



Boris Waldeck

IEC 61131 und C# im Verbund

Das Mischen der Programmier-Sprachen C# und IEC 61131-3 vereinfacht viele Automatisierungs-Projekte. Die Firma KW-Software hat eine solche Zusammenführung nun auf .NET-Basis in ihrem SPS-Laufzeitsystem umgesetzt.

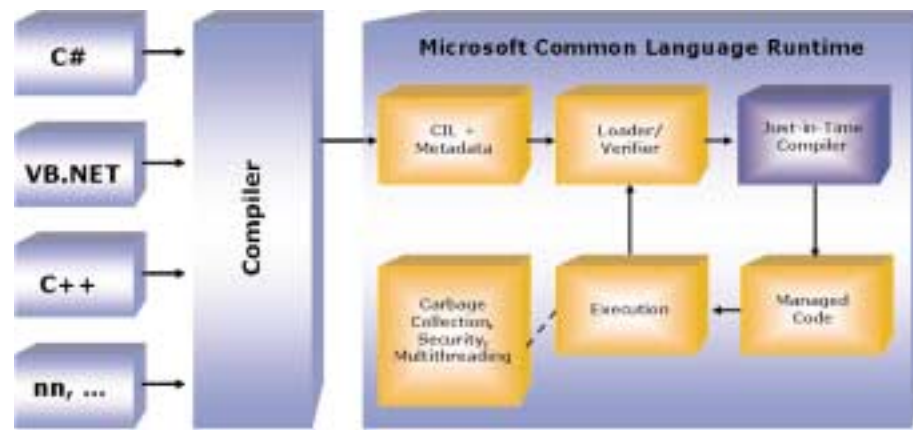
Die .NET-Plattform von Microsoft stellt mit der Common Language Infrastructure (CLI) eine allgemeine Basis zur Ausführung von .NET-Programmen her, die von den unterschiedlichsten Programmiersprachen verwendet wird. Wichtige Komponente dieses Konzepts ist die Laufzeitumgebung Common Language Runtime (CLR), die den nach ISO/IEC 23271 standardisierten Zwischencode Common Intermediate Language (CIL) ausführt. Erst zur Laufzeit übersetzt ein Just-in-Time-Compiler (JIT) den Zwischencode in managed code um und arbeitet diesen ab. Die Common Intermediate Language

(CIL) ist der standardisierte Zwischencode, der die Mischung von verschiedenen Programmiersprachen erlaubt. Komponenten einer Applikation können in unterschiedlichen Sprachen programmiert werden, wodurch das Nebeneinander mehrerer Programmiersprachen in einer Applikation möglich ist. Basis hierfür ist ein eigens definiertes sprachübergreifendes System von objektbasierten Datentypen, die das reibungslose Zusammenspiel zwischen Komponenten, die in verschiedenen Sprachen geschrieben wurden, gewährleisten. Es gibt heute schon rund 40 verschiedene Programmiersprachen die CIL erzeugen.

Echtzeit und .NET?

Die CLR geht mit dem Just-in-Time-Compiler (JIT) einen Mittelweg zwischen Interpretation und Compilierung des Programmcodes. Der JIT compiliert zur Laufzeit den benötigten Teil des Codes und die Garbage Collection sorgt für die automatische Speicherverwaltung. Dieses Programmierkonzept hat jedoch auch seinen Preis: Es werden mehr Ressourcen benötigt und die gesamte Ausführungszeit ist etwas langsamer als bei compiliertem Code. Die Garbage Collection gibt nicht mehr benötigten Speicher nicht direkt frei, sondern die CLR entscheidet über den Zeitpunkt; dadurch

(Grafiken: KW Software)



Die Common-Language-Runtime(CLR)-Systemarchitektur von Microsoft .NET – speziell für echtzeitkritische Aufgaben nicht zu empfehlen.

können die Antwortzeiten auf Ereignisse in Mitleidenschaft gezogen werden. Dies ist natürlich besonders für Embedded-Geräte nachteilig. Wenn es um die Erstellung Ressourcen-sparender oder Laufzeit-kritischer Embedded-Anwendungen geht, ist die Verwendung der Microsoft CLR speziell für Echtzeit-Aufgaben nicht zu empfehlen.

C# – die robuste Programmiersprache

Die wichtigste .NET-Programmiersprache ist C#. Sie ist die systemeigene Sprache für die .NET Common Language Runtime und fügt sich nahtlos in die .NET-Laufzeitumgebung ein. C# wurde für die Entwicklung robuster, langlebiger objektorientierter Komponenten konzipiert. Entwickler können leicht von C++ auf C# umsteigen, da die Syntax sehr ähnlich ist.

Die komponentenbezogenen Konzepte, beispielsweise Eigenschaften, Methoden und Ereignisse, stellen die Kernelemente der Sprache und der zugrundeliegenden Laufzeitumgebung dar. Deklarative Informationen (Attribute) können auf Komponenten angewendet werden, um anderen Bestandteilen des Systems Entwurfszeit- und Laufzeit-Informationen zu einer Komponente zu liefern. Die Dokumentation kann innerhalb der Komponente geschrieben und in XML exportiert werden.

C# bietet eine einfache, sichere und intuitive Umgebung, ist typensicher und bietet dem Programmierer Schutz vor nicht initialisierten Variablen, unsicheren Typumwandlungen und weiteren häufig auftretenden Programmierfehlern, wie etwa implizites Konvertieren von Bool nach Integer oder „Durchfallen“ bei „switch“-Anweisungen. Viele Fehler werden bereits beim Editieren und Compilieren im Visual Studio erkannt. Zusätzlich stellt C# den Programmierern eine Vielzahl von Features zur Verfügung, die die Arbeit erleichtern, wie beispielsweise ein weitreichendes Intellisense auch für Interfaces und Eventhandler.

Durch die Verbreitung von .NET und C# gibt es heute schon sehr viele weitere Tools rund um C#, die zur Testautomation, Gene-

Steuerungsebene

rieren von API-Hilfen, Messen von Performance oder dem Überprüfen von Design-Richtlinien eingesetzt werden.

Die echtzeitfähige Embedded-CLR

Die Microsoft CLR der .NET-Plattform ist nicht für Echtzeit-Anwendungen auf Embedded-Zielsystemen geeignet. Vor allem deshalb, weil die automatische Speicher-verwaltung (Garbage Collector) das Abarbeiten von Programmen in Echtzeit nicht zulässt. Für KW-Software war durch diese Problematik der Weg vorgezeichnet: Die Entwicklung einer kleinen, echtzeitfähigen CLR, basierend auf der weltweit akzeptierten .NET-Technologie, welche die Vorzüge der neuen Technologie mit bewährten Standards vereint und die Echtzeit-Fähigkeit für Embedded-Applikationen sichert.

Was genau verbirgt sich hinter diesem echtzeitfähigen und .NET-basierten Laufzeitsystem, dem KW-Software den Namen „Proconos embedded CLR“ verlieh?

Seit ihrer Einführung im Jahre 1993 ist die Norm IEC 61131 anerkannter Standard im Bereich der Automatisierung und der SPS-Programmierung. Einheitlich hinsichtlich der Struktur von Automatisierungsgeräten, den verwendeten Programmiersprachen und Datentypen, ermöglicht diese Norm die unabhängige Entwicklung von SPS-Anwendungen und sichert Portierbarkeit zu, um nur zwei der Vorzüge zu

nennen, die sich auf die IEC 61131 zurückführen lassen.

IEC 61131-Kompatibilität alleine reicht im .NET-Zeitalter jedoch nicht mehr aus. Gefordert ist heute unter anderem auch die Unterstützung der Programmiersprachen, die in der .NET-Welt gängig sind, allen voran C#.

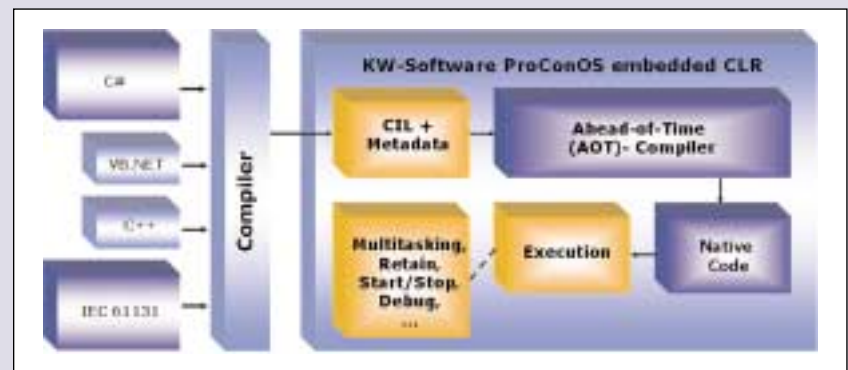
Proconos embedded CLR ist mit einer offenen Programmierschnittstelle auf Basis des .NET-eigenen Zwischencodes CIL ausgestattet. Damit kann die Embedded-CLR mit jeder .NET-Sprache programmiert werden, die in der Lage ist, CIL-Code zu generieren. Darüber hinaus ist das MS Visual Studio.NET durch vorgefertigte Templates erweitert worden, mit denen Projekte in C# als Firmware Library für das IEC-Programmiersystem Multiprog von KW-Software generiert werden. Im Programmiersystem eingebunden, können die in C# entwickelten Programmteile wie IEC-61131-Funktionen und -Funktionsbausteine verwendet werden. So wird es möglich

- ▷ im IEC-Programmiersystem von KW-Software IEC-Sprachen und beispielsweise C# innerhalb eines Projekts zu mischen;
- ▷ durch die Verwendung von C# portable Firmware zu entwickeln, dabei den gesamten Funktionsumfang und die Vorzüge der Hochsprache C# zu nutzen;
- ▷ weder auf die Vorzüge des IEC-Programmiersystems, noch auf die Effizienz

Die Echtzeit-Kriterien

Hausinterne Messungen bei KW-Software ergaben, dass die sehr schnelle Ausführungsmaschine der Embedded-CLR 1000 Anweisungen auf einem NIOS II 50 MHz in 150 µs, auf einem Pentium I 150 MHz in 24 µs oder einem AMD 2.200+ in 0,5 µs abarbeitet. Erreicht wird die Echtzeit-Fähigkeit durch den Einsatz eines Ahead-of-Time

(AOT)-Compilers: Im Unterschied zur Microsoft CLR wird dabei der Zwischencode CIL nicht erst bei Anforderung (Just in Time) in nativen Maschinencode compiliert, sondern dies geschieht bereits im Vorfeld (Ahead of Time), wodurch sich die Echtzeit und die Speicherverwaltung sicherstellen lässt.



Die Embedded-CLR-Systemarchitektur

Programmier-Beispiel

von MS Visual Studio verzichten zu müssen;

- ▷ eigene Tools und Sprachen die CIL erzeugen, zur Programmierung zu nutzen;
- ▷ die vorhandenen .Net-Tools zur Dokumentation, zum Generieren von API-Hilfen oder zum Überprüfen von Design-Richtlinien einzusetzen.

In der Embedded-CLR werden spezifische Eigenschaften der IEC 61131 oder der Embedded-Hardware über .NET-Attribute abgebildet (zum Beispiel [Function block], [RETAIN] oder [NATIVE]).

Diese in C# verfügbaren Attribute definieren das Systemverhalten und werden vom AOT-Compiler verwaltet.

Eine native Bibliothekenschnittstelle ermöglicht und verwaltet den Aufruf von nativem Code über .NET-Namespaces aus C#.

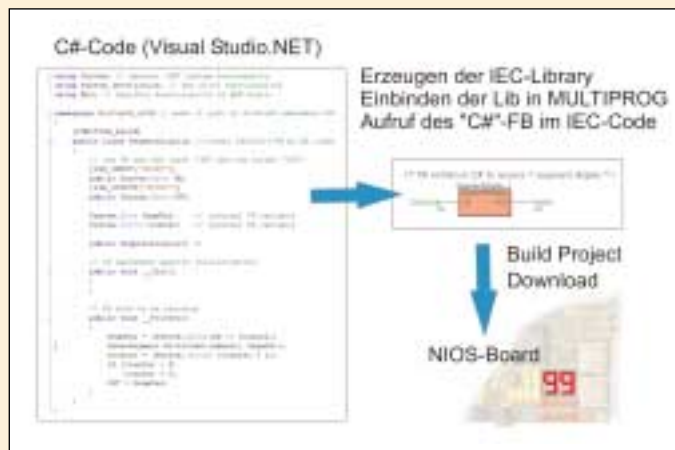
SPS-Funktionalität für Embedded-Systeme

Neben ihrer Echtzeit-Fähigkeit bringt die Embedded-CLR alle Funktionalitäten mit,

Ansteuerung einer 7-Segment-Anzeige

Für die Ansteuerung einer 7-Segment-Anzeige wird aus dem IEC-61131-FBD-Programm über einen C#-Funktionsbaustein „SegmentDisplay“ die Hardware angesteuert. Dieser Baustein wird aus einer mit dem Visual Studio.NET erzeugten Firmware-Library aufgerufen. Dabei

wird der Hardwarezugriff über die Namespaces von C# realisiert. Für die Klasse „SevenSegment“ mit der Methode „WriteCode“ wird über das Native Library Interface ohne Overhead ein performanter Funktionscall in den nativen Code erzeugt.



Managed und unmanaged Code

Unter Managed Code sind alle .NET-Programme zu verstehen, die zum Beispiel in C# programmiert sind und von der CLR ausgeführt und überwacht werden. Die CLR sorgt für die Freigabe von Speicher und stellt sicher, dass geschützte Speicherbereiche nicht direkt angesprochen oder überschrieben werden können. Der Code von nicht .NET-Programmiersprachen, insbesondere C/C++-Compilern für Embedded-Geräte wird als unmanaged bezeichnet.

die eine Embedded-Plattform zu einer SPS oder sogar zu einem Programmable Automation Controller (PAC) machen. So bildet die Embedded-CLR ein gemeinsames Laufzeitsystem, mit dem verschiedenste Automatisierungsanwendungen (SPS, CNC, HMI) auf einem Zielsystem kombiniert werden können. Zu den realisierbaren Funktionalitäten gehören:

- ▷ Multitasking;
- ▷ Unterstützung verschiedener Task-Typen, wodurch externe Ereignisse wie auch System-Events abgefangen werden können;
- ▷ Bereitstellung einer herstellerunabhängigen Geräteschnittstelle für Debug-, Diagnose- und Überwachungszwecke;
- ▷ eine definierte Zustandsmaschine, die ein Maximum an Transparenz und Sicherheit bietet;
- ▷ Retain-Handling, wodurch ein Warmstart auch nach Programm-Änderungen an remanenten Variablen möglich wird;
- ▷ Systemvariablen, die den direkten Zugriff auf die Hardware und somit das einfache Auslesen von Systeminformationen ermöglichen.

Die erwähnte Debug-Schnittstelle ermöglicht unter anderem den Download (und Upload) der SPS-Anwendung zum (vom) Zielsystem, das Starten (Heiß, Warm und Kalt) und Stoppen der SPS, das Anzeigen von Variablenwerten im Online-Modus, das Forcen und Überschreiben von Variablen und letztlich OPC-Funktionalität auch für C#-Variablen.

Viele mögliche Plattformen

Die Embedded-CLR lässt sich schnell auf unterschiedliche 16-Bit-, 32-Bit- oder 64-Bit-CPU-Systeme portieren. Firmware in C# wird plattformunabhängig entwickelt und ist somit ohne Zusatzaufwand einfach portierbar. Durch den Einsatz von C# für Embedded-Geräte lassen sich rund 80 % des Codes Hardware-unabhängig (managed) und nur zirka 20 % als Hardwarezugriff (Treiber) in C oder C++ (unmanaged) entwickeln. Das reduziert den Aufwand bei der Entwicklung der Gerätesoftware deutlich und ermöglicht das Zusammenfassen von unterschiedlichen Automatisierungsanwendungen auf einem Gerät.

Um die Portierbarkeit auf verschiedene Embedded-Plattformen nicht durch physikalische Grenzen einzuschränken, ist der geringe Speicherbedarf der Embedded-CLR ein weiterer bedeutender Aspekt: Ein Footprint von zirka 100 KByte ermöglicht den Einsatz auf allen heutigen Embedded-Systemen. *hap*



**Dipl.-Ing.
Boris Waldeck**

ist Sales and Marketing Director
bei KW-Software.