Java in Real-Zeit (mit Public -domain-Software)

Java vereinigt Sicherheit mit der geliebten C (und C++) Syntax. Aber noch sind einige Steine aus dem Weg zu räumen, bevor aus Java die ideale Programmiersprache für sicherheitsrelevante eingebettete Real-Zeit-Systeme wird. Sachlich gesehen, wäre ADA immer noch die erste Wahl für solche Applikationen, aber die persönliche Neigung des Programmierers spielt auch eine wichtige Rolle. Deswegen wird so viel in Java investiert. Sie ist noch eine sehr junge Sprache und ihre Mankos sind bekannt. Das Interesse, aus Java eine ideale Programmiersprache für Real-Zeit eingebettet Systeme zu machen, ist enorm. Weltweit wird sehr intensiv daran gearbeitet. Das wird aber noch seine (real) Zeit brauchen.

Für die Beurteilung des jetzigen Java benutzen wir folgende Kriterien: Sicherheit (**++**), I/O-Anbindung (**--**), Real-Zeit Unterstützung (**--**) , Effizienz (**-**), Parallelität (**++**), Lesbarkeit (**+**), Flexibilität (**+**) und Portabilität (**++**). Sicherheit ist für sicherheitsrelevante eingebettete Real-Zeit-Systeme das wichtigste und gerade hier liegen die Schwächen von C/C++ und die Stärcke von Java. Die I/O-Anbindung und Real-Zeit-Unterstützung sind wichtig, um die Kontrollsoftware in ihre Umgebung (HW + Zeit) einzubetten. Genau hier und bei der Effizienz liegen die Schwächen von Java. Die Defizite von Java bei der I/O-Anbindung sind: 1) das Fehlen eines Konstrukts, um direkt auf Hardware-Adressen und/oder Register zuzugreifen 2) das Fehlen von Konstrukten, um asynchrone externe Ereignisse (Interrupts) zu behandeln (daran wird gearbeitet).

Die größten Defizite bei der Real-Zeit-Unterstützung zeigt Java in der Implementierung des Garbage Collectors, der extrem dynamischen Speicherverwaltung und der schwierigen Ausführungszeitvorhersage. An diesen Punkten wird weltweit sehr intensiv gearbeitet (wir auch).

Die Java-Effizienz hat zwei Defizite: Die langsamere Ausführung auf Grund der Interpretation und/oder “on the fly”-Übersetzung des Bytecode und der relativ große Speicherverbrauch für den Interpreter oder für den Just-In-Time-Compiler. Um diesen Flaschenhals zu beseitigen, werden spezielle Java-Prozessoren, die den Bytecode mittels Hardware interpretieren, entwickelt. Für die, die nicht schon wieder einen neuen Prozessor einsetzen können, entwickeln wir einen Übersetzer von Bytecode auf den Assembler verschiedener Standard-Prozessoren. Damit werden diese zwei Effizienz-Defizite von Java in eingebetteten Systemen ausgeglichen. Die Übersetzer werden wir in einigen Monaten als Public-domain-Software unter
http://www.first.gmd.de/~sergio/public_domain anbieten.

Die Unterstützung der Parallelität ist wichtig, weil Real-Zeit-, Steuerungs- und Regelungs-Systeme meistens als parallel laufende Tasks implementiert werden. Die Lesbarkeit ist wichtig, weil sie das Überprüfen (Audits) von Code vereinfacht, was die Sicherheit steigern kann. Die Flexibilität ist wünschenswert, um das Programmieren zu vereinfachen. Dies kann wiederum die Sicherheit steigern, denn komplexere Programme sind eine Gefahrenquelle.

Portabilität ist für viele Applikationen sehr wichtig. Für eingebettete Real-Time-Systeme ist sie nicht so relevant. Denn meistens wird die Software speziell für eine gegebene Hardware (I/O, Interrupts, Redundanz,...) entwickelt, abgestimmt und optimiert. Um die Kontrollsoftware in einer anderen Anlage laufen zu lassen, reicht es meistens nicht aus, sie einfach erneut zu kompilieren.

Sergio Montenegro, GMD-FIRST, Rudower Chaussee 5, D-12489 Berlin-Adlershof
Tel +49/30/6392-1878, email: sergio@first.gmd.de

Die Vorteile von Java gegenüber C/C++ sind: Sie ist eine kleinere und verständlichere Sprache als C. Sie wird weiterentwickelt. Sie hat eine standardisierte Threads- und Parallelitätsunterstützung. Sie hat eine größere standardisierte Klassen-Bibliothek (>510 Klassen). Das Programmieren ist effektiver: man macht weniger Programmierfehler. Die Java primitiven Daten-Typen haben immer dieselbe Größe. Sie hat 64-Bit Arithmetik. Sie ist sicherer durch: Strengere Typüberprüfung (z.B. boolean ist nicht gleich integer), Arrays-bounds check zur Laufzeit, Typüberprüfung zur Laufzeit , Garbage Collector und einen Ausnahmenmechanismus, den der Programmierer nicht umgehen kann. (Sie können weitere Informationen zu diesen Themen unter
http://www.first.gmd.de/~sergio/public finden)

Die größten Schwierigkeiten von Java für Real-Zeit-Applikationen sind:

Speicherverwaltung

Für die Freigabe von alloziierten Speicherbereichen benutzt Java keine delete- oder free-Funktion, sondern wenn mehr Speicher benötigt wird (new-Operator) als z.Z. im Heap vorhanden ist, werden alle laufenden Tasks für eine Weile suspendiert, während der Garbage Collector den gesamten Speicher nach Blöcken durchsucht, zu denen es keine Referenz mehr gibt. Diese Blöcke werden als frei markiert und können wieder alloziiert werden. Man kann nicht vorhersagen, wann und für wie lange der Garbage Collector aktiviert wird, und er kann nicht unterbrochen werden bis er fertig ist. Dies kann das gesamte Timing der Applikation über Bord werfen und kritische Deadlines können verpaßt werden. Dies kann wiederum zu einem Systemversagen führen.

I/O

Die Einbettung von Java in eine spezielle Hardware (mit Sensoren und Aktuatoren) ist nicht vorgesehen. Es ist nicht möglich auf bestimmte Register oder Adressen zuzugreifen. Man muß eine abstrakte Schnittstelle benutzen, die mittels einer andere Programmiersprache (z.B. C) implementiert wurde. Bei der Erstellung von Java-Prozessoren soll sich das ändern. Diese Prozessoren sollen einen erweiterten Bytecode haben, der es erlaubt, auf bestimmte Adressen zuzugreifen. Es ist aber noch nicht definiert, wie man aus Java diese Funktionalität benutzen kann. Das zweite Problem ist, daß Java nicht für asynchronen I/O (interruptgesteuert) konzipiert ist. Eine einfache Lösung wäre es, einen Thread (Task) pro Interrupt zu haben. Beim Eintreffen eines Interrupts wird der passende Thread vom Laufzeitsystem aktiviert. Dies gehört nicht zur Standard-Java-Umgebung, kann aber einfach implementiert werden, wenn das System eingebettet wird.

Scheduler

Der Standard-Java-Scheduler ist nicht für Real-Zeit Applikationen konzipiert. Deadlines und Zeitfenster werden nicht unterstützt. In der Regel hat man einen nicht preemptiven Scheduler, der nicht in der Lage ist, Tasks ohne ihre aktive Beteiligung zu unterbrechen. Dadurch sind Timeouts nicht möglich. Wir haben einen experimentellen Real-Zeit-Scheduler geschrieben, bei dem die Tasks ihre Zeitanforderungen anmelden können und er versucht (ohne Garantie) diese Anforderungen zu erfülen. Dieser Scheduler wird als "Public-Domain-Software" unter
http://www.first.gmd.de/~sergio/public_domain angeboten.

Sergio Montenegro, GMD-FIRST, Rudower Chaussee 5, D-12489 Berlin-Adlershof
Tel +49/30/6392-1878, email: sergio@first.gmd.de