Steuerungsprogramme 4.0
Struktur und Aufbau von Steuerungsprogrammen für Industrie 4.0

Control Programs 4.0
Structure and Format of Control Programs in the Context of Industry 4.0

Gerd Kainz, **Nadine Keddis**, Dr. **Alois Zoitl**,
Dr. **Christian Buckl**, fortiss GmbH, München;
Dr.-Ing. **Dirk Pensky**, Festo Didactic GmbH & Co. KG, Denkendorf;
Bernd Kärcher, Festo AG & Co. KG, Esslingen/Berkheim

Kurzfassung

Um künftigen Anforderungen an die Produktion gerecht zu werden, ist es erforderlich, Wandlungsfähigkeit in der Produktion und im speziellen im Steuerungskonzept umzusetzen. Ausgangsbasis dafür bilden die Strukturen der Steuerungsprogramme.

Zur Unterstützung von Wandlungsfähigkeit ist es notwendig, die Steuerungsprogramme zuerst zu modularisieren, indem man die Algorithmen für die Ansteuerung der Sensoren und Aktoren in wiederverwendbare Module mit höherer Funktionalität kapselt, wie z.B. Modul *Zylinder* mit *fahreAufPosition()*, Modul *Sensor* mit *messeWert()*.

Aufbauend auf diesen Modulen wurde eine Rezeptfahrweise umgesetzt, die es ermöglicht Steuerungsprogramme während der Laufzeit zu tauschen. Da nicht die gesamten Steuerungsprogramme während der Laufzeit getauscht werden, sondern lediglich übergeordnete Rezepte eingespielt werden, wird sichergestellt, dass es zu keinen Schäden an der Anlage kommen kann.

Zusätzlich zu diesem Themenbereich gibt der Beitrag einen Einblick in weitere Ergebnisse des Forschungsprojekts „AutoPnP – Plug & Play für Automatisierungssysteme“, welches für die Ableitung der Anforderungen an die neue Steuerungsarchitektur herangezogen wurde.

Abstract

Upcoming requirements on the production of tomorrow demand for adaptability of the production and especially of the control architecture in use.

The structure of the underlying control programs serves as the starting point for increasing adaptability. To support adaptability it is required to first modularize the control programs.

Therefore, the algorithms used to interact with connected sensors and actuators need to be divided into separate modules encapsulating all the necessary code needed to interact with the attached devices. Those modules can offer higher level functions for the usage of devices, e.g., module *Cylinder* with *goToPosition()*, module *Sensor* with *getMeasurementValue()*.
On top of the modules for sensors and actuators a recipe-based control strategy has been realized. The recipe-based control strategy allows the adaptation of control programs during run-time. As only the recipes are exchanged during run-time, it can still be ensured that the controlled system does not get damaged while reprogramming the respective parts. All the locking mechanisms are active all the time preventing any unintended misbehavior of the plant.
In addition to the field of control programs, further results of the research project “AutoPnP – Plug & Play for Automation Systems” are presented. AutoPnP was used for determining the requirements on future control architectures.

1. Einleitung

Volatiler werdende Märkte haben in den letzten Jahren dazu geführt, dass die Nachfrage nach wandelbaren, adaptiven Produktionssystemen stark gestiegen ist und weiterhin steigt. Gründe für die Turbulenzen sind neue verfügbare Technologien, geänderte Umweltanforderungen, wie z.B. knapper werdende Ressourcen, sowie weitreichende gesellschaftliche, politische und ökonomische Veränderungen [1]. Darüber hinaus ist ein klarer Trend weg von Massenprodukten Richtung personalisierter an den Kunden angepasster Produkte erkennbar [2].
Außerdem werden die Lebenszyklen der Produkte immer kürzer. Gleichzeitig steigt die Anzahl der Varianten eines Produkts an und bewirkt einen Rückgang bei den gefertigten Stückzahlen einer einzelnen Produktvariante. Aus diesen Gründen ist eine Erhöhung der Adaptivität von Produktionssystemen zwingend erforderlich, damit die Produktion rasch und problemlos an die geänderten Anforderungen für die Herstellung neuer Produkte und Varianten angepasst werden kann. Oft müssen dabei unterschiedliche Varianten eines Produktes bzw. verschiedene Produkte zur selben Zeit hergestellt werden. Wesentliche Punkte bei der Erfüllung dieser Anforderungen sind eine Reduktion der Rüstzeiten zwischen Varianten und Produkten und eine Erhöhung der Wandlungsfähigkeit innerhalb einer Fabrik [3].
Dieser Beitrag gliedert sich wie folgt: Nach einer kurzen Beschreibung der aktuellen Situation in der Automatisierungsindustrie und den aktuellen Hindernissen bei der Umsetzung adaptiver Produktionssysteme gehen wir auf die notwendige Veränderung der Steuerungsprogramme ein. In Kapitel 3 wird zuerst eine Modularisierung der Steuerungsprogramme vorge-

stellt. Aufbauend darauf wird in Kapitel 4 die Unterstützung von höherwertigen Funktionen aufgezeigt. Kapitel 5 beschreibt dann im Detail die entwickelte Rezeptfahrweise. Weitere Ergebnisse zur Realisierung von Wandlungsfähigkeit aus dem Forschungsprojekt AutoPnP sind Inhalt von Kapitel 6. Abschließend gibt Kapitel 7 eine kurze Zusammenfassung.

2. Die wandelbare Fabrik

Fabriken benötigen heutzutage eine aufwändige Steuerungsinfrastruktur mit einer Vielzahl von Sensoren und Aktoren, eingebettet in ein digitales Kommunikationsnetz, das sowohl Maschinen und Anlagenteile innerhalb eines Standortes untereinander vernetzt als auch standortübergreifend Vorgänge verbindet. Trotz des hohen Grades an Automatisierung sind Änderungen in der Fabrik immer noch mit einem hohen manuellen Aufwand verbunden. Dieser manuelle Aufwand betrifft insbesondere die Planung und die Inbetriebnahme [4] sowie die Neukonfiguration der IT-Systeme. Neben den bereits erfolgreichen Bemühungen die Modularisierung von Fabriken und Maschinen auf mechanischer, elektrischer und pneumatischer Ebene voranzutreiben, müssen nun insbesondere auch die IT-Systeme modularisiert und flexibilisiert werden [5]. Nur indem sich neue Komponenten einfach in das bestehende System integrieren lassen, wird die Umsetzung der Vision von adaptiven Produktionsumgebungen im Kontext von Industrie 4.0 [6] sichergestellt.

Eine wesentliche Anforderung an zukünftige Produktionssysteme ist die selbständige Anpassung an neue Fertigungsaufgaben [6]. Um dies mit minimalem Konfigurationsaufwand zu erreichen, müssen weitere Dienste wie Selbstbeschreibung, Selbstkonfiguration, Datenbeschaffung, Echtzeitüberwachung und Produktionsplanung angeboten werden. Diese automatische Konfiguration ist nicht nur auf die Steuerungsebene beschränkt, sondern muss auch vertikal über alle Ebenen der Automatisierungspyramide geführt werden [5]. Dadurch wird ermöglicht, dass z.B. „Manufacturing Execution Systems“ (MESes) zu jedem Zeitpunkt den tatsächlichen Fabrikzustand verfügbar haben.

Im Zuge der Vernetzung vieler unterschiedlicher Systeme bringt das Konzept der „Cyber-Physical Systems“ (CPSes) viele Potenziale und damit einen Mehrwert mit sich [6]. Die Vorteile liegen unter anderem in den Bereichen Interoperabilität, Engineering und Modularität. CPSes beschreiben den Trend, dass einzelne Module (Subsysteme) immer intelligenter werden und sich selbständig mit anderen Modulen zu größeren Systemen koppeln können, obwohl diese spezielle Kopplung im Ursprünglichen Modulentwurf nicht berücksichtigt wurde. Übertragen auf die Automatisierungstechnik (AT) bedeutet dies, dass Subsysteme die Fähigkeit erhalten sich (teil-)autonom zu Produktionsanlagen zu verbinden. Das CPS-Konzept

spielt daher auch im Bereich der AT als Basis für und zur Erhöhung der Wandlungsfähigkeit eine immer größere Rolle [7].

Die notwendige Modularität für adaptive Produktionssystem sowohl auf mechatronischer aber insbesondere auch auf IT-Ebene zu erreichen ist das Ziel des Verbundprojekts „AutoPnP - Plug & Play für Automatisierungssysteme“[1], welches im Rahmen des AUTONOMIK-Programms durch das deutsche Bundesministerium für Wirtschaft und Energie (BMWi) gefördert wird. Hierfür wurde ein modell-basierter Plug & Produce-Ansatz entwickelt, welcher den manuellen Aufwand für die Inbetriebnahme und (Re-)Konfiguration von Fabriken in Verbindung mit IT-Systemen reduziert. Notwendige Abläufe werden automatisiert, sodass die Fabrik kostengünstig und schnell anpassbar und somit wandlungsfähig wird. Dies soll es in Zukunft ermöglichen je nach Bedarf Produktionslinien in ihrer Struktur beliebig, mit minimalem Aufwand für die Änderungsinbetriebnahme des Gesamtsystems, umzukonfigurieren (Bild 1).

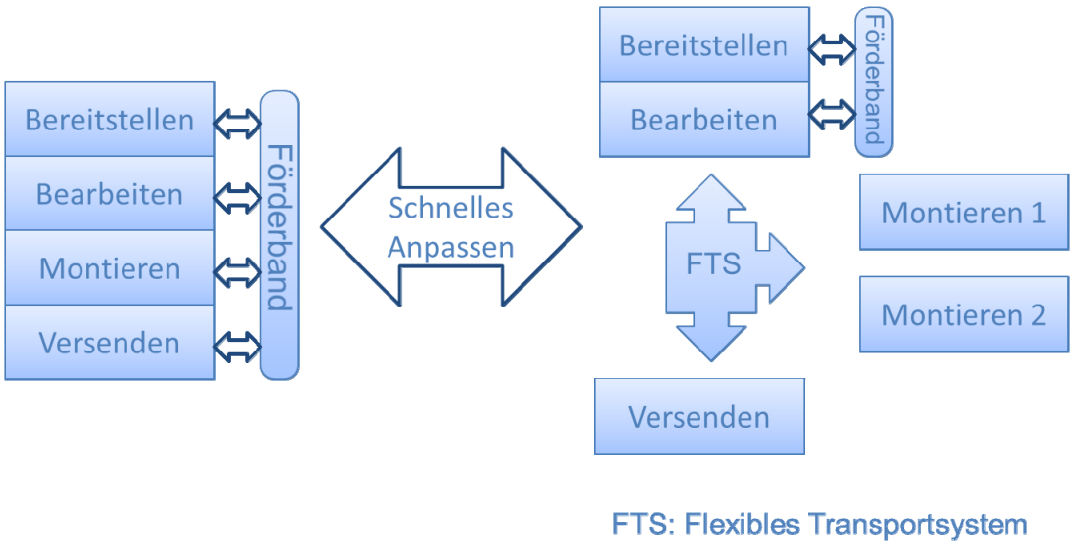

Bild 1: Anpassung des Produktionssystems. Die Stationen und die Produktionslinie werden umkonfiguriert, um den Durchsatz zu erhöhen oder um neue Produkte oder deren -Varianten zu fertigen.

Eine wesentliche Voraussetzung für eine schnelle Anpassung von IT-Systemen ist die vollständige Information über die aktuelle Systemkonfiguration (in unserem Fall das Layout der Produktionsanlage) sowie eine maschinenlesbare Beschreibung der Funktionen und Fähigkeiten derselben. Insbesondere müssen die Teilsysteme ihre Beschreibungen selbst bereit-

[1] AutoPnP-Projekt: http://www.autopnp.com/

stellen. Diese Beschreibungen werden in Modellen abgelegt, die zwischen verschiedenen Steuerungen aber auch über Systemgrenzen hinweg ausgetauscht werden können. Die Modelle kommen hier nicht nur während der Entwicklung, sondern insbesondere auch zur Laufzeit zum Einsatz. Dieses Konzept ist allgemein unter Models@Run-time bekannt [8]. Für die wandelbare Fabrik bedeutet das, dass das Produktionssystem die Modelle in Abhängigkeit des aktuellen Fabrikzustands (z.B. Ausfall einer Station) während der Laufzeit verändern kann. Dies erfolgt durch eine automatisierte Stations- und Nachbarschaftserkennung, welche es dem Anwender ermöglicht mit Hilfe von IT-Werkzeugen nötige Änderungen im Produktionsablauf aber auch in der Anlagenstruktur auf Modellebene automatisiert einzuplanen und anschließend auf Systemebene anzustoßen. Ein MES kann dieses Laufzeitmodel unter anderem für die Produktionsplanung und -steuerung zur automatischen Generation von Abläufen für die Abarbeitung von Produktionsaufträgen nutzen [9-11].

Um den hier beschriebenen Ansatz zur Umsetzung von wandelbaren Fabriken auch auf Steuerungsebene zu ermöglichen, war es notwendig die Struktur der bestehenden Steuerungsprogramme, verfasst in strukturiertem Text der IEC 61131-3 [12], anzupassen und zu verändern. Hierfür mussten zuerst die Steuerungsprogramme modularisiert und durch die Einführung von höheren Funktionen eine Anhebung des Abstraktionsgrades vorgenommen werden. Basierend auf diesem Grundgerüst wurde eine Rezeptfahrweise umgesetzt, die es erlaubt das Ausführungsprogramm einer Station während des Betriebes auszutauschen und so das Verhalten der Station individuell an das aktuell zu fertigende Produkt anzupassen.

Die einzelnen Schritte auf dem Weg von klassischen Steuerungsprogrammen zu Steuerungsprogrammen für Industrie 4.0 werden in den nachfolgenden Kapiteln im Detail erläutert.

3. Modularisierung der Steuerungsprogramme

Ausgangsbasis für Industrie 4.0-taugliche Steuerungsprogramme ist ihre Modularisierung. Dieser Beitrag bezieht sich auf SPS-Programme, kann aber auch auf andere Steuerungsprogramme, wie z.B. Roboterprogramme, übertragen werden. In klassischen Steuerungsprogrammen wird üblicherweise die übergeordnete Steuerungslogik (meist ein Zustandsautomat) mit dem Code für die konkrete Ansteuerung der Sensoren und Aktuatoren vermischt (siehe Bild 2a). Die starke Durchdringung der Steuerungslogik mit Code für die Ansteuerung der Sensoren und Aktuatoren führt aber zwangsweise zu Problemen, falls zu einem späteren Zeitpunkt entweder die Steuerungslogik verändert oder Sensoren und Aktuatoren ausgetauscht werden müssen.

Im Falle der Veränderung der Steuerungslogik ist es notwendig, dass die richtigen Erweiterungsstellen identifiziert werden können. Dies wird durch die Tatsache erschwert, dass der

eigentliche Steuerungscode durch den Code für die Ansteuerung der Geräte versteckt wird und dadurch schwerer auszumachen ist. Im Extremfall übersteigt der Anteil an Code für die Ansteuerung der Geräte den Code für die übergeordnete Steuerungslogik um ein Vielfaches. Beim Erweitern kann es auch vorkommen, dass Code für die Geräteansteuerung, der eigentlich zusammen gehört, versehentlich aufgeteilt wird und dadurch ein Fehlverhalten bei der Anlagensteuerung hervorgerufen wird.

Im Falle des Austauschs oder der Umverdrahtung eines Sensors oder Aktuators ist es wichtig, dass alle Programmteile identifiziert werden, die sich um die Ansteuerung des betroffenen Geräts kümmern. Da diese Programmteile üblicherweise mehrfach im Programm auftreten und meist aus mehreren zusammengehörigen Befehlen bestehen, kann es leicht passieren, dass bei der Anpassung des Codes die Überarbeitung einiger Programmteile vergessen wird oder diese nur teilweise angepasst werden. Dies führt dann zu Inkonsistenzen innerhalb des Steuerungsprogramms, welche aufwendig bei der Inbetriebnahme erkannt und behoben werden müssen. In vereinzelten Fällen kann es sogar passieren, dass nicht alle Inkonsistenzen bei der Inbetriebnahme gefunden werden und sich diese daher erst später während des Betriebes bemerkbar machen.

Um die oben genannten Probleme zu beheben, reicht eine einfache Modularisierung des Steuerungsprogramms aus. Dabei werden die Befehle zur Ansteuerung eines Gerätes in separaten Funktionen (bei Funktionen kann es sich sowohl um Funktionen als auch Funktionsblöcken handeln) zusammengefasst, die wiederum an einem gemeinsamen Ort (z.B.: Datei oder Ordner) gesammelt abgelegt werden. Die Funktionen können dann an den entsprechenden Stellen im Programmcode dazu verwendet werden, die gewünschte Interaktion mit dem Gerät durchzuführen. Die Ablage der Ansteuerungsbefehle in eigenen Funktionen an einem gemeinsamen Ort erleichtert zudem die Anpassung des Ansteuerungscodes, falls einmal ein Geräteaustausch oder eine Umverdrahtung stattfindet.

Lange Ketten von Steuerungsbefehlen können ebenfalls in eigene Funktionen ausgelagert werden, um die Programmteile der übergeordneten Steuerungslogik besser vom Ansteuerungscode zu trennen und dadurch die Lesbarkeit des Steuerungsprogramms zu erhöhen.

Bild 2 zeigt eine beispielhafte Modularisierung eines Programmteils, dass die Zustandsmaschine der übergeordneten Steuerung samt Maschinensteuerung umsetzt.

```
blink := NOT blink;

CASE state OF
    STARTING_UP:
        "signalTowerRedLight" := TRUE;
        "signalTowerYellowLight" := FALSE;
        "signalTowerGreenLight" := FALSE;
        IF "restartButtonPressed" THEN
            state := INITIALIZE;
        END_IF;

    INITIALIZE:
        // Initialize
        ...
        "signalTowerRedLight" := FALSE;
        "signalTowerYellowLight" := blink;
        "signalTowerGreenLight" := FALSE;
        IF finished THEN
            state := OPERATION;
        END_IF;

    OPERATION:
        // Operate
        ...
        "signalTowerRedLight" := FALSE;
        "signalTowerYellowLight" := FALSE;
        "signalTowerGreenLight" := blink;
        IF finished THEN
            state := INITIALIZE;
        END_IF;
END_CASE;
```

```
CASE state OF
    STARTING_UP:
        SetSignalTower(signal := RED, blinking := false);
        IF ShallStationBeRestarted() THEN
            state := INITIALIZE;
        END_IF;

    INITIALIZE:
        Init.InitDB();
        SetSignalTower(signal := YELLOW, blinking := true);
        IF InitDB.finished THEN
            state := OPERATION;
        END_IF;

    OPERATION:
        Main.MainDB();
        SetSignalTower(signal := GREEN, blinking := true);
        IF MainDB.finished THEN
            state := INITIALIZE;
        END_IF;
END_CASE;
```

(a) Steuerungsprogramm vor Modularisierung(b) Steuerungsprogramm nach Modularisierung

Bild 2: Beispiel für die Modularisierung von Steuerungsprogrammen.

4. Bereitstellung höherer Funktionen

Aufbauend auf einer Modularisierung, wie sie im vorherigen Kapitel vorgestellt wurde, lassen sich weitere Vereinfachungen dadurch erreichen, dass oft verwendeter Code in separaten Funktionen zusammengefasst wird. Diese Funktionen können dann höhere Funktionalitäten bereitstellen. Unter höheren Funktionen werden hierbei Funktionen verstanden, die auf mehreren Funktionen aufbauen und diese geeignet miteinander verbinden, um eine höherwertige Funktionalität zu realisieren. Ein Beispiel einer höheren Funktion wäre eine Funktion *transport()*, die ein Werkstück auf einem Transportband vom Anfang bis zum Ende transportiert. Das Erreichen des Endes kann dabei entweder durch das Verstreichen einer definierten Zeit oder durch das Unterbrechen einer Lichtschranke festgestellt werden. Die höhere Funktion *transport()* verbindet dabei die Einschaltfunktion des Transportbands mit der Funktion zum Erkennen des Erreichens des Transportbandendes.

Die Verwendung von höheren Funktionen innerhalb eines Steuerungsprogramms kann dabei helfen Codeduplizierung zu vermeiden, indem immer wieder verwendeter Code einmal zusammengefasst in einer Funktion hinterlegt wird, die dann bei Bedarf aufgerufen werden kann. Dabei muss der Code nicht immer vollständig identisch sein. Eventuelle Unterschiede lassen sich bis zu einem gewissen Grad auch durch Funktionsparameter und entsprechende Verzweigungen im Funktionscode ausgleichen. Durch das Zusammenfassen des gemeinsamen Codes wird aber auf jeden Fall sichergestellt, dass bei etwaigen Änderungen der Code nur an einer einzigen Stelle angepasst werden muss. Das geänderte Verhalten wird dann automatisch bei allen Aufrufen berücksichtigt. Dies erleichtert es deutlich, ein vorliegendes Steuerungsprogramm konsistent zu halten.
Die Lesbarkeit von Steuerungsprogrammen wird durch die Verwendung von höheren Funktionen signifikant erleichtert [13]. Dadurch verringert sich einerseits die Anzahl der Programmzeilen. Andererseits erleichtern gute Funktionsnamen das schnellere Verständnis von existierendem Programmcode.
Aufbauend auf einer vorhandenen Bibliothek von höheren Funktionen lassen sich zudem neue Steuerungsprogramme viel schneller entwickeln. Des Weiteren kann ein Großteil der Programmierfehler durch die Wiederverwendung von bereits getestetem Code vermieden werden, was wiederum zu einer deutlichen Verringerung des Aufwands bei der Entwicklung und Inbetriebnahme von Steuerungsprogrammen führt. Diese Vorteile spielen besonders im Zusammenhang mit Industrie 4.0 eine entscheidende Rolle, da hier die Anzahl der Neuinstallationen und Änderungen von Produktionsanlagen deutlich zunimmt.

5. Rezeptfahrweise

Zur weiteren Erhöhung der Wandlungsfähigkeit wurde basierend auf der Modularisierung und den höheren Funktionen eine Rezeptfahrweise implementiert. Die entwickelte Rezeptfahrweise entspricht in ihrer Arbeitsweise der eines Befehlsinterpreters, welcher ein Programm (Rezept) erhält und dieses entsprechend der enthaltenen Befehle ausführt. Ein Rezept enthält dabei die Steuerungsanweisungen für die jeweilige Maschine. Die zulässigen Steuerungsanweisungen sind vom konkreten Maschinentyp abhängig und können sich zwischen unterschiedlichen Maschinentypen stark unterscheiden.
Bevor auf die genaueren Details von Rezepten eingegangen wird, soll zuerst die allgemeine Programmstruktur der hierfür verwendeten Steuerungsprogramme näher betrachtet werden. Die Steuerungsprogramme bestehen aus vier Teilen: (1) Programmgrundgerüst, (2) Interfaces, (3) Modultypen und (4) stationsabhängige Programmteile.

Das Programmgerüst enthält Code, der für alle Stationen gleich ist. Darunter fällt unter anderem die übergeordnete Stationszustandsmaschine (ähnlich der OMAC-Zustandsmaschine [14]), die Rezeptsteuerung samt Speicher, Kommunikationsmechanismen zur Anbindung an ein übergeordnetes MES und Code zur Interaktion mit dem generischen Stationsbedienpult und der Stationszustandssignalisierung. Dieser Sachverhalt wird in Bild 3 gezeigt.

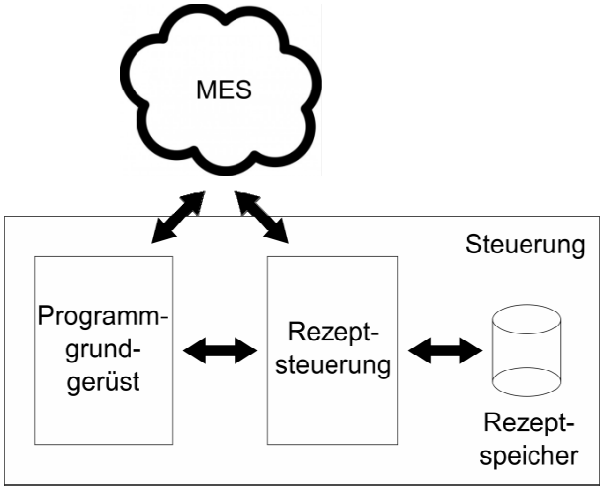

Bild 3: Grundstruktur eines Steuerungsprogramms mit Rezeptfahrweise.

Die Rezeptsteuerung implementiert einen Programminterpreter, der zyklisch Befehle aus dem Rezeptspeicher liest, dekodiert und ausführt. In seiner Funktionsweise ähnelt er handelsüblichen CPUs (siehe Bild 4). Der Befehlssatz der Rezeptsteuerung ergibt sich dabei aus der Summe der im System bekannten Interfaces. Sobald ein Rezept vollständig abgearbeitet wurde, kann dieses durch ein anderes Rezept ersetzt werden und dessen Abarbeitung angestoßen werden.

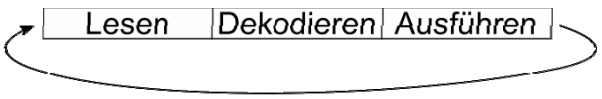

Bild 4: Funktionsweise der Rezeptsteuerung.

Zur Integration von stationsabhängigem Programmcode werden vordefinierte Funktionen aufgerufen, welche für die jeweilige Station entsprechend implementiert werden müssen. Innerhalb des Programmgerüsts ist auch bereits ein Großteil der Verriegelungslogik umgesetzt.
Interfaces erlauben es die erforderlichen Funktionen mit ihren Parametern zur Umsetzung einer Funktionalität festzulegen, z.B. enthält das Interface eines Zylinders Funktionen zur

Abfrage der aktuellen Position und zum Verfahren des Zylinders. Dazu wird auch die Dekodierung der Interface-Befehle direkt innerhalb des Interfaces implementiert. Die Verwendung von Interfaces ermöglicht einen sehr einfachen Austausch von Modulen, solange der ersetzende Modultyp auch alle verwendeten Interfaces des zu ersetzenden Modultyps implementiert.

Interfaces werden wiederum von Modultypen implementiert. Bei einem Modultyp handelt es sich um alle Algorithmen, die zur Ansteuerung dieses Modultyps, wie z.B., ein SLT Zylinder von Festo oder eine SOEG Lichtschranke von Festo, notwendig sind. Dabei kann ein Modultyp auch hierarchisch aus anderen Modulen (Instanzen von Modultypen) aufgebaut sein und bei Bedarf auch höhere Funktionen umsetzen. Um ein Interface zu implementieren, muss ein Modultyp Implementierung für alle vom Interface spezifizierten Funktionen anbieten und die Implementierung des Interfaces in seiner Systemschnittstelle anzeigen. Die Umsetzung ein und desselben Interfaces kann dabei für zwei verschiedene Modultypen unterschiedlich ausfallen. Die Modultypen bilden eine Bibliothek, die sicherstellt, dass die Implementierung eines Modultyps nur ein Mal erfolgen muss. Basierend auf den erstellten Modultypen lassen sich sehr rasch ganze Stationen auf- bzw. umbauen.

Die stationsabhängigen Programmteile umfassen neben den aus dem Programmgrundgerüst aufgerufenen Funktionen auch die Konfiguration der Station. Unter der Konfiguration einer Station versteht man die Instanziierung von Modulen basierend auf den vorher definierten Modultypen. Bei der Instanziierung eines Moduls wird unter anderem die Verkabelung des Gerätes gegen die SPS angegeben. Beim CODESYS Application Composer von 3S[2], der eine vergleichbare Modularisierung umsetzt, können die Ein- und Ausgänge der Steuerung komfortabel per Drag & Drop mit den Modulen verdrahtet werden. Neben der Spezifikation der Geräteverdrahtung können abhängig vom jeweiligen Modultyp auch noch weitere Daten, wie z.B. die Angabe, ob ein Sensorwert invertiert anliegt oder nicht, benötigt werden.

Rezeptstruktur

Nachdem die allgemeine Programmstruktur beschrieben wurde, soll nun auf die genaue Struktur der Rezepte eingegangen werde. Bei einem Rezept handelt es sich um eine Sequenz von Befehlen, die in der angegebenen Reihenfolge abgearbeitet wird. Ein Befehl bezieht sich im Normalfall auf einen Aufruf einer Interfacefunktion für ein Modul. Die Abarbeitungszeit eines Befehls kann auch mehrere Zyklen umfassen und je nach Befehl stark variie-

---

[2] CODESYS Application Composer: http://de.codesys.com/produkte/codesys-engineering/application-composer.html

ren. Neben den Funktionsaufrufen für Module gibt es auch Programmsteuerbefehle. Diese dienen zur Realisierung von Verzweigungen und Schleifen.

In Bild 5 wird der Aufbau eines Rezepts dargestellt. Ein einzelner Befehl besteht aus einem Befehlsidentifizierer, der sich aus der Angabe des Moduls, des Interfaces und des Befehls zusammensetzt, und dessen Parametern. Sollten mehr als drei Parameter von einem Befehl verwendet werden, werden die restlichen Parameter in den darauffolgenden Rezeptzeilen abgelegt. Neben dem eigentlichen Befehl werden auch Angaben für die Behandlung von Verzweigungen in einer Rezeptzeile abgelegt. Evaluierung gibt an, ob es sich bei dem aktuellen Befehl um eine Überprüfung handelt und entsprechend des Ergebnisses verzweigt werden soll. In diesem Fall muss der aufgerufene Befehl als Ergebnis einen Wahrheitswert liefern. Die Evaluierungsmerker dienen dazu, die Zugehörigkeit eines Befehls zu einem Verzweigungszweig festzulegen. Aktuell wird hierfür eine maximale Verzweigungstiefe von acht unterstützt.

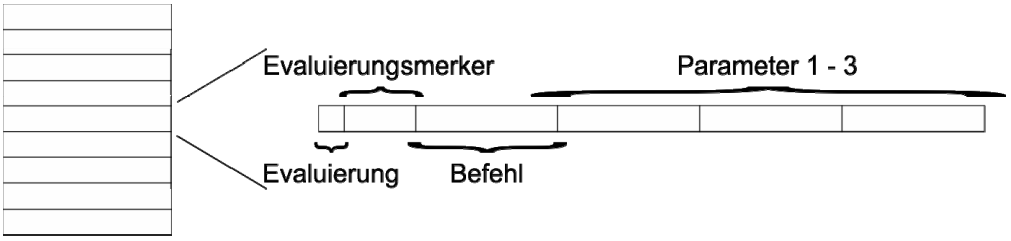

Bild 5: Aufbau eines Rezepts samt Befehl.

Zur besseren Veranschaulichung wird in Bild 6 ein einfaches Rezept für ein Transportband gezeigt. Die Funktion *call()* ruft die ausgewählte Interfacefunktion für das übergebene Modul auf. Die Befehlsweiterschaltung innerhalb des Rezepts erfolgt immer erst nach vollständiger Abarbeitung der aufgerufenen Modulfunktion. Die Zeilen 1 - 3 implementieren eine Schleife, die solange wartet, bis ein Werkstück am Anfang des Transportbands erkannt wird. In Zeile 4 erfolgt dann der Aufruf der Transportfunktion für das Transportband. Zum Abschluss wird in Zeile 5 das Rezeptende signalisiert.

```
1   IF NOT call(module := DETECT_WORKPIECE, interface := INTERFACE_SENSOR,
                command := IS_WORKPIECE_DETECTED) THEN
2       GOTO 1
3   END_IF
4   call(module := CONVEYOR_BELT, interface := INTERFACE_TRANSPORTATION,
        command := TRANSPORT, parameters := [PLACE_START, PLACE_END])
5   call(module := RECIPE_CONTROLLER, interface := INTERFACE_RECIPE_CONTROL,
        command := RECIPE_END)
```

Bild 6: Rezept für die Steuerung eines Transportbands in lesbarer Form.

In Bild 7 wird exemplarisch die Konfiguration des Moduls *DETECT_WORKPIECE* des Rezepts aus Bild 6 gezeigt. Dafür wird der Eingang angegeben, an dem der Sensor an die Steuerung angeschlossen ist (Eingang 0.0). Außerdem wird noch festgelegt, dass der Sensor nicht invertiert ist, d.h. eine Werkstückerkennung wird durch den Zustand logisch 1 angezeigt.

```
TYPE SensorConfig :
    STRUCT
        sensorValue : POINTER TO BIT;
        inverted : BOOL;
    END_STRUCT
END_TYPE
```

```
WorkpieceDetectConfig : SensorConfig := (
    sensorValue := ADR(%IX0.0),
    inverted := FALSE
);
```

Bild 7: Konfiguration eines Werkstückerkennungssensors mit zugehöriger Sensorkonfigurationsdatenstruktur.

Da SPSen normalerweise einen flachen Namensraum aufweisen (die Namen aller Funktionen und Datentypen befinden sich in einem gemeinsamen globalen Namensraum), ist es zwingend erforderlich eine geeignete Unterteilung des Namensraumes zu verwenden, da es ansonsten zu ungewollten Namenskollisionen kommen kann. Beispielsweise können benamte Elemente zusammengruppiert werden, indem ihnen ein gemeinsamer Präfix gefolgt von einem Unterstrich vorangestellt werden. Es ist aber zu beachten, dass die maximale Länge für Namen nicht überschritten wird.

6. Weitere Ergebnisse aus AutoPnP

Neben der Modularisierung und Einführung einer Rezeptfahrweise bei den Steuerungsprogrammen wurden im Rahmen des Verbundprojekts „AutoPnP“ weitere Ergebnisse für die Umsetzung von Industrie 4.0-Aspekten erzielt. Diese sollen im Folgenden grob vorgestellt werden. [9-11]

Eine wichtige Erkenntnis für die Umsetzung von Wandlungsfähigkeit ist die Notwendigkeit der Entkopplung der Arbeitsvorgänge innerhalb eines Arbeitsplans von den Stationen (Arbeitsplätzen), an denen diese durchgeführt werden. Die Zuordnung der einzelnen Arbeitsvorgänge zu den Stationen erfolgt dann im Rahmen der Feinplanung. Durch die späte Kopplung von Arbeitsvorgängen und Stationen kann flexibel auf die aktuelle Situation innerhalb der Fertigung eingegangen werden. So können z.B. ausgelastete oder neu verfügbare Maschinen direkt berücksichtigt und ausgefallene Maschinen einfach übergangen werden. So-

mit müssen die bestehenden Arbeitspläne nicht mehr an veränderte Situationen in der Fertigung angepasst werden, da dies im Rahmen der Feinplanung automatisch erfolgt. Die Entkopplung wurde durch die Einführung von Fähigkeiten erreicht. Arbeitsvorgänge erfordern für die Durchführung gewisse Fähigkeiten, die von Stationen bereitgestellt werden. Abhängig von der Station können von dieser auch mehrere Fähigkeiten angeboten werden. Die Feinplanung ist dann dafür verantwortlich, geeignete Stationen für die einzelnen Arbeitsvorgänge auszuwählen. Dabei ist zu beachten, dass zwischen aufeinanderfolgenden Stationen bei der Fertigung auch eine entsprechende Materialtransportmöglichkeit besteht.

Ein weiterer wichtiger Aspekt für die Wandlungsfähigkeit von Produktionsanlagen ist eine maschinenlesbare Beschreibung der Stationen. Diese Beschreibung kann dazu verwendet werden, eine Station beim zuständigen MES anzumelden und diesem die angebotenen Fähigkeiten zu übermitteln. Neben den bereitgestellten Fähigkeiten können die Beschreibungen dem MES auch noch weitere Informationen, wie z.B. Bearbeitungsdauern und Wartungsintervall, zur Verfügung stellen, welche bei der Planung der Fertigung Berücksichtigung finden. Um eine einfache Verarbeitung der Beschreibungen durch das MES zu ermöglichen, ist es erforderlich, dass man sich bei der eingesetzten Beschreibungssprache auf einen gemeinsamen herstellerunabhängigen Standard einigt.

Basierend auf der Stationsbeschreibung und der Entkopplung der Arbeitspläne von den Stationen kann als weitere Verbesserung eine automatische Erkennung von Stationen und des dazwischen liegenden Materialfluss umgesetzt werden. Dies erlaubt es dem MES direkt die zur Verfügung stehenden Stationen bei der Feinplanung zu berücksichtigen. Im Rahmen der Anbindung der Stationen an das MES können auch detaillierte Informationen über den aktuellen Zustand an das MES übermittelt werden. So können z.B. Ausfälle direkt gemeldet und eine entsprechende Umplanung vorgenommen werden. Die Erkennung des Materialflusses ist dann von Bedeutung, wenn entweder ein automatischer Materialtransport zwischen den Stationen durchgeführt wird oder Material zwischen den einzelnen Stationen nicht beliebig hin- und zurücktransportiert werden kann. Der mögliche Materialfluss muss in diesen Fällen bei der Feinplanung durch das MES mitberücksichtigt werden. Gegebenenfalls müssen die notwendigen Transporte durch das MES festgestellt und entsprechende Transportaufträge generiert werden.

Alle vorgestellten Ergebnisse wurden innerhalb des Forschungsprojekts AutoPnP im Rahmen des Demonstrators „Wandelbare Fabrik“ umgesetzt und erprobt. Der Demonstrator wird in Bild 8 gezeigt. Durch die vorgestellte Modularisierung von SPS-Programmen konnten neue Stationen rasch und mit wenig Aufwand integriert und in Betrieb genommen werden, da viele Module über die Stationen hinweg wiederverwendet werden konnten. Die Rezeptfahr-

weise ermöglicht es dem MES die Stationen flexibel für die unterschiedlichen Produktionsaufträge zu nutzen.

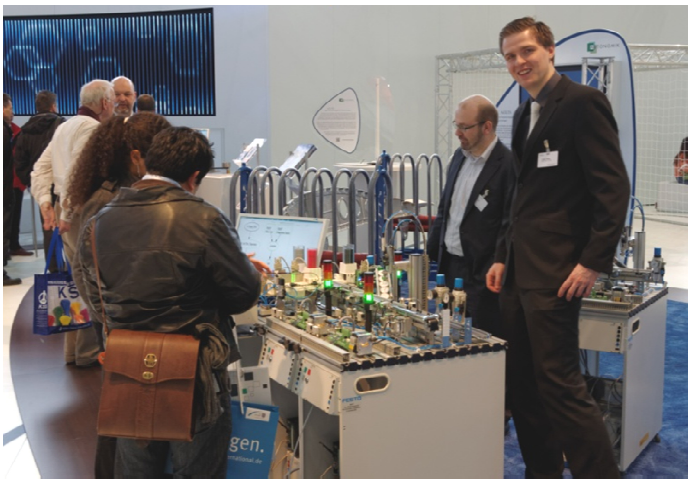

Bild 8: Demonstrator „Wandelbare Fabrik“ des Forschungsprojekts AutoPnP.

7. Zusammenfassung

Basierend auf den Anforderungen von Industrie 4.0 an die Produktion von morgen wurde eine Modularisierung von Steuerungsprogrammen vorgestellt. Aufbauend darauf wurde eine Rezeptfahrweise präsentiert, die es erlaubt das Verhalten einer Station während der Laufzeit grundlegend zu verändern. Da die Änderung des Verhaltens auf dem Austausch eines Rezepts basiert, bleiben die Verriegelungslogiken davon unberührt, wodurch ein grobes Fehlverhalten der Station ausgeschlossen werden kann. Abschließend wurden weitere Erkenntnisse zu wandlungsfähigen Fabriken aus dem Forschungsprojekt AutoPnP vorgestellt.

Danksagung

Die vorgestellten Ergebnisse wurden innerhalb des Forschungsprojekts „AutoPnP – Plug & Play für Automatisierungssysteme“ erarbeitet. AutoPnP ist Teil des vom deutschen Bundesministerium für Wirtschaft und Energie (BMWi) geförderten AUTONOMIK-Programms, welches durch das deutsche Zentrum für Luft- und Raumfahrt (DLR) als Projektträger betreut wird.

Referenzen

[1] Nyhuis, P.: Wandlungsfähige Produktionssysteme: Heute die Industrie von morgen gestalten. Produktionstechnisches Zentrum 2008.

[2] Koren, Y.: The Global Manufactoring Revolution. John Wiley & Sons, Inc., 2010.

[3] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., Brussel, H.V.: Reconfigurable Manufacturing Systems. Annalen von CIRP, Vol. 48, Nr. 2, S. 527-540, 1999.

[4] Zäh, M.-F., Beetz. M., Shea, K., Reinhart, G., Stursberg, O., Ostgathe, M., Lau, C., Ertelt, C., Pangercic, D., Rühr, T., Ding, H., Paschedag, T.: An Integrated Approach to Realize the Coginitive Machine Shop. In: Proc. of the 1st Int. Workshop on Cognition for Technical Systems, Techische Universität München, 2008.

[5] Sauer, O., Jasperneite, J.: Adaptive information technology in manufacturing. In: Proc. Of the 44th CIRP International Conference on Manufacturing Systems. S., Omnipress 2011.

[6] Geisberger, E., Broy, M.: agendaCPS – Integrierte Forschungsagenda Cyber-Physical Systems, acatech – Deutsche Akademie der Technikwissenschaften 2012.

[7] Vogel-Heuser, B., Kegel, G., Bender, K., Wucherer, K.: Global Information Architecture for Industrial Automation, atp – Automatisierungstechnische Praxis, Heft 1, S. 108-115, 2009.

[8] Blair, G.; Bencomo, N.; France, R.B.: Models@run.time, IEEE Computer, Vol. 42 (10), S. 22-27, 2009.

[9] Nadine Keddis, Gerd Kainz, Christian Buckl, and Alois Knoll. Towards adaptable Manufacturing Systems. In Proceedings of the 2013 IEEE International Conference on Industrial Technology (ICIT 2013), February 2013.

[10] Gerd Kainz, Nadine Keddis, Dirk Pensky, Christian Buckl, Alois Zoitl, Reinhard Pittschellis und Bernd Kärcher. AutoPnP – Plug-and-Produce in der Automation: Wandelbare Fabrik als cyberphysisches System. In atp edition. Vulkan Verlag, April 2013.

[11] Alois Zoitl, Gerd Kainz, and Nadine Keddis. Production Plan-Driven Flexible Assembly Automation Architecture. In Proceedings of the 6th International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS 2013), August 2013.

[12] IEC 61131-3: Programmable controllers - Part 3: Programming languages. 2013.

[13] Tenny, T., "Program readability: procedures versus comments", IEEE Transactions on Software Engineering, vol.14, no.9, pp.1271-1279, September 1988.

[14] D. Arens, T. Hopfgartner, T. Jensen, M. Lamping, M. Pieper, D. Seger, “Packaging Machine Language V3.0 Mode & States Definition Document,” OMAC Motion for Packaging Working Group, June 2006.