Kostenoptimierte Lösungen für frei konfigurierbare Touch-Panels und IP-Gateways mit JControl

Intuitive Bedieneroberflächen mit Touch-Panels sowie die Integration in vorhandene Kommunikationsstrukturen (z.B. über IP-Gateways) sind Anforderungen, denen heute viele Geräte und Systeme gewachsen sein müssen. Prinzipiell verfolgen beide Anwendungen das gleiche Ziel: Sie realisieren eine möglichst benutzerfreundliche Steuerungs- und Überwachungsebene für ein oder mehrere Geräte, ob lokal oder aus der Ferne. Dabei können die zu steuernden Geräte direkt oder auch indirekt (z.B. über einen Feldbus) an das Touch-Panel oder IP-Gateway angeschlossen werden. Während es Anwendungsbereiche gibt, in denen die Geräte-Umgebung fest vorgegeben ist (z.B. in der Automobiltechnik), existieren andere Anwendungsbereiche, bei denen die Anzahl und die Art der zu steuernden Geräte und Funktionen von Anwendungsfall zu Anwendungsfall variieren. Dies erfordert flexible Konfigurationskonzepte, die es ermöglichen, dass das Touch-Panel oder IP-Gateway ohne großen Aufwand an die jeweilige Geräteumgebung angepasst werden kann.

Dieses Papier diskutiert unterschiedliche Ansätze zur Konfiguration von Geräten und stellt eine hochgradig flexible und gleichzeitig extrem ressourcensparende Lösung vor, die auf JAVA-Bytecode-Generatoren basiert.

Konfigurierbare Geräte

Bild 1 zeigt beispielhaft das Blockschaltbild eines IP-Gateways als Embedded System, das für die Anbindung beliebiger CAN-Bussysteme an das Internet konzipiert wurde. In sehr ähnlicher Weise ließen sich auch Gateways für EIB- oder LON-Netzwerke entwerfen. Das Schaltungskonzept basiert auf dem relativ neuen ColdFIRE-Prozessor MCF5280 von Motorola, der z.B. eine CAN- und eine Ethenet-Schnittstelle mit auf dem Controller integriert, was ausgesprochen kompakte und vor allem auch kostengünstige Embedded Systems ermöglicht. Aufgrund der hohen Integrationsdichte der Einzelkomponenten ist das Schaltungskonzept weitgehend vorgezeichnet – natürlich könnte es noch bzgl. der Stromversorgung oder Speicherausstattung leicht variiert werden, aber dennoch ist ein und die selbe Hardware für ein breites Spektrum unterschiedlicher Anwendungen geeignet; beispielsweise für Heizungsanlagen, für die Automobiltechnik oder für Industriesteuerungen.

Komplizierter wird die Sache jedoch wenn es um die Software geht, denn hier müssen spezielle Anwendungsvarianten berücksichtigt werden. Egal ob IP-Gateways oder Touch-Panels – die Art, Anzahl und Konfiguration der zu steuernden Geräte spielt dabei ebenso eine Rolle wie das grafische Design der Benutzeroberfläche. Um nicht für jeden einzelnen Anwendungsfall eine neue Software implementieren zu müssen, ließe sich mit einer *konfigurierbaren Software* ein gewisses Spektrum an Anwendungsvarianten abdecken.

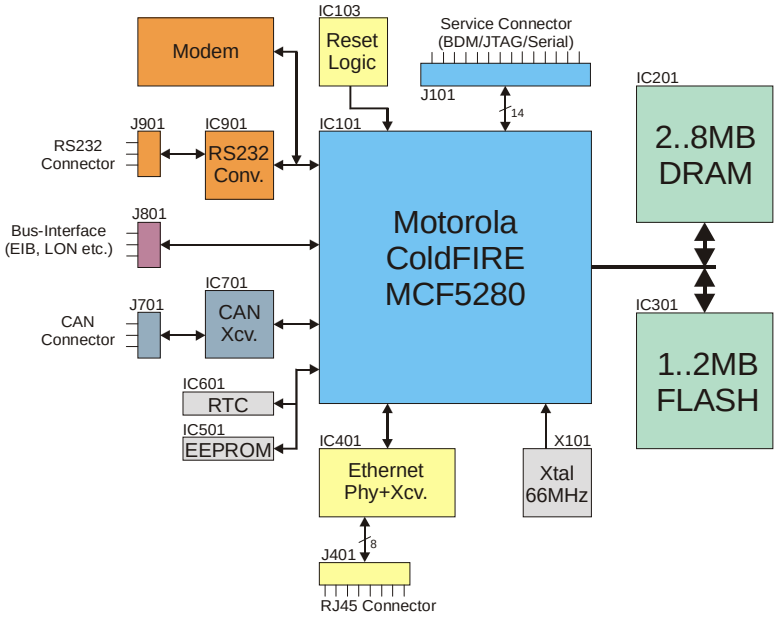

Bild 1: Blockschaltbild eines kostenoptimierten IP-Gateways auf Basis des Motorola ColdFIRE MCF5280

Während es Bereiche gibt, in denen sich unterschiedliche Anwendungsvarianten immer noch relativ ähnlich sind und es leicht möglich ist, dass Laien z.B. mit einem integrierten Konfigurationsmenü eine Standard-Software an unterschiedliche Anforderungen anpassen, so gibt es andere Anwendungsbereiche, in denen nur sehr wenig Gemeinsamkeiten zu finden sind und eine Standard-Software nicht ausreicht, um alle sinnvollen Anwendungsvarianten vorzusehen.

Ein Beispiel für einen sehr komplexen Anwendungsbereich ist die Home-Automation. Hier ist zu erwarten, dass in nahezu jeder Installation andere Geräte von unterschiedlichen Herstellern in anderen Konfigurationen und an verschiedenen Orten eingesetzt

werden. Erschwerend kommt hinzu, dass zumeist auch nachträglich noch Geräte unterstützt werden sollen, die bei der Entwicklung des IP-Gateways oder Touch-Panels noch gar nicht bekannt waren. Es muß für derart komplexe Anwendungsbereiche nach Konfigurationsmöglichkeiten gesucht werden, die einfach handhabbar sind (möglichst auch vom Laien), aber dennoch genügend Flexibilität bieten, um damit eine große Zahl unterschiedlicher Anwendungen realisieren zu können.

Konfigurationsgrad und Konfigurationsaufwand

Mit zunehmenden Möglichkeiten, die eine Technologie vorsieht (*Konfigurationsgrad*) steigt auch der Aufwand, mit dem sich ein bestimmtes Problem lösen läßt (und die damit verbundenen Kosten, der *Konfigurationsaufwand*). Im Extremfall müßte für jedes einzelne Gerät eine neue Betriebssoftware entwickelt werden, was zwar ein Höchstmaß an Flexibilität bedeutet, jedoch in nur ganz wenigen Anwendungsbereichen überhaupt finanzierbar ist. Insbesondere für den Endkundenmarkt sind andere, vor allem kostengünstigere Lösungsansätze gefragt.

Konfigurationsmenüs, Konfigurationstabellen, Skriptsprachen oder Programmiersprachen sind mögliche Ebenen zur Konfiguration eines Gerätes (siehe Bild 2). Konfigurationsmenüs beispielsweise lassen sich einfacher bedienen als Tabellen oder Skriptsprachen, allerdings ist der Konfigurationsgrad auch eingeschränkt. Das gilt vor allem dann, wenn aus Kostengründen nur eine einfache Bedieneroberfläche auf dem Gerät realisiert werden kann. In der Praxis kommen die ersten drei Technologien schnell an ihre Grenzen – sie sind entweder in ihrer Anwendung zu kompliziert oder sie sind zu unflexibel. Oft zeigt sich auch später erst, dass eine besondere Funktion gewünscht wird, die bei der Geräte-Entwicklung noch gar nicht absehbar war. Insbesondere offene

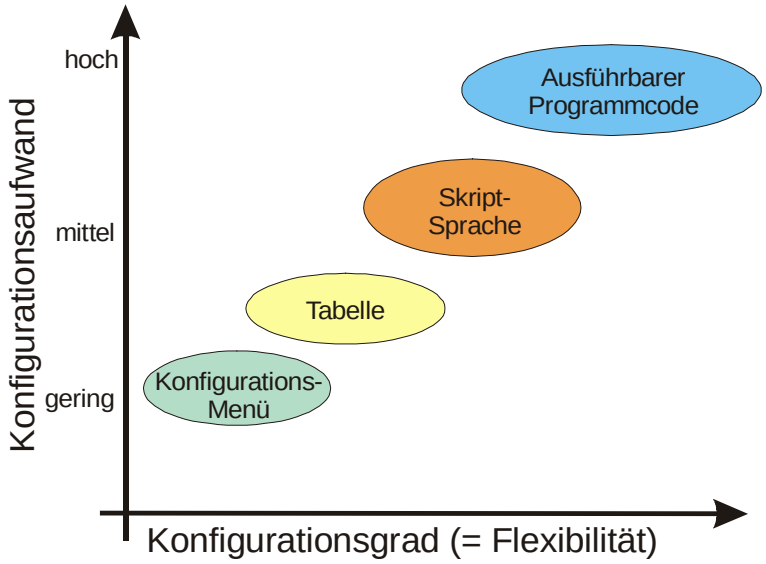

Bild 2: Zusammenhang zwischen Konfigurationsaufwand und Konfigurationsgrad: Je flexibler eine Technologie ist, desto aufwendiger ist es, ein bestimmtes Problem damit zu lösen

Feldbussysteme, über die zu steuernde Geräte an das Touch-Panel oder IP-Gateway angeschlossen werden sollen, stellen sehr hohe Anforderungen an die Konfigurierbarkeit – denn neue busfähige Geräte sollen nach Möglichkeit auch nachträglich noch in die Steuerung integriert werden können, selbst wenn sie über völlig neuartige Funktionen verfügen (besondere Benutzerschnittstellen oder spezielle Busobjekte). Dazu müssten dann ggf. die Tabellenstruktur oder Skriptsprache erweitert, eine neue Firmware eingespielt und zusätzlich noch alle Projektdateien angepaßt werden, was sehr zeit- und kostenintensiv ist.

Konfigurations-Software

Mit einer speziellen Konfigurations-Software, die nicht auf dem zu konfigurierenden Gerät selbst sondern auf einem externen PC ausgeführt wird, läßt sich der Konfigurationsaufwand deutlich reduzieren (siehe Bild 3). Auf einem PC steht mit Windows oder Linux eine leistungsfähige grafische Bedieneroberfläche zur Verfügung, die auf einem kostenoptimierten und damit eingeschränkten System in der Form nicht realisierbar ist. Außerdem lassen sich automatische Konfigurationsmechanismen für einen PC deutlich leichter implementieren als für ein Embedded System. Zur Inbetriebnahme würde das Touch-Panel oder IP-Gateway dann über eine einfache serielle Schnittstelle oder per USB am PC angeschlossen, an dem die Einstellungen vorgenommen und anschließend übertragen werden. Für die Konfiguration von Telefonanlagen beispielsweise sind solche PC-gestützten Lösungen weit verbreitet.

Als Format für die Konfigurationsdaten, die mit PC-Unterstützung erstellt werden könnten, lässt sich auf Tabellenstrukturen, Skriptsprachen oder ausführbare Programmcodes zurückgreifen. Größtmögliche Flexibilität bieten auch hier die ausführbaren Programmcodes, die ein im Konfigurations-Werkzeug integrierter Code-Generator automatisch erzeugt. Damit können die Vorteile eines geringen Konfigurationsaufwandes mit den Vorteilen größtmöglicher Flexibilität kombiniert werden: Der Anwender nutzt die Konfigurations-Software am PC wie jedes andere Programm auch und muss gar nicht realisieren, dass er mit seinem Tun mehr oder weniger komplexe Programmstrukturen erzeugt. Auf diese Weise können per Mausklick einzelne Bedienelemente für bestimme Geräte oder sogar ganze grafische Benutzeroberflächen ausgewählt

und zusammengestellt werden. Auch die Integration spezieller Funktionen wie Konvertierungsroutinen für Meßwerte, Zeitschaltprogramme, Alarmüberwachungen oder automatische Steuerungen (sog. *Logik-Module*) sind mit ausführbaren Programmcodes sogar nachträglich noch möglich.

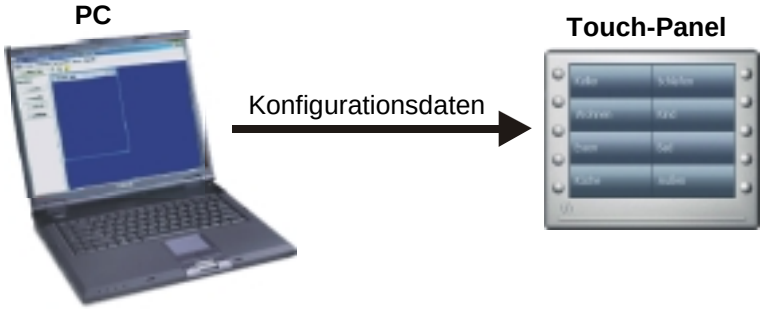

Bild 3: Eine anwendungsspezifische Konfiguration für das Touch-Panel wird mit PC-Unterstützung erstellt

Interpreter und Zwischensprachen

Die Geräte-Konfiguration auf Basis einer vom System direkt ausgeführten Maschinensprache durchzuführen wäre ausgesprochen umständlich. Einerseits lassen sich auf dieser Ebene separate Programmteile nur mit unverhältnismäßig hohem Aufwand „verketten“ (es müßten z.B. binäre Sprungziele verändert werden), andererseits ist eine umfangreiche Funktionssicherheit kaum zu gewährleisten, da nachgeladene Konfigurationsdaten theoretisch Zugriff auf alle Systemressourcen haben (zumindest wenn auf die Hardware-Unterstützung einer MMU o.ä. verzichtet wird). Ein weiteres Problem ist, dass mit einem Wechsel der Prozessortechnologie nicht nur die Firmware, sondern auch größere Abschnitte der Konfigurations-Software angepaßt werden müssten.

Diese Nachteile lassen sich sehr elegant mit einem Interpreter umgehen, der in die Firmware des Embedded Systems integriert wird und eine universelle *Zwischensprache* ausführt. Diese Zwischensprache unterscheidet sich von Skriptsprachen dadurch, dass sie wirklich universell ist und damit geeignet, alle denkbaren konfigurierbaren Vorgänge zu formulieren. Ein entsprechender Interpreter ließe sich für jeden beliebigen Prozessor implementieren, so dass ein Wechsel der Prozessortechnologie unter Beibehaltung der Konfigurations-Software möglich wäre. Unerlaubte Vorgänge in den Konfigurationsdaten, wie z.B. ununterbrochenes Senden auf dem Bus, könnten vom Interpreter abgefangen werden. Damit ließe sich ein hohes Maß an Funktionssicherheit gewährleisten.

Ein Interpreter für eine universelle Zwischensprache kann auch als *virtuelle Maschine* bezeichnet werden, da er sich nach außen wie ein Mikroprozessor verhält, obwohl es sich genau genommen um eine Software handelt. Die objektorientierte Programmiersprache JAVA verwendet eine solche virtuelle Maschine für die Ausführung der JAVA-Programme, die sog. *JAVA Virtual Machine*. JAVA-Quelltexte werden mit Hilfe eines Compilers in die Zwischensprache (den *JAVA-Bytecode*) umgesetzt, die anschließend von der virtuellen Maschine ausgeführt werden kann. Das JAVA-Konzept eignet sich aus vielerlei Gründen hervorragend für die Programmierung und Konfiguration eingebetteter Systeme:

- *Modularität*: JAVA bietet auf der Bytecode-Ebene ausgefeilte Mechanismen zur Verkettung separater Programmteile (Klassen), so dass einzelne Software-Module problemlos nachgeladen und funktional verknüpft werden können
- *Entwicklungswerkzeuge:* Leistungsfähige Entwicklungswerkzeuge mit Debug-Möglichkeiten (Compiler und IDEs) werden kostenlos im Internet angeboten. Damit können schon während der Entwicklungsphase einzelne Software-Module in Hochsprache entwickelt und getestet werden.
- *Erweiterungen:* Neben der eigentlichen Programmiersprache bietet JAVA automatische Speicherbereinigung (Garbage Collection), Multi-Threading sowie sehr umfangreiche Klassenbibliotheken, was die Software-Entwicklung deutlich vereinfacht
- *Plattformunabhängigkeit:* Die JAVA Virtual Machine kann für völlig unterschiedliche Systeme implementiertwerden, was die Unabhängigkeit von der jeweiligen Hardware gewährleistet (Hardware-Abstraktion). Dadurch wird ein deutlich längerer Lebenszyklus der Software-Konzepte erreicht.
- *Verbreitung:* JAVA hat in den vergangenen Jahren eine große Verbreitung erfahren, so dass auch Dritte (betriebsfremde Personen) wesentlich unkomplizierter einzelne Software-Module beisteuern können (*Third Party Programming*). Selbst komplexe interne Funktionen lassen sich einfach über vorgegebene Schnittstellen, sog. APIs (Application Programming Interfaces), bedienen, ohne dass hierfür Systeminternas offengelegt werden müssen.

Die Implementierung eines JAVA-Bytecode-Generators erfordert die Kenntnis der virtuellen JAVA-Maschine. Eine entsprechende Spezifikation wird von der JAVA-Community kostenlos im Internet zur Verfügung gestellt. Aufgrund der sehr leistungsfähigen Mechanismen ist es deutlich leichter, einen Code-Generator für den JAVA-Bytecode zu entwickeln als für eine prozessorspezifische Maschinensprache.

Bild 4 zeigt, wie die Konfigurations-Software für ein Touch-Panel zur Steuerung von Bus-Systemen strukturiert sein könnte. Der Anwender erstellt zunächst mit Hilfe der grafischen Benutzeroberfläche die von ihm gewünschten Menüstrukturen. Dazu wählt er die einzelnen Schaltflächen mit der Maus aus, plaziert sie auf dem Bildschirm und beschriftet sie. Um die Arbeit weitmöglichst zu vereinfachen, können die verfügbaren Geräte und Funktionen automatisch aus vorhandenen Projektdateien und Produktdatenbanken extrahiert und vorschlagen werden. Ggf. lassen sich die Informationen sogar direkt über ein am PC angeschlossenes Bus-Interface aus den vernetzten Geräten auslesen. Sobald der Anwender die Konfiguration fertiggestellt hat, generiert die Software eine individuelle Geräte-Konfigurationsdatei in JAVA-Bytecode und lädt diese anschließend in das Touch-Panel.

JAVA-Profile

Für diverse Anwendungsbereiche hat die JAVA-Community unterschiedliche Profile definiert, die alle auf der virtuellen JAVA-Maschine basieren, sich aber in Art und Umfang der Klassenbibliotheken unterscheiden. So gibt es z.B. ein JAVA-Profil für mobile Geräte wie Handys (*Micro Edition*), für Applikationen im Desktop-Bereich (*Standard Edition*) oder auch für Server-Anwendungen (*Enterprise Edition*). Die Klassenbibliotheken spezifizieren die Software-Grundausstattung für ein System, z.B. Funktionen für grafische Benutzeroberflächen, diverse Eingabemedien, Netzwerkzugang etc. Sie sollen Kompatibilität garantieren zwischen Geräten, die ein und das selbe Profil implementieren.

Es gibt allerdings auch viele Anwendungsbereiche, für die kein optimales JAVA-Profil existiert. Das ist insbesondere bei kostenoptimierten Systemen der Fall, bei denen Speicherausstattung und Prozessor-Leistung möglichst gering gehalten werden müssen – hier sind die Standard-Profile oft viel zu überladen. Nicht umsonst werden die Eigenschaften „langsam“ und „hoher Speicherverbrauch“ leicht mit JAVA in Verbindung gebracht – allerdings zu unrecht.

JControl

JControl ist eine JAVA-basierte Technologie, die auf kostenoptimierte Systeme zugeschnitten wurde. Das zentrale Element ist die *JControl Virtual Machine*, ein auf Geschwindigkeit optimierter und sehr kompakter JAVA Bytecode-Interpreter, der in der 32-Bit-Variante zusammen mit den Klassenbibliotheken für ein Touch-Panel nur ca. 280kByte belegt (Motorola ColdFIRE unter uClinux). Einsatzfähige Geräte können mit einer Speicherausstattung ab 2MB DRAM realisiert werden (inkl. Betriebssystem und Arbeitsspeicher). Die 8-Bit-Minimal-Variante belegt inkl. der wichtigsten Klassenbibliotheken (Taster, grafisches LC-Display, Schnittstellen etc.) nur ca. 60kByte.

Neben der kompakten virtuellen Maschine umfaßt das JControl-Konzept leistungsfähige Entwicklungswerkzeuge. Dazu gehört das Projektverwaltungs-Werkzeug *JCManager*, mit dem Projekte zusammengestellt und in das Embedded System geladen werden können. Ein Simulator ermöglicht die exakte Simulation der Geräte schon während der Entwicklungsphase am PC (Bild 5). Außerdem stehen Editoren für Bilder, Schriftsätze und Melodien zur Verfügung, die auch spezielle gerätespezifische Dateiformate und Display-Auflösungen unterstützen.

JControl bietet Speicherplatz-optimierte Klassenbibliotheken für grafische Benutzeroberflächen, Web-basierte Anwendungen oder Feldbussysteme, was auch bei der Entwicklung von Touch-Panels oder IP-Gateways zugute kommt. Eine weitere Besonderheit ist die Unterstützung weicher Echtzeit-Verarbeitung. Außerdem steht ein spezieller Compiler zur Verfügung, der XML-basierte Beschreibungsdateien (z.B. von grafischen Benutzeroberflächen und Busfunktionen) direkt in ausführbaren JAVA-Bytecode übersetzt.

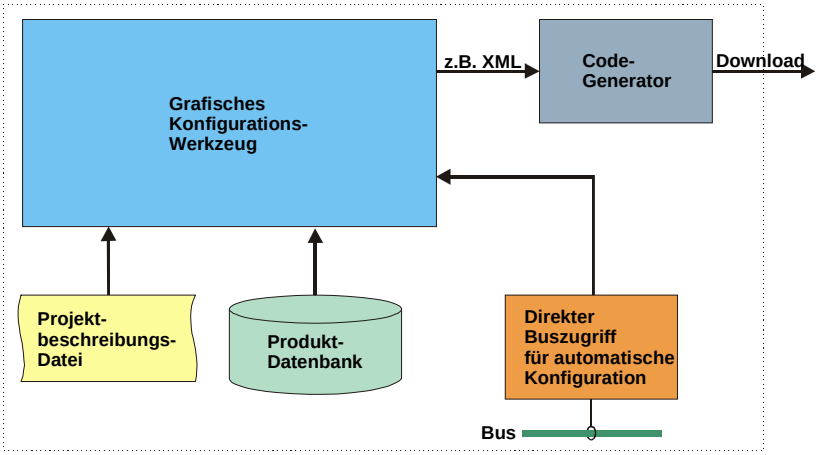

Bild 4: Struktur einer Konfigurations-Software für Touch-Panels

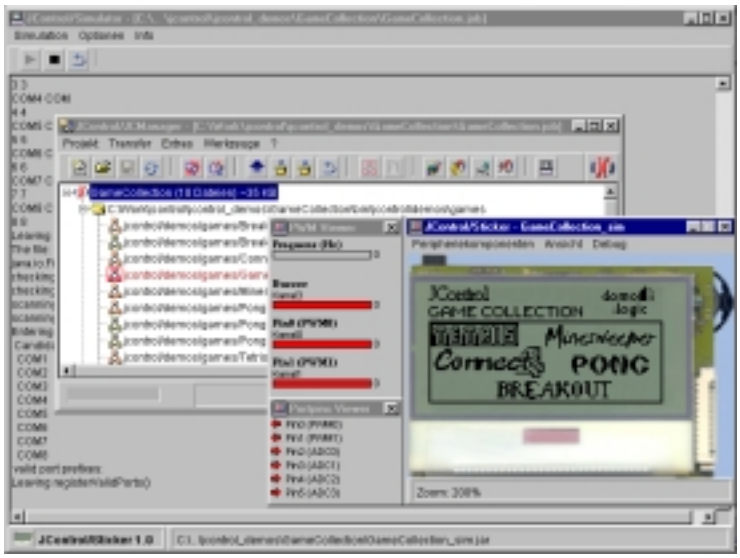

Bild 5: Screenshot einer JControl-Simulator-Sitzung bei der Simulation eines einfachen Gerätes mit grafischem LC-Display (Auflösung 128x64 Pixel)

Betriebssoftware

Bild 6 zeigt vereinfacht die Struktur der Anwendungssoftware, so wie sie bereits für ein Touch-Panel realisiert wurde. Neben der vom Code-Generator speziell erzeugten Anwendung können z.B. ein Zeitschaltprogramm sowie ein sog. Logik-Modul in das Gerät geladen werden (oberer Bereich). Werkseitig sind außer dem Betriebssystem (hier: uClinux) und der JControl Virtual Machine eine Basis-Klassenbibliothek sowie Funktionen für die grafische Benutzeroberfläche und die Bus-Anbindung vorgesehen. Alle separaten Software-Module werden von der virtuellen Maschine automatisch zur Laufzeit über festgelegte Namen verkettet, so dass hier keine weiteren Schritte notwendig sind.

Die Software für ein IP-Gateway ließe sich in ganz ähnlicher Weise konzipieren, mit dem Unterschied, dass anstelle der grafischen Benutzeroberfläche ein HTML-Seitengenerator vorgesehen werden müßte. Auf Anforderung von außen (über das Internet) könnte dieser dann eine spezielle Web-Seite mit variablen Werten erzeugen und über den integrierten Web-Sever bereitstellen.

Für beide Systemvarianten ist ein 32-Bit-Controller aus dem unteren Preissegment (ca. 5-7 USD) sowie eine Speicherausstattung von insgesamt 2 MB Flash und 2-4 MB DRAM ausreichend.

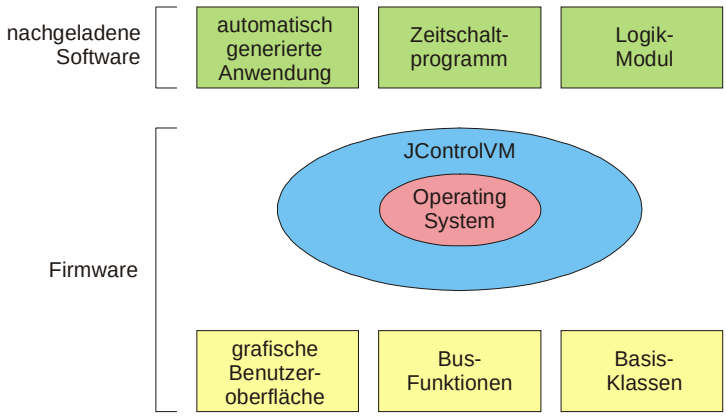

Bild 6: Beispiel: Software-Struktur eines Touch-Panels

Entwicklungsaufwände

Bei der Entwicklung von Software für neue Geräte kann leicht auf fertige Module aufgesetzt werden, so dass sich ein größerer Teil der Aufwände auf Anpassungsarbeiten beschränkt. Hierzu zählen:

- Anpassung der virtuellen Maschine an Betriebssystem und Hardwareausstattung (z.B. uClinux)
- Erstellung einer anwendungsspezifischen Konfigurationssoftware, die die Anwendungsvarianten und vorhandene Systemausstattung berücksichtigt
- Ggf. Erstellung/Anpassung einer anwendungsspezifischen grafischen Benutzeroberfläche (Corporate Design) für Touch-Panels oder eines HTML-Seitengenerators für IP-Gateways
- Erstellung weiterer anwendungsspezifischer Software-Module in JAVA (System-Menüs, Zeitschaltprogramme, Logik-Module etc.)

Fazit

Steuereinheiten wie z.B. Touch-Panels oder IP-Gateways erfordern gerade im Zusammenhang mit Feldbussystemen hochgradig flexible Software-Konzepte, damit sie auf stark unterschiedliche Marktbedürfnisse angepaßt werden können. Dabei spielen die Herstellungs- und Installationskosten vor allem im Endkundengeschäft eine tragende Rolle, so dass für Systeme mit niedriger Speicherausstattung und Prozessorleistung sowie einfachen Konfigurationsmechanismen eindeutige Wettbewerbsvorteile entstehen.

Die hier vorgestellten Konzepte verlagern einerseits den größten Teil der Konfigurations-Funktionalität auf den PC, weshalb hierfür keine Ressourcen im Gerät selbst bereitgestellt werden müssen. Andererseits bietet die JControl Virtual Machine ein Höchstmaß an Flexibilität, wodurch nicht nur zusätzliche Anwendungsmöglichkeiten entstehen, sondern auch im späteren Produktleben noch weitreichende Funktions-Erweiterungen ohne Firmware-Updates durchgeführt werden können. Dabei bietet die JAVA-basierte Technologie vielerlei Vorteile gegenüber Programmen auf Maschinencode-Ebene: Die automatische Verkettung separater Software-Module, Sicherheitsmechanismen dank der Interpreter-Technologie, objektorientierte Programmierung mit automatischer Speicherverwaltung sowie eine sehr hohe Hardware-Unabhängigkeit. Das erleichtert nicht nur die Programmierung einzelner Module „von Hand“, sondern es verbessert auch die Möglichkeiten für eine automatische Programmcode-Erzeugung mit Hilfe eines Code-Generators.

JControl ist eine JAVA-Plattform, die vor allem für Embedded Systems mit geringer Speicherausstattung und Performance konzipiert wurde. Fertige Software-Module und leistungsfähige Software-Werkzeuge und Klassenbibliotheken erleichtern dabei die Anwendungsentwicklung.