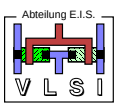
JControl – Einfache Implementierung und Evaluierung von eingebetteten Anwendungen mit JAVA

Tagungsbeitrag *10. E.I.S.-Workshop 2001*, Helge Böhme[1], Gerrit Telkamp[2]



[1]Abteilung E.I.S., TU BRAUNSCHWEIG
Gaußstraße 11
38106 Braunschweig
ϟ h.boehme@tu-bs.de
☎ 0531/391-3108, Fax 0531/391-5840

[2]Domologic Home-Automation GmbH
Rebenring 33
38106 Braunschweig
ϟ g.telkamp@domologic.de
☎ 0531/3804-340, Fax 0531/3804-342

Kurzfassung

JControl ist eine JAVA-Realisierung, die speziell für Steuerungsaufgaben konzipiert wurde, wie sie z. B. in der Haus- und Gebäudeleittechnik (Home-Automation) vorkommen. Eine nur 20KB große virtuelle JAVA-Maschine verfügt über alle für Steuerungsaufgaben notwendigen Eigenschaften (*Objektorientierung*, *Garbage-Collecton*, *Multithreading*, *Echtzeit* etc.). Besonderes Augenmerk lenkt dieses Papier auf die JControl-Entwicklungsumgebung: Da speziell vorgefertigte Klassenbibliotheken (APIs) die Ansteuerung von Peripheriekomponenten und Kommunikationsschnittstellen unterstützen, lässt sich der Implementierungsaufwand von Steuerungsanwendungen erheblich reduzieren. Dazu tragen auch eine eigene speziell darauf ausgelegte Entwicklungsoberfläche und die Möglichkeit der Simulation von eingebetteten Anwendungen auf dem Entwicklungssystem bei.

1 Motivation

Eingebettete Syteme sind zu einem festen Bestandteil unseres täglichen Lebens geworden. In einer Vielzahl einstmals rein mechanischer Geräte arbeiten heute Mikrocontroller, die die Herstellungskosten senken, die Zuverlässigkeit erhöhen und zudem den Funktionsumfang der Geräte steigern. Die Steuerung für diese Systeme hat in der Regel einen reaktiven Charakter und die Algorithmen dafür erfordern nur eine relativ geringe Rechenleistung, die bereits von *8-bit*-Mikrocontrollern erbracht werden kann. Viele Chip-Hersteller haben solche Controller im Lieferprogramm und arbeiten permanent an neuen Modellen. Es gibt ganze Produktfamilien mit zahlreichen auf dem Chip integrierten Peripherieeinheiten, z. B. Speicher, Ein-/Ausgabeports, Timer, A/D–D/A-Wandler, PWM, diverse serielle Schnittstellen und Bus-Systeme. Zumeist existiert eine Vielzahl von Varianten, die je nach Ausstattung für eine mehr oder weniger große Palette von Anwendungen einsetzbar sind.

Doch je universeller diese *Systems-on-Chip* sind, desto spezieller muss die Software auf die jeweilige Anwendung zugeschnitten werden. Um so ausgefallener ein Gerät ist und je geringer dessen geplante Stückzahl, um so stärker schlägt sich die Entwicklung dieser Software im Produktpreis nieder. Ferner fordert der Markt neue Funktionen von den Geräten in immer kürzeren Abständen, somit ist es von besonderer Bedeutung, nach effizienten Ansätzen für die Software-Entwicklung zu suchen.

Für gewöhnlich ist die Beschreibung der eigentlichen Anwendung (z. B. die Implementierung eines Regelalgorithmus) nur ein Teil des gesamten Entwicklungsaufwandes. Nicht zu vernachlässigen ist der Aufwand für die Ansteuerung der Peripheriekomponenten und der damit verbundenen weiteren Hardware, insbesondere dann, wenn der Mikrocontroller umfangreiche Funktionen anbietet und es auf ein spezielles Timing ankommt. Ein weiterer Schwierigkeitsfaktor ist die Anbindung an vorhandene Software-Module, wie z. B. an einen Protokoll-Stapel für ein spezielles Bus-System. Auch zukünftige Änderungen sind in einem solchen Programm schwierig (wenn z. B. zur Kommunikation eine andere Schnittstelle verwendet werden soll).

Unsere JControl-Technologie bietet die Möglichkeit, eingebettete Anwendungen in der Programmiersprache JAVA zu entwickeln und auf einem Mikrocontroller ablaufen zu lassen. Dabei kann dank des komponenten- und objektorientierten Konzepts von JAVA der Entwicklungsaufwand von Software für eingebettete Systeme minimiert werden. JControl stellt die Anwendung (also das eigentliche Problem) in den Vordergrund und hält sie dabei leicht änderbar und wartbar.

2 Die JControl-Plattform

JAVA hat sich in den letzten Jahren einen hervorragenden Stellenwert in der Software-Landschaft erobert, die Gründe dafür dürften bekannt sein. Wie Tabelle 1 zeigt, gibt es die JAVA-Plattform in zahlreichen Ausprägungen unterschiedlicher Komplexität von der JAVA2 Enterprise Edition für Server-Systeme bis hinunter zur JAVACard [5].

Unser JControl bietet eine Lösung zwischen der für eingebettete 16–32-bit-Systeme vorgesehenen JAVA2 Micro Edi-

Variante	Anwendungsgebiete	Anforderungen
J2EE	• Datenbanken • Webserver	GigaBytes Multiproz. @ GigaHz
J2SE	• Desktop-Anwendungen • Internet-Anwendungen	> MegaByte 32-bit-Proz. > 100 MHz
J2ME	• Set-Top-Boxen • PDAs	< MegaByte 16–32-bit-P. @ 100 MHz
JControl	• Embedded Control • Home-Automation	< 50 KiloBytes 8–32-bit-Proz. > 8 MHz
JavaCard	• Mobile Datenbank • Kryptographie	< 50 KiloBytes 8-bit-Prozessor < 4 MHz

Tabelle 1: Die verschiedenen JAVA-Varianten

tion und der JAVACard und ist dabei komplexitätsmäßig vergleichbar mit der JAVACard. JControl hat aber wegen der anderen Aufgabengebiete zusätzliche Eigenschaften, die von den komplexeren Implementierungen bekannt sind. Dies zeigt sich auch bei der verwendeten Hardware. Abbildung 1 zeigt z. B. eine JControl-Variante mit einer einfa-

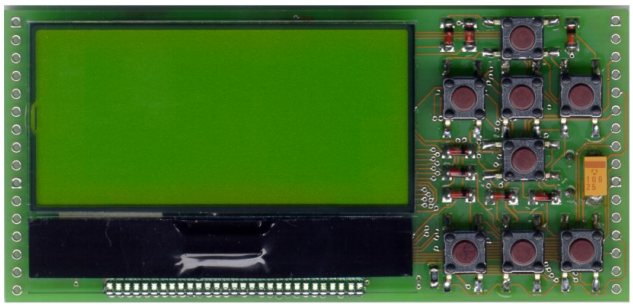


Abbildung 1: Eine JControl-Variante mit Grafik-Display

chen Tastatur und einem grafischen LC-Display mit 132x64 Pixel. Es existieren noch weitere Varianten, z. B. eine tastaturlose Version als Anzeigeterminal (elektronische Visitenkarte) oder mit zusätzlichen Peripherieeinheiten (CAN-Bus Anbindung). Als Basis für diese Modelle dient in jedem Fall ein 8-bit-Mikrocontroller. Der Stromverbrauch beträgt bei maximaler Taktfrequenz von 16 MHz lediglich 12mA.

2.1 Anwendungsbeispiel

JControl eignet sich hervorragend für die Programmierung einfacher Steuerungen. In einer konkreten Anwendung wurde das Steuermodul eines vernetzten Kühlschranks mit JControl entwickelt, wobei im Vergleich zu C nennenswerte Vereinfachungen bei der Software-Entwicklung erzielt werden konnten. Der Grund dafür liegt neben den grundsätzlichen Vorteilen von JAVA darin, dass Funktionen wie die Temperaturmessung, die Ansteuerung der Kühlmittelpumpe, eines Displays und des Netzwerks bereits zur Grundausstattung von JControl gehören. Diese Anwendung konnte mit etwas mehr als 200 Zeilen Quelltext implementiert werden. In der Vergangenheit wurde bereits eine vergleichbare Anwendung in C erstellt (allerdings ohne Display), wofür 1400 Zeilen Quelltext erforderlich waren.

2.2 Die JControlVM in aller Kürze

Der vom JAVA-Compiler generierte Bytecode wird in der Regel nicht direkt auf einem Prozessor ausgeführt, sondern von einer virtuellen Maschine interpretiert, so auch bei JControl. Da JControl auf eingebetteten Systemen mit nur wenigen KByte RAM zum Einsatz kommt, scheiden Optimierungen wie das Just-in-Time-Compiling aus. Nebenbei vereinfacht das Interpreter-Konzept noch einige Dinge, was sich z. B. beim Thread-Scheduling noch zeigen wird (siehe 2.2.2).

Die JControlVM wurde im Vergleich zu Desktop-VMs um einige Funktionen reduziert. Beispielsweise wurde auf Datentypen mit einer Größe von mehr als 16 Bit verzichtet, womit der RAM-Bedarf auf beinahe die Hälfte reduziert werden konnte. Ferner ist die JControlVM nicht auf ein Betriebssystem aufgesetzt, sondern beinhaltet selbst alle notwendigen Betriebssystem-Funktionen. Die Vorteile sind u. a. weniger Speicherverbrauch und schnellere Reaktionszeiten der Anwendung auf externe Ereignisse.

Die JControlVM ist eine spezifikationsgemäße JAVAVM, die über einen Mikrokernel mit lediglich 12KByte ROM-Bedarf verfügt. Mikrokernel heißt, dass lediglich die für die direkte Ausführung der Bytecodes benötigten Komponenten (*Execution-Engine*, *Heap-Manager* und *Thread-Scheduler*) vorhanden sind (siehe Abbildung 2). Das übri-

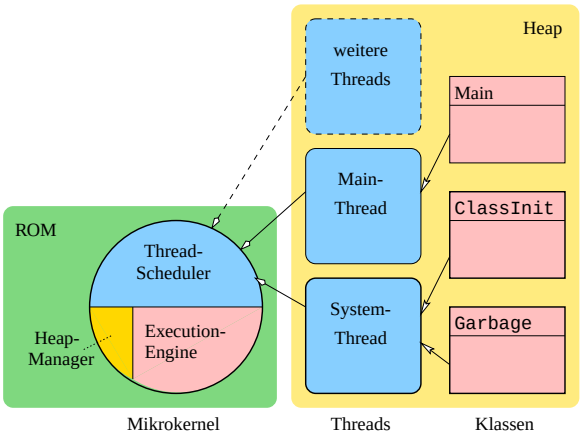


Abbildung 2: Der JControlVM-Systemaufbau

ge „Beiwerk“ (*Garbage-Collector*, *Klassen-Initialisierung* und *ROM-Filesystem*) wurde in einen System-Thread mit 8KByte ROM-Bedarf ausgelagert. Diese Elemente laufen im Kontext der virtuellen JAVA-Maschine und können somit auch deren Möglichkeiten nutzen. Der System-Thread ist der Anwendung gegenüber gleichberechtigt und arbeitet vollkommen nebenläufig. So können der Garbage-Collector und die Klassen-Initialisierung inkrementell arbeiten, d. h. sie unterbrechen die Anwendung immer nur für einen kurzen Moment, um jeweils einen kleinen Teil ihrer Aufgaben abzuarbeiten.

2.2.1 Speicherverwaltung

Die JControlVM ist für Systeme ab 2KByte RAM konzipiert. Im RAM werden nicht nur die JAVA-Objekte abgelegt, sondern auch weitere Laufzeit-Objekte wie z. B. Klassenzugriffstabellen und Threads. Um den knappen Speicher mit möglichst wenig Verschnitt zu nutzen, werden JAVA- und Laufzeitobjekte gleichberechtigt behandelt und in einem Unified Heap abgelegt. Dieser belegt fast das gesamte RAM und unterliegt komplett der kompaktierenden Garbage-Collection.

2.2.2 Thread-Scheduler

Die JAVAVM steuert selbst die Verteilung der Rechenzeit auf die einzelnen Threads. Dies hat den entscheidenden Vorteil, dass ein Wechsel *immer* nur zu einem kontrollierbaren Zeitpunkt stattfindet (bei der JControlVM ist dies stets der Übergang zwischen zwei Bytecodes). Somit kann auf eine Variablen-Zugriffskontrolle verzichtet werden (keine lokalen Kopien je Thread), da sie immer und überall einen gültigen Wert aufweisen.

Der Scheduler arbeitet mit einer im Millisekundenraster einstellbaren Zeitscheibe. Er ermöglicht neben den bei JAVA üblichen Prioritäten auch die Verarbeitung von Zeitschranken für einzelne Threads, die dynamisch zur Laufzeit gesetzt und verkettet werden können (EDF-Scheduling; EDF = Earliest Deadline First). Das erlaubt eine rudimentäre Echtzeitabarbeitung von Anwendungen. Bei der Überschreitung einer Zeitschranke wird eine Exception ausgelöst und die Anwendung kann geeignet darauf reagieren.

Jeder Thread verfügt über einen eigenen Speicherbereich auf dem Heap, in dem ein lokaler Operandenstack verwaltet wird. Neben dem JAVA-Stack werden dort auch die Laufzeitdaten aufgerufener Methoden und deren lokale Variablen abgelegt.

2.2.3 ROM-Filesystem

Bei der Ausführung von JControl-Anwendungen wird auf die benötigten Class-Files zurückgegriffen. Diese können – zusammen mit der JAVAVM – im ROM des Controllers abgelegt worden sein oder sich in einem internen oder externen Flash-Speicher befinden. Für den Zugriff auf einzelne Klassen und auf andere Ressourcen dient eine verkettete Liste als Filesystem, die zusätzlich noch weitere Informationen aufnehmen kann (z. B. Binde-Informationen bei Klassen, die dann nicht zur Laufzeit der Anwendung erzeugt werden müssen; siehe auch Abschnitt 4).

Die Programmiersoftware für den Flash-Speicher ist Teil des JControl-APIs und befindet sich im ROM des Controllers; neue Anwendungen werden z. B. über die RS232-kompatible serielle Schnittstelle programmiert. Hierbei ergibt sich eine klare Trennung zwischen den austauschbaren Anwendungen und der fest mit dem Controller (und der daran angeschlossenen Peripherie) verbundenen API-Klassen.

3 Das JControl-API

Da es Ziel von JControl ist, einen plattformunabhängigen Zugriff auf die Hardware des Controllers und die damit verbundene Peripherie zu erlangen, werden die Anwendungen von jeglichem Hardwarezugriff entbunden. Als Schnittstelle zur Außenwelt existiert das JControl-Framework, dem *alleine* der direkte Zugriff auf die Hardware gestattet ist. Dafür wurde ein Native-Interface entwickelt, mit dem controller-spezifischer Maschinencode direkt in die API-Klassen eingebettet werden kann (inline). Aus dem externen Flash ist es hingegen prinzipbedingt nicht möglich, nativen Code auszuführen, da dieser vom Mikrocontroller nicht als Programm- sondern als Datenspeicher angesprochen wird.

JControl unterstützt eine Untermenge der gewöhnlichen JAVA-Klassenbibliothek, z. B. aus den Paketen `java.lang` (`Object`, `Thread`, `Exception`, `String`) und `java.io`. Diese Untermenge garantiert die Kompatibilität zum JAVA-Sprachkonzept und stellt die von JAVA gewohnte Funktionalität zur Verfügung. Zusätzlich verfügt JControl über einen Satz eigenständiger API-Klassen, die auf die speziellen Eigenschaften von JControl zugeschnitten sind. Diese API-Klassen befinden sich in einem eigenständigen Paket `jcontrol`.

Zunächst existiert ein Satz von JControl-System-Klassen, die die Funktionalität der virtuellen Maschine erweitern bzw. als Ergänzung der JAVA-API-Klassen dienen. Beispielsweise weisen Threads bei JControl erweiterte Funktionalitäten auf (z. B. EDF-Scheduling, siehe 2.2.2), die die Klasse `java.lang.Thread` nicht unterstützt. Daher wurden diese in der Klasse `jcontrol.system.TimedThread` zusammengefasst. Hier ist es u. a. möglich, einen Thread mit einer Zeitschranke zu versehen oder den Scheduler zu steuern.

Die Klasse `jcontrol.system.Download` stellt die Schnittstelle zu einer Entwicklungsplattform dar. Sie wird automatisch aufgerufen, wenn keine Anwendung vorhanden ist oder wenn dies explizit (z. B. per Tastenkombination) angefordert wird und versetzt dann den Controller in den Downloadmodus (2.2.3). Die Klasse kann später auch explizit von der Anwendung aufgerufen werden, um eine Neu-Programmierung zu errreichen.

Zur Ansteuerung der mit dem Controller verbundenen Peripherieeinheiten existiert ein Satz Klassen im Paket `jcontrol.io`, die hierarchisch in drei Ebenen unterteilt sind:

Hardware-Ebene lediglich einfachste digitale Ein- und Ausgabefunktionen (z. B. für das Setzen oder Lesen einzelner Port-Pins oder von Bussen),

komponentenorientierte Ebene anwendungsorientierte Funktionen für die Steuerung elektronischer Komponenten (z. B. Display, Tastatur, Schnittstellen),

geräteorientierte Ebene zur Steuerung komplexer Geräte (z. B. Waschmaschinen, Kühlschränke etc.).

Die höheren Ebenen setzen auf die Implementierung der unteren Ebenen auf. Dem Applikationsprogrammierer stehen die Klassen aller drei Ebenen zur Verfügung, nicht jedoch, wie schon erwähnt, der direkte Zugriff auf die Hardware des Controllers.

Bei diesen Klassen wird konsequent das objektorientierte Konzept von JAVA umgesetzt. Ein Beispiel sind Streams: Ob nun Zeichen von der Tastatur entgegengenommen werden, von der seriellen Schnittstelle oder von einer Chip-Karte, ist für die Anwendung völlig unerheblich, genauso wie das Schreiben auf ein Display oder den CAN-Bus.

Die Ansteuerung und die Initialisierung der Hardware wird komplett von den API-Klassen übernommen. Bei der Erzeugung eines Objekts wird die Komponente initialisiert und bei der Freigabe (z. B. via Garbage-Collector) die entsprechende Hardware abgeschaltet. Für die Anwendung ist das völlig transparent.

4 Die JControl-Programmierumgebung

Bei der Konzeption von JControl wurde Wert darauf gelegt, dass prinzipiell zur Entwicklung von Anwendungen keine speziellen Werkzeuge benötigt werden. Erforderlich sind lediglich ein JAVA-Compiler, ein JAR-Archivierer und ein einfaches Terminal-Programm (für das Laden der Anwendungen auf den Controller). JControl unterstützt genau ein JAR-Archiv im Flash-Memory. Die zu startende Anwendung kann dabei – wie bei JARs üblich – mittels des eingebundenen Manifest-Files festgelegt werden. Dieses Verfahren garantiert eine größtmögliche Kompatibilität zum JDK. Um den Entwicklungsprozess zu vereinfachen, zu optimieren und komfortabler zu gestalten, wurden einige unterstützende Werkzeuge implementiert. Diese stehen auf allen Plattformen zur Verfügung, die JAVA2 und das grafische Toolkit Swing unterstützen.

4.1 Anwendungs-Entwicklung für JControl

Das wichtigste Werkzeug ist die integrierte Entwicklungsumgebung VJControl (siehe Abbildung 3). Diese ermöglicht nicht nur die Eingabe und Compilierung der JAVA-Anwendungen, sondern bindet auch die übrigen JControl-Werkzeuge zum Archivieren und Upload auf den Controller in eine komfortable Benutzeroberfläche ein. So steht z. B. ein Projekt-Management zur Verfügung, das die Zusammenstellung der Klassen, die für eine Anwendung benötigt werden, weitgehend automatisiert. Diese Klassen können mit wenigen Handgriffen in den Flash-Speicher des Controllers geladen werden.

Eine weitere Funktion der JControl-Werkzeuge ist die Unterstützung des speziellen JCA-Formats. Das eigenständige JControl-Archiv-Format ist nicht nur kompakter als unkomprimierte JAR-Archive, sondern es erlaubt auch das Laden von mehreren APIs und Anwendungen auf den Controller (und ein selektives Löschen einzelner Archive), da es speziell auf die Architektur des Flash-Speichers ausgelegt ist. Der JControl-Archivierier arbeitet dabei implizit optimierend, d. h. durch Weglassen irrelevanter Information in den Class-Files werden diese verkürzt, ohne dass sie an Funktionalität verlieren. Auf dem Entwicklungsrechner kann noch eine zweite Version des Archivs (als JAR) mit kompletten Debugging-Informationen erzeugt werden. Dies kann dann für den JAVA-Compiler als Grundlage dienen, für den Fall dass auf dem Controller geladene APIs von zukünftigen Anwendungen benutzt werden sollen.

Schließlich wird noch die Möglichkeit gegeben, die Klassen vorab um Binde-Informationen zu erweitern (Vorverlinkung). Die Klassen gewinnen in diesem Fall zwar etwas an Größe und belegen mehr Flash-Speicher bzw. ROM, die Symbolreferenztabellen müssen dann aber nicht zur Laufzeit im knappen RAM abgelegt werden. Ein positiver Nebeneffekt dabei ist ein schnellerer Zugriff der JControlVM auf Klassen-Elemente, da die symbolischen Referenzen beim ersten Zugriff nicht aufwändig ermittelt werden müssen.

4.2 Simulation, Verifikation und Präsentation

Ein wichtiger Aspekt bei der Entwicklung von Anwendungen für eingebettete Systeme im Allgemeinen ist die Testbarkeit auf dem Entwicklungssystem. Die Plattformunabhängigkeit von JAVA bietet hier einige Möglichkeiten.

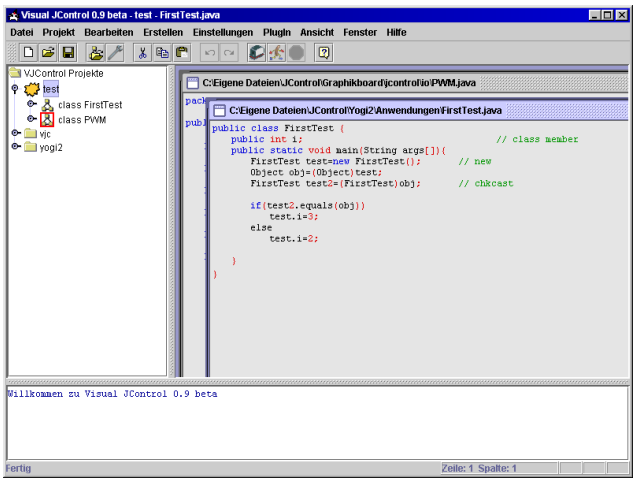


Abbildung 3: Die JControl Entwicklungsumgebung

So existiert auch ein JControl-Applet, das die JControl-Hardware grafisch darstellt und für den Anwender ein API mit derselben Programmierschnittstelle zur Verfügung stellt. Dies kann dann dieselben JControl-Anwendungen ausführen, wie die reale Hardware (siehe Abbildung 4). Natürlich kann auf diese Weise nicht die mit dem Controller

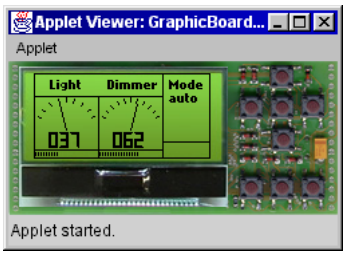


Abbildung 4: Eine simulierte JControl-Anwendung

verbundene Hardware getestet und deren Interaktion überprüft werden. Die Mensch-Maschine-Schittstelle (Tastatur, Display etc.) lässt sich so jedoch viel schneller perfektionieren, da die Test-Zyklen kürzer werden.Nebenbei können diese eigentlich zur Simulation bestimmten Anwendungen auch dazu dienen, die Produkte auf Web-Pages zu präsentieren [1].

Sollen reale Peripherieeinheiten in den Simulationsvorgang einbezogen werden, muss in jedem Fall die Anwendung direkt auf dem Controller ausgeführt werden. Tritt dann dort ein Fehler auf, so ist dessen Lokalisation gewöhnlich schwierig. Daher wird es zukünftig mit JControl auch möglich sein, Remote-Debugging durchzuführen. Dabei überträgt die virtuelle Maschine kontinuierlich oder auf Anfrage ihren Status z. B. über die serielle Schnittstelle an den Host. Auf diesem befindet sich ein Debugger-Frontend, das in die VJControl-Oberfläche integriert ist und den momentanen Status der virtuellen Maschine visualisiert. Zur Verwendung kommt dabei ein reduziertes Protokoll gemäß der JPDA-Spezifikation [8].

5 Fazit

JControl setzt die Programmiersprache JAVA auf sehr kompakte Mikrocontroller-Systeme um. Dadurch eignet sich JControl auch für Anwendungen, bei denen es besonders auf einen günstigen Preis, kleine Ausmaße und niedrigen Stromverbrauch ankommt. Sogar Single-Chip-Systeme sind realisierbar. Trotz der extremen Kompaktheit bietet JControl aber dennoch die Vorteile der Programmiersprache JAVA. Das JControl-API realisiert eine plattformunabhängige Schnittstelle zur Implementierung von Steuerungen. Die Entwicklung von Anwendungen kann mit gewohnten Werkzeugen erfolgen, wird aber auch mit eigens dafür zugeschnittenen Werkzeugen unterstützt. Die Software ist sowohl auf dem Entwicklungsrechner wie auf dem eingebetteten System verifizierbar und kann ohne Aufwand im Internet präsentiert werden.

Literatur

[1] JControl-Homepages:
`http://www.jcontrol.de`
`http://www.jcontrol.org`

[2] Böhme, Helge; Telkamp, Gerrit; Embedded Control mit JAVA; Embedded Intelligence 2001, 14. bis 16. Februar 2001, Nürnberg

[3] Böhme, Helge; Telkamp, Gerrit; Eine JAVAVM für Anwendungen in der Home Automation; 9. E.I.S.-Workshop, 22. bis 24. September 1999, Darmstadt

[4] Böhme, Helge; Konzeption und Implementierung einer virtuellen JAVA-Maschine auf einem 8-bit-Mikrocontroller für Steuerungsaufgaben in den Home-Automation; Diplomarbeit, Abteilung E.I.S., Technische Universität Braunschweig, 1998

[5] JAVA™ Technology Products & APIs
`http://java.sun.com/products/`

[6] Lindholm, Tim; Yellin, Frank; JAVA™ Die Spezifikation der virtuellen Maschine, Die offizielle Dokumentation von JAVASOFT; Addison-Wesley, 1997
`http://java.sun.com/docs/books/vmspec`

[7] Gosling, James; Joy, Bill; Steele, Guy; The JAVA™ Language Specification; Addison-Wesley, 1996
`http://java.sun.com/docs/books/jls`

[8] The JAVA™ Platform Debugger Architecture (JPDA):
`http://java.sun.com/j2se/1.3/docs/guide/jpda/index.html`