C versus Ada in sicherheitskritischen Applikationen

Michael Jungmann
MTU Aero Engines GmbH
Systemauslegung/Softwareentwicklung
Münchner Strasse 665
80995 München
Telefon ++49(0)89 1489 9149
Michael.Jungmann@muc.mtu.de

Abstract: In Zukunft wird es verstärkt notwendig sein, sicherheitskritische Systeme statt in Ada in der Programmiersprache C zu implementieren. Die Verwendung von C ist getrieben vom Mangel an Software-Entwicklungswerkzeugen für verwendete Plattformen. Diese Arbeit zeigt auf, welche zusätzlichen Risiken durch die Verwendung der Sprache C entstehen, und wie diese Risiken durch eine Anpassung des Softwareentwicklungsprozesses reduziert wurden.

Motivation

Zu den Kernkompetenzen der MTU Aero Engines GmbH gehört die Entwicklung und Herstellung von Regel- und Überwachungssystemen für Luftfahrtantriebe. Speziell Regelsysteme sind entsprechend der höchsten Sicherheitsstufe des jeweils anzuwendenden SW-Entwicklungsstandards (RTCA DO178A/B Level A etc.) zu implementieren und zu testen.

Die Methoden und Prozesse der Softwareentwicklung und -zertifizierung wurden mit den - häufig behördlichen - Kunden im Rahmen früherer Projekte abgestimmt und ständig weiter verfeinert. Ein zentraler Punkt des Softwareentwicklungsprozesses war die Verwendung der Programmiersprache Ada. Aufgrund dieses etablierten Prozesses war es natürlich wünschenswert, diesen auch für zukünftige Projekte anwenden zu können. Die weitere Verwendung der Programmiersprache Ada war ein bedeutendes Kriterium neben anderen bei der Konzeption einer neuen Rechner-Plattform. Ausschlaggebend für die Auswahl der Rechner-Plattform waren letztlich Anforderungen an die Hardware (Rechenleistung, geringe Stromaufnahme, Fertigungstechnologie etc.)

Ein Signalprozessor von TI erfüllte die Kriterien am besten. Leider ist für diesen Prozessor kein Ada Entwicklungswerkzeug verfügbar. Es war deshalb notwendig, die Software in C zu implementieren und den kompletten Softwareentwicklungsprozess zu überarbeiten, um ihn an die Eigenschaften von C anzupassen.

Darüber hinaus gibt es weitere Gründe, sicherheitskritische Applikationen in Zukunft häufiger in C zu implementieren. Beispielsweise halten sicherheitskritische Systeme verstärkt Einzug in den Automobilbau, wo bisher nahezu ausschließlich in C implementiert

wurde. Zum anderen finden in der Softwareentwicklung zunehmend automatische Sourcecode Generatoren Verwendung, wobei eine Vielzahl zur Erzeugung von C-Code, aber nur eine sehr geringe Zahl zur Erzeugung von Ada-Code am Markt verfügbar sind.

Anpassung des Softwareentwicklungsprozesses

Bei der Definition des etablierten Softwareentwicklungsprozesses konnte man sich auf die bekannten Eigenschaften eines validierten Ada Compilers (wie zum Beispiel strenge Typprüfung) stützen. Programmierrichtlinien wurden etabliert, um die Verwendung der Programmiersprache auf die für sicherheitskritische Systeme zulässigen Merkmale zu beschränken.

Eine Grundregel bei der Entwicklung sicherheitskritischer Systeme ist, hohes Gewicht auf die Fehlervermeidung zu setzen und sich nicht darauf zu verlassen, durch eine hohe Testtiefe alle Fehler zu finden.

Da die Programmiersprache C mehr Eigenschaften bietet, die bei fehlerhafter Verwendung zu einem Fehlverhalten des Systems führen könnten und zusätzlich ein C Compiler weniger Prüfungen durchführt, war eine Anpassung des Softwareentwicklungsprozesses für C notwendig. Ausgehend von einer Sicherheitsanalyse für C wurden Programmierrichtlinien aufgestellt, welche die Verwendung der Programmiersprache auf einen für sicherheitskritische Systeme akzeptablen Umfang reduzieren. Zusätzlich wurden Tests und Analysen definiert, um die Einhaltung dieser umfangreicheren Programmierrichtlinien zu überprüfen. Diese Analyse fand weitgehend unabhängig von bereits existierenden Regelwerken statt. Die entstandenen Programmierrichtlinien wurden anschließend einer Prüfung auf Vollständigkeit unterzogen. Dazu wurde überprüft, ob Regeln aus gängigen Programmierrichtlinien für C (z.B. MISRA) genauso oder noch restriktiver enthalten sind. Weiterhin wurde geprüft, ob die Wirkung von Regeln, welche für die Programmierung mit Ada bestehen, auch mit dem vorliegenden Regelwerk erzielt wird. Darüber hinaus mussten auch programmiersprachenunabhängige Regeln (z.B. IEC61508 Teil 3) berücksichtigt sein.

Da nur wenige Informationen über die Zuverlässigkeit des einzusetzenden Compilers existierten, wurde eine Analyse des Compilerverhaltens durchgeführt und daraus Verwendungshinweise für den Compiler abgeleitet.

Programmierrichtlinien

Bei der Aufstellung der Programmierrichtlinien wurde deren Prüfbarkeit bereits berücksichtigt. Das führte in Einzelfällen zu Regeln, die strenger sind als notwendig, sich dafür aber leichter überprüfen lassen. Im Folgenden sind die Schwerpunkte der Programmierrichtlinien aufgeführt.

Das definierte Subset von C schließt Konstrukte mit besonders hohem Fehlerpotential aus. Dazu gehören ‚function-like' Makros, die ‚goto' Anweisung, die Inkrement- und Dekrementoperatoren, alternative Darstellungsformen etc.

Die Verwendung von C-Standardtypen ist ausgeschlossen. Alle Daten sind unter Nutzung von projektspezifischen Typen zu definieren. Bezüglich Strukturen, Bitfeldern und Arrays werden genaue Verwendungshinweise gegeben. Aufzählungstypen und Unions sind von der Verwendung ausgeschlossen. Es existieren Regeln, welche die Konsistenz zwischen Definition und Deklaration von Funktionen und Daten sicherstellen.

Die Verwendung von Pointern ist limitiert. Insbesondere Pointerarithmetik, Konvertierung von Zahlen in Pointer und umgekehrt sowie Pointer auf Funktionen sind verboten.

Die Strukturierung der Software und die damit zusammenhängende Sichtbarkeit von Daten und Funktionen, welche in Ada durch das Package-Konzept unterstützt ist, wird durch Programmierrichtlinien detailliert vorgeschrieben.

Die Forderungen zur Lesbarkeit enthalten Vorschriften zur Verwendung von Kommentaren, zur Einrückung von Blöcken, zur standardisierten Formatierung von Konstrukten wie Schleifen, Alternativen, Auswahlanweisugen etc.

Darüber hinaus regeln die Programmierrichtlinien Einschränkungen bei der Verwendung erlaubter Konstrukte wie Schleifen Alternativen, Funktionsaufrufe etc.

Test- und Analyseverfahren

Hier soll jetzt nur auf die zusätzlichen Tests und Analysen eingegangen werden, welche wegen der Verwendung der Programmiersprache C notwendig werden. Die bei der Entwicklung sicherheitskritischer Systeme üblichen Tests wie funktionale Unit-Tests, SW-Integrationstests sowie Acceptance-Tests sind ebenso durchzuführen.

Die Überprüfung der Einhaltung der Programmierrichtlinien erfolgt durch verschiedene Analysetools. Beispielsweise sind etliche der Regeln identisch zu den MISRA-Regeln, so dass für deren Überprüfung ein kommerzielles MISRA-Tool zum Einsatz kommt. Die Prüfung der Einhaltung des zulässigen Sprach-Subsets erfolgt durch einen Parser. Beim Test auf Einhaltung des Typkonzepts findet ‚PC-Lint' Anwendung. Die Überprüfung des Daten- und Kontrollflusses erfolgt mit ‚Testbed for C' (LDRA). Für Überprüfungen, welche nicht durch ein kommerzielles Tools abgedeckt wurden, war im Einzelfall abzuwägen, ob ein entsprechendes Tool selbst zu entwickeln ist, oder aber diese Überprüfungen manuell durchzuführen sind. Derzeit liegt der Anteil der manuellen Überprüfungen bei ca. 10% der Regeln mit Schwerpunkt auf Regeln zur Lesbarkeit und Verständlichkeit.

Überprüfung des Compilers

Bei der Auswahl möglicher Compiler wurde berücksichtigt, ob diese bereits vom Hersteller mit existierenden Validierungssuiten überprüft wurden. Die Ergebnisse liegen in Form von Listen bekannter Fehler vor und wurden für die Programmierrichtlinien berücksichtigt. Zur eigenen Überprüfung des Compilers wurden charakteristische Programmstrukturen (Schleifen, Alternativen etc.) entsprechend der Programmierrichtlinien in C implementiert und compiliert. Anschließend wurde der entstandene Assemblercode manuell untersucht. Dabei identifiziertes Fehlverhalten des Compilers wurde zur Formulierung weiterer Programmierrichtlinien herangezogen.

Zusammenfassung/Ausblick

Es ist grundsätzlich möglich, die Programmiersprache C zur Entwicklung sicherheitskritischer Systeme zu verwenden. Wegen den Eigenschaften der Programmiersprache und der dafür verfügbaren Entwicklungssysteme ist es notwendig, den Softwareentwicklungsprozess grundlegend zu überarbeiten. Sowohl die Programmierrichtlinien als auch die Test- und Analyseverfahren können nicht von einem Entwicklungsprozess für Ada übernommen werden, sondern müssen ausgehend von einer grundsätzlichen Sicherheitsanalyse neu definiert werden.

Der Aufwand für die Softwareentwicklung für ein sicherheitskritisches System in C ist höher als für die Entwicklung in Ada. Demgegenüber steht eine größere Auswahl an Hardwareplattformen, für welche Entwicklungsumgebungen für C verfügbar sind.

Sicherheitskritische Systeme werden in Zukunft häufiger in der Programmiersprache C implementiert werden. Deshalb wird es weitere Untersuchungen geben, deren Ziel es ist, ein maximales Maß an Fehlerfreiheit von in C implementierter Software zu erreichen und die Effizienz der notwendigen zusätzlichen Tests und Analysen weiter zu steigern.

Derzeit werden die Voraussetzungen geschaffen, auch automatische Codegeneratoren für sicherheitskritische Systeme einzusetzen. Dabei werden große Teile der Software mit Tools wie ‚Simulink'/'Stateflow' (TheMathWorks) grafisch modelliert. Die Erzeugung des C-Sourcecodes erfolgt mit einem kommerziellen Codegenerator wie ‚TargetLink' (dSPACE). Damit der entstehende Sourcecode den Ansprüchen für sicherheitskritische Systeme genügt, sind zusätzliche Bearbeitungsschritte notwendig.

Literaturverzeichnis

[MISRA] Guidelines for the use of the C language in vehicle based software, The Motor Industry Software Reliability Association, April 1998.

[RTCA] Software Considerations in Airborne Systems and Equipment Certification DO-178B, RTCASC-167/EUROCAE WG12