S 2-1
Universelle Programmierschnittstelle für Motion-Logic Systeme

Struktur, Funktionen und Anwendung in der Forschung und Lehre

Von Elmar Engels[1], Thomas Gabler[2],

1 Abstract

Dieser Artikel beschreibt eine universelle Programmierschnittstelle zur Entwicklung von Motion-Logic Applikationen für industrielle Motion-Controller. Sie ermöglicht die Entwicklung von Software mit modernen Hochsprachen, wie C/C++, Visual Basic, C#, Java, Objective-C und visuellen Programmiersprachen, wie LabVIEW in Koexistenz mit klassischen Programmiersprachen für speicherprogrammierbare Steuerungen nach IEC 61131-3. Daraus ergeben sich neue Möglichkeiten industrielle Antriebs-und Steuerungssysteme mit ihrem IT-Umfeld zu verbinden.

2 Einleitung

Motion-Logic Systeme werden in der Automatisierungstechnik üblicherweise mit speicherprogrammierbaren Steuerungen (SPS) realisiert. Die Programmiersprachen, die auf diesen Steuerungen eingesetzt werden, können herstellerspezifisch oder herstellerübergreifen, z.B. nach IEC-61131-3, sein [2].

Es gibt jedoch auch Anwender, die Anforderungen nach Programmiersprachen stellen, wie sie in der Anwendungsprogrammierung im PC-Bereich eingesetzt werden. Gründe für diese Forderungen können das größere Erfahrungswissen mit diesen Programmiersprachen sein, oder es existieren bereits gewachsene Programmpakete, die auf eine neue Plattform portiert werden sollen.

Aus technologischer Sicht kommt hinzu, dass die Programmierwerkzeuge und die Programmierparadigmen für die Anwendungsprogrammierung im PC-Bereich, nicht zuletzt auch durch mobile Anwendungen, wie SmartPhones, Netbooks und ähnliche Geräte, immer effizienter und komfortabler werden. Nun unterscheiden sich die Anforderungen an Anwenderprogramme im PC-Bereich in vielen Dingen von Anforderungen an zuverlässige und echtzeitfähige Motion-Logic-System, aber eine Schnittstelle zwischen beiden Welten ist dennoch möglich. Diese kann dadurch realisiert werden, dass Softwarekomponenten mit harten Echtzeitanforderungen gekapselt werden, und neben der SPS Programmierschnittstelle eine offene Programmierschnittstelle für Programmiersprachen aus dem PC-Bereich zu Verfügung stellen.

Die Autoren haben eine solche Schnittstelle für das System IndraMotion der Firma Bosch Rexroth Electric Drives and Controls GmbH entwickelt [1]. Diese als Motion-Logic-Programming-

[1] Prof. Dr.-Ing. Elmar Engels, FH Aachen, Aachen
[2] Dipl.-Ing. Thomas Gabler, Bosch Rexroth Electric Drives and Controls GmbH, Lohr am Main

Interface (MLPI) bezeichnete Programmierschnittstelle ermöglicht dem Anwender den Zugriff auf Funktionalitäten, die in der Firmware des Motion-Logic-Controllers gekapselt sind. So können schnell und einfach Motion-Control Applikationen in den Programmiersprachen C/C++, C#, Visual Basic oder Objective-C entwickelt werden. Auch eine grafische Programmierung mit dem Entwicklungswerkzeug LabVIEW wird unterstützt (vgl. Bild 1).

In dem vorliegenden Beitrag wird die Schnittstelle vorgestellt und erläutert, welche Möglichkeiten sich zukünftig durch die Programmierschnittstelle bei der Umsetzung von Forschungs- und Entwicklungsprojekten ergeben.

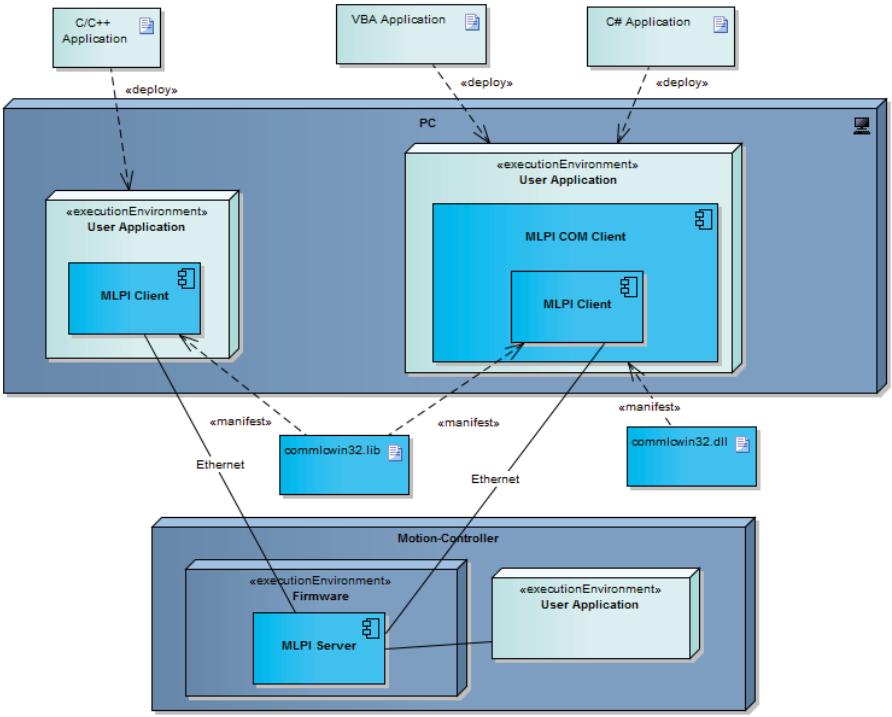

Bild 1: Übersicht der Motion-Logik Programmierschnittstelle als Deployment Diagram

3 Einleitung

Motion-Logic Systeme für die Automatisierungstechnik werden üblicherweise mit speicherprogrammierbaren Steuerungen (SPS) oder sogenannten Motion-Controllern realisiert. Für die Programmierung dieser Motion-Controller werden teilweise proprietäre Programmiersysteme einzelner Hersteller eingesetzt. Stand der Technik ist jedoch der Einsatz von Engineeringum-

gebungen, die in großen Teilen konform zu den Vorgaben der internationalen Norm IEC 61131 sind. Die Norm definiert in Teil 3 Programmiersprachen, die es erlauben Software so zu entwickeln, dass sie in Teilen herstellerunabhängig ist und somit interoperabel auf verschiedenen Plattformen eingesetzt werden kann. Zwar implementieren Steuerungs- und Antriebshersteller Funktionalitäten, die Alleinstellungsmerkmale für ihre Systeme sind und damit die Interoperabilität einschränken, jedoch hat die Normierung der Programmiersprachen dazu geführt, dass sich Entwickler wesentlich schneller in unterschiedliche Systeme einarbeiten können und Teilkomponenten der Software wiederverwendet werden kann [3].

Obwohl die Verwendung der grafischen oder textuellen Programmiersprachen der IEC 61131-3 anwendungsspezifische Vorteile besitzt, gibt es Anwender, die speziell für ihr Anwendungsgebiet Programmierwerkzeuge und Programmierparadigmen einsetzen möchten, wie sie in der modernen Informatik durch Microsoft Windows oder Linux Betriebssysteme zur Verfügung stehen. Gründe dafür sind einerseits die breitere Basis von Anwendern und andererseits die kürzeren Innovationszyklen im Bereich der PC-Programmiersprachen und -werkzeuge.

Speicherprogrammierbare Steuerungen und Motion-Controller sind in der Regel echtzeitfähige Systeme, die kurze und definierte Antwortzeiten erfordern. In der Vergangenheit wurden die Anwendungen daher mit Hilfe von Echtzeitbetriebssystemen oder proprietären Betriebssystemen in C programmiert, um die erforderlichen Systemgeschwindigkeit zu erreichen. Für komplexe Anlagen sind so im Laufe der Produkthistorie enorme Investitionen in die Applikationssoftware geflossen. Es ist offensichtlich, dass ein Plattformwechsel hin zu einem IEC basierten System nicht in kurzer Zeit und mit geringem Aufwand realisiert werden kann. Zwar gibt es moderne und leistungsfähige Hardware mit umfangreichen Schnittstellen und Funktionsbibliotheken für C/C++, C#, LabVIEW, Visual Basic oder Objective C, die einen Plattformwechsel erleichtern würden, allerdings sind dies in aller Regel Systeme, die eher in die Klasse der Rapid-Prototyping Systeme fallen und mit den damit einhergehenden Kosten verbunden sind [6].

Bei Anlagen mit Seriencharakter, auch wenn es Sondermaschinen sind, setzen die Anlagenhersteller daher üblicherweise industrielle Automatisierungstechnik ein. Gründe dafür sind die Langzeitverfügbarkeit, die breite Unterstützung von Feldbussystemen und die nachgewiesene Zuverlässigkeit. Die Herausforderung, die sich daraus ergibt, ist die Entwicklung eines zuverlässigen industriellen Motion-Controllers, der sowohl mit IEC 61131-3 Programmiersprachen als auch mit modernen Hochsprachen aus der PC Umgebung programmiert werden kann, wobei idealerweise beide Softwaresysteme auf einem Gerät parallel ausgeführt werden können. Diese Anforderungen wurden mit der im Folgenden beschriebenen universellen Motion-Logic Programmierschnittstelle (MLPI) umgesetzt.

4 Zielsetzung bei der Entwicklung der Programmierschnittstelle

Aus technologischer Sicht sind die Innovationszyklen moderner Programmiersysteme im PC Bereich deutlich kürzer, als in der Automatisierungstechnik. In der modernen Informatik sind Anwendungen für Smart Phones, Netbooks oder Tablet-PC Innovationstreiber, die sich auf die Applikationsentwicklungsumgebungen aber auch auf die Programmierparadigmen auswirken.

Es ist zu beobachten, dass die Entwicklungsumgebungen als auch die Methodik schneller und effizienter werden.

In einigen Bereichen unterliegen Produktionsanlagen wie Werkzeugmaschinen, Handlingssysteme oder Industrieroboter ähnlichen Kriterien, wie Consumer-Elektronik, aber in vielen Aspekten mit deutlich unterschiedlicher Gewichtung der Faktoren. Anforderungen, wie die Echtzeitfähigkeit, die Verfügbarkeit, der Service über Jahrzehnte und auch die Zuverlässigkeit mit Hinblick auf Prozess- und Maschinensicherheit unterscheiden sich deutlich zwischen Büroanwendungen und Industrieanwendungen. Kapitalintensive Anlagen, wie Umformpressen beispielsweise, müssen über mehr als ein Jahrzehnt betrieben werden und die entsprechenden Ersatzteile verfügbar sowie Funktionserweiterungen möglich sein.

5 Schnittstellen eines Antriebssystems

Ausgangspunkt für die Entwicklung einer Motion-Logic Programmierschnittstelle sollte der Blick auf die Schnittstellen eines Motion-Controllers sein. In Bild 2 ist eine vereinfachte Sicht auf die beiden Hauptkomponenten des Antriebssystems, den Motion-Controller und das Antriebsregelgerät (Drive), dargestellt. Mit Blick auf das Antriebssystem aus Sicht der Architektur sind die Schnittstellen des Systems nach außen wichtig, auch wenn die interne Struktur der Komponenten weitaus komplexer ist. Es ist im ersten Schritt unerheblich, ob Motion-Logic und Antriebsregelgerät in einem Gerät untergebracht sind oder auf unterschiedliche Targets verteilt sind. Aus der Abbildung geht hervor, dass sich die Schnittstellen der beiden Komponenten ähneln. Antriebsregler und Motion-Controller verfügen über Benutzerschnittstelle mit Display und Bedienelementen, Peripherieschnittstellen für die zentrale oder dezentrale Signalerfassung und Signalausgabe und Schnittstellen für die Querkommunikation zwischen Steuerungen oder zwischen Antrieben.

Wichtig sind ebenfalls die Schnittstellen, die über die Anschlussports der Komponenten adressiert werden können. Diese Schnittstellen können in Parameter-, Kommando- und allgemeine Datenschnittstelle gegliedert werden.

Über die Parameterschnittstellen können Konfigurationen und Einstellungen der Komponenten vorgenommen werden. Kommandos lösen Aktionen in den Komponenten aus und die allgemeine Datenschnittstelle ermöglicht Zugriff auf Informationen, die nicht in die beiden genannten Klassen fallen.

Es ist leicht nachvollziehbar, dass Motion-Controller und Antriebsregler nicht über die gleichen Parameter, Kommandos und allgemeinen Daten verfügen müssen, allerdings sollte auf Information, die identisch in beiden Komponenten verfügbar sind, auch identisch zugegriffen werden können. Dies lässt sich einfach formulieren, allerdings liegt zwischen den beiden Komponenten die wohl wichtigste Schnittstelle, die Antriebsschnittstelle. Typische Schnittstellen, mit denen von Motion-Controllern auf Antriebsregler zugegriffen werden kann, sind Feldbusschnittstellen, wie beispielsweise PROFINET, PROFIBUS, Ethernet/IP, Powerlink, DeviceNET, EtherCAT oder sercos. Diese Schnittstellen unterscheiden sich gravierend, was die Ge-

schwindigkeit, Protokolle, Zugriffsverfahren und Eignung für synchrone Achsbewegungen angeht.

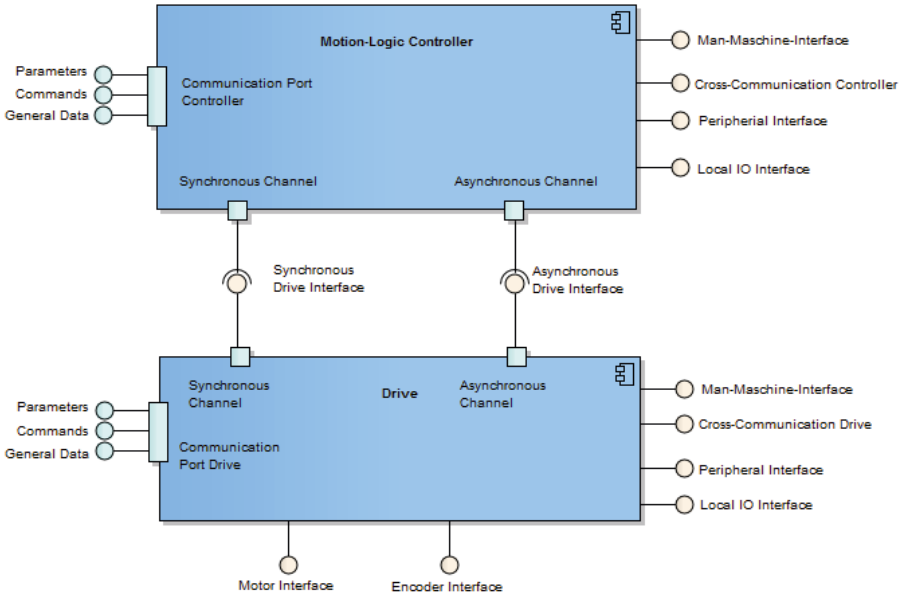

Bild 2: Schnittstellen eines Motion-Logic Systems

Bei all den Unterschieden haben sie jedoch auch Ähnlichkeiten. Sie unterscheiden zwischen asynchroner und synchroner Datenkommunikation, wobei sich die zugehörigen Protokolle deutlich unterscheiden. Synchrone Datenkommunikation ist erforderlich, wenn dem Antrieb beispielsweise zyklisch Lagesollwerte vorgegeben werden sollen, die bei synchronen Achsbewegungen minimale Totzeiten und Jitter aufweisen müssen. Allerdings muss es auch während des zyklischen Austauschs von Sollwerten möglich sein größere Daten zwischen Motion-Controller und Antriebsregler zu übertragen, die nicht innerhalb eines Telegrammzyklus übermittelt werden können. Dazu dient die asynchrone Datenkommunikation, die die Gesamtdaten in kleine Datenpaket unterteil, die dann sequentiell als Teil der einzelnen zyklischen Telegrammen übertragen werden können.

Zwar wurden in den vergangenen Jahren seitens der Antriebsanwender Anstrengungen unternommen, um die Schnittstellen zu den Antriebsregelgeräten zu harmonisieren, allerdings lässt sich das bestenfalls in der Definition von Profilen erkennen, die jedoch für die einzelnen Feldbusse stark differieren. Beim Einsatz von Antriebsfunktionalitäten, die durch die Profile definiert sind lässt sich Interoperabilität und Austauschbarkeit von Antriebskomponenten errei-

chen. Will der Anwender aber die volle Funktionalität der Antriebsregler verwenden, so wird die Anwendungssoftware schnell herstellerabhängig.

Neben der Definition von Profilen für die Feldbusse wurden Anstrengungen unternommen Harmonisierungsschichten für unterschiedliche Feldbusse zu definieren [5]. Dieser Ansatz zielt auf die Angleichung der Konfiguration und die Verwendung unterschiedlicher Feldbusse ab und liefert somit nicht automatisch auch eine Harmonisierung der Antriebsreglerschnittstellen.

Die universelle Programmierschnittstelle bewegt sich auf einer etwas höheren Architekturebene, da sie eine Schnittstelle für Anwendungen darstellt. Allerdings ist sie aufgrund der unterlagerten Schnittstelle des Feldbusses des Antriebs von diesem anhängig. Die Implementierung des Motion-Logic Programming Interfaces (MLPI) basiert auf dem Automatisierungsbus sercos [4].

sercos ist ein international anerkanntes und genormtes Feldbussystem, das ursprünglich für Antriebssysteme entwickelt wurde und geschätzt wird wegen seiner Echtzeitperformance, des Funktionsumfangs der Antriebsregler und seiner Robustheit. Die Funktionalitäten und Eigenschaften sind für den Motion-Controller in Form von vielzähligen Parametern und Kommandos verfügbar. Damit wird klar, dass die MLPI Zugriff auf diese Funktionalitäten liefern muss und nicht vollständig unabhängig vom verwendeten Feldbus des Antriebs sein kann.

6 Software Architektur der MLPI

Für den Entwurf der MLPI wurden einige Anforderungen aufgestellt, die die universelle Anwendbarkeit benötigt. Diese Anforderungen waren:

- Unabhängigkeit von der verwendeten Programmiersprache,
- Definition einer Architektur entsprechend eines Schichtenmodells,
- Strukturierung in Funktionalitäten,
- Unterstützung von objektorientierten Programmiersprachen,
- Netzwerkfähigkeit,
- hohe Ausführungsgeschwindigkeit,
- geringe Codegröße und
- Multi-Instanzfähigkeit.

In Bild 3 ist in einem Paketdiagramm übersichtlich die Strukturierung der Funktionalitäten dargestellt.

Die API Bibliothek enthält die grundlegenden Funktionalitäten um sich mit der Programmierschnittstelle des Motion-Controllers zu verbinden oder die Verbindung wieder zu lösen. Nachdem eine Verbindung zum Motion-Controller aufgebaut wurde, können mit der Funktionalität der System Bibliothek Systemeinstellungen vorgenommen werden. Dies sind Ethernet Einstellungen, Gerätenamen, Betriebsmodi, Diagnoseinformationen oder das Laden und Speichern von Parameterdateien.

Die Logic Bibliothek ermöglicht den Zugriff auf symbolische als auch Direktvariablen. Für einen sehr schnellen Datenaustausch können zudem entsprechende Datenpufferbereiche angelegt und adressiert werden.
Auf Einzel- als auch Listenparameter sowie deren Attribute wird über die Parameter Bibliothek zugegriffen.

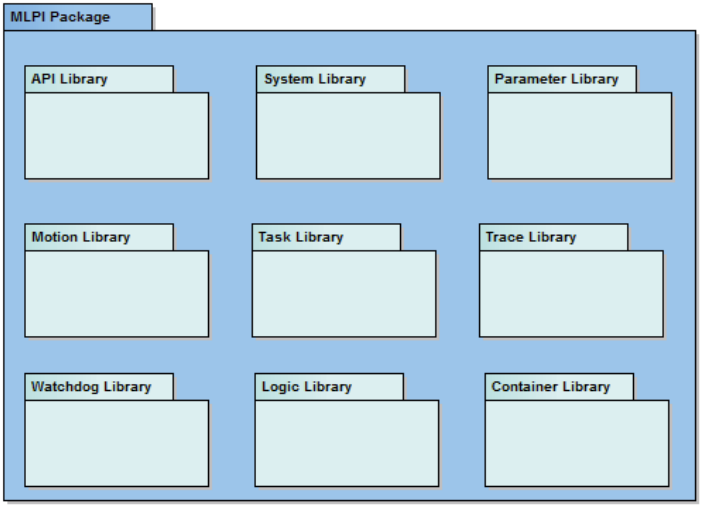

Bild 3: Aufteilung der Funktionalitäten der MLPI auf Bibliotheken

Eine ursprüngliche Forderung war die nach einem parallelen Betrieb von klassischen IEC Programmen parallel zu Applikation, die über die MLPI auf den Motion-Controller zugreifen. Dazu besitzt der Motion-Controller eine SPS Runtime Umgebung, die der MLPI-Applikation ebenfalls zur Verfügung steht. So können beispielsweise SPS Programme geladen, gelöscht, gestartet und gestoppt werden.

Wird eine MLPI-Applikation auf dem Motion-Controller ausgeführt, so muss diese Applikation in das Tasksystem des Motion-Controllers eingebunden werden. Die Task Bibliothek erlaubt das Einstellen der erforderlichen Werte, wie Priorität und Scheduling, und die Watchdog Bibliothek erlaubt das Überwachen der Ausführungszeiten entsprechender Systemreaktionen.
Mit diesen Funktionalitäten können typische SPS Applikationen in verschiedenen Programmiersprachen entwickelt werden, die über die MLPI an den Motion-Controller angebunden sind. Auch die Kommunikation zwischen einer MLPI-Applikation und einer IEC Applikation ist mit Hilfe globaler SPS Variablen möglich.

Die Kernaufgabe, nämlich die Realisierung von Bewegungssteuerungen mit Hilfe der MLPI, erfordert die Motion Bibliothek. Sie beinhaltet zahlreiche Funktionen, mit denen Achsobjekte angelegt und konfiguriert werden können. Die Motion Bibliothek stellt dazu eine umfangreiche Funktionalität zur Verfügung. Über die Schnittstelle lassen sich für Achsobjekte die typischen Eigenschaften einstellen. Dazu gehören beispielsweise die Bewegungsgrenzwerte für Position, Geschwindigkeit, Beschleunigung, Verzögerung und Ruck aber auch Statusinformationen zu den Achsen. Nach der Parametrierung der Achse kann diese über die in der PLCopen definierten Funktionen kommandiert werden. Selbstverständlich lassen sich hierüber auch die herstellerspezifische Funktionen aufrufen, welche die Leistungsfähigkeit des Antriebssystems ausmachen. Bei Verwendung mehrerer Achsen lassen sich mit der MLPI neben den diskreten und kontinuierlichen Einzelachsbewegungen auch synchrone Multi-Achsanwendungen realisieren.

Die Funktionalitäten der PLCopen sind im Motion-Kernel des Motion-Controllers realisiert und werden über die MLPI aufgerufen. Für eigene Bewegungsbefehle oder Regelungsalgorithmen stellt die MLPI aber Kanäle für die Vorgabe zyklischer Positions- oder Geschwindigkeitssollwerte zur Verfügung. Mit der Möglichkeit zwischen diesen Methoden der Bewegungskommandierung zu wechseln bietet die MLPI eine enorme Flexibilität für den Anwender.

Die genannten Bibliotheken beinhalten die Funktionalitäten, die über die Schnittstelle den Anwenderapplikationen zur Verfügung gestellt werden. Darüber hinaus muss jedoch auch die Architektur der Anwenderapplikation an die Anforderungen von Bewegungssteuerungen angepasst werden.

7 Architektur der Anwenderapplikation

In Bild 4 ist ein Komponentendiagramm einer Applikation dargestellt, die auf der MLPI aufsetzt. Die Applikation ist entsprechend der zeitlichen Anforderungen an die Motion-Logic Applikation in synchrone Threads, asynchrone Threads und freilaufende Threads aufgeteilt, die jeweils über den MLPI-Client auf die MLPI Schnittstelle zugreifen. Für harte Echtzeitanforderungen sollte eine Applikation mit synchronem Thread auf dem Motion-Controller direkt ausgeführt werden, während die Applikationsanteile mit asynchronen und freilaufenden Threads über ein Netzwerk auf den Motion-Controller zugreifen können. Unter der Randbedingung, dass das Netzwerk wenig belastet ist und die Reaktionszeiten kurz genug sind, kann aber auch der synchrone Anteil über ein Netzwerk auf den Motion-Controller zugreifen.

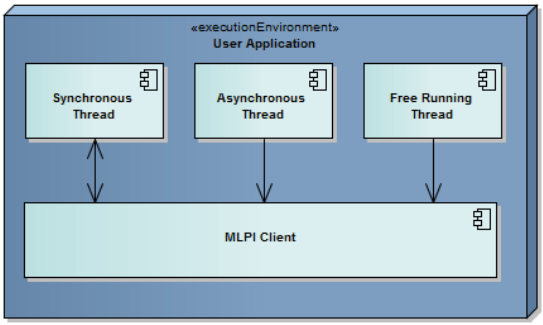

Bild 4: Anpassung der Anwenderapplikation an die zeitlichen Anforderungen

Software für performante Bewegungssteuerungen sollte typischerweise in C/C++ implementiert und auf dem Target ausgeführt werden, so dass die Aufrufe der synchronen Threads durch den MLPI-Client mit minimalen Laufzeiten ausgeführt werden. Solange die MLPI für das entsprechende Betriebssystem zur Verfügung gestellt wird, kann die Applikation aber auch in Visual Basic oder C# implementiert werden. Zurzeit unterstützt die MLPI die Betriebssysteme Microsoft Windows, Windriver VxWorks, Android und iOS.

Auf Windows Betriebssystemen kann die weit verbreitete und bekannte Entwicklungsumgebung Microsoft Visual Studio für die Applikationsentwicklung verwendet werden. Mit dieser Entwicklungsumgebung und der Unterstützung der genannten Programmiersprachen ist es naheliegend, dass objektorientierte Softwarearchitekturen eingesetzt und mit dem .NET Framework ein breites Spektrum an grafischen Benutzerschnittstellen für HMI-Geräte entwickelt werden können. Die MLPI stellt dazu entsprechend dem Microsoft Component-Object-Model (COM) ein COM-Interface zur Verfügung. Diese Schnittstelle ermöglicht die Implementierung von nicht-echtzeitfähigen Applikationen.

So lassen sich beispielsweise SCADA-Systeme (Supervisory Control and Data Acquisition) entwickeln, die über die MLPI Produktionsdaten aus dem Motion-Controller lesen und in einer Datenbank ablegen können. Die existierenden Schnittstellen, wie OPC (OLE for Process Control) oder OPC-UA (OPC Unified Architecture), stehen dem Anwender als Kommunikationstechnik zwischen Motion-Controller und SCADA System weiterhin zur Verfügung, allerdings bietet die MLPI hier weitere Möglichkeiten der Netzwerkanbindung.

Die MLPI ist für den wissenschaftlichen Bereich von besonderem Interesse, da sie neben den genannten Programmiermöglichkeiten eine effiziente und performante Schnittstelle zwischen einem Motion-Controller und einem Rapid-Prototyping System eröffnet. Eine konkrete Nutzung ist beispielsweise die Anbindung eines Motion-Controllers an LabVIEW (siehe Bild 5). So lassen sich mit Hilfe der in LabVIEW definierten grafischen Programmierung Anwendungen realisieren, die die Achsen eines industriellen Antriebssystems kommandieren. Für die Erstellung einer Motion-Logic Applikation stehen in Analogie zu den genannten Bibliotheken der MLPI

grafische Programmierelemente zur Verfügung. Die Entwicklungszeit und der Entwicklungsaufwand der für die grafische Programmierung einer LabVIEW Applikation erforderlich ist, ist weitaus geringer, als die Programmierung einer IEC 61131-3 Applikation für den Motion-Controller. Ein weiterer Vorteil ist die Nutzung der sehr umfangreichen Funktionalitäten, die durch die LabVIEW Entwicklungsumgebung zur Verfügung gestellt werden und üblicherweise nicht als Standardfunktionen in einer IEC Entwicklungsumgebung zur Verfügung stehen.

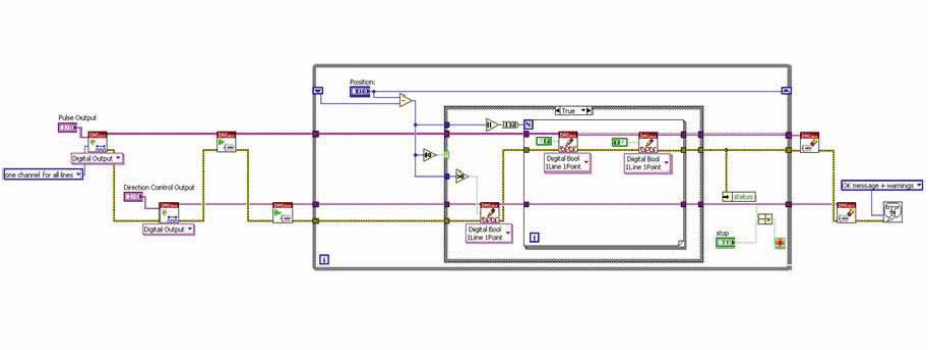

Bild 5: Erstellung einer Motion-Logic Applikation mit LabVIEW

8 Ausblick

Mit bereits verfügbaren Funktionalitäten und unterstützten Programmiersprachen bietet die MLPI einen zukunftsweisenden Ansatz Motion-Logic Applikationen mit Industrieantrieben zu realisieren. Dies ist einerseits vorteilhaft für die Forschung und Entwicklung an Hochschulen, da so Schnittstellen zwischen modernen Entwicklungswerkzeugen implementiert werden können. Andererseits bietet dies aber auch Chancen die Entwicklungszeiten beim Entwurf und Design von Sondermaschinen deutlich zu reduzieren.

9 Literatur

[1] Engels, E.; Krauskopf, S.: Innovation in Motion-Logic Programming – A versatile Interface; Konferenzbeitrag REM 2011, Kocaeli, Türkei, 2011

[2] PLCopen. Technical Specification – Function blocks for motion control. Version 1.1. PLCopen, 2007

[3] H. Lepers. SPS-Programmierung nach IEC 61131-3 - Mit Beispielen für CoDeSys und Step7. Franzis Verlag, 2007

[4] IEC 61491. Serial data link for real time communication between controls and drives – SERCOS, 2002

[5] OSADL. FAPI - Architektur. https://www.osadl.org/projects/svn/fieldbus-framework/trunks/fapia-trunk/, 09.07.2011

[6] PMC. Motion Control API. http://www.pmccorp.com/products/api.php#feat. 09.07.2011