New Java variants target deeply embedded systems

PORTABILITY, SECURITY, SIMPLICITY, AND SHRINK-WRAPPED TOOLS: JAVA HAS A LOT TO OFFER FOR EMBEDDED DESIGN. NOW, FORMALISED DEFINITIONS OF JAVA'S COMPACT VARIANTS, TOGETHER WITH NEW LICENSING TERMS, DEMAND A FRESH LOOK AT THE LANGUAGE AS AN OPTION FOR SMALLER SYSTEMS.

Were it not for Java’s obvious attractions, there would probably be little pressure from system designers to enable its use in deeply embedded systems. In its original form, Java is slow, occupies far too much memory, and cannot offer deterministic performance. But the appeal of its ability to run binary identical code on any suitable host architecture is simply too great to ignore. This ability has spawned a great deal of effort by independent tool and operating-system (OS) vendors, as well as by Sun Microsystems, to get Java working in the embedded-system domain. However, as yet, few demonstrable commercial applications exist.

Sun developers wrote Java as a pure object-oriented language. Originally, a virtual machine (VM) ran the language in interpreted form; the VM was an application that you could port to a variety of OSs and processors. Although none of these components was new, together they sparked a vision of the language as the cornerstone of a new model of networked computing.

BUILT FOR DESIGN REUSE

System developers in networked computing regard architecture independence as Java’s overwhelming advantage; this architecture is usually expressed as write once, run anywhere (WORA). WORA is attractive for deeply embedded designs, although it is not as important as it is in the networked-computing environment. Using Java is appealing because, in prin-
ciple, you can write the code on any platform, using a rich variety of off-the-shelf development tools. You can then move the unaltered code to your target, where it executes identically. Almost by definition, Java enables design reuse: Code modules written entirely in Java to execute a particular function will function identically in a later design, even if you totally revise the hardware environment.

By the same token, you can issue specifications for software functions—engineers with no knowledge of the project’s hardware can write the code for these specifications—and you can expect that code to work. In an industry in which contract design plays an increasingly important part, you may be able to access a wider pool of coding expertise in the shape of programmers who have become adept at Java for commercial applications.

If you want to move your software development to a “purer” object-oriented approach, Java is smaller and simpler to use than C++. Tool vendors offer differ-
ent figures, but all agree that today, most high-level-language coding for deeply embedded design is in C rather than in C++. They also agree that C++ is often used as an enhanced C, rather than as a pure object-oriented style.

FUTUREPROOF PRODUCTS

Software vendors report that the most important factor that attracts designers of deeply embedded systems to Java is the ability the language gives them to upgrade and add new functions to products in the field.

It is also important that the Java VM protects the underlying OSs and hardware. When you put a product into the field today, you write the code for any upgrades installed on the product. Your service organisation or the product's user—following your instructions—installs firmware revisions. You have a reasonable degree of control over the process. Java gives you a much greater degree of flexibility for product upgrades. Code to give your product new functionality may not originate within your company, so it is essential that new code cannot crash the product. The Java VM gives you that protection. Before it interprets and runs the Java byte code (which is the instruction set for the VM), it carries out checks to verify that there is no improper access to the OS or hardware.

A layered architecture (hardware/OS/VM) also places limits on Java systems, because it adds to the memory footprint and slows execution. (However, the use of an interpreter is a bigger factor in limiting performance.)

If you are working with any sort of real-time system, your biggest reservation about Java is likely to be its lack of deterministic response. The main contributor to that lack of response is Java's system of memory garbage collection. The VM dynamically allocates memory to, and loads, classes and methods at runtime. When it begins to run out of memory, it marks objects that have finished execution, deletes them, and returns the memory to its pool of available memory. The problem is that the original Java garbage collector runs when available memory gets low, with timing that you cannot predict.

LOOSEN THE REINS, HOLD THE STANDARD

Sun has gone to both legal and technical lengths to maintain the integrity of its definition of Java. One result has been that a number of software companies have written implementations of the Java VM that the companies claim are compliant with the published specifications of the language. As the authors of these versions started with a clean sheet and without reference to Sun's own technology, these versions are known as "clean-room" versions. In several cases, these suppliers have attempted to extend Java's applicability by addressing some of the limitations of the VM in footprint and performance. This situation is still evolving. Sun now recognises a wider range of uses for Java with its family of Java variants, including PersonalJava and EmbeddedJava. Sun published its latest specification for EmbeddedJava in December 1998. Together with changes in Sun's licensing terms (see sidebar "Widening the Java community"), this change has brought back into the fold several of the clean-room versions. Whether this convergence will continue, with Sun remaining at the heart of the mechanism that controls the definition of the "true Java," or whether you shall see a proliferation of Java variants is an open question.

Sun targets the EmbeddedJava application environment at devices that have dedicated functionality and severely limited memory, with no requirement for general-purpose Web browsing. The application environment allows you to construct a suitably restricted set of libraries for each application. The process basically eliminates from the general set the fields and methods that your application does not require, and automated tools exist to help in this process. Once you construct your minimal set of resources (and bearing in mind any resources that you might need for future code upgrades), you can then place an executable-code image in ROM. Sun's plans also anticipate that EmbeddedJava applications will run with a conventional real-time operating system (RTOS). Future releases of the EmbeddedJava specifications will have their own real-time extensions.

MINIMUM SYSTEM IMPLEMENTATIONS

Tool and OS vendors are applying a number of techniques that promise to make Java more usable on a smaller scale.
and with an RTOS. No one expects Java (at least, when you run it with the interpreter and class libraries loaded) to be capable of hard real-time response. Many vendors aim their efforts at allowing Java to coexist with established RTOSs. The result is, typically, a port of a Java VM that runs as a separate task or process on the RTOS. Java offerings exist that allow you to run Java with a much-reduced memory footprint and better performance. Some of these offerings reduce the RAM demand by running as much of the language out of ROM as possible. A number of approaches exist to speed the garbage-collection process and improve its predictability.

One approach to improving performance is the just-in-time (JIT) compiler. Like an interpreter, this compiler operates at runtime, recompiling each time you run the code. Rather than incurring a time penalty, which occurs when the VM interprets each instruction in turn into the appropriate byte code, the JIT compiler operates ahead of program execution. As well as saving time, the JIT compiler also incorporates compile optimisations that are unavailable to an interpreter. A JIT compiler is also memory-resident, and it requires substantially more memory than the interpreted version. Therefore, most vendors do not offer it as an appropriate option for resource-constrained, deeply embedded designs that are memory-constrained.

INTEGRATION WITH RTOSs

Every RTOS vendor now has a Java strategy. Usually, this strategy involves a port of a Java VM together with measures to keep its operating footprint within manageable size. The changes that the vendors most commonly apply involve minimising the loaded libraries, placing as many of Java’s resources as possible in ROM, and changing the action of the garbage collector.

QNX claims that there are similarities between the way it has engineered certain key aspects of its Neutrino OS and some objectives and strategies of the Java language. Java has WORA features; Neutrino’s drivers, file systems, and other OS services can be source-code-identical across CPU platforms. QNX licenses the Chai VM from Hewlett-Packard. Chai is a clean-room VM that has a full bytecode interpreter, a Java native interface for switching into C or C++ code within the program flow, a dynamic class loader, and an incremental garbage collector. Chai is ROMable and has its own set of core libraries. Like other clean-room VMs for embedded applications, you can optimise and “subset” Chai ac-

**INSTRUMENT DSP RUNS IN JAVA**

An embedded-Java application that is already in the marketplace is the new series of Tektronix (www.tek.com) TDS 500/600/700 oscilloscopes (Figure A). Tektronix’s main reason for selecting Java was to use it as the means of adding more application-specific features to the instruments, although the Tektronix team also mentions virtually every other feature to which you route the language’s proponents usually quote in its favour.

Write once/run anywhere is an attractive feature for Tektronix designers, because it allows them to reuse code across multiple platforms, and it eases the porting of applications between instruments. Also in the language’s favour is the aspect for which Doug Shiffle, software-applications engineering manager, coins the term “write anywhere/run anywhere” (WARA). With development teams in Beaverton, OR, and Singapore and code writers in India, China, and Japan, using secure and reusable techniques to rapidly integrate code becomes an important factor.

Tektronix sees this trend going beyond the initial development team to enable field-application engineers and, ultimately, customers to write application-specific routines for the instruments, says Tom Weiss, worldwide-business-development manager. Tektronix intends to exploit the security features of the language to ensure that you cannot accidentally overwrite, by errant code, fundamental operating parameters, such as stored calibration constants. The architecture of the scopes provides a further level of protection; in these scopes, a thin layer of services sits on what is essentially a general-purpose instrument, and GPIB is part of that layer. The Java virtual machine (VM) sits on top of the same service layer and communicates with the measurement system via the GPIB. The Java application is therefore also “location-independent,” in that it can be at any location to which you route the GPIB.

Tektronix already had experience with object-oriented tools and chose to use Wind River System’s port of Java onto VWorks for development of this series of scopes. Previous object-oriented programming was mostly in SmallTalk. There is obvious appeal in writing code on low-cost platforms, such as PCs, with a rich set of low-cost tools, and then downloading the exact same code to the instrument. The architecture carries out DSP in Java. John Calvin, software-applications engineer, notes that the instruments’ basic measurement functions are real-time operations that return values to the Java program, which carries out the processing.

Tektronix has found that in writing custom applications for these instruments, typically 75% of the code is reused from previous work. Going forward, Tektronix expects that the technology will contribute to making instruments more multifunctional as software more easily manipulates basic measurement sets that you can, in turn, quickly rebuild to carry out new tasks.

---

**Figure A**

Tektronix intends that Java should provide a framework for customer programming of special instrument functions.
cording to available resources.

QNX runs every driver and OS module in its own memory space, which the memory manager protects. As a result, QNX claims it can provide reliability consistent with Java objectives. For example, the OS can recover from a failure of a device driver anywhere in the system, without impact on any other process and without endangering the entire system.

More than three years ago, when Wind River Systems (WRS) began to port the Sun VM onto its VxWorks OS, one of the company’s objectives was to maintain the high priority that the company places on system security. First efforts at cutting the base Java system down to embedded size focused on scaling the runtime components by removing certain components, such as the graphics libraries. Within the next few weeks, WRS aims to release a port of PersonalJava, closely followed by a port of EmbeddedJava, for the VxWorks OS.

According to Jorg Berthold, WRS’s Java-product specialist, the core Java VM for all of the implementations is the same, but the memory footprint is reduced to about 2.5 Mbytes with graphics. Berthold says that the WRS implementation tightly integrates Java into the RTOS. The WRS port maps each thread in the Java program, one-to-one, onto corresponding VxWorks threads, so the RTOS never actually sees Java. But the Java task acquires optimized thread scheduling and preemption from the RTOS. When WRS carried out the first port of the full Sun VM, the company retained the original garbage-collection scheme, which, once initiated, must run to completion and cannot be pre-empted. This specific

**SUN CHANGES TACK ON JAVA IN SILICON**

For several years, Sun proposed a hierarchy of Java solutions with full software implementation at one end of the scale. This implementation would run on machines in which the overhead of the virtual machine was not an issue. For embedded applications of all types, Sun planned a series of µP cores that would directly execute the byte codes that the Java compiler produced. The byte codes would be the µPs’ instruction set. Free of the need to interpret the code, this solution would be as efficient as a conventional µP. There were to be pico-, micro-, and ultrajava cores at a range of performance points.

In the last few weeks, however, Sun has changed its approach. Sun will not manufacture or market the silicon. Rather, it will license the core technology to silicon-vendor partners. Sun’s strategy focuses support on the companies that license the core Java-processor technology: picoJava. The company will emphasize development of new picoJava cores and the associated infrastructure for its licensees. Sun has built a processor based on picoJava, microJava 701, that it will use as a proof-of-concept and reference design, but the company will not actively market the product. Currently, there are four licensees (LG, NEC, IBM, and Fujitsu) who will sell picoJava- based CPU designs and complete product solutions for the embedded market.

LG is developing Java processors for the set-top and wireless market, working with ETRI, a Korean research institute for developing information and telecommunications technology. ETRI’s designs include NetTheater, an Internet set-top-box reference that the company developed using a picoJava-based chip and all Java software. The box allows execution of the Java applet/application through the World Wide Web.

Fujitsu is targeting digital consumer and networking personal-computing and communications products, including smart, handheld devices; digital-still-communications markets; and car navigation systems. Fujitsu offers a JavaOS evaluation board, J- StarterKit for SparcLite: a corresponding J- StarterKit for picoJava will follow.

NEC believes that the picoJava core is well-suited for use as an IP core, and company officials say that NEC will focus on using the picoJava core as an IP core in system-on-chip embedded controllers. The company has developed an evaluation chip of the picoJava core, which it has run on an evaluation board. In the near future, the company aims to add a real-time-operating-system kernel and Java runtime to the software environment.

Meanwhile, Sun continues to use the microJava 701 CPU as a reference platform for benchmarking and development, validation of the picoJava-II core, software porting, and rapid prototyping. Sun will also offer a microJava 701-based reference board with a complete working environment (based on Java OS) and a set of development tools (including C/C++ compilers from Metaware).

In future development, Sun officials say the company will produce the occasional microJava core to further develop the line and to act as a demonstrator. Development and production of application variants will rest with the silicon licensees in a fairly conventional IP relationship.
A shared library mechanism that loads only when required by program flow. Microware provides a mechanism, which the company bases on Sun/JavaSoft tools, to convert class files into a library module for loading into ROM. The VM can run such files from ROM on a shared basis, rather than loading instances of them into RAM, saving memory.

You can choose multiple Java ports from Integrated Systems (ISI). The company has at least three offerings, the newest of which is the HP Chai VM. For small footprint, real-time applications, ISI has licensed and ported the Portable Executive for Real-Time Control (pERC) from NewMonics to its pSOS RTOS. The product is a Java clean-room implementation that adds real-time extensions and runs in a memory space of 0.5 Mbytes. NewMonics aims pERC at deeply embedded applications. Tools for pERC include a means of creating native methods and a ROMizer that creates an image of pERC code that you can write into ROM. ISI also has a port of the Sun VM running as a single task in pSOS.

Tao Systems specialises in very-small-footprint OSs, including its Elate product, which have mostly found application in Internet appliances. However, company chairman Francis Charig sees no reason why you should not use the OS, and the Java port that runs on it, in a deeply embedded application. The RAM and ROM footprint is kept very small (1.5 Mbytes for PersonalJava) by a system that dynamically loads not entire class libraries, but only individual methods as a Java program requires them. The Java implementation can also run on other OSs. Even for deeply embedded systems, Charig believes that reducing runtime size by dropping graphics is not a viable option in the long term; as intelligent devices proliferate, the graphics will be the means of building a compelling interface and brand identity. Tao’s Java has its own clean-room version, but with the new Sun licensing arrangements, Tao now has a Sun licence and will be certified as compliant with the Sun specification. Charig is also a vehement supporter of single standards and says that progress toward a real-time Java should be made gradually and always with the objective of maintaining a single standard for the language. “The impor-
Java in deeply embedded systems

Important issues are not about the basic virtual machine,” Charig says, “but about the added value around it: graphics and security.” He concludes, “Tao has the techniques to build a real-time Java implementation, but we will do it in line with agreed standards that come from a decision of the market.”

JENE THERAPY

When Sun changed its approach to Java licensing, Insignia Solutions was one of the companies that took immediate advantage. Insignia previously had a clean-room implementation of Java; now it has access to Sun’s source code for the Java platform. Insignia can also gain access to the test suites required to certify its implementation as fully compliant with the Sun-defined specification.

Insignia designed its JENE VM to be highly configurable and to run in the smallest possible memory space; the product’s speed and predictability also suit embedded applications. JENE comprises the Embedded Virtual Machine (EVM) and JENE Class embedded class libraries. The EVM has the entire Java feature set, including dynamic class loading, dynamic compilation, a full library set, and standard interfaces. Insignia has written a dynamic compiler that it adapts to the application and to the available resources (particularly memory). Although it is compiling, in effect, “on the fly,” the compiler never interferes with program flow in either time or memory occupied; Insignia structured the device this way. The compiler operates on the program in segments, an approach that occupies less memory space and allows optimisation techniques that are closed to a JIT compiler. The total EVM occupies less than 1.5 Mbytes, and you can optimise this amount further. Some of the available settings include the number and size of buffers for compiled code (more and bigger equates to a bigger footprint but more flexibility for the compiler), the size of the object heap and system heap, the optimisation level, and memory the compiler uses for data structures. Insignia describes its garbage collector as precise, accurate, and con-

FOR MORE INFORMATION...

For more information on devices such as the ones discussed in this article, circle the appropriate numbers on the Information Retrieval Service card or use EDN Europe’s Express Request service. When you contact any of the following manufacturers directly, please let them know you read about their products in EDN Europe.

Diab Data
Munchen, Germany
+49 89 9393191
www.ddi.com
Circle No. 480

Geoworks
Marklesfield, UK
+44 1625 503305
www.geoworks.com
Circle No. 481

Hewlett-Packard
Bracknell, UK
+44 1344 360000
www.hp.com
Circle No. 482

Insignia Solutions
High Wycombe, UK
+44 1628 539500
www.insignia.com
Circle No. 483

Integrated Systems (ISI)
Stevenage, UK
+44 1438 756511
www.isi.com
Circle No. 484

Lucent Technologies Bell Labs
Murray Hill, NJ, USA
+1-317-322-6684
www.lucent.com
Circle No. 485

Micro Digital via Comsol
Addlestone, UK
+44 1932 829460
www.smicinfo.com
www.computer-solutions.co.uk
Circle No. 486

Microteq Division, Mentor Graphics
Newbury, UK
+44 1635 811600
www.mentor.com/microteq
Circle No. 487

Microwave
Burnham, UK
+44 1628 667578
www.microwave.co.uk
Circle No. 488

NewMonics
Ames, IA, USA
+1-515-296-0897
www.newmonics.com
Circle No. 489

NSICom (Negev Software Industries)
OrYehuda, Israel
+972 3 533 1976
www.nsicom.com
Circle No. 490

QNX
Chorleywood, UK
+44 1923 28
www.qnx.com
Circle No. 491

SDS
Berhamsted, UK
+44 1442 870605
www.sds.com
Circle No. 492

Sun Microsystems
Bagshot, UK
+44 1276 451440
www.sun.com
www.java.sun.com
Circle No. 493

Tao Systems
Reading, UK
+44 118 910 2999
www.tao.co.uk
Circle No. 494

Transvirtual Technologies
Berkeley, CA, USA
+1-415-492-9062
www.transvirtual.com
Circle No. 495

VenturCom
Cambridge, MA, USA
+1-617-661-1230
www.vci.com
Circle No. 496

Wind River Systems Europe
Swindon, UK
+44 1793 831831
www.wrs.com
Circle No. 497

PicoJava silicon and technology licensees

Fujitsu Microelectronics USA
San Jose, CA, USA
+1-408-922-9104
www.fujitsu.com
Circle No. 498

IBM Microelectronics
White Plains, NY, USA
+1-914-872-5582
www.ibm.com
Circle No. 499

LG Semiconductor
Seoul, Korea
+82 4312702231
www.lg.co.kr
Circle No. 500

NEC
Tokyo, Japan
+81 44 548 7398
www.nec.com
Circle No. 501

Sun Microelectronics
Frimley, UK
+44 1276 416342
www.sun.com
Circle No. 502

SUPER CIRCLE NUMBER

For more information on the products available from all of the vendors listed in this box, circle one number on the reader service card. Circle No. 503
servative. Here, garbage collection is a concurrent process that runs as a thread within the VM, with a priority that you can set. You can also signal from the application when it is safe for memory compaction to take place, so that the largest possible contiguous block of memory is free for use. Insignia ports JENE to OSs including VxWorks and Windows CE.

COMPiled JAVA KEEPS MOST BENEFITS

Diab Data offers an alternative route into Java; its FastJ compiles Java (source or byte codes) to native machine instructions just as compilers do for any other language (Figure 2). This strategy sacrifices Java’s binary portability, but the sacrifice may be of little consequence if you have one target environment. You still get most of the other advantages of the language, Diab says, and you don’t have the overhead of a VM. If you want to take a gradual approach to learning and using Java, you can mix modules of Java code with others (C++, C, or assembler, for instance). Diab has written the compiler front end to fully comply with the Java language specification 1.1, while allowing the back-end optimisations already in its compiler suite to operate. You can write Java native methods in C, so that you can gain direct access to hardware resources from within the Java code. You can select whether to have many of the checks that Java performs, as these checks carry an overhead in compiled code size. You can develop with the checks on and recompile production code with them turned off.

Diab bases its compiler on cleanroom, configurable versions of the Java core libraries. The company removed many of the interdependencies so that the compiler loads only the library functions it requires. Alternatively, you can configure the libraries to match what your code uses. Diab addresses the garbage-collection issue with several options. You can choose not to run a garbage-collection routine at all but to use a facility to deallocate memory when it is no longer required, or you can pre-allocate memory at the start of a program. That option may use more memory, but it also removes all the main obstacles to deterministic behaviour. If you prefer to retain Java’s dynamic memory management and protection against memory leaks, Diab provides both a standard nonincremental routine that halts tasks while it runs (if your application can tolerate it) and an incremental, preemptible routine that runs as a low-priority background thread. A suite of tools supports using Java in a mixed-language, build/make environment.

With the compiled solution and without the VM, of course, you lose the ability to implement field upgrades and create new hardware functions without compiling and loading new code, but this approach does provide a route into using Java. In the United Kingdom, SDS represents Diab Data. SDS’s European Vice President Geoff Revill notes that users would like to have the benefit of all the features that currently carry a penalty (for example, WORA attributes and graphics capabilities), but these features are less important to many users in the embedded space than the ability to impact time to market and access a wider range of code-writing talent.

“We need to watch developments in the commercial software sector. There’s a lot of work going in the universities around Java; abstracting away from the hardware may allow us to tap into a bigger pool of programmers,” Revill says. Other advantages to using Java, Revill notes, are its simplicity and the short time (compared with C++) it takes for a designer to become effective in writing it. The Diab Data offering is a no-cost option to its compiler and debugger tool set.

From his SDS perspective, Revill also has some observations on the likely shape of an effective embedded debugger for a VM-based embedded-Java approach. The product would have to be three-level, with source, VM, and real-machine views. Should a correctly functioning VM not dispense with the need to view the hardware? Not if you want to use native methods to directly access hardware (as you certainly do in an embedded design). And you would need access to a view of hardware activity to port the VM in any case, Revill notes.