(embedded)

TOP TEN REASONS FOR USING JAVA IN EMBEDDED APPS

Learn why Java is the best language for the Internet of Things.

BY SIMON RITTER

For many years now, we've been promised the Internet of Things—with everything from lightbulbs to cars all networked together with built-in "intelligence." The combination of Moore's Law and economies of scale is finally making this a reality. For example, Philips recently announced that it would be selling internet-enabled lightbulbs through the Apple store. For these networked, programmable devices to be really useful, they need applications to collect, process, and transmit data. Traditionally, code for embedded systems has been written in assembler or C, but for the rapid explosion of programmable, connected devices, we need something better: Java. What makes Java the best language for embedded devices? Here are ten great reasons (with thanks to Roger Brinkley's blog entry for inspiration):

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY BOB ADLER

1 WE ALREADY HAVE A WHEEL; LET'S NOT KEEP INVENTING NEW ONES.

Java SE 7 has almost 4,000 standard class libraries covering everything from collections through concurrency to networking. Having this enormous collection of standard APIs means significantly less time spent rewriting standard functionality for embedded applications. While smaller-footprint parts of the Java platform, such as Oracle Java ME Embedded, don't have as many standard classes by default, they still have a wealth of functionality available. Many of the Oracle Java ME Embedded APIs are targeted at the needs of embedded systems: for example, the device API provides standardized ways of working with low-level protocols such as Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C). However, Java's API power doesn't stop with the classes available as part of the platform. Because Java is such a popular language, developers have created libraries that address almost anything you can think of: sensors, serial port communication, and a ZigBee API, to name a few. Many of these libraries are open sourced and can easily be integrated into new embedded applications.

ONE FOR ALL

Porting code in Java is a non-event. As long as none of the APIs used by the application have changed, it's simply a matter of redeploying the existing class or JAR files.

2 "SEGMENTATION FAULT" IS NEVER GOOD, SO WE'LL JUST AVOID IT.

One of Java's strengths, and one of the things that has made it so popular as a programming language, is how robust application code is. Languages such as C and C++ use explicit pointers to reference memory. In Java, all object references are implicit pointers that cannot be manipulated by application code. This avoids potential problems, such as buffer overruns and memory access violations, through incorrect pointer calculations. Situations such as these can cause an application to stop abruptly. In embedded systems, these types of errors can be much harder to track down because often there is no device such as a screen where error messages can be displayed.

3 ONE PLATFORM RUNS THEM ALL.

Unlike desktop machines or even servers, embedded systems vary widely in terms of their architecture, resources, and OS capabilities. When embedded systems are updated, there's also a good chance that the processor and OS will change significantly from the previous release of the product. Migrating applications written in assembler or C to a new platform can be time consuming, costly, and error prone. From the very beginning, Java has been write once, run anywhere. Porting code in Java is a non-event. As long as none of the APIs used by the application have changed, it's simply a matter of redeploying the existing class or Java archive (JAR) files. When moving to a newer (and therefore better-performing) version of Java, all that is required is a simple recompile.

4 LET THE VIRTUAL MACHINE TAKE CARE OF THE MUNDANE STUFF.

The Java Virtual Machine (JVM) provides a wealth of benefits to embedded developers that they don't get with natively compiled code.

Memory management is all handled automatically by the JVM; memory is allocated by instantiating an object, rather than having to use explicit calls to library functions such as malloc. Developers don't have to keep track of object references and explicitly deallocate memory either; the garbage collector handles all this. This substantially reduces the potential for memory leaks, which can have a far greater impact on embedded systems where applications might need to run for very long periods in memory-constrained

DON'T GO NATIVE

Some people wrongly think that by using a VM rather than native instructions, performance will be compromised.

environments without needing to be restarted.

Concurrency support for applications is also handled by the JVM. The ability to create and synchronize different threads of execution has been built in from the very beginning with Java. Even if the embedded platform on which the JVM is deployed does not support multithreading directly (something that's increasingly rare these days), it's still possible to emulate the functionality through the concept of green threads.

Some people wrongly think that by using a virtual machine (VM) rather than native instructions, performance will be compromised. With nearly 20 years of development in the JVM, these issues are no longer a significant factor. In certain situations, having precise knowledge, which is available only as the application is running, can lead to *better* performance than the performance of natively compiled code.

5 PICK AN EMBEDDED PLATFORM—ANY EMBEDDED PLATFORM.

Embedded systems are generally designed to solve a specific problem, whether it is how to provide in-car entertainment or how to monitor the pH level of part of an industrial process. As such, each system will have hardware tailored to the solution being developed, which gives designers far greater choice than when creating a new desktop or laptop system.

Many embedded chip manufacturers, such as Freescale and Broadcom, create processors and systems on a chip (SoCs) using architectures from companies such as ARM. Although this makes certain aspects of the different processors and SoCs standard, there are often small differences in terms of aspects such as floating-point processing, bus architectures, and so on. Java abstracts these differences away from the developers, making their lives significantly easier.

Reference implementations of the Oracle Java Embedded Client are available for Intel x86 and MIPS as well as ARM v5, v6, and v7 architectures running Linux (the most popular OS for embedded devices). Oracle Java SE Embedded is available for Intel x86 and IBM Power e500v2 and e600 systems as well as ARM v5, v6, and v7 architectures, again running Linux. This gives embedded systems designers plenty of choice about which platform to use.

6 THINGS HAVE CHANGED SINCE 1995.

Java was originally developed in the early 1990s to allow applications to be written for the Star7 PDA, which was an embedded device. When Java was first launched as a general-purpose computing platform in 1995, the average personal computer had 8 MB of RAM and a processor running at less than 200 MHz. Even today, the full Java SE runtime requires only a minimum of 64 MB of RAM to run on Windows XP. Typical low-end embedded systems now match or exceed these specifications.

Java was developed from the very beginning to work in resource-constrained environments, making it ideally suited to the needs of embedded systems today. Oracle Java ME Embedded will run in as little as 350 KB of ROM and 130 KB of RAM (look carefully, that's *kilobytes*, not *megabytes*), growing to 1.5 MB of ROM with 700 KB of RAM for the full, standard configuration. Oracle Java SE Embedded is designed for more-powerful systems, but still requires only 39 MB of ROM and 32 MB of RAM.

Having to deal with lower clock speeds also means that the Java platform works well when you want to maximize the time between charges for battery-powered devices. When resources are tight, Java can still get the job done.

Java Embedded: Start Developing

Tori Wieldt, Java Magazine *technology editor, caught up with Kevin Smith of Oracle Technical Business Development and got the latest on Oracle Java ME Embedded.*

Java Magazine: What's new for partners in the Java ME embedded space?

Smith: We announced Oracle Java ME Embedded at JavaOne San Francisco 2012. It's pretty exciting because it represents a new Java platform that addresses the needs of the machine-to-machine [M2M] market and the Internet of Things. We have been working closely with Cinterion and Qualcomm on this. Oracle Java ME Embedded contains JSR 228, Information Module Profile—Next Generation. The Java ME team has been working for many years connecting people to the internet with PCs, wireless handsets, tablets, and laptops, and Oracle Java ME Embedded represents the platform that will connect things to the internet. Depending on which analyst report you read, by 2020 there will be 30 to 60 billion things connected. Now, you can use your existing Java skills to program for these many devices.

Java Magazine: What is the Qualcomm Orion board?

Smith: The Orion board is Qualcomm's developer board for M2M and the Internet of Things. Since the inception of Qualcomm's new M2M business unit in early 2012, we have been working with Qualcomm on Oracle Java ME Embedded support for Orion.

Java Magazine: Why does this matter to a Java developer?

Smith: Unlike most developer boards, Orion is different in that the system on a chip [SoC] is attached to the board via a daughter card. Qualcomm has many different SoC combinations with different capabilities and modems that will provide developers one board that spans many different device types. Included on the motherboard are WiFi a/b/g/n, an SD card slot, four DB9 connectors, five LEDs, JTAG, USIM, GPIO, SPI, and I2C. The board even runs on the included battery. It also comes with sensors for temperature and light; GPS; and a 3-D accelerometer, so developers will be able to prototype solutions for container management, building automation, automotive, and the next wave of cool devices on the board. We also have a Java ME Software Development Kit integrated with the NetBeans IDE that emulates the features of the board. We really think that this is going to accelerate time to market for developers because they can get their hands on the platforms as soon as they are available, rather than waiting for OEM [original equipment manufacturer] prototypes. It's very exciting. Additional good news for developers: these prototype boards usually cost around $5,000 for a single solution, and with the Orion board they can get a scalable platform for around $500.

Kevin Smith of Oracle Technical Business Development explains what's new for partners and Java developers in the embedded space.

7 **"GIVE US THE TOOLS, AND WE WILL FINISH THE JOB."***

Developing code requires tools; the better the tools, the easier the job is to finish and the more quickly it gets done. Today, time to market is just as critical as functionality for a product's success.

Tools for writing assembler and C code for embedded systems tend toward text editors and command-line tool chains controlled by esoteric tools such as Make. These work, but they severely limit the productivity of developers. Java developers have a great deal of choice of commercial as well as free and open source integrated development environments (IDEs). Tools such as NetBeans and Eclipse make writing code much easier and less error prone. Features such as automatic code completion, syntax checking as you type, and integrated documentation all help to get the job done more quickly and make the code more robust. As rich client platforms (RCPs), these tools are easy to extend, so they can be tailored to the needs of specific types of application development. Development time can be greatly reduced by building emulators for the target hardware to work with a desktop-based IDE.

* WINSTON CHURCHILL, BBC RADIO BROADCAST, FEBRUARY 9, 1941

8 DEVELOP ON ONE MACHINE, DEPLOY TO ANOTHER.

Embedded systems by their very nature tend not to resemble desktop or laptop systems. Frequently they are what is called *headless*, meaning that they have no display attached. This adds extra challenges and complexity to software development in terms of how to compile the code on one machine and deploy it to another.

Because Java uses a VM, it doesn't matter where you compile your code, so there's no need to set up complex cross-compilation tool chains. Use your favorite Java IDE to develop the code in comfort on your desktop, and then simply copy the class files or JAR files to the target device. To make developers' lives even easier, most IDEs support the concept of remote debugging through a network connection. If the code is doing what you told it to do, rather than what you wanted it to do, simply use the debugging facilities in your IDE to figure out what the issues are. This will be a lot faster than using print statements.

9 NINE MILLION DEVELOPERS CAN'T BE WRONG.

Depending on whose survey you consult, there are up to 9 million developers in the world who know how to program in Java. Almost all universities use Java as a teaching language for object-oriented programming fundamentals, which means that the number of Java developers is likely to increase.

If you want to develop Java code, you have a huge pool of programming talent to draw on. Because the JVM and the class libraries abstract away much of the complexity of developing embedded system code, you don't need to find people who have lots of experience in this field. For situations where more-domain-specific knowledge is required, there will still be a larger number of Java-capable developers to fill these positions.

BIG TALENT POOL

Because the JVM and the class libraries abstract away much of the complexity of developing embedded system code, you don't need to find people who have lots of experience.

10 FROM DEVICE TO DATA CENTER . . .

For embedded systems to have real value, they need to be networked, not just to other embedded systems but also to places where data can be aggregated, analyzed, and searched. This is what data centers are good at, recording and processing huge amounts of data from many different sources and then mining the data for useful information. What is the most popular platform for developing enterprise applications? Java. Using Java in embedded devices, in the data center for big data processing, and on the desktop for presenting and controlling the whole system makes architecting a complete solution simpler, quicker, and cheaper.

As you can see, Java offers benefits for developing code for embedded systems. With lower cost, more choices of platform, and readily available skilled developers, there's nothing holding you back from helping to build the Internet of Things. `</article>`

Simon Ritter is a Java technology evangelist at Oracle. He has worked in both Java technology development and consultancy. He now focuses on the core Java platform and Java for client applications.