API für VM Debugging Awareness

Seit Mitte 2006 unterstützt Lauterbach das Debugging von Java-Anwendungen für die Java Virtual Machines J2ME CLDC, J2ME CDC und Kaffe. Da sich virtuelle Maschinen einer wachsenden Beliebtheit erfreuen, steigt auch die Zahl der Anbieter. Und längst sind nicht mehr alle virtuellen Maschinen Open-Source. Um VM-Anbietern und ihren Kunden die Möglichkeit zu geben, das Debugging schnell und flexibel für ihre VM anzupassen, arbeitet Lauterbach seit Mitte 2010 an einer offenen Lösung.

Als Referenz für die Entwicklung einer VM API für das Stop-Mode Debugging wird die für ARM Cores implementierte Android *Dalvik Virtual Machine* verwendet.

Zwei Debug-Welten

Für alle, die mit Android nicht vertraut sind, zunächst eine kleine Einführung. Aus Entwicklersicht ist Android ein Open-Source Software-Stack, der aus folgenden Komponenten besteht (siehe Bild 4):

- Ein Linux Kernel mit seinen Hardware-Treibern.
- Die *Android Runtime* mit der *Dalvik Virtual Machine* und einer Reihe von Bibliotheken: klassische Java Core Libraries, Android-spezifische Libraries, in C/C++ erstellte Libraries.
- In Java programmierte Anwendungen und das sie unterstützende *Application Framework*.

Software für Android wird in verschiedenen Programmiersprachen geschrieben:

- Der Linux Kernel, einige Libraries und die *Dalvik Virtual Machine* werden in C bzw. C++ oder Assembler codiert.

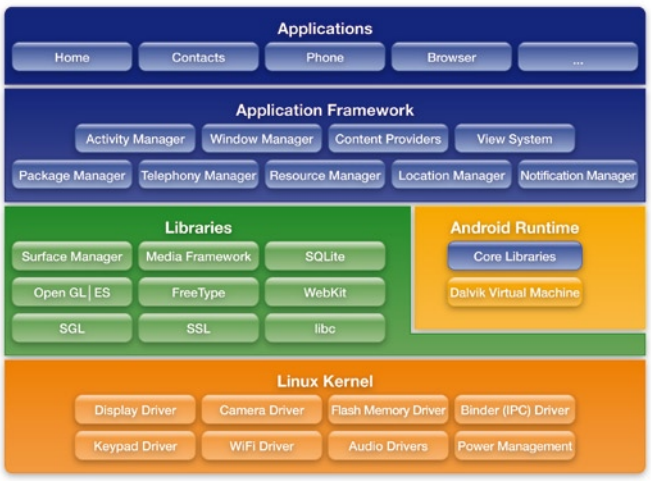

Bild 4: Der Open-Source Android Software-Stack.

- VM-Anwendungen und das sie unterstützende *Application Framework* werden in Java programmiert.

Getestet wird der jeweilige Code in seiner eigenen Debug-Welt.

Debugging von C/C++ und Assembler Code

Den in C/C++ und Assembler codierten Teil von Android kann man hardwarenah über die JTAG-Schnittstelle im Stop-Mode debuggen. Lauterbachs TRACE32 kommuniziert dazu direkt mit dem Prozessor der Android Hardware-Plattform (siehe Bild 5).

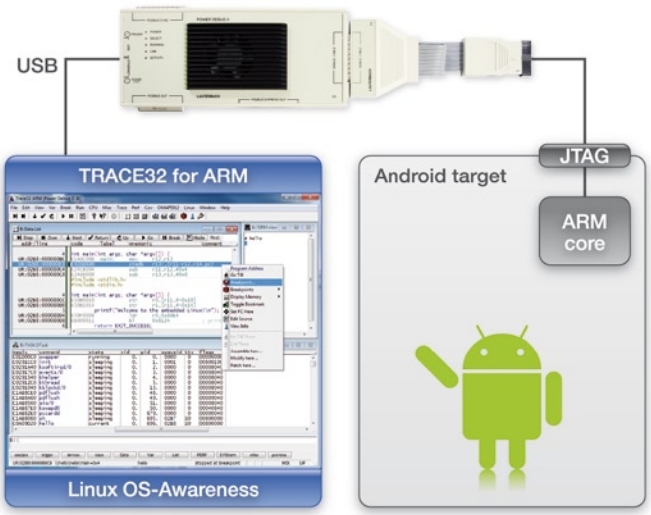

Bild 5: Beim Stop-Mode Debugging kommuniziert der Debugger direkt mit dem Prozessor auf der Android Hardware-Plattform.

Charakteristisch für das Stop-Mode Debugging ist, dass immer dann, wenn der Prozessor für das Debugging angehalten wird, das gesamte Android System stoppt.

Stop-Mode Debugging hat einige große Stärken:

- Es benötigt nur die funktionierende JTAG-Kommunikation zwischen Debugger und Prozessor.
- Stop-Mode Debugging benötigt keinen *Debug Server* auf dem Target und eignet sich deshalb auch gut für das Testen von Release-Software.
- Stop-Mode Debugging erlaubt ein Testen unter realen Zeitbedingungen und ermöglicht darum eine effiziente Fehlersuche für Probleme, die erst unter Echtzeitbedingungen auftreten.

»

Stop-Mode Debugging unterstützt im Augenblick jedoch noch nicht das Debugging von VM-Anwendungen, z.B. auf der Dalvik VM. Daher ist ein transparentes Debugging durch alle Software-Schichten bisher nicht möglich.

Debugging von Java-Code

Java-Code für Android wird üblicherweise mit den in Eclipse integrierten *Android Development Tools* (ADT) getestet. Der *adb server* – adb steht für Android Debug Bridge – auf dem Host kommuniziert dazu über USB oder Ethernet mit dem *adb daemon* auf dem Target (Bild 6).

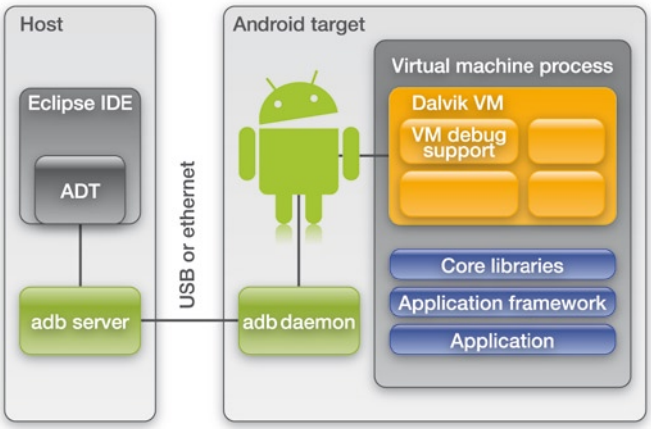

Bild 6: Die in Eclipse integrierten Android Development Tools (ADT) zum Debuggen von Java-Code.

Voraussetzung für das Testen mit ADT sind speziell für Debugging kompilierte VM-Anwendungen und eine Deklaration der Debug-Erlaubnis im *Application Manifest*. Zudem muss Debugging für die Hardware-Plattform generell freigeschaltet sein.

Das Debugging des Java-Codes mit ADT ist recht komfortabel. Es gibt jedoch Fälle, in denen man mit ADT nicht weiterkommt. Dies sind:

- Fehler, die erst mit dem Release-Code auftreten.
- Fehler, die erst dann auftreten, wenn die Java-Anwendung mit einem in C/C++ angebotenen Dienst oder einem Linux Hardware-Treiber interagiert.
- Debugging nach dem Zusammenbruch der Kommunikation zwischen *adb server* und *adb daemon*.

VM Aware Stop-Mode Debugging

Um ein durchgängiges Testen eines Android Systems von der Java-Anwendung bis hinunter zum Linux Hardware-Treiber unter Realzeitbedingungen zu ermöglichen, erweitert Lauterbach aktuell das Stop-Mode Debugging um eine *VM Debugging Awareness*.

Der JTAG-Debugger kommuniziert direkt mit dem Prozessor auf der Android Hardware-Plattform. Daher kann der Debugger nach dem Anhalten des Prozessors auf alle Systeminformationen zugreifen. Die „hohe Kunst“ für den Debugger besteht nun darin, die richtigen Informationen zu finden und für den Nutzer leicht verständlich, von Bits und Bytes abstrahiert, darzustellen.

Eine Abstraktionsebene bot bisher dem TRACE32-Nutzer die Möglichkeit, Betriebssystem-Software, ggf. über mehrere virtuelle Adressräume hinweg, zu debuggen. Eine andere, bisher vom Betriebssystem-Debugging unabhängige Abstraktionsebene, stellt das Java-Debugging dar.

Um in Systemen wie Android auf VMs laufende Anwendungen zu debuggen, wobei die VMs ihrerseits in Betriebssystem-Prozessen instanziiert sind, muss Betriebssystem- und Java-Debugging nun kombiniert werden. Zur Umsetzung dieser neuen Komplexität entwickelt Lauterbach eine neue, offene und leicht erweiterbare Lösung.

Die offene Lösung

Das Stop-Mode Debugging von Lauterbach unterstützt zukünftig folgende Abstraktionsebenen:

- Hochsprachen-Debugging
- Target-OS Debugging Awareness
- VM Debugging Awareness

Das **Hochsprachen-Debugging** ist fester Bestandteil der TRACE32-Software und wird mit dem Laden der Symbol- und Debug-Informationen für ein Programm passend konfiguriert.

Die **Target-OS Debugging Awareness** muss immer vom TRACE32-Nutzer konfiguriert werden. Dies ist einfach möglich, da es für alle gängigen Betriebssysteme ferti- »

Dalvik Virtual Machine

Dalvik ist der Name der in Android benutzten virtuellen Maschine. Die *Dalvik Virtual Machine* ist ein in Software programmiertes Modell eines Prozessors, das einen von Java abgeleiteten Byte-Code ausführt. Virtuelle Maschinen erlauben es, prozessorunabhängige Software zu schreiben. Beim Umstieg auf eine neue Hardware-Plattform muss lediglich die virtuelle Maschine portiert werden.

Für VM kompilierte Software läuft automatisch auf jeder Plattform, auf die diese VM portiert ist.

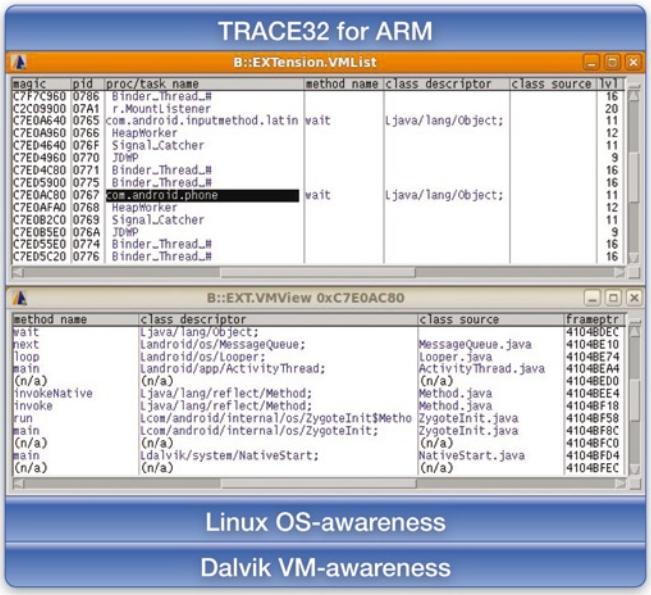

Bild 7: Für die Referenzimplementierung muss die Linux OS-Awareness und die Dalvik VM-Awareness in TRACE32 geladen werden.

ge Konfigurationen gibt. Für proprietäre Betriebssysteme bietet die RTOS API eine offene Möglichkeit zur Anpassung.

Die **VM Debugging Awareness** ist für J2ME CLDC, J2ME CDC und Kaffe direkter Bestandteil der TRACE32-Software. Alle anderen virtuellen Maschinen können mit der VM API individuell angepasst werden. Für die sehr populäre Android Dalvik VM steht, ähnlich wie für die *Target-OS Debugging Awareness,* eine fertige Konfiguration zur Verfügung.

Die offene Lösung, sowohl für das Betriebssystem als auch für die virtuelle Maschine, erlaubt es auch Anbietern von Closed-Source VMs, eine TRACE32 VM-Awareness für ihr Produkt zu erstellen und ihren Kunden anzubieten.

Die Referenzimplementierung

Um auf einem ARM-basierten Android Target von der Java-Anwendung bis hinunter zu den Linux Hardware Treibern durchgängig debuggen zu können, benötigt TRACE32 folgende Erweiterungen (siehe Bild 7):

- Eine Linux OS-Awareness, die von Lauterbach bereits seit 1998 angeboten wird.
- Eine VM-Awareness für Dalvik, die von der Lauterbach Web-Seite heruntergeladen werden kann. Diese muss lediglich für die benutzte Plattform konfiguriert werden.

www.lauterbach.com/vmandroid.html

Aktuell ist es möglich, alle Java-Anwendungen, die gerade ausgeführt werden, zu ermitteln und aufzulisten (EXTension.VMList im Bild 7). Ebenso lässt sich der VM-Stack für eine ausgewählte Java-Anwendung analysieren und darstellen (EXTension.VMView im Bild 7). Als nächster Schritt ist geplant, den von der VM gerade ausgeführten Quellcode anzuzeigen. Das Ziel der Entwicklung ist natürlich das Stop-Mode Debugging für VM-Anwendungen mit allen Funktionen eines modernen Debuggers.

Neu unterstützte RTOS	
DSP/BIOS für ARM	Q2/2011
OSEK/ORTI SMP	Q2/2011
T-Kernel für ARM	verfügbar
VDK für Blackfin	verfügbar
Windows Embedded Compact 7 für ARM	verfügbar
µC/OS-III für ARM	verfügbar