* sowie die Ausgänge *bActPosLeft* und *bActPosRight*, welche die Signale der Endlagenschalter repräsentieren. Diese Ein- und Ausgänge sind jeweils vom Typ BOOL (siehe Abbildung 3).

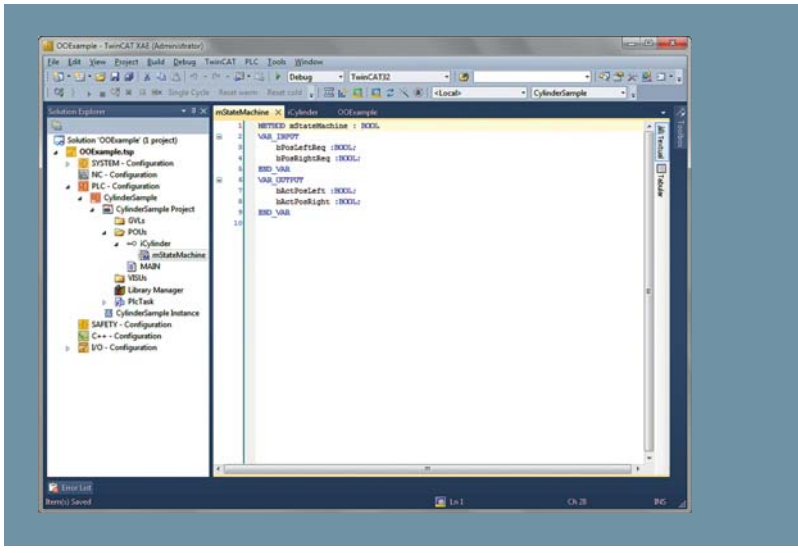


Abbildung 3: Interface iCylinder mit der Methode mStateMachine

Basierend auf diesem Interface kann nun ein Basisfunktionsbaustein *fbBaseCylinder* erstellt werden, in dem die Zustandsmaschine der Zylinder implementiert wird. Wählt man beim Anlegen des Funktionsbausteines

bereits aus, dass das Interface *iCylinder* implementiert werden soll, wird die Methode *mStateMachine* samt ihrer Ein- und Ausgänge automatisch angelegt und muss nur noch ausprogrammiert werden. Welche Interfaces ein Funktionsbaustein implementiert, wird mithilfe des Schlüsselwortes IMPLEMENTS definiert (siehe Abbildung 4). Die Ableitung eines Funktionsbausteines von mehreren Interfaces ist dabei möglich.

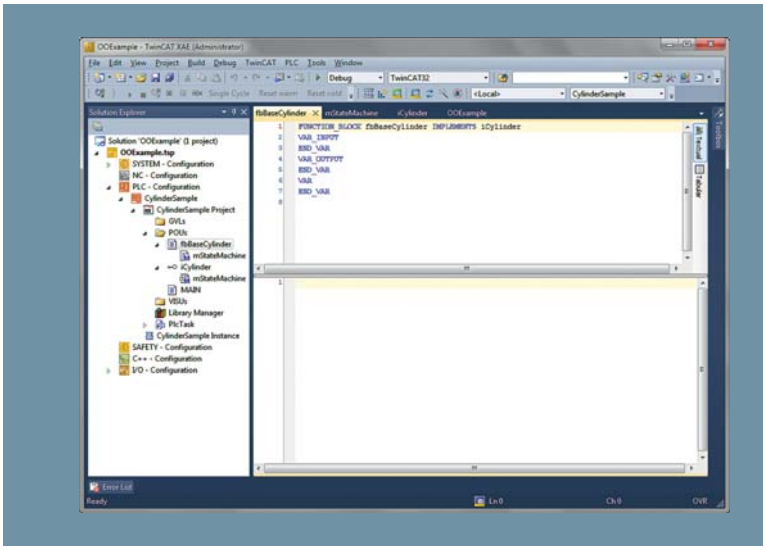


Abbildung 4: Basisfunktionsbaustein der Zylinder

Ist der Basisfunktionsbaustein ausprogrammiert, können die Funktionsbausteine der verschiedenen Zylindertypen von ihm abgeleitet werden. Dies geschieht mit dem Schlüsselwort EXTENDS.

Die abgeleiteten Funktionsbausteine erben nun alle Methoden und Eigenschaften des Basisbausteines. In unserem Beispiel bedeutet dies, dass die Funktionsbausteine der abgeleiteten Zylinder die Methode der *mStateMachine* vom Basisfunktionsbaustein erben. Der Vorteil hierbei ist, dass Erweiterungen in der Methode *mStateMachine*, die alle Zylinder-Funktionsbausteine betreffen, nur im Basisfunktionsbaustein nachgepflegt werden müssen und dann sofort in allen abgeleiteten Funktionsbausteinen zur Verfügung stehen. Soll die Zustandsmaschine eines abgeleiteten Zylinders ein anderes Verhalten aufweisen, als die des Basisfunktionsbausteines, ist es möglich, diese Methode zu überschreiben. Dies geschieht, indem der abgeleitete Funktionsbaustein eine Methode mit demselben Namen implementiert, wie die des Funktionsbausteins der Basisklasse.

Aufgabenstellung dieses Beispiels war es, einen Lösungsansatz zu finden, bei dem der Sollablauf der Anlage unabhängig ist von Zylindertyp, der tatsächlich in der Anlage verbaut wird. Um zu zeigen, dass die Aufgabenstellung mit diesem Lösungsansatz erfüllt wird, werden nun zwei Funktionsbausteintypen *Cylinder_TypA* und *Cylinder_TypB* vom Basisfunktionsbaustein abgeleitet und entsprechend ausprogrammiert.

Dabei werden beiden Funktionsbausteintypen noch weitere, teilweise unterschiedliche, Ein- und Ausgänge hinzugefügt. Von beiden Typen wird im Programm *Main* jeweils eine Instanz angelegt (siehe Abbildung 5). Da der Zugriff auf diese Funktionsbausteine über das Interface geschehen soll, wird ebenfalls ein Interface-Pointer angelegt, der auf das Interface *iCylinder* zeigt.

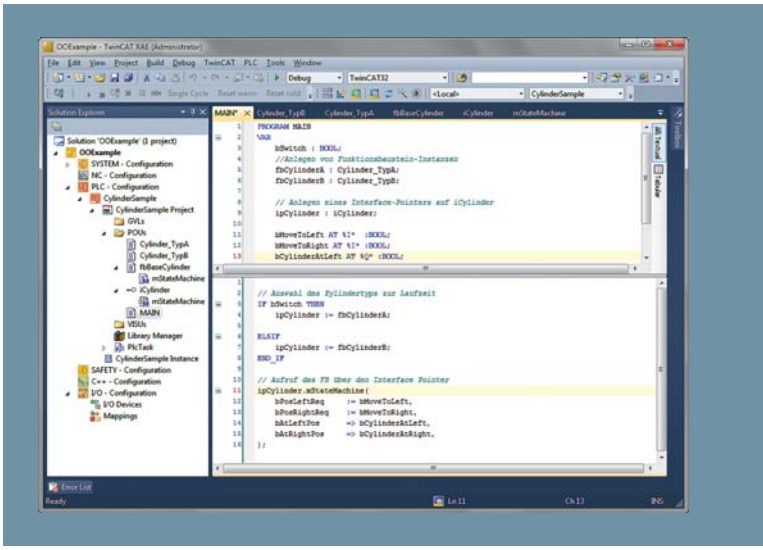


Abbildung 5: Aufruf der Zylinder-Funktionsbausteine

Wie in der Abbildung zu sehen ist, erfolgt der eigentliche Aufruf unserer Zustandsmaschine nun über den Aufruf der Methode *mStateMachine* des Interface-Pointers. Durch die Zuweisung einer Funktionsbaustein-Instanz auf den Interface-Pointer kann nun sogar zur Laufzeit entschieden werden, welche Methode *mStateMachine* tatsächlich aufgerufen wird. Dies bezeichnet man als „Späte Bindung".

Eine weitere schöne Möglichkeit für die Anwendung von Interfaces ist zum Beispiel auch die Betriebsartenumschaltung von Modulen einer Anlage. Wird sichergestellt, dass alle Module einer Anlage (z. B. auch unser Zylinder) ein Interface implementieren, das die Basisschnittstelle für die Betriebsartenumschaltung enthält, kann mittels nur einer Schleife über eine Struktur vom Typ dieses Interfaces eine Umschaltung der Betriebsarten von allen Anlagenkomponenten herbeigeführt werden.

Fazit

Fasst man das Gezeigte noch einmal zusammen, ist zu erkennen, dass durch die Einführung dieser Schlüsselworte neue Möglichkeiten zur Verfügung stehen, um Steuerungscode nicht nur funktional zu kapseln, sondern auch zu strukturieren. Es ist somit nicht nur das Ziel, die Wiederverwendbarkeit des Steuerungscodes zu erhöhen, was zu deutlichen Zeit- und damit auch Kosteneinsparungen führt, sondern auch die Lesbarkeit und Erweiterbarkeit. Man erhält damit Steuerungscode von deutlich höherer Qualität.

www.beckhoff.de/TwinCAT3