Jetter

WHITEPAPER

Hochsprache versus SPS-Sprachen in der Automatisierungstechnik

Autor: Andreas Leu, Jetter AG, Germany

2013

Zusammenfassung

Eine Aufgabe und zwei völlig unterschiedliche Lösungsansätze. So könnte man die Problematik von zwei Programmierphilosophien in der Automatisierungstechnik zusammenfassen. Dieses Whitepaper beleuchtet die Entstehung der klassischen SPS-Sprachen und der Hochsprache für Automatisierungsanwendungen. Es zeigt auf, welche Faktoren darüber entscheiden, welche Sprache oder Technologie eingesetzt wird oder werden sollte.

Historie der speicherprogrammierbaren Steuerung

Der Siegeszug der SPS (**S**peicher**p**rogrammierbare **S**teuerung) oder englisch PLC (**P**rogammable **L**ogic **C**ontroller) begann Anfang der siebziger Jahre. Automatisierungsaufgaben wurden bis zu diesem Zeitpunkt mit diskret verdrahteter Relaistechnik realisiert. Es versteht sich von selbst, dass diese Systeme durch das häufige Schalten der Relais wartungsintensiv waren. Änderungen am Ablauf waren nur durch Umverdrahtung möglich. Um eine gewisse Flexibilität zu erreichen, wurden Schaltungen aufgebaut, bei denen durch Veränderungen von Steckverbindungen Abläufe verändert werden konnten.

In den USA wurden 1970 vier Unternehmen beauftragt, die Relaistechnik durch Mikrocontroller zu entwickeln. Ein wichtiges Kriterium war, dass die Fachleute, welche bisher die Elektropläne erstellten, einfach auf eine Programmiersprache umzuschulen waren. So entstand als erste Programmiersprache der Kontaktplan (KOP). Er stellte die überwiegend bitorientierten Aufgaben optimal dar. Die Entwickler konnten wie gewohnt den Maschinenablauf in bekannter Darstellung abstrahieren und die Umschulungskosten waren dementsprechend gering. Um die Flexibilität zu erhöhen, kamen später die Sprachen Funktionsplan (FUP) und Anweisungsliste (AWL), eine an Assembler angelehnte Sprache, dazu.

Die heute gültige Norm IEC-61131 beinhaltet nebst KOP, FUP und AWL zusätzlich den an Pascal angelehnten strukturierten Text (ST) und die Ablaufsprache (AS), welche eine Art Zustandsdiagramm als Schrittkette darstellt. Diese Norm wurde 1996 verabschiedet und ist bis zum heutigen Zeitpunkt praktisch unverändert gültig.

Steigende Ansprüche

War es am Anfang eine reine Verknüpfung von digitalen Ein- und Ausgangssignalen - also das direkte Ersetzen von Relaistechnik - kamen bald analoge Sensoren und Aktoren in der Automation von Anlagen zum Einsatz, beispielsweise Temperaturfühler oder analoge Stellglieder. Dafür werden in den normierten Programmiersprachen KOP und FUP sogenannte Funktionsblöcke geschaffen, die mit den Ein- und Ausgangssignalen beschaltet werden können. Konkret geht es wieder darum, eine Hardware-Schaltung zu entwickeln, welche den Prozess steuern würde.

Anfang der achtziger Jahre gewannen positionierbare Antriebe in der Automation immer mehr an Bedeutung. In der Regel waren es Schrittmotoren oder, bei höheren Dynamikansprüchen, geschwindigkeitsgeregelte Gleichstrommotoren (Servoantriebe). Diese ressourcenintensiven Regelungen konnten nicht mehr in der SPS selbst gelöst werden. Deshalb wurden Interface-Funktionsblöcke in die Programmiersprachen integriert, die über normierte Schnittstellen mit externen Schrittmotor- oder Servoreglern kommunizierten.

Um die Kosten bei der Verdrahtung von größeren Anlagen zu reduzieren, wurden die Signale von den Sensoren und zu den Aktoren in dezentralen Modulen zusammengefasst. Die längere Distanz vom Modul zur zentralen Steuerung wurde anstatt mit vielen Einzeladern durch ein Kabel mit wenigen Adern überwunden. Es entstanden die Feldbusse Profibus, Interbus, DeviceNet etc. Um diesen Anforderungen Rechnung zu tragen, wurden in den SPS-Sprachen wieder Funktionsblöcke eingesetzt.

Die heutigen Anforderungen an ein Automatisierungssystem gehen weit über das reine Steuern und Regeln von mechanischen Bewegungen hinaus. Produktionsprozesse sind datenbankgesteuert, Prozesse müssen über die Bussysteme auf die Millisekunde synchronisiert werden, komplexe Berechnungen sind notwendig, Steuerungen tauschen Daten untereinander aus etc.

Die klassischen Programmiersprachen für SPS-Systeme KOP, FUP und AWL sind von ihrer Historie her wegen ihrer Bitstruktur weniger für diese Aufgaben geeignet. Deshalb lässt die Norm IEC-61131-3 mit strukturiertem Text eine Quasihochsprache zu, um die Anforderungen zumindest zum Teil erfüllen zu können.

Hochsprache in der Automatisierung

Hochsprachen in der Informatik sind schon wesentlich älter als die SPS-Sprachen. Bereits zu Beginn der fünfziger Jahre entstanden Assemblersprachen, die den Maschinencode - also den Code, den ein Prozessor „versteht“ - in Textform darstellten. Mitte der fünfziger Jahre ergaben sich mit den ersten höheren Programmiersprachen Fortran und Cobol neue und auch komfortablere Möglichkeiten der Programmeingabe. Später folgten die bekannten Sprachen Basic und Pascal. Heute sind die in der Informatik meistverbreiteten Hochsprachen Java, C, C++, C#, Delphi etc. Hauptmerkmale dieser Hochsprachen sind folgende:

- Unterschiedliche Datentypen wie Boolean, Integer, Real, Char und auch strukturierte Datentypen wie Array und Struct
- Programmstrukturen wie Funktionen und Prozeduren
- Steuerkonstrukte *if, then, else,* oder Schleifen wie *for, while* und *repeat*

In der Automatisierung fanden Hochsprachen anfangs keine Verwendung. Wie bereits erwähnt, waren die Aufgaben für die Automation größtenteils bitorientiert. Dafür waren die Hochsprachen ursprünglich nicht geschaffen.

Anfang der achtziger Jahre bot die Firma Jetter mit SYMPAS auf dem Automatisierungsmarkt erstmals eine Alternative zu den klassischen SPS-Sprachen, und zwar eine hochsprachen-ähnliche textliche Programmierung. Damit wollten die Entwickler von SYMPAS dem Anwender ein Werkzeug zur Verfügung stellen, welches den wachsenden Anforderungen hinsichtlich Antriebsintegration, Visualisierung und Datenverwaltung eher gerecht wird. Diese Art der Programmierung wurde schon damals von einer Minderheit an Programmierern, die eine Hochsprache beherrschten, geschätzt, auch wenn die Sprache noch diverse Einschränkungen gegenüber einer echten Hochsprache hatte. Die Freiheitsgrade wurden mit SYMPAS in der Programmierung gegenüber klassischen SPS-Sprachen wesentlich erhöht.

Heute bietet die Jetter AG mit JetSym STX dem Anwender eine Programmiersprache, die gegenüber den klassischen Hochsprachen kaum mehr Defizite aufweist. STX bedeutet *Structured Text Extended.* Die normierte Programmiersprache *Structured Text* bildet die Grundlage für JetSym STX, der Sprachumfang wurde aber wesentlich erweitert. Seit dem Jahr 2005 ist auch die objektorientierte Programmierung möglich. Kenntnisse dieser Programmierart sind aber nicht zwingend erforderlich, da das Programm auch als prozeduraler Code erstellt werden kann. In JetSym STX ist auch die Mischung von prozeduraler und objektorientierter Programmierung möglich.

Hochsprachen können heutzutage in vielen SPS-Programmierumgebungen eingebunden werden. Diese Möglichkeit wird vor allem dann genutzt, wenn es sich um komplexere Algorithmen handelt, die in den klassischen SPS-Sprachen schlecht lösbar sind. Zur reinen Ablaufdarstellung werden Hochsprachen in der Regel nicht verwendet. Durch diese Mischung von verschiedenen Sprachen wird allerdings der Programmcode von Dritten schlecht les- und interpretierbar.

Kriterien für die Auswahl der Technologie

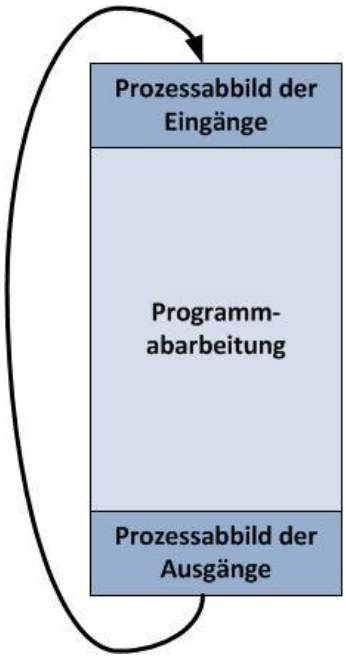
Auf dem weltweiten Automatisierungsmarkt existieren heutzutage über hundert Anbieter von freiprogrammierbaren Steuerungen. Für den Anwender stellt sich die Frage, welche Technologie- und Programmier-Philosophie zur Lösung seiner Automatisierungsaufgabe die richtige ist, beziehungsweise, welche Kriterien bei der Auswahl zum Zug kommen. Hier spielen folgende Faktoren eine Rolle:

- Komplexität der Aufgabe(n)
- Ausbildung des Programmierers
- Marktverbreitung und -akzeptanz des Systems
- Softfacts wie Sympathie oder Antipathie gegenüber der Marke, den Vertriebsmitarbeitern etc.

Wie die einzelnen Punkte gewichtet werden, ist schwierig zu beantworten und hängt auch von gewissen Sachzwängen ab. Hat sich ein Unternehmen oder eine Person auf ein System „eingeschossen“, wird es nicht immer wieder gewechselt. Auch die Lagerhaltung von möglichst wenigen Komponenten spielt bei der Systemauswahl eine Rolle.

Zyklusorientiert contra ereignisgesteuert

Der Großteil der auf dem Markt erhältlichen speicherprogrammierbaren Steuerungen arbeitet zyklusorientiert. Dies bedeutet, dass über das herstellerspezifische Betriebssystem immer das gesamte Programm abgearbeitet wird. Zu Beginn des Programmablaufs werden alle Eingänge (PAE = Prozessabbild der Eingangsinformationen) miteinander verknüpft, und erst am Ende werden alle Ausgänge geschrieben (PAA = Prozessabbild der Ausgangsinformationen). Folgende Grafik zeigt, wie ein SPS-Programm abgearbeitet wird.



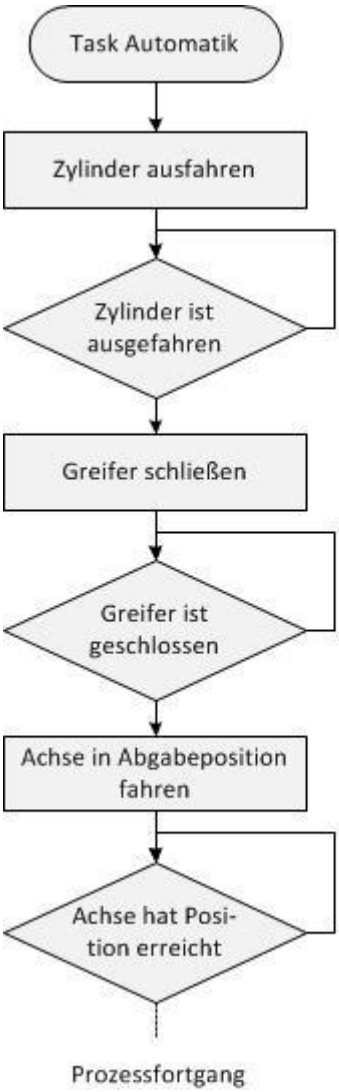
PAE: Der Zustand aller Eingangsinformationen (Sensoren) wird eingelesen

PAA: Der Zustand aller Ausgangsinformationen (Aktoren) wird ausgegeben

In der Automation sind viele Abläufe sequentiell - das heißt, sie werden schrittweise in Abhängigkeit eines vorhergegangenen Zustandes ausgeführt. In der Praxis bedeutet dies, dass nur der Schritt ausgeführt werden darf, der gerade aktuell ist. Da das Programm als Ganzes abgearbeitet wird, verwendet der Programmierer Zwischenmerker. Damit speichert er eine Information wie zum Beispiel „Greifer ist geöffnet“ oder „Zylinder ist ausgefahren“. Mit dieser Information wird der jeweilige Schritt verknüpft.

Das Betriebssystem einer Steuerung von Jetter arbeitet ereignisgesteuert und ist multitaskingfähig. Eine Eingangsinformation wird jeweils an der Stelle eingelesen, an der das Signal im Programm abgefragt wird. Eine Ausgangsinformation wird zu dem Zeitpunkt beschrieben, wenn diese im Programmcode stattfindet.

Der entscheidende Unterschied zum zyklusorientierten Ablauf ist der, dass der Programmcode in JetSym STX direkt den Prozess abbildet. Dieser lässt sich in abstrahierter Form zum Beispiel als Flussdiagramm darstellen wie unten gezeigt. Die einzelnen Blöcke lassen sich dann direkt in Programmcode umwandeln.

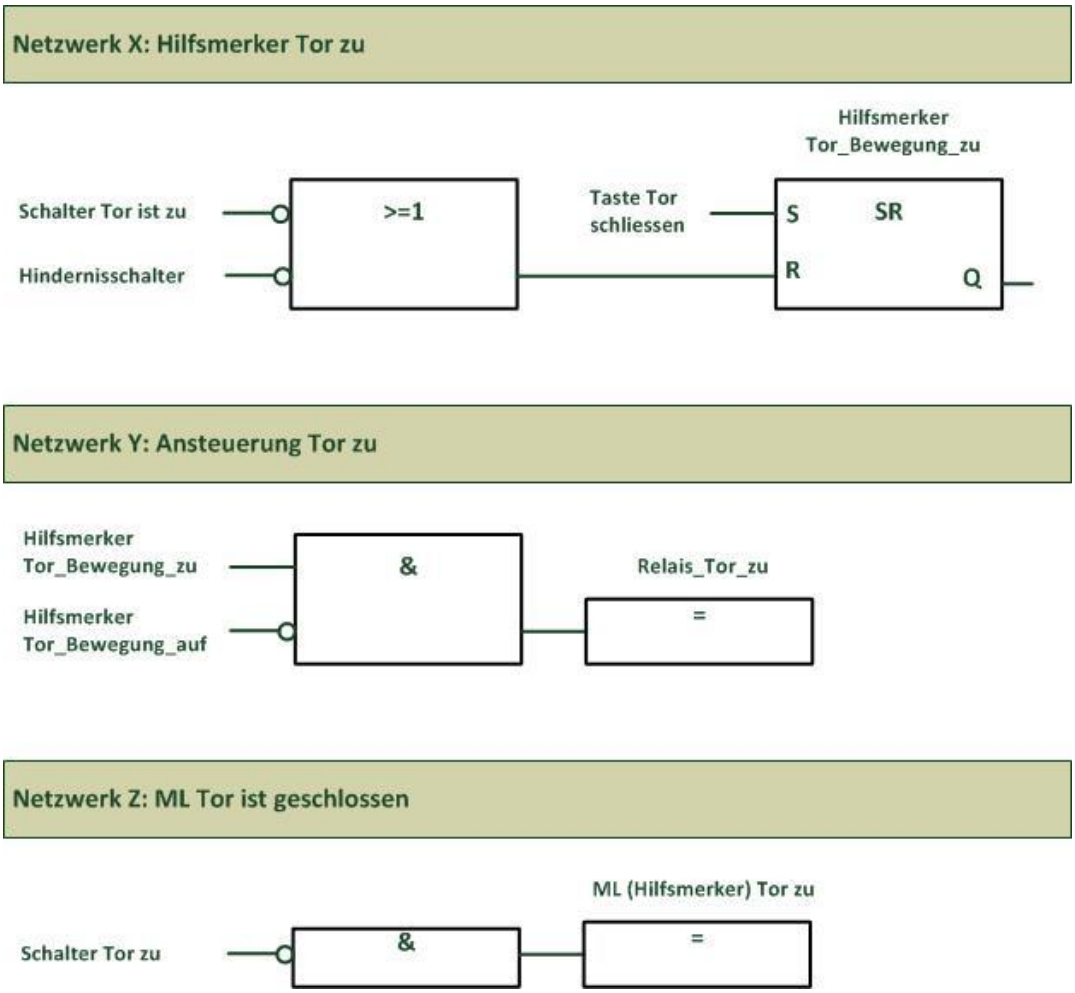



```
task Automatik

boZylinder := true;

when biZylinder continue;

boGreifer := true;

when biGreifer_zu continue;

ax_X.MovePtP.StartNewTargetposition();

when ax_X.IsInTargetWindow continue;
```

Bei dieser Form der Programmabarbeitung richtet sich der Programmcode rein nach dem Prozessablauf. In der Praxis bedeutet das, dass der Code in prozeduraler Form abgebildet wird. Da bei einer Maschine oder Anlage mehrere solcher Prozesse parallel ablaufen, ist eine Aufteilung in mehrere Tasks möglich und notwendig. Die Anzahl der parallel ablaufenden Prozesse entscheidet über die Anzahl der zu verwendenden Tasks. Bei den Jetter-Steuerungen sind bis zu hundert voneinander unabhängige Tasks möglich.

Unterschiede der Sprachen

Die Programmierer von klassischen SPS arbeiten in der Regel mit einer grafischen Programmiersprache wie FUP oder KOP. Anhand einer Torsteuerung wird die konkrete Umsetzung im Funktionsplan (FUP) erläutert. Das Beispiel zeigt, dass die Zustände wie „Tor_Bewegung_zu“ oder „Tor ist zu“ mittels Hilfsmerkern abgespeichert werden.



Im Gegensatz dazu ist die Hochsprache JetSym STX ablauforientiert. Da die Abfrage der Bedienelemente und die Programmsequenz „Tor_schliessen“ parallele Prozesse sind, werden dazu zwei Tasks verwendet. Die Beispielsequenz könnte damit etwa wie folgt abgebildet werden:

```
15
16          if bi_Taste_Tor_schliessen then
17              b_Start_Tor_schliessen := true;
18              when not bi_Taste_Tor_schliessen continue;
19          end_if;
20
```

Programmausschnitt aus dem Bedientask

```
37
38 task tTor_schliessen autorun
39     loop
40         when b_Start_Tor_schliessen and not bi_Hindernisschalter continue;
41         bo_Relais_Tor_zu := true;
42
43         when bi_Schalter_Tor_zu continue;
44         bo_Relais_Tor_zu := false;
45         b_Start_Tor_schliessen := false;
46
47     end_loop;
48 end_task;
49
```

Programmausschnitt: Tasks "Tor_schliessen"

Der Vergleich zeigt deutlich die unterschiedlichen Lösungsansätze. Es handelt sich hier um eine klassische SPS-Aufgabe, da es sich vor allem um Bit-Verarbeitungen handelt. Deshalb sind keine Vorteile für die Verwendung einer Hochsprache ersichtlich. Es ist gut erkennbar, dass sich die klassischen SPS-Sprachen an der Hardware einer Schaltung orientieren. Die Hochsprache JetSym STX hingegen bildet den Prozess ab.

Wann der Einsatz einer Hochsprache sinnvoll ist

Bei einem weiteren Beispiel erkennt man rasch die Grenzen der klassischen SPS-Programmierung. Der Quellcode zeigt eine relativ simple und häufige Aufgabe: Zehn Positionen werden mit einer Servoachse mittels einer *for*-Schleife über ein indiziertes Array (afPositionen) abgefahren. Nach Erreichen jeder der Positionen wird ein Ausgang (Bohrer), in diesem Beispiel für 250 Millisekunden, ausgefahren und anschließend wieder eingefahren. Der Ablauf ist als Text auch ohne Kommentare lesbar und verständlich. Eine Änderung (zum Beispiel der Verzögerungszeit) ist simpel, und Erweiterungen lassen sich einfach realisieren. Offensichtlich ist es deshalb nicht sinnvoll, diesen - wenn auch kurzen und eher simplen - Ablauf grafisch zu programmieren. Darüber hinaus wäre er kaum lesbar.

```
47     loop
48         when bStartPosition continue;
49
50         for nIndex := 1 to 10 do
51             axSpindel.MovePtp.StartNewTargetPosition(AxisPositioningModes.AbsNormal,
52                                 afPositionen[nIndex],fZielfenster);
53
54             when axSpindel.IsInTargetWindow continue;
55
56             boBohrer := true;
57             delay(t#250ms);
58             boBohrer := false;
59             delay(t#200ms);
60          end_for;
61
62     end_loop;
63
```

Erweiterte Anwendungsbereiche

Das Programmbeispiel zeigt eine relativ einfache und häufige Applikation. Die heutigen Anforderungen jedoch gehen nicht nur einen, sondern mehrere Schritte weiter. Ganz offensichtlich ist für komplexere Anwendungen - wie höhere mathematische Operationen, Netzwerkhandling, strukturierte Ausnahmebehandlung oder String- und Dateioperationen - die klassische SPS-Programmierung eher ungeeignet. Der Befehlsumfang der Hochsprache JetSym STX dagegen bedient auch diese Operationen und Funktionen. Falls der Programmierer die objektorientierte Programmierung beherrscht, steht ihm diese in JetSym STX ebenfalls offen.

Fazit: Die grafische Programmierung zeigt ihre Vorteile allenfalls in der rein bitorientierten Verknüpfungslogik und ist für einfache Anwendungen mit Kleinsteuerungen ein taugliches Werkzeug. Steigen die Anforderungen, macht die Verwendung einer der klassischen grafischen SPS-Sprachen oft wenig Sinn. In solchen Fällen ist eine Hochsprache wie JetSym STX ein äußerst taugliches Mittel, um den Anforderungen gerecht zu werden. Mit dem Einsatz einer Hochsprache erhält der Anwender einen höheren Freiheitsgrad bei der Problemlösung als mit normierten SPS-Sprachen.

Die Rolle der Ausbildung

Geht eine Person eine Aufgabe an, wird sie immer versuchen, das Problem mit ihrem spezifischen Wissen zu lösen. Ein Mechaniker wird versuchen, die Lösung, wann immer es geht, mechanisch zu realisieren. Dasselbe gilt auch für den Einsatz der Steuerungstechnologie. An vielen Schulen für Auszubildende der Automation oder der Elektrik wird nach wie vor die klassische SPS-Programmierung gelehrt. Zum einen, weil die Marktpräsenz von klassischen SPS-Systemen höher ist, zum anderen, weil die Umsetzung von der elektrischen Schaltung zum Programmcode einfacher ist. An den Fachhochschulen und speziell an Universitäten erlernen die Studenten eine Hochsprache. In der Regel bevorzugt diese Gruppe eine Hochsprache wie JetSym STX.

Erfahrungen aus der Schulung für angehende JetSym-STX-Programmierer zeigen, dass Hochschulabgänger den Einstieg in diese Hochsprache sehr schnell schaffen, da für sie die Abstraktion einer Automatisierungsaufgabe auf diese Weise natürlich ist. Programmierer mit langjähriger SPS-Erfahrung tun sich in der Regel schwerer bei der Umsetzung in eine Hochsprache, da ihnen die prozessorientierte Denkweise nicht vertraut ist.

Fazit: Die Ausbildung des Anwenders spielt auch eine Rolle bei der Auswahl der Technologie. Solange kein wesentlicher Zwang zum Systemwechsel besteht, wird er die Aufgabe mit der ihm bekannten Programmiersprache lösen.

Unterschiedliche Lösungsansätze

In einem sind sich die Anwender einig: Maschinen und Anlagen werden auch in Zukunft noch komplexer. Daher die Anforderung an die Hersteller von Automatisierungssystemen, dass

gerade die entsprechenden Geräte und vor allem deren Tools einfacher und intuitiver bedienbar sind. Mit JetSym STX bietet die Jetter AG eine echte Alternative zu den klassischen SPS-Sprachen, aber auch einen komplett anderen Lösungsansatz. Um den Umsteigern von der SPS-Programmierung den Einstieg zu erleichtern, wurde JetSym STX bewusst auf der SPS-Quasihochsprache Structured Text aufgebaut. Die wenigsten SPS-Anwender programmieren allerdings nur in Structured Text sondern verwenden meistens die grafischen Sprachen oder die Anweisungsliste. Daneben gibt es noch eine Gruppe von Anwendern, die eine textbasierte Programmiersprache rundweg ablehnen.

Genauso wie für einen Elektriker ein Kontaktplan den richtigen Ansatz für die Lösung einer Automatisierungsaufgabe darstellt so fühlt sich ein Hochsprachenprogrammierer in JetSym STX wohler. Mit Sicherheit werden in Zukunft Hochsprachen in der Automatisierungstechnik an Bedeutung gewinnen, denn die Komplexität der zu realisierenden Applikationen macht dies unumgänglich. Die Jetter AG wird JetSym STX hinsichtlich Komfort und Einfachheit weiterentwickeln, um diesen Anforderungen des Marktes auch in Zukunft gerecht zu werden.

© 2013, Jetter AG, Germany

Der Autor

Andreas Leu
Telefon +49 7141 2550 - 466
Telefax +49 7141 2550 - 484
aleu@jetter.de

Jetter AG
Gräterstraße 2
71642 Ludwigsburg
Germany

www.jetter.de

Die Jetter AG

Seit 30 Jahren ist die Jetter AG im Bereich der Automatisierungstechnik tätig. Ihre Steuerungslösungen kommen im Maschinen- und Anlagenbau und der mobilen Automation (Arbeitsmaschinen und Nutzfahrzeuge) zum Einsatz.