

Device Software-Entwicklung mit mehreren Programmiersprachen

Der Trend zur TCP/IP-Kommunikation macht das Mixed Language Design zunehmend auch für sicherheitsrelevante Projekte interessant



In der Vergangenheit war die Wahl einer Programmiersprache für ein A&D-Projekt eine ziemlich klare Sache. In den USA war der Einsatz von Ada für DoD-Projekte bis zum Jahr 1997 sogar zwingend vorgegeben. Trotz der Öffnung in den letzten Jahren verwenden viele hochintegrierte und sicherheitsrelevante Projekte in den USA auch heute noch Ada als Programmiersprache und der Einsatz von Ada wird bei sicherheitsrelevanten Projekten auch in einer Reihe anderer Länder immer noch gefördert. Erst in letzter Zeit ist ein deutlicher Zuwachs bei der Anwendung von mehreren Sprachen für die Entwicklung von Device Software zu verzeichnen. Am häufigsten wird Ada dabei mit C gepaart.

n Paul Parkinson

Dass gerade in letzter Zeit auch bei sicherheitsrelevanten Entwicklungen immer häufiger ein Mixed Language Design bspw. aus Ada und C verwendet wird, hat mehrere klar umrissene Gründe: Erstens werden vorhandene sicherheitsrelevante Systeme, die in Ada implementiert wurden, häufig um zusätzliche Funktionen erweitert, die den Einsatz eines üblicherweise in C geschriebenen Echtzeitbetriebssystems erfordern. Zwei-

tens sind durch die Einführung von TCP/IP und die damit verbundenen Netzwerktechnologien – die oftmals in C implementiert sind – verteilte Anwendungen weit verbreitet; TCP/IP wird durch Strategien wie z.B. Network Centric Warfare (NCW) und Network Enabled Connectivity (NEC) sogar in NATO-Verteidigungssystemen vorangetrieben. Drittens sind Display-Systeme heute offen für die Verbindung sicherheitsrelevanter, in Ada >



Paul Parkinson
ist Senior Systems Architect bei Wind River
T +44/1793/83533-0
Paul.Parkinson@windriver.com

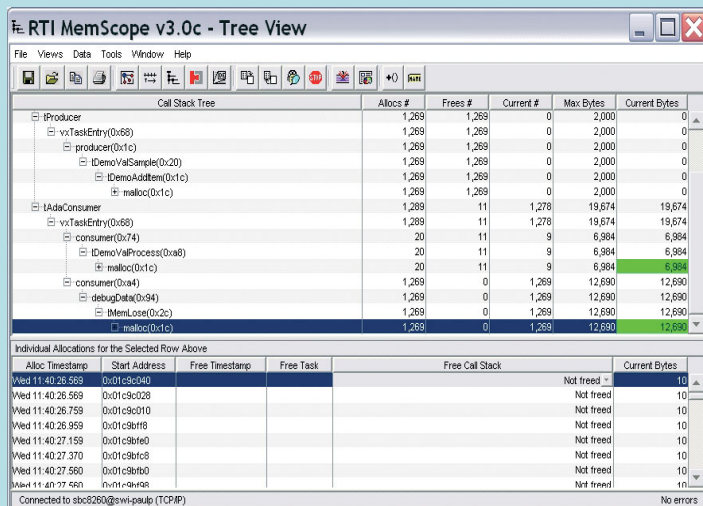


Abb. 1: RTI MemScope Baumstruktur-Ansicht einer Mixed-Language-Device Software



Abb. 2: Der Wind River System Viewer zeigt den Ada-Aufruf der VxWorks C API

implementierte Device Software mit OpenGL und anderen in C implementierten Grafikbibliotheken. Und schließlich herrscht bei der Software-Entwicklung ein wachsender Trend zur Code-Wiederverwendung, der die Wahrscheinlichkeit eines gemischten Systems aus Ada und C erhöht.

In Luft- und Raumfahrtssystemen können all diese Faktoren zusammen auftreten. Ein typisches Beispiel hierfür sind die Integrated-Modular-Avionics-Systeme (IMA), bei denen Device Software-Programme in Ada und C nebeneinander auf dem gleichen Prozessor laufen, grafische Displays steuern und über ein Ethernet- oder AFDX-Netzwerk Daten übertragen.

Mixed-Language-Programmierung

Um Mixed-Language Device Software effizient entwickeln und optimieren zu können, müssen Entwickler Ada-Prozeduren von C aus und umgekehrt aufrufen und Mixed-Language Source-Level Debugging durchführen können sowie die Speichernutzung der Mixed-Language-Anwendung und das Verhalten der Mixed-Language-Anwendung auf Systemebene verstehen. Diese Voraussetzungen werden neben praktischen Implementierungsbeispielen in den folgenden Abschnitten näher untersucht.

Zusammenspiel

Viele Programmiersprachen unterstützen die Anbindung an andere Programmiersprachen nur begrenzt; folglich bieten auch Sprach-Compiler und Debugger nur eine eingeschränkte Unterstützung.

Deshalb mussten Entwickler bisher ihre eigenen Bindeglieder zwischen den Sprachen im Assembler implementieren – eine fehleranfällige Tätigkeit, die zu nicht portierbarem Code führen kann. Die Programmiersprache Ada95 stellt eine gut definierte Schnittstelle zur Sprache C zur Verfügung, so dass Entwickler C-Funktionen von Ada-Prozeduren und umgekehrt einfach und sicher aufrufen können. So kann bspw. ein in Ada implementiertes Befehlssteuerungsprogramm, das in der Vergangenheit über einen Bus oder über serielle Geräte kommuniziert hat, heute über die Verbindung mit einem in C implementierten IPv6-Stack netzwerkfähig gemacht werden.

Im Hinblick auf künftige Wartungsmöglichkeiten ist es jedoch eventuell vorzuziehen, keinen IPv6-spezifischen Code innerhalb des Ada-Befehlssteuerungsprogramms einzubetten. Stattdessen sollte vorzugsweise eine Abstraktionsschicht benutzt werden, um die Schnittstelle mit dem Netzwerk-Stack zu verbergen oder zu minimieren. Das kann zum Beispiel durch die Verwendung von Interprozesskommunikations-Fähigkeiten (IPC) geschehen, die von dem zugrunde liegenden Echtzeitbetriebssystem (RTOS) bereitgestellt werden. Bei einer auf VxWorks basierenden Implementierung können VxWorks-Message-Queues für IPC eingesetzt werden, und eine GNAT Ada-Prozedur kann die API msgQReceive() aufrufen, die wie folgt in C implementiert ist:

```
function Msg_Queue_Receive
(Msg_Queue_Id : Integer;
 Buffer       : System.Address;
 Max_NBytes  : Interfaces.C.unsigned;
 Timeout    : Integer)
return Integer;
pragma Import (C, Msg_Queue_Receive, „msgQReceive“);
```

Hierdurch werden ganz unkompliziert Aufrufe von GNAT Ada an C ermöglicht, vorausgesetzt, die entsprechenden Ada-Datentypen werden als Schnittstelle zu den in den Parametern der C-Funktion verwendeten Datentypen ausgewählt. Ein Blocking Call von der Ada-Anwendung, msgQReceive(), sieht dann wie folgt aus; die umgekehrte Prozedur kann auf gleiche Weise mit dem pragma-Export erfolgen:

```
Return_Value := Msg_Queue_Receive
(Data_Queue_Id, Ada_Buffer_Address, Result_Structure_Size, -1);
```

Debugging

Wenn Entwickler einmal C-Funktionen von Ada-Prozeduren und umgekehrt aufgerufen haben, werden sie diesen Mixed-Language-Anwendungscode auch auf Source-Ebene debuggen wollen. Hier stellen sich jedoch ganz andere Herausforderungen, da Compiler und Debugger in der Vergangenheit oft eine Vielfalt von Object-Module-Formaten (OMF) verwendeten. Entwickler, die versucht haben, von verschiedenen Compilern erstellte Anwendungen in C++ zu binden, sind mit diesem Problem nur zu gut vertraut. In den vergangenen Jahren gab es in der Branche Bemühungen, das „Executable & Linking Format“ (ELF) sowie das „Debugging With Arbitrary Record Format“ (DWARF) als Standards zu etablieren. Hierdurch ist das Parsing von Objektdateien zwar unkomplizierter geworden, doch müssen sich diese Tools noch immer an ihre eigenen Namenskonventionen halten und zudem die der anderen Tools berücksichtigen, mit denen zu debuggender Objektcode erstellt wurden.

