Dieser Artikel erschien in leicht gekürzter Form unter dem Titel „Objekt Steuerung“ in der Zeitschrift 'Elektronik' Heft 8/2001. Die Veröffentlichung an dieser Stelle erfolgt mit freundlicher Genehmigung des WEKA Fachzeitschriften-Verlag GmbH.

Objektorientierte und ereignisgesteuerte Programmierung von Industrie-Computern

„Komponenten“ für die Fertigungsautomation

Der Einsatz von Industrie-Computern (IPC) für die Fertigungsautomation eröffnet eine Vielzahl neuer und effizienter Möglichkeiten. An erster Stelle steht hier die Anwendung von modernen objektorientierten und ereignisgesteuerten Software-Architekturen.
Ihre Anwendung ist jedoch auch mit einer Vielzahl von Problemen verbunden. Dieser Artikel zeigt diese auf und beschreibt eine Methodik, die es ermöglicht, echtzeitfähige 32-Bit Steuerungsprogramme direkt in einer modernen Hochsprache der 5. Generation zu entwickeln und auf einem IPC unter dem Betriebssystem DOS ausführen zu lassen.

von Immo Wache und Andrzej Jardzioch

Einführung

Gegenwärtig sind im Bereich der Fertigungsautomation vor allem Speicherprogrammierbare Steuerungen (SPS) und in geringerem Maße auch Industrie-Computer anzutreffen.
Während für IPC als Programmiersprachen vor allem Assembler und C Anwendung findet, dominieren bei den SPS sogenannte Fachsprachen, die in der IEC-61131-3 [1, 2] genormt sind.

Diese Sprachen stehen im Vergleich zum aktuellen Entwicklungsstand bei den universellen Programmiersprachen aus dem IT-Bereich deutlich zurück. Gegenwärtig lassen sich die universellen Programmiersprachen sinnvoll in 5 Generationen einteilen. Die Tabelle 1 gibt dazu einen Überblick. Diese Einteilung entspricht dabei nicht ganz der „klassischen“ Darstellung, wie sie Ende der 80‘er Jahre aufgestellt wurde, sondern versucht vielmehr, zwischenzeitliche Entwicklungen angemessen wiederzuspiegeln.

Bild 1: Laboraufbau mit Host-System, IPC und Feldbus zur Steuerung eines automatisierten Materialflusssystems

Tabelle 1: Generationen der Programmiersprachen in Analogie zu den SPS-Fachsprachen nach IEC 61131-3 [3]

Generationen und Vertreter von universellen Programmiersprachen	Merkmale	SPS – Fachsprachen (IEC 61131-3)
1. Generation Maschinensprache	numerisch codierte Befehle an die Verarbeitungseinheit • maschinenspezifisch	nur in der Anfangszeit von SPS verwendet, nicht mehr anzutreffen
2. Generation Assemblersprachen	Mnemonics zur Bezeichnung von Maschinenbefehlen • elementare math. u. log. Funktionen • Sprünge, Unterprogramme	IL - Instruction List (Anweisungsliste) LD - Ladder Diagram (Kontaktplan) FBD – Function Block Diagram (Funktionsplan)
3. Generation (1. Phase) prozedurale Sprachen z.B. Fortran, Basic	erste Hochsprachen • Verarbeitung von Zeichenketten, • höhere mathematische Funktionen	keine Entsprechung
3. Generation (2. Phase) strukturierte Sprachen z.B. Pascal, C	strukturierte Programmierung • Wiederholung, • Fallunterscheidung usw.	ST – Structured Text
4. Generation Objektorientierte Sprachen z.B. C++, Object Pascal	objektorientierte Programmierung • Datenkapselung, • Vererbung, • Polymorphismus	keine Entsprechung
5. Generation Komponentenbasierte (visuelle) Sprachen z.B. Delphi, Java, C++Builder (Visual Basic)	visuelle Programmierung • Komponenten • Ereignisse • Two Way Tools (RAD)	keine Entsprechung

Es zeigt sich, dass die marktführenden SPS-Compilerbauer bei der Sprachentwicklung in den letzten Jahrzehnten recht konservativ vorgegangen sind. Viel Zeit ging verloren, um gemeinsame Sprachnormen zu finden. Damit bleibt bisher die Nutzung moderner Software-Entwicklungsmethoden wie die objektorientierte Programmierung [4], UML [5] oder gar die Verwendung von Design-Patterns [6] für den Bereich der Fertigungsautomation verschlossen. Aus diesem Grund soll hier der Versuch unternommen werden von der SPS-Fachsprachen hin zur Anwendung einer modernen objektorientierten-Programmiersprache für die Programmierung von IPC für die Fertigungsautomation zu gelangen.

Eine Methodik zur objektorientierte Programmierung von IPC

Die objektorientierte Programmierung für Steuerungen in der Fertigungsautomation ist zum gegenwärtigen Zeitpunkt mit einer Reihe von Problemen verbunden, für die geeignete Lösungen gesucht werden müssen. Vor allem sind dies folgende Punkte:

- Gewährleistung einer Echtzeitverarbeitung in Verbindung mit einem dazu passenden Betriebssystem,
- Anpassung eines Programmiersprache der 5. Generation an das gewählte Betriebssystem,
- Realisierung einer industrietauglichen Verbindung des IPC mit den Ein- und Ausgangssignalen des zu steuernden Prozesses (der gesteuerten technischen Anlage),
- Anwendung des objektorientierten und ereignisgesteuerten Paradigmas auf steuerungstechnische Software,

- Gewährleistung der charakteristischen zyklischen SPS-Programmabarbeitung (sequentielle Eingabe, Verarbeitung und Ausgabe) innerhalb eines ereignisorientierten Steuerungsprogramms.

Die nachfolgenden Erläuterungen stellen eine praktisch anwendbare Methodik zur Lösung der dargestellten Probleme vor. Die Lösung basiert auf der Verwendung der Programmiersprache Delphi in der aktuellen Version 5 zur Programmentwicklung in einer integrierten Entwicklungsumgebung (IDE) auf einem Host-PC unter Windows und der anschließenden Programmübertragung auf einen IPC mit dem Betriebssystem DOS. Bild 2 stellt das dazu verwendete Entwicklungssystem schematisch dar.

Besondere Vorteile der hier vorgestellten Methodik liegen in der universellen Wiederverwendung und Erweiterung von einzelnen Software-Bausteine (den sogenannten Komponenten), die eine einfache und schnelle Anpassung an unterschiedlichste zu automatisierende Anlagen ermöglichen.

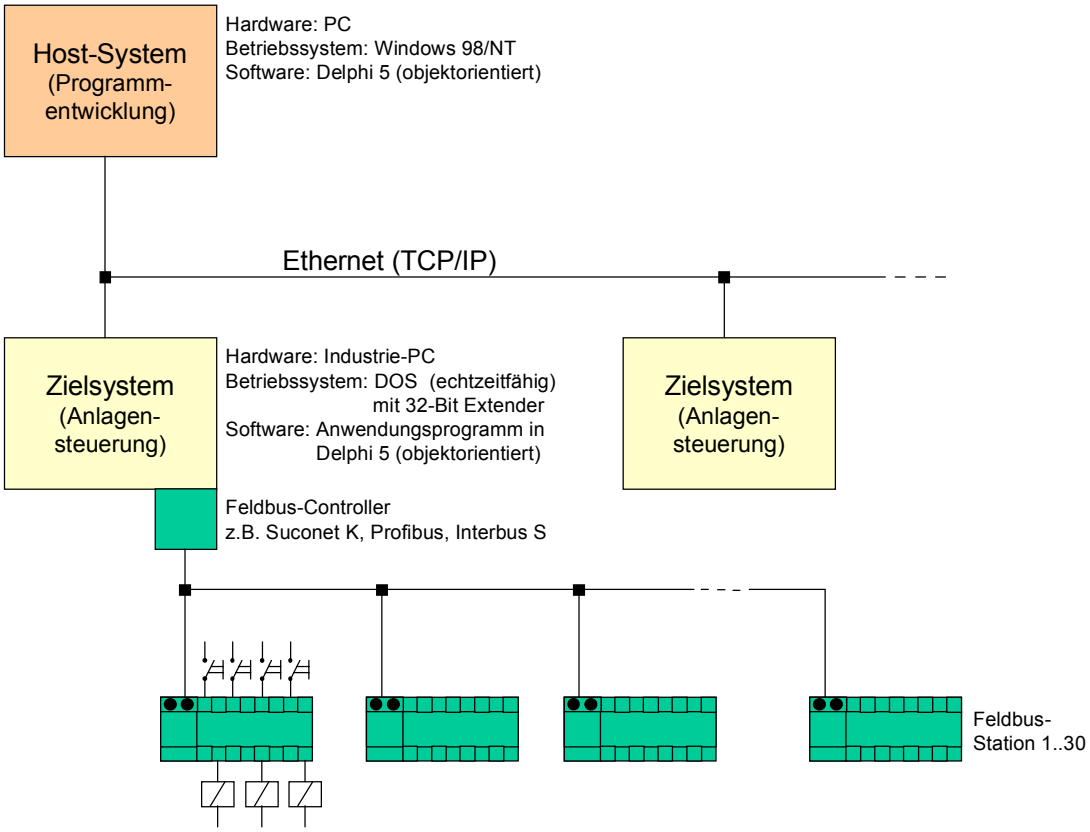

Bild 2: Entwicklungssystem zur Programmentwicklung von objektorientierten IPC-Steuerungsprogrammen für die Fertigungsautomation

Auf der IPC-Seite wird ein 32-Bit DOS-Extender verwendet, der die Ablauffähigkeit von 32-Bit Programmen in einem flachen Speichermodell auch unter dem Betriebssystem DOS ermöglicht. Weiterhin wird die standardmäßig vorhandene Echtzeituhr (Real-Time-Clock) des IPC zur Generierung von Hardware-Interrupts eingesetzt, um eine konstante Zykluszeit im Bereich von ca. 10ms zu ermöglichen, wie sie als Anforderung in vielen industriellen Anwendungsfällen zu erfüllen ist. Im Gegensatz zur konventionellen Vorgehensweise erfolgt die Programmierung der Steuerungssoftware nicht mehr in einer SPS-Fachsprache nach IEC61131-3 sondern direkt in einer objektorientierten Hochsprache.

Die Kopplung zwischen dem Host zur Programmentwicklung und dem IPC als Zielsystem wird durch ein gewöhnliches Ethernet-Netzwerk unter dem Protokoll TCP/IP verwirklicht. Zur Anbindung der zu steuernden Anlage an den IPC kommt vorzugsweise ein Feldbus zur Anwendung. Die einzelnen Feldbusstationen mit den entsprechenden Ein- und Ausgängen können bei modular aufgebauten, und räumlich weit verteilten Anlagen (wie z.B. bei Materialflusssystemen üblich) den Anlagenteilen serienmäßig zugeordnet werden. Hierfür sind unterschiedliche Feldbus-Standards nutzbar (z.B. Profibus, Interbus-S, Sukonet-K1 o.ä.). Das hier vorgeschlagene Steuerungskonzept sieht die Verwendung des Feldbus-Standards Sukonet-K1 der Fa. Moeller vor [7], da hierzu bereits umfangreiche Erfahrungen und entsprechende Software-Module vorlagen. Vorteil dieser Feldbus-Variante ist die Möglichkeit der einfachen Programmierung des Feldbus-Controllers unter dem Betriebssystem DOS. Eine Anpassung des Steuerungssystems auf andere Feldbus-Standards steht dabei jedoch nichts im Wege, da sie konzeptionell bereits berücksichtigt wurde.

Die Anwendung von IPC in der Fertigungsautomation

Für Steuerungsaufgaben der Fertigungsautomation ist die Echtzeitverarbeitung der verwenden Steuerungshardware von grundlegender Bedeutung. Unter Echtzeit soll hier (entsprechend der Definition aus [8]) die sichere Einhaltung einer Antwortzeit der Steuerung auf eine Anforderung aus dem gesteuerten Prozess verstanden werden. Die Einhaltung des Echtzeitverhaltens wird dabei sowohl durch die Gestaltung des eigentlichen Steuerungsprogramms aber auch grundlegend durch die Wahl des verwendeten Betriebssystems bestimmt. Als Betriebssysteme für IPC werden gegenwärtig vor allem die Betriebssysteme Windows, Linux und DOS eingesetzt. Aufgrund der internen Realisierung der Multitasking-Eigenschaften von Windows und Linux müssen diese Betriebssysteme als nicht echtzeitfähig im oben genannten Sinne eingestuft werden. Unter DOS hingegen wird nach einem Programmstart die Kontrolle über den IPC praktisch vollständig an das ausgeführte Programm übergeben. Wenn dieses bestimmte Regeln einhält (z.B. keine Zugriffe auf das Dateisystem, keine Ein- und Ausgaben an die Standard-Console etc.), so kann ein Steuerungsprogramm tatsächlich harte Echtzeit-Kriterien erfüllen. Aus diesem Grund wurde für die hier vorgestellte Lösung DOS als kostengünstig verfügbares Betriebssystem (z.B. durch FreeDOS, ein Freeware-Programm [9]) ausgewählt.

Das Hauptproblem bei der Verwendung von DOS als Betriebssystem ist jedoch die Verfügbarkeit von Programmiersprachen der 5. Generation, die auch in der Lage sind, entsprechend geeignete Programme zu erzeugen. Praktisch alle gegenwärtig verfügbaren Programmiersprachen der 5. Generation sind auf die Erzeugung von Programmen ausgerichtet, die ausschließlich unter dem Betriebssystem Windows lauffähig sind. Glücklicherweise wurde diese Lücke erkannt und es existiert z.B. der 32-Bit DOS-Extender WDosX [10] (ein weiteres Freeware-Programm), der durch die Emulation von bestimmten Windows-Kernel-Bibliotheken die Möglichkeit eröffnet, auf Windows ausgerichtete Programme auch unter DOS ausführen zu können. Um nun das Programmiersystem Delphi 5 für die Verwendung von WDosX anzupassen, sind einige Schritte erforderlich, die detailliert unter [11] beschrieben sind. Damit ist die Grundlage für die Realisierung von Komponenten gegeben, um eine einfache Erstellung von Steuerungsprogrammen für die Fertigungsautomation zu ermöglichen.

Eine Software-SPS als Delphi Komponente

Zentrale Element der hier vorgestellten Steuerungsprogramme mit Delphi und WDosX stellt die Komponente *TwdxPlc* dar. Sie gewährleistet die typische, zyklische Arbeitsweise einer SPS in den Teilschritten

- Eingabe der Prozesseingänge in das Prozessabbild,

- Verarbeitung der Signale im Prozessabbild und
- Ausgabe der Verknüpfungsergebnisse aus dem Prozessabbild an die Ausgänge.

Jeder SPS-Zyklus wird dafür durch ein Timer-Ereignis von einer Zeitgeber-Komponente angestoßen. Der eigentliche Zugriff auf die Prozessein- und –ausgänge erfolgt über spezielle Komponenten, die für das jeweils verwendete Feldbussystem ausgewählt werden und mit der Komponente *TwdxPlc* verbunden sind. Die Verarbeitung des Prozessabbildes erfolgt entweder in speziellen Ereignissen *TwdxPlc.AfterInput* bzw. *TwdxPlc.BeforeOutput* oder übersichtlicher mittels universeller oder spezialisierter Petri-Netz-Komponenten. Diese werden ebenfalls mit der Komponente *TwdxPlc* verbunden (siehe dazu Bild 3).

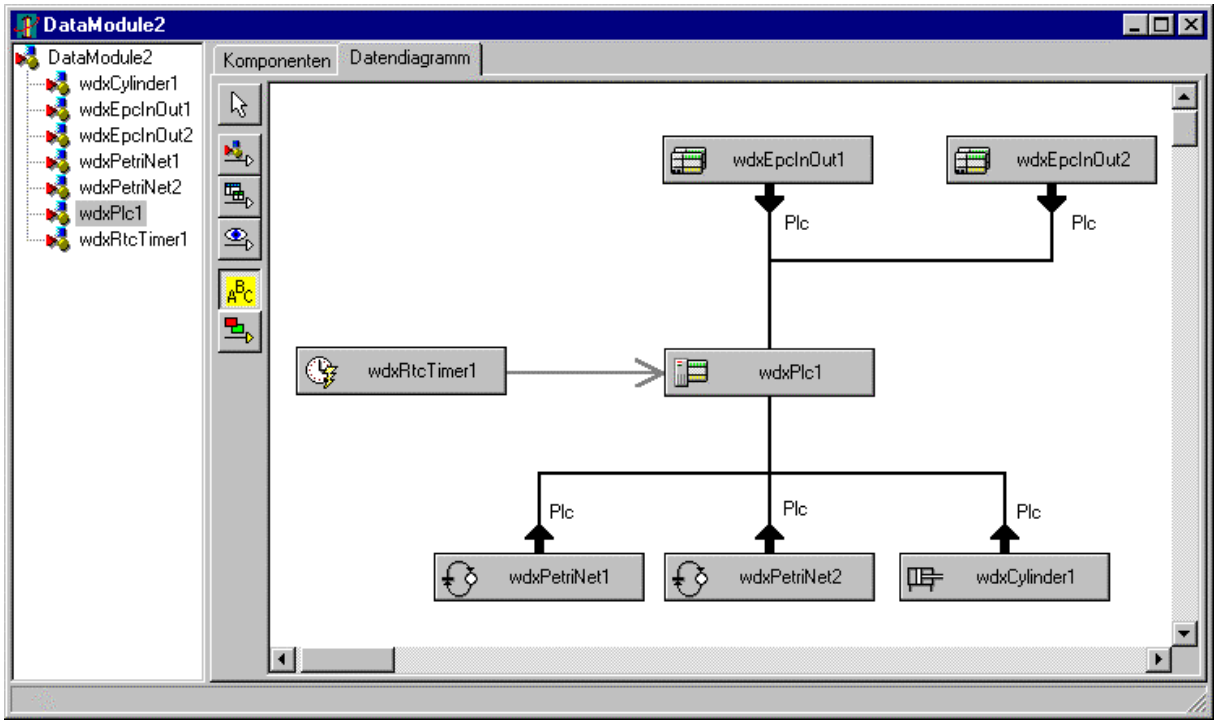

Bild 3: Die Software-SPS aus Delphi Komponenten

Der Zugriff auf das interne Prozessabbild der Komponente *TwdxPlc* innerhalb des Verarbeitungszyklus ist mit unterschiedlichen Datenformaten (Bit, Byte) über eine Vielzahl von Eigenschaften möglich. Listing 1 zeigt dazu einige Möglichkeiten:

```
procedure TForm1.wdxPlc1AfterInput(Sender: TObject);
begin
  with wdxPlc1 do
  begin
    { Zugriff auf Ein- und Ausgangs-Bits }
    Bit[A, 0, 0, 0] := True;
    BusBit[A, 1, 2, 1, 7] := Bit[NE, 0, 0, 2];

    { Zugriff auf Ein- und Ausgangs-Bytes }
    Byt[NA, 0, 1] := BusByt[E, 1, 2, 0];
  end;
end;
```

Listing 1: Zugriff auf Bits und Bytes von zentralen und busorientierten Ein-/Ausgabeeinrichtungen im Prozessabbild von *TwdxPlc*

Bild 4 verdeutlicht den Adressaufbau in Form von Beispielen für zentrale und busorientierte Ein-/Ausgabeeinrichtungen. Mit Ausnahme der logischen Interpretation des Signals erfolgt die Trennung der einzelnen Adressbestandteile mit einem Punkt gemäß IEC61131-3.

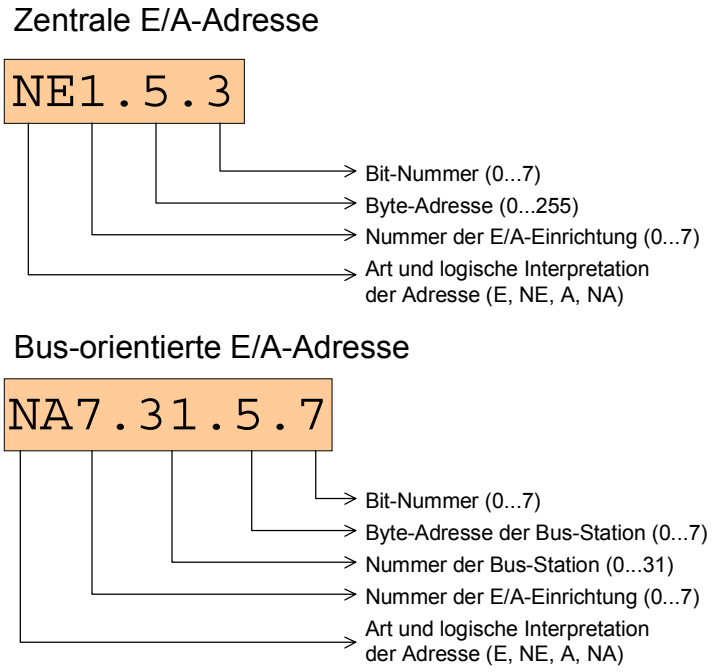

Bild 4: Der Adressaufbau für den Zugriff auf das Prozessabbild

Weiterhin kann ein spezieller Datentyp *TBitAddr* zum Zugriff auf einzelne Bits des Prozessabbildes über eine Array-Eigenschaft *TwdxPlc.IO[]* verwendet werden. Dadurch können die zugehörigen Ein- und Ausgänge von Petri-Netze als deren Eigenschaften direkt über einen Komponenteneditor (in der Delphi-IDE als Objektinspektor bezeichnet) eingegeben werden.

Die Komponente *TwdxPlc* bildet die möglichen Betriebszustände (Init, Halt, Run und Error) einer konventionellen SPS nach. Zur Abfrage und Beeinflussung des Betriebszustandes stehen Methoden und Eigenschaften zur Verfügung. Der Zustandsgraph (Bild 5) beschreibt die möglichen Zustände (Eigenschaft *Mode*) und Übergangsbedingungen (Methoden).

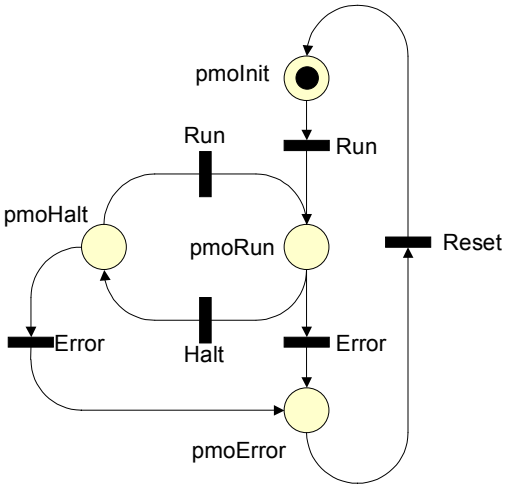

Bild 5: Der Zustandsgraph von *TwdxPlc*

Realisierung einfacher Steuerungsaufgaben

Die bisher beschriebene Komponente *TwdxPlc* bildet gewissermaßen die Infrastruktur jeder Anlagensteuerung. Was noch fehlt ist die Abbildung der eigentlichen Steuerungsaufgabe, d.h. die Verknüpfung der Eingangssignale sowie innerer Zustandssignale der Anlagensteuerung zu entsprechenden Ausgangssignalen. Grundsätzlich lassen sich einzelne Steuerungsaufgaben in kombinatorische und sequentielle Steuerungen einteilen. Praktische Steuerungen bilden im allgemeinen eine Kombination beider Steuerungsarten.

Für die Realisierung von boolschen Verknüpfungen stehen in der hier verwendeten Programmiersprache Object-Pascal boolesche Variablen zur Speicherung von binären Signalen sowie unäre (die NEGATION) und binäre Operatoren (die UND-Verknüpfung, die ODER-Verknüpfung und EXCLUSIV-ODER-Verknüpfung) zu deren Verknüpfung bereit. Bild 6 zeigt die Darstellung einer einfachen Steuerungsaufgaben als Kontaktplan (LD) und als Funktionsplan (FBD) nach IEC61131-3.

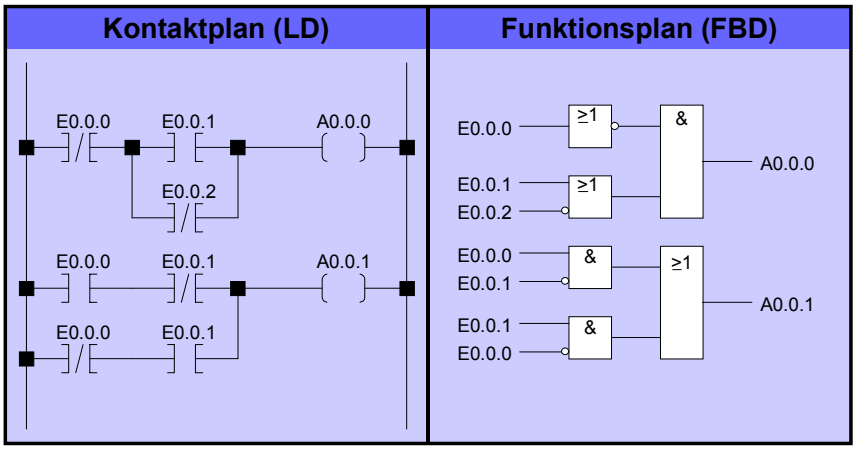

Bild 6: kombinatorische Steuerungsaufgaben als LD und FBD

Für die Implementation einfacher Steuerungsfuntionen bietet sich die Verwendung der Ereignisse *TwdxPlc.AfterInput* bzw. *TwdxPlc.BeforeOutput* an. Hier wurde das Ereignis *AfterInput* gewählt. Die Umsetzung der kombinatorischen Steuerungsaufgaben aus Bild 6 zeigt das Listing 2:

```
procedure TForm1.wdxPlc1AfterInput(Sender: TObject);
begin
  with wdxPlc1 do
  begin
    { erste Verknüpfung }
    Bit[A, 0, 0, 0] := not Bit[E, 0, 0, 0]
        and (Bit[E, 0, 0, 1] or not Bit[E, 0, 0, 2]);
    { zweite Verknüpfung }
    Bit[A, 0, 0, 1] := Bit[E, 0, 0, 0] xor Bit[E, 0, 0, 1];
  end;
end;
```

Listing 2: Realisierung von kombinatorischen Steuerungsaufgaben in Delphi

Einfache sequentielle Steuerungsfunktionen lassen sich analog abbilden. Auf eine detaillierte Darstellung wurde im Rahmen dieses Artikels verzichtet.

Zustandsautomaten als Petri-Netz-Komponenten

Die bisher gezeigten Methode zur Programmierung von kombinatorischen und sequenziellen Steuerungsfunktionen besitzt noch nicht das gewünschte Potential einer effektiven Modularisierung und Wiederverwendung von Software. Hier zeigt sich, dass die Verwendung der objektorientierte Programmierung (mit den Merkmalen Datenkapselung, Vererbung und Polymorphismus) für die Realisierung von Steuerungsaufgaben sehr gut geeignet ist. Grundsätzlich muss das Steuerungsprogramm in der Anlagensteuerung das Verhalten des gesteuerten Prozesses im Rahmen der erforderlichen Genauigkeit abbilden. Es spricht daher nichts dagegen, den Systemaufbau der gesteuerten Anlage mit dem charakteristischen Eigenschaften und Verhaltensmerkmalen ihrer Elemente und Relationen entsprechend abzubilden. Genau dafür bietet sich eine Gliederung des Steuerungsprogramms in Klassen und deren Instanzen an, die miteinander in Kommunikation treten können und damit den Steuerungsprozess verwirklichen.

Alle Klassen, die Bestandteil der Steuerungsstruktur sind, müssen in der Lage sein, Eingangssignale sowie Ereignisse (von anderen Komponenten) in Verbindung mit ihrem momentanen Zustand zu verknüpfen und daraus die entsprechenden Ausgangssignale zu bilden. Zur Verwirklichung dieses Verhaltens (auch als Zustandsautomat bezeichnet) bietet sich die Modellierung auf Basis von Petri-Netzen an [12]. Das grundlegende Verhalten aller zu verwendenden Petri-Netze muss dabei nicht in jeder Klasse des Steuerungsprogramms neu implementiert werden, sondern wird aus einer abstrakten Basisklasse geerbt.

Zentrales Merkmal eines Petri-Netzes ist sein innerer Zustand. Dieser gibt an, welcher Platz innerhalb des Netzes gerade markiert (aktiv) ist. Dieser innere Zustand wird in der Basisklasse durch einen Zahlenwert symbolisiert. Nach außen hin kann der Zustand über die Eigenschaft *State* abgefragt werden. Eine Änderung des Zustandes ist ausschließlich über eine spezielle Methode möglich.

Zur konkreten Umsetzung von Petri-Netzen müssen entsprechend abgeleitete Klassen implementiert werden. Eine solche ist die universell verwendbare Komponente *TwdxPetriNet*, die alle abstrakten Methoden der Basisklasse implementiert und diese auf die Ereignisse *OnStateCtrl*, *OnBeginState*, *OnAtState* und *OnEndState* abbildet, welche vom Anwender auf einfache Weise anwendungsspezifisch untersetzt werden können. Darüber hinaus besitzt *TwdxPetriNet* Eigenschaften zur einfachen Zuordnung von 8 Eingängen und 16 Ausgängen zum gesteuerten Prozess (über die Eigenschaften *Inp01...Inp16* und *Out01...Out08).* Bild 7 zeigt, wie eine Komponente *TwdxPetriNet* im Formulareditor eingefügt wird, sowie deren Eigenschaften und Ereignisse.

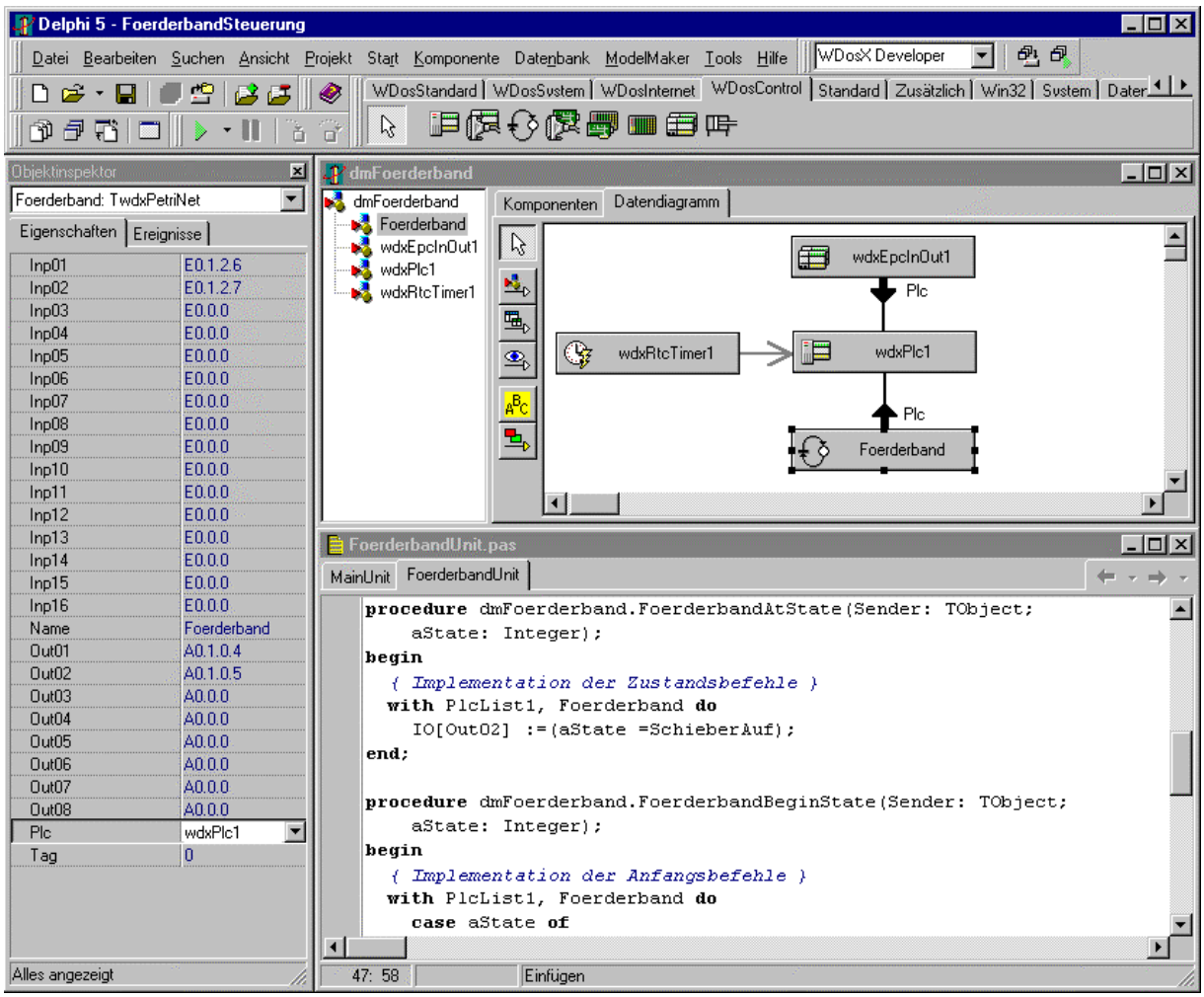

Bild 7: Die Komponente *TwdxPetriNet* (Foerderband) mit ihren Eigenschaften und Ereignissen in der Delphi-IDE

Wie aus der Beschreibung der Eigenschaft *State* deutlich wurde, können mittels *TwdxPetriNet* nur Petri-Netze modelliert werden, bei denen zu einem gegebenen Zeitpunkt genau ein Platz markiert ist. Petri-Netze mit nebenläufigem bzw. parallelen Vorgängen sind damit nicht darstellbar. Diese Einschränkung hat den Vorteil, dass ein unkontrolliertes Verschwinden oder Vermehren von Marken wie es durch Programmierfehler auftreten könnte, prinzipiell verhindert wird. Derartige Verhaltensweisen von Petri-Netzen lassen sich nur durch aufwendiges Debugging herausfinden und beseitigen (nebenläufige Prozesse können hingegen durch Aufteilung auf mehrere Petri-Netze realisiert werden, die über Ereignisse und Methoden miteinander kommunizierten).

Folgende Beispiel (nach einer Idee aus[13]) soll die Anwendung der Komponente *TwdxPetriNet* näher erläutern.

Ein Förderband transportiert Füllgut aus einem Bunker in einen Transportwagen. Beim Einschalten benötigt der Bandmotor eine Anlaufzeit von 5 sec. Anschließend wird der Schieber Y geöffnet. Beim Abschalten des Förderbands wird der Schieber geschlossen und das Bandmotor läuft noch 10 sec. nach, um das Band leerlaufen zu lassen.

Das entsprechende technologische Schema mit dem mechanischen Aufbau und den daraus resultierenden technischen Zusammenhängen zeigt Bild 8. Zur Beschreibung der ablaufenden Prozesse sind steuerungstechnisch interpretierte Petri-Netze gut geeignet.

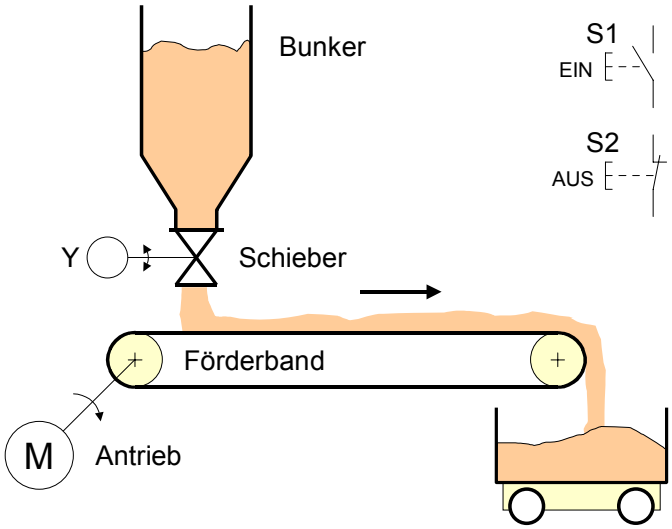

Bild 8: Technologisches Schema zum Beispiel „Förderband“

Das entsprechende Petri-Netz zur Beschreibung des Ablaufs ist in Bild 9 dargestellt.

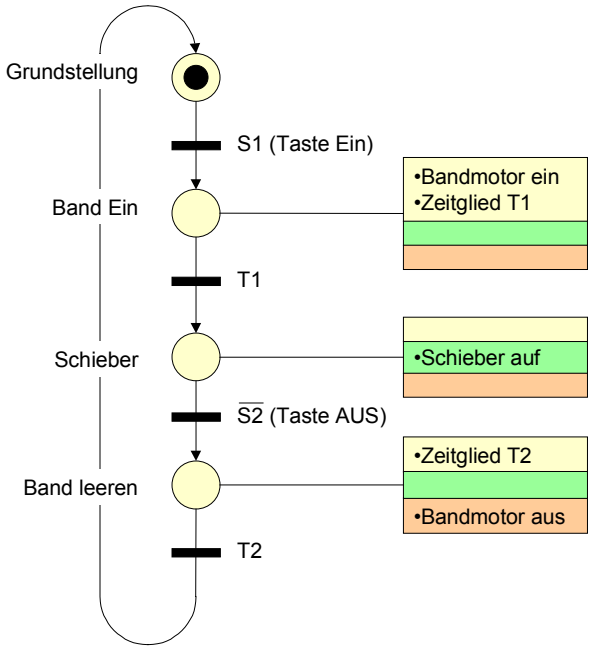

Bild 9: Zeitbewertetes Petri-Netz zum Beispiel „Förderband“

Jedem Platz im steuerungstechnisch interpretierten Petri-Netz werden in Anlehnung an den Funktionsplan nach DIN 40719 Teil 6 die von ihm beeinflussten Befehle bzw. Ausgangssignale zugeordnet. Gegenüber der DIN-Norm wird hier jedoch eine Dreiteilung der Befehle vorgenommen. Im oberen Abschnitt werden die Befehle aufgeführt, die bei der Markierung des Platzes genau einmal ausgeführt werden. Der mittlere Abschnitt beschreibt Befehle, die so lange aktiv sind, wie auch der zugeordnete Platz markiert ist. Beim Verlassen der Marke aktiviert der Platz den dritten, unteren Abschnitt, dessen Befehle analog zum ersten Abschnitt genau einmal ausgeführt werden. Diese Dreiteilung der Befehle wird über die Ereignisse *OnBeginState*, *OnAtState* und *OnEndState* der Komponente *TwdxPetriNet* implementiert. Die Umsetzung der Transitionen erfolgt über das Ereignis *OnStateCtrl*.

Im Listing 3 ist beispielhaft die programmtechnische Umsetzung der Ereignisse *OnStateCtrl*, *OnBeginState*, und *OnAtState* wiedergegeben. Das vollständige Programm ist unter der am Ende des Artikels angegebenen Internet-Adresse verfügbar [14].

Die Umsetzung der Transitionen in der Methode *FoerderbandStateCtrl* erfolgt auf effektive Weise mit Hilfe einer Fallauswahl. Abhängig vom Zustand der Variablen *aState*, die den bisher markierten Platz im Petri-Netz beschreibt, wird nur der entsprechend zugeordnete Fall bearbeitet. Innerhalb des Falls werden die Eingangssignale der zugehörigen Transition(en) ausgewertet und im Falle des Feuerns der Transition die Variable *aState* mit dem entsprechenden Nachfolgeplatz belegt. Die Implementierung der Befehle in den Methoden *FoerderbandBeginState* (und *FoerderbandEndState*) erfolgt analog über eine entsprechende Fallauswahl. Eine Ausnahme bildet die Methode *FoerderbandAtState*. Hier muss eine direkte Auswertung der Aktivität eines bestimmten Platzes im Petri-Netz erfolgen und das logische Ergebnis dem zugeordneten Zustandsbefehl zugewiesen werden.

```
procedure TdmFoerderband.FoerderbandStateCtrl(Sender: TObject;
  var aState: Integer);
begin
  { Implementation der Transitionen }
  with wdxPlc1, Foerderband do
    case aState of
      Grundstellung:
        if Io[Inp01] then aState := BandEin;
      BandEin:
        if TimeOut(T1) then aState := SchieberAuf;
      SchieberAuf:
        if not Io[Inp02] then aState := BandLeeren;
      BandLeeren:
        if TimeOut(T2) then aState := Grundstellung;
    end;
end;

procedure TdmFoerderband.FoerderbandBeginState(Sender: TObject;
  aState: Integer);
begin
  { Implementation der Anfangsbefehle }
  with wdxPlc1, Foerderband do
    case aState of
      BandEin:
      begin
        Io[Out01] := True;
        T1 := StartTimer(5000);
      end;
      BandLeeren:
        T2 := StartTimer(10000);
    end;
end;

procedure TdmFoerderband.FoerderbandAtState(Sender: TObject;
  aState: Integer);
begin
  { Implementation der Zustandsbefehle }
  with wdxPlc1, Foerderband do
    Io[Out02] := (aState = SchieberAuf);
end;
```

Listing 3: Implementierung des Petri-Netzes zur Förderbandsteuerung

Zusammenfassung

Dieser Artikel beschreibt eine neuartige Methodik zur Realisierung von objektorientierten, komponentenbasierten und ereignisgesteuerten Programmen für IPC in der Fertigungsautomation. Er zeigt, dass die damit verbundenen Probleme hinsichtlich des Echtzeitverhaltens unter dem Betriebssystem DOS in Verbindung mit dem Programmiersystem Delphi in der aktuellen Version 5 lösbar sind. Das Komponenten-Konzept von Delphi hat sich als praxistaugliche Basis für die Modellierung einer objektorientierten Software-SPS erwiesen. Die Programmierung von kombinatorischen und sequenziellen Steuerungsfunktionen ist unter Verwendung leistungsfähiger Prozessoren direkt in einer Hochsprache möglich. Eine Modularisierung des Steuerungsprogramms kann durch Verwendung von speziellen Petri-Netz-Komponenten erreicht werden, wodurch die Vorzüge der objektorientierten Programmierung zur Geltung kommen. Die praktische Anwendbarkeit der Methodik wurde anhand einer Materialflusssteuerung für eine Versuchsanlage eines automatischen Palettentransportsystems nachgewiesen [15]. Gegenwärtig befindet sich eine Anpassung dieser Materialflusssteuerung für das Hängefördersystem „Conway 3000“ der Fa. Henke Maschinenbau [16] in der Entwicklung. Weiterhin erfolgt z.Z. eine Anpassung der WDosX-Delphi-Bibliotheken an den Borland C++Builder 5, um entsprechende IPC-Steuerungsprogramme auch direkt in C++ entwickeln zu können.

Literatur

[1] *John, K.-H.; Tiegelkamp, M.:* SPS-Programmierung mit IEC 1131-3. Springer Verlag, 1997.

[2] *Wellers, H.:* SPS-Programmierung nach IEC 1131-3: Normierung, Beispiele, Übungen. Cornelson Girandet Verlag, 1996.

[3] *Wache, I.:* Object oriented modelling of material flow controls. Jornal of Material Processing Technology. Vol. 76 No. 1-3. April 1998.

[4] *Booch, G.:* Objektorientierte Analyse und Design. Mit praktischen Anwendungsbeispielen. Addison-Wesley Verlag, 1995.

[5] *Douglas B. P.:* Real-Time UML. Addison-Wesley Verlag, 1998.

[6] *Gamma E., Helm R., Jonson, R. E.:* Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Addison-Wesley Verlag, 1996.

[7] *Firmenschrift*: EPC335.1 Businterface für PC, Moeller GmbH Bonn, 1996.

[8] *Furrer, F.:* Ethernet-TCP/IP für die Industrieautomation. Hüthig Verlag, 2. Auflage, 2000.

[9] The FreeDOS Project. http://www.freedos.org/

[10] WDosX. http://michael.tippach.bei.t-online.de/wdosx/

[11] Delphi WDosX Page (DWP). http://www.dwp42.org/

[12] *Jardzioch A., Honczarenko J.:* Machine systems control with help of Petri-net method. Proceeding of „International Conference CIM'99" Zakopane 1999". WNT Warszawa 1999, Vol. 1, pp.196-205.

[13] *Piotrowski, A., Maier, H.:* Messen, Steuern, Regeln mit IBM-Kompatiblen PCs. Loseblattausgabe, Interest Verlag, 1998.

[14] Beispielprogramm zum Artikel. http://www.dwp42.org/pub/public/foerderband.zip

[15] *Wache, I.:* Das Materialflußsystem ‚Kugelpalette' für die flexible Fertigung. Wissenschaftliche Zeitschrift des Instituts für Fabriksysteme und Logistik. Universität Rostock 1997.

[16] *Firmenschrift:* Hängefördersystem „Conway 3000". Henke Maschinenbau GmbH Lübbecke-Alswede, 2000.

Was sind Komponenten?

Komponenten bilden zunächst eine Form von Klassen, wie sie aus der objektorientierten Programmierung (OOP) bekannt sind. Sie besitzen jedoch zusätzliche Merkmale, die sie zu einem Schlüsselelement von visuellen Programmiersprachen machen. Diese sind:

- Persistenz
- Laufzeittypinformationen (RTTI) und
- Methodenzeiger (Ereignisse).

Komponenten besitzen eine besondere Elementkategorie (zusätzlich zur privaten, geschützten und öffentlichen Kategorie), die als veröffentlicht (published) bezeichnet wird und für welche Laufzeittypinformationen verfügbar sind. Damit können Komponenten bereits zur Entwurfszeit eines Programms instanziiert und parametriert werden. Durch die Persistenz der Komponenten können diese parametrierten Komponenten über ein Streaming-System als Bestandteil des compilierten Programms gespeichert und beim späteren Programmstart mit Hilfe der RTTI wieder aufgebaut werden.
Methodenzeiger stellen veröffentlichte Eigenschaften dar, in denen die Adressen von Methoden anderer Objekte gespeichert werden können. Über sie wird die Ereignissteuerung bzw. der Botschaftenaustausch in komponentenbasierten Programmen verwirklicht.

Das „Geheimnis“ der Ereignisse

Ereignisse stellen einen Mechanismus dar, der es unabhängig voneinander entwickelten Komponenten ermöglicht, miteinander zu kommunizieren. Damit dieses Ziel erreicht werden kann, müssen die Komponenten in einem ‚Container‘-Objekt (dem Mediator-Pattern) eingefügt werden. Die Methoden dieses Mediators bilden den ‚Klebstoff‘ zur Verbindung der enthaltenen Komponenten. Dazu folgendes Beispiel (vergl. Bild A): Der *Mediator1* enthält eine Komponente *Switch1*, die Ereignisse beim Wechsel eines Eingangssignals auf *True* oder *False* auslöst. Diese sind mit Methoden von *Mediator1* verknüpft, welche ihrerseits die Methoden der Komponente *Cylinder1* aufrufen. Im Beispiel wird damit der Zylinder geöffnet, sobald der Schalter eingeschaltet wird.

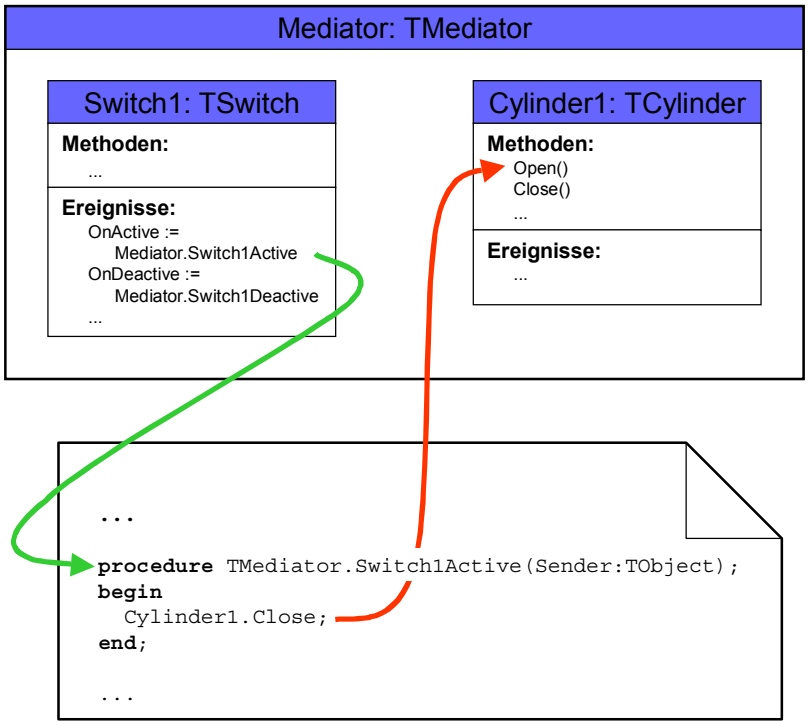

Bild A: Pneumatikzylindersteuerung mittels Ereignissen

Dipl-Ing. Immo Wache
arbeitet gegenwärtig als Software-Entwickler am Informationstechnologie-Zentrum Rostock. Er wurde 1963 in Berlin geboren und studierte von 1985 bis 1989 Technologie des Maschinenbaus an der TH Wismar. Bis 1998 war er wissenschaftlicher Mitarbeiter am Institut für Fabriksysteme und Logistik an der Universität Rostock. Dort befasst er sich im Bereich der Produktionsautomatisierung mit der Entwicklung von PC-basierten Steuerungssystemen für automatische Transportsysteme, um auf dem Gebiet zu promovieren.

E-Mail: immo.wache@gmx.net

Dr.-Ing. Andrzej Jardzioch
ist Ph.D. am Fachbereich Automatisierte Fertigungssysteme des Institutes für Fertigungstechnik der TU Szczecin. Er wurde 1959 geboren, studierte von 1979 bis 1984 in der Fachrichtung Maschinentechnologie an der TU Szczecin. Von 1985 bis 1995 war er wissenschaftlicher Assistent am Institut für Fertigungstechnik der TU Szczecin. Herr Dr. Jardzioch ist z.Z. Leiter des von der polnischen Akademie der Wissenschaften finanzierten Projektes "Anwendung der Künstlichen Intelligenz zur Planung und Steuerung von Fertigungssystemen". Die Ergebnisse dieses Projektes bilden die Grundlage für seine Habilitation.

E-Mail: jard@safona.ps.pl