API for VM Debugging Awareness

Since 2006, Lauterbach has supported the debugging of Java applications for the Java Virtual Machines J2ME CLDC, J2ME CDC and Kaffe. Since virtual machines are increasing in popularity, the number of providers is growing. Nowadays not all of these virtual machines are open-source. To enable VM providers and their customers to adapt debugging flexibly for their VM, Lauterbach has been working on a solution since mid-2010.

The Android Dalvik Virtual Machine implemented for ARM cores is used as a reference for the development of a VM API for stop-mode debugging.

Two Debug Worlds

For developers, Android is an open-source software stack consisting of the following components (see Fig. 5):

- A Linux kernel with its hardware drivers.
- Android Runtime with Dalvik Virtual Machine and a series of libraries: classic Java core libraries, Android-specific libraries, and libraries written in C/C++.
- Applications programmed in Java and their supporting Application Framework.

Software for Android is written in various languages:

- The Linux kernel, some libraries, and the Dalvik Virtual Machine are coded in C, C++, or Assembler.
- VM applications and their supporting Application Framework are programmed in Java.

Each block of code is tested in its own separate debug world.

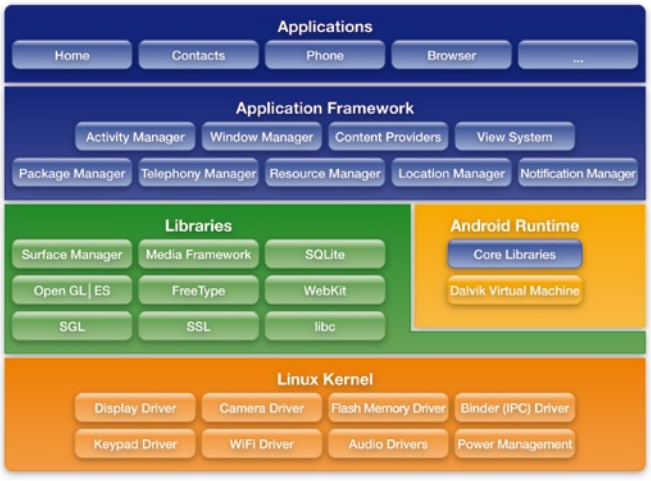

Fig. 5: The open-source Android software stack.

Debugging C/C++ and Assembler Code

The Android part coded in C/C++ and Assembler can be debugged on the target hardware over the JTAG interface in stop-mode. In stop-mode debugging, the TRACE32 debugger communicates directly with the processor of the Android hardware platform (see Fig. 6).

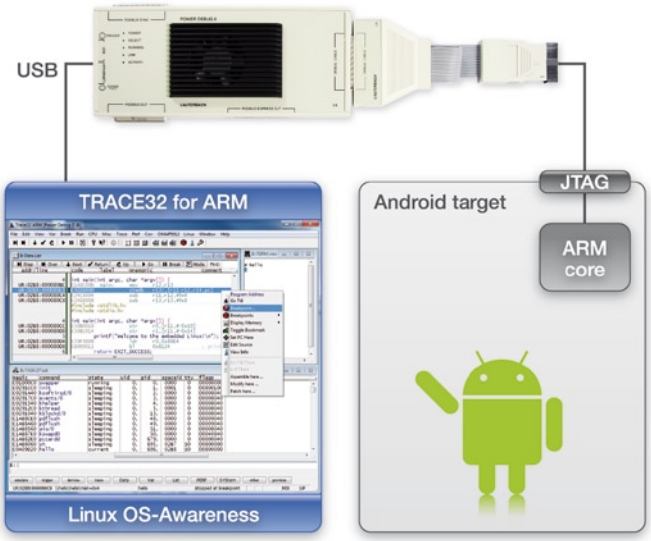

Fig. 6: In stop-mode debugging, the debugger communicates directly with the processor on the Android hardware platform.

A characteristic of stop-mode debugging is that when the processor is stopped for debugging, the whole Android system stops.

Stop-mode debugging has some big advantages:

- It needs only a functioning JTAG communication between the debugger and the processor.
- It needs no debug server on the target and is therefore very suitable for testing release software.
- It permits testing under real-time conditions and therefore enables efficient troubleshooting for problems that only occur in such conditions.

At present, stop-mode debugging does not support the debugging of VM applications such as on the Dalvik VM. Therefore transparent debugging through all of the software layers is not yet possible.

Debugging Java Code

Java code for Android is usually tested with the Android Development Tools (ADT) integrated into Eclipse. »

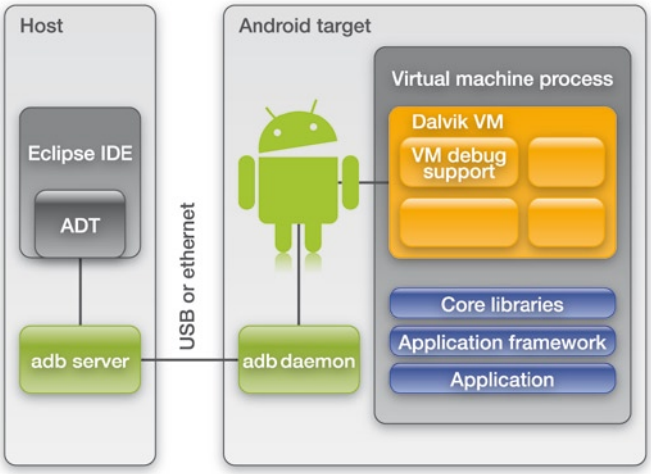

Fig. 7: The Android Development Tools (ADT) integrated in Eclipse for debugging Java code.

The adb server – adb stands for Android Debug Bridge – on the host communicates over USB or Ethernet with the adb daemon on the target (Fig. 7).

Prerequisites for debugging with ADT are VM applications specially compiled for debugging and Android debug support (adb daemon) running on the hardware platform.

Debugging Java code with ADT is comfortable. However, there are a few cases in which ADT cannot help you. These are:

- Errors that first occur with the release code.
- Errors that first occur when the Java application interacts with a service offered in C/C++ or a Linux hardware driver.
- Debugging following a communication breakdown between adb server and adb daemon.

VM Aware Stop-Mode Debugging

To enable thorough testing of an Android system from the Java application down to the Linux hardware driver under real-time conditions, Lauterbach is currently adding VM debugging awareness to its stop-mode debugging.

The JTAG debugger communicates directly with the processor on the Android hardware platform. The debugger can therefore access all system information after the processor stops. The “fine art” for the debugger is now to find the correct information and make it easy to understand for the user, abstracted from bits and bytes.

One abstraction level has given TRACE32 users the option of debugging operating system software even over several virtual address spaces. Another abstraction level, up to now independent of operating-system debugging, is Java debugging.

To debug applications running on VMs in systems like Android, where the VMs themselves are instantiated within the operating-system processes, operating-system debugging and Java debugging now have to be combined. To implement this new complexity, Lauterbach is developing a new, open, and easy-to-expand solution.

The Open Solution

In the future, stop-mode debugging from Lauterbach will support the following abstraction levels:

- High-level language debugging
- Target-OS debugging awareness
- VM debugging awareness

High-level language debugging is a fixed component of the TRACE32 software and is configured for a program with the loading of the symbol and debug information.

Target-OS debugging awareness must always be configured by the TRACE32 user. There are example configurations available for all common operating systems. The RTOS API provides an option to be customized for proprietary operating systems.

VM debugging awareness is a fixed component of the TRACE32 software for J2ME CLDC, J2ME CDC and Kaffe. All other virtual machines have to be adapted individually with the VM API. A ready-to-use configuration is available for the very popular Android Dalvik VM.

The open solution, both for the operating system and for the virtual machine, enables providers of closed-source VMs to write a TRACE32 VM awareness for their product and offer it to their customers. »

Dalvik Virtual Machine

Dalvik is the name of the virtual machine used in Android. The Dalvik Virtual Machine is a software model of a processor that executes byte code derived from Java. Virtual machines permit the writing of processor-independent software. If you switch to a new hardware platform, you only have to port the virtual machine.

Software compiled for a VM runs automatically on any platform to which this VM is ported.

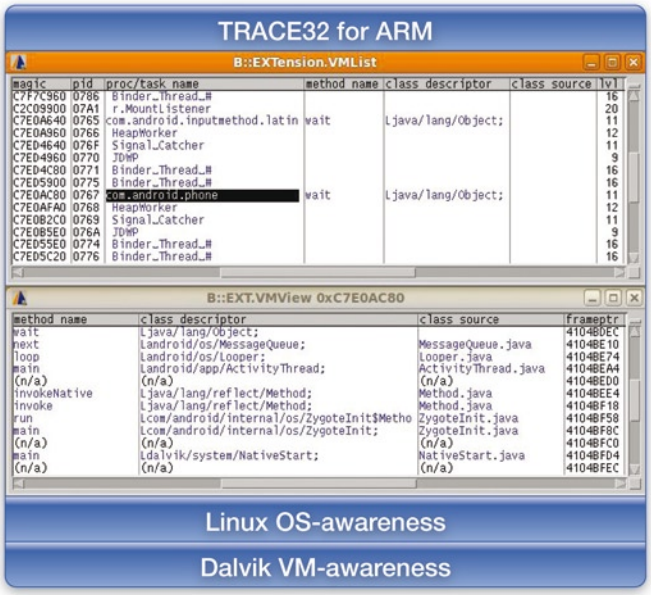

Fig. 8: For the reference implementation, Linux OS-awareness and Dalvik VM-awareness have to be loaded in TRACE32.

The Reference Implementation

To be able to debug on an ARM-based Android target TRACE32 requires the following extensions (see Fig. 8):

- A Linux OS-awareness as provided by Lauterbach since 1998.
- A Dalvik VM-awareness, which can be downloaded from the Lauterbach homepage. This just has to be configured for the platform used.

  www.lauterbach.com/vmandroid.html

It is now possible to identify and list all Java applications now being run (EXTension.VMList in Fig. 8) and to analyze and view the VM stack for a selected Java application (EXTension.VMView in Fig. 8). The next step planned is to display the source code currently being run by the VM. The aim of the development is of course stop-mode debugging for VM applications with all the functions of a modern debugger.