

# Java gets real for embedded SW development

*Java is a clean, modern, easy to learn, network centric, object oriented language originally meant for embedded devices. After some initial interest in Java by the embedded community Java has not yet become mainstream because of memory and processor constraints. With the emergence of new low cost, low power application specific 32 bit embedded microprocessors at an astounding rate, embedded engineers find themselves relying heavily on software to add value to their products. This fact is triggering renewed interest in Java as a productive platform for embedded applications of the future. This article describes the current state of Java and some interesting (RT) Java implementations.*

## JAVA LOOKS BACK FOR A BRIGHT FUTURE

The embedded industry has typically always been in a catch 22 situation needing more functionality for less money. A low consumer price target had the net engineering result that embedded SW engineers faced the struggle with limited performance, limited memory and limited bandwidth constraints. If we look at the desktop market as a predictor for the embedded market then it is clear that about 10 years ago desktop (PC) SW development was bound by the same constraints in price, performance, memory and bandwidth. At that time, 1989, in an editorial column of the Journal of Object Oriented Programming Wilf Lalonde saw a strong growth and interest in OO techniques and he predicted that it would presumably take the SW industry another 10 year to absorb OO thinking and methods into their day to day software development activities. In hindsight a very true statement. With Java, C++, Java Beans and COM, OO and component development have been firmly set on the agenda. I am confident that within the Embedded sector we are at a crossroad. As new low cost and low power 32 bit embedded microprocessors are announced at an astounding rate and embedded engineers are finding themselves relying heavily on software to add value to their products. Java definitely brings advantages for embedded development. The dominant question however when thinking about Java is: can Java meet the embedded world requirements?

## JAVA: THE BENEFITS FOR EMBEDDED SW DESIGN

Java is a modern, simple, safe, OO based language with a network centric architecture and a good security model. Java is defensive, in the way that it does not allow direct memory manipulation or pointers and it has automatic garbage collection to free memory after use. Java is platform independent through a virtual processor architecture in software which gives an

abstraction layer to the underlying hardware. Java source code is compiled into bytecode which is stored in class files. Bytecodes are executed on the virtual processor. Portability is thus very, very simple compared to C/C++ also labeled to be languages for portable SW systems.

Java is productive. First code-metrics from the admin world are available (ref. [1]) and Java is about 2 times as productive compared to C++. Despite what is frequently regurgitated in the press I found Java generally not slow and non-GUI stuff is comparable in speed to C++. At Mountside we did our own benchmarking comparing C++ and Java on a number of Operating Systems (JDK 1.1.6 based) and we found number crunching and I/O to screen or network comparable and sometimes faster than equivalent C++ programs. Multithreading, GUI operations and reading or writing to files was comparable but sometimes slower than equivalent C++ solutions. These findings are just a moment in time as we now have Java 2 with more advanced compilation, optimized multithreading and performance optimized class functionality. Our general conclusion about performance issues is: find out for yourself and don't believe the press because it's a complex issue and it doesn't fit into a few 'soundbytes'. For embedded benchmarking the embedded caffeine marks are sometimes quoted when reporting about embedded Java performance and they give at least some impression about system performance.

While the Java language and its performance may be critical, more important is functionality. And in this area no embedded development platform can match Java. Leverage is gained using the Java Development Kits which contain vast amounts of predefined functionality: pretested, open and at your disposal. Communication, 2D/3D graphics, Multithreading, GUI functionality, Automotive, Telephony and TV API's. All these features should make Java platform of choice for high level embedded design. However tradeoffs have to be made and also with Java technology there is no free lunch.

## JAVA PROBLEM AREA'S FOR EMBEDDED DESIGN

When evaluating Java as platform of choice for embedded design memory footprint, performance as well as predictability and real time capabilities are recurring issues. The many different JDK's are large and so is their hunger for memory and cycles. For efficient Java use, a 32 bit processor is almost a must. When Java was first introduced about 4 years ago this was too steep a hurdle for many embedded designers. Nowadays 32 bits processors are a growing part of the embedded market.

Footprint is also an issue. Java, as mentioned, uses a software implementation of a processor model: the Java Virtual Machine. This adds an abstraction layer to the underlying hardware which increases demand on memory.

Typically embedded RTOS applications use priorities for guaranteeing the correct level of responsiveness in an application. In Java there are only 10 priority levels. This may be a problem when developing complicated real time software.

In Java direct memory manipulation is impossible. Integrating legacy code, drivers or specific HW functionality must be done via Java Native Interfaces to mostly C code. This isolates spots of non-portability but if many JNI calls are used, code becomes as hard to port as regular C code. Memory management, a frequent and nasty source of problems in C/C++ code, is done automatically in Java. Gone are the days of weeding through C/C++ source to find your allocation errors. Through Garbage Collection memory no longer in use is freed. GC is powerful feature however it makes the behavior of standard Java programs in time critical sections unpredictable as you don't know when Garbage Collection takes over your process. Many commercial RTOS vendors of Java systems have implemented alternative GC's to avoid his behavior by implementing special garbage collection schemes which still guarantee responsiveness.

## JAVA: SOME PROBLEMS BUT HOPE FOR EMBEDDED SW DEVELOPMENT

Because Java holds such an unparalleled promise to bring embedded software development up to the next level of productivity many commercial vendors have addressed the problems sketched just now.

- Of course all the major standard RTOS vendors like VxWorks, OS/9, QNX and many, many others also support some form of Java on their platforms. Many of these implementation guarantee only soft real time response and leave hard real time response

mechanisms to RTOS native control tasks whereby a mechanism is provided to communicate from the Java environment to the native OS. Also garbage collection has mostly been enhanced as to guarantee (soft) real time response when using Java. Garbage collection in Java enabled RTOS systems is mostly done using a 'mark and sweep' garbage collector which may be interrupted if properly designed.

- **Kjava:** Sun has released Kjava on JavaOne 99 suited for 16 Mhz+, 16/32 bit RISC/CISC micro-controllers. Only 128 KB of RAM are required for Kjava where the KJava VM takes up only 40 KB in size. This Java implementation is 'bare bones' and does not support doubles or longs and uses specially adapted minimal JDK's. Furthermore KJava needs an underlying OS. PalmOS on the Palmpilot has been chosen by SUN as reference platform for KJava.
- **Jeode:** Insignia solutions has released the Jeode Java Environment for x86, ARM, MIPS, PowerPC and SH processors for operating systems like CE, VxWorks, ITRON. Insignia uses a highly optimizing Just In Time (JIT) compiler with proprietary optimizations which will deliver fast and small code. (250K-2.3 MB depending on JDK support) Insignia delivers quite a full JDK with a rich set of functionality as you see from Figure 1.

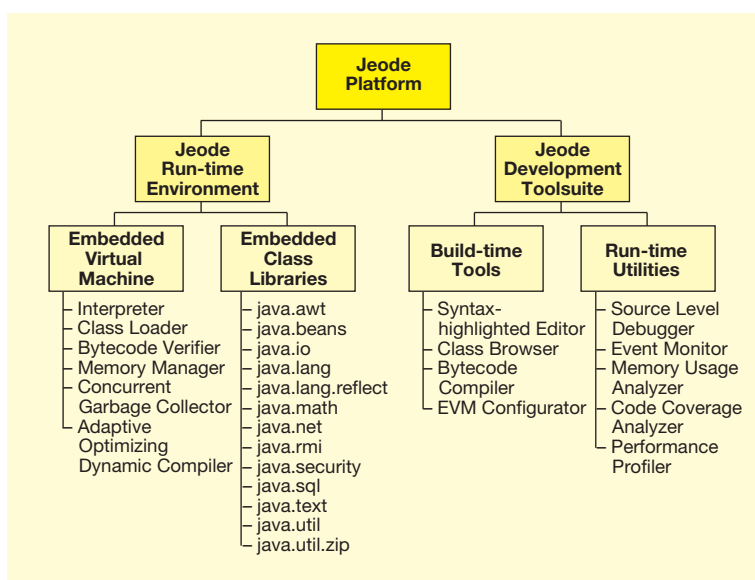


Figure 1. The Insignia Jeode Classes

- **Chai:** Chai is an initiative by HP supported by Microsoft at producing a Java compatible Virtual Machine for embedded applications. The ChaiVM uses about 228 Kbytes excluding further necessary support classes (these are about 189 Kbyte). The ChaiVM uses also a Mark-Sweep algorithm capable of running in the background for memory management.
- **Tao's Elate J-Engine** combines also small footprint and fast execution. Elate is an object oriented networked RTOS using a JIT compiler. Most code,

including kernel and device drivers are written as Virtual Processor code, an imaginary 32 bit RISC processor model (VP2 code). This ensures a highly portable architecture. At present Elate is available for: (Strong)ARM, x86, ColdFire, M-Core, PowerPC, SH3/4, and MIPS.

The Virtual Processor technology ensures that Elate is truly binary compatible across different platforms and CPU's. Either during statically at build time or dynamically at run time VP2 code is converted to native code. Dynamic binding is therefore possible in an embedded system only 70kB in size. Multimedia capabilities are added to Elate by means of an Audio Visual Environment (AVE), written in VP2. All AWT classes are recoded in VP2 using AVE, making a full Personal Java 1.1 set available in appr. 1,5MB ROM space.

It is clear that traditional RTOSses, Kjava, Jeode, Chai and Tao's Elate may complicate the choice for a particular Java environment. In general one should be very careful when selecting an RTOS Java environment because often important classes, methods or features have been left out by the RTOS vendor in order to minimize footprint and to maximize performance. Also look very careful which JDK's are supported. For instance, with a vendor not supporting the AWT classes there is little chance of getting a decent GUI running on such a system and all other JDK's depending on the AWT classes may not be used on such a system.

## JAVA AND HARD REAL TIME RESPONSE

Combining a traditional RTOS with Java opens up legacy code. Sometimes this is not necessary and fresh new idea's blossom: like Jbed by Esmertec. Jbed has a number of remarkable features. Implementing a JVM may be done in several ways (see Figure 2). Jbed has chosen to embed the OS in the JVM immediately on the bare hardware. In this way typical OS functionality like: thread scheduling, memory management, I/O drivers, network drivers and file systems are implemented using pure Java. In order to do this Jbed provides a couple of special library functions which allow direct memory manipulation and interrupt handling. These functions are only used in system drivers and are not available for application development.

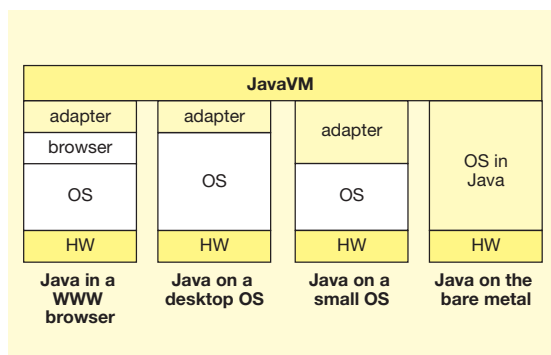


Figure 2. Possible JVM Implementations

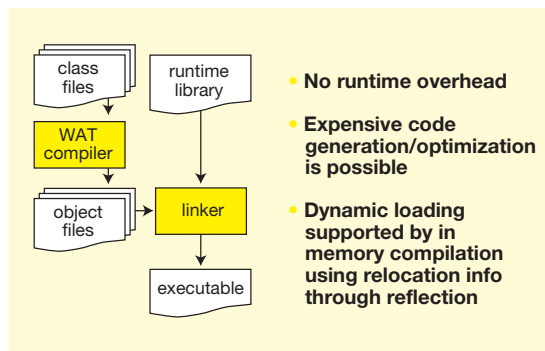


Figure 3. Way Ahead of Time Compilation (WAT)

The Jbed RTOS is based on the safe language concept Java as such a safe language inherently does not allow direct memory manipulation. So why not abandon memory protection schemes which are must when using unsafe C/C++ and traditional RTOS systems. In this way a scalable small, fast system may be produced, very light on overhead. The size of a Jbed system with RT-tasks, threads and GC can be scaled down to about 100K, Timeslices on a 50MHz PowerPC go down to 100µs.

For embedded code generation Jbed employs Way-Ahead-of-Time compilation (WAT) (see Figure 3). Java class files are compiled into native code which is issued on the target. Dynamic loading of classes is difficult in WAT compilation. Jbed uses Java reflection in a clever way extracting relocation information from it in order to link in and load classes at run-time.

The Java thread model is too weak to handle hard real-time requirements. In Jbed Esmertec extended the thread model with so called tasks. Threads are standard Java-Threads, whereas Tasks support hard real-time constraints. At creation time every task specifies a duration and deadline. The scheduler runs the task with the closest deadline first (see Figure 4). Earliest deadline first scheduling enables admission testing in this way up front it can be guaranteed that when a task is admitted that it will meet it's deadline.

**The task nearest to it's deadline is scheduled first. Task r is admitted because there is time left to execute this task. Admission testing ensures that tasks added dynamically are guaranteed to execute.**

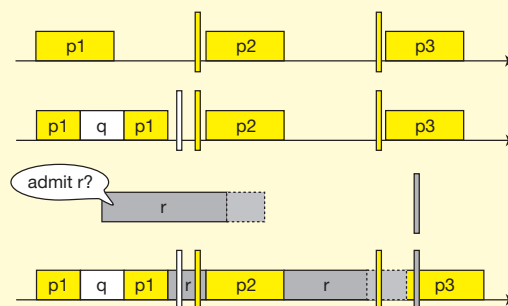


Figure 4. Earliest Deadline First Scheduling

## DOING IT YOURSELF JAVA

Based upon the Elate J-Engine Snijder Micro Systems has developed a Personal Java platform for embedded applications. The unit has been developed for OEM applications and may be used as a stand-alone or networked Personal Java Platform. The 144 pin SO-DIMM form factor provides many I/O capabilities and makes design-in fast and straightforward (see Figure 5).

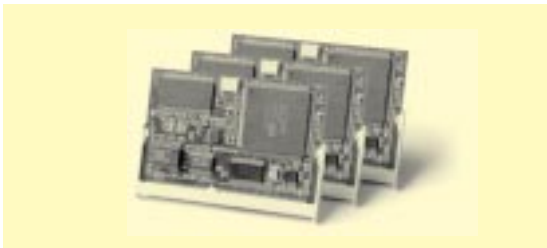


Figure 5. Snijder Java hardware

The module is built around a highly integrated CPU with an ARM7 32 bit RISC core and onboard peripherals such as serial ports, LCD controller and digital I/O lines (see Figure 5). The main external components are an Ethernet controller and memory.

Native ARM code is written for booting, configuration and first level interrupt handling. The device drivers for Ethernet, graphics and serial I/O are all written in elate's virtual processor code (VP2). No special drivers need to be written for the JVM as the Elate J-Engine is written in VP2 itself and uses the AVE tools for the GUI implementation.

Writing a VP2 device driver is comparable to writing a driver in assembly. For different CPU's like MIPS or PowerPC exactly the same VP2 driver source may be re-used. Headaches about moving from a little endian to a big endian CPU and vice versa are nonexistent. The filing system, graphics toolkit and network protocols are based on the low level drivers and need not to be rewritten.

With this project the Object-Oriented nature of the Java-language has been extended completely down to the OS the underlying hardware. Building blocks like Ethernet or video chipsets come with their (portable) drivers. A fully customized PersonalJava controller comes within reach, also for smaller quantities.

The effort resulted in a 68x45mm sized SO-DIMM module with an ARM7 CPU, 10BASE-T Ethernet, three serial ports (RS232, TTL and RS485), 8MB DRAM, 4MB FLASH and an expansion bus. It only requires power to run Java Byte Code instantly.

This project was executed in a few man months by a small team. It demonstrates that a full featured specific Java hardware environment may be written on custom specification with only limited resources.

## CONCLUSIONS

Java comes of age. In 'advanced functionality' application areas as for example settop boxes use of Java is definitely growing. This will also trickle down to more

constrained embedded settings. Java is a safe language and new and exciting ways of thinking about RTOSes are emerging as in Jbed. Even making a JVM work on your own hardware is not that hard as the experiences from Snijder microsystems show.

In functionality Java is almost unbeatable. The functionality provided in the JDKs makes reuse possible and Java as a clean OO language is an enabling factor for future embedded devices. Many Java initiatives are released or under way for almost every significant platform. The hardest part will be the cultural change in the embedded community from a relatively constrained hardware base with highly tailored applications to a more wasteful use of hardware resources in order to build applications faster, more extensible and network savvy. I see Java picking up in the embedded sector. I hope it takes less than 10 years for the culture to change and for Java/OO to become mainstream in the embedded world. Maybe we should ask Wilf Lalonde about his 1999 predictions for OO and the embedded sector? ■

## REFERENCES

- [1] Percentage cost savings in 9 Java projects, Source: IDC, May '98, "Java Pays - Positively".

## WEB RESOURCES

[www.insignia.com](http://www.insignia.com)  
[www.esmertec.com](http://www.esmertec.com)  
[www.java.sun.com](http://www.java.sun.com)  
[www.snijder.com](http://www.snijder.com)  
[www.hp.com/emso](http://www.hp.com/emso)  
[www.jug.nl](http://www.jug.nl)

---

*Joost Backus joined the management team of Mountside software engineering in 1996, previously he has held various functions in the IT industry and applied IT research. Joost is currently also chairman of the Dutch Java User Group called Dutch Melange.*

*Nick Snijder founded his own HW company called Snijder Microsystems in 1986 and is actively involved in the embedded Java market. Snijder Microsystems has released it's latest embedded Java board featuring an ARM processor at JavaOne 99. Snijder Microsystems is member of the Dutch Java User Group.*