Can Real-time Software Engineering be Taught to Java Programmers?

Sally Smith, Shaun W. Lawson, and Alistair Lawson
School of Computing, Napier University,
10 Colinton Road, Edinburgh,
United Kingdom, EH10 5DT.
{s.smith, s.lawson, al.lawson}@napier.ac.uk

Abstract

For several years we have been watching with envy as specialist labs are developed for multimedia students which, together with software licenses, are now costing upwards of $5,000 per seat. We would like to be able to offer as rich a learning experience for our software engineers who study a final year module on real-time software engineering. In persisting with our students' main taught programming language of Java we argue that it is still possible to demonstrate the issues of software development for real-time systems whilst also offering realistic and rewarding practical work. Although the real-time community is still largely working in C, we believe we can educate the real-time developers of the future, and we use, as leverage for this, the ever growing body of reported work in making Java technology more suitable for time critical and embedded systems development. In this paper we present our case for a relevant real-time undergraduate laboratory based around Java.

1. Introduction

Taught modules in real-time systems design are considered an established component of undergraduate degree courses in software engineering [1]. The ability to design, implement and evaluate time critical applications is a skill package sought after by not only traditional engineering employers but also, in our recent experience, increasingly by commercial product providers such as the telecommunications and entertainment industries. In the past our own software engineering graduates have largely made use of their real-time skills on large safety-critical installations such as those deployed by the aerospace industry. By contrast, recent graduates are finding use for their real-time and embedded skills in creating applications for devices such as PDAs and other in-the-field data acquisition systems, mobile phones, and point of sale transaction units.

Our current real-time module, CO42018 Real Time Software Engineering (RTSE)[†], is delivered to final year Software Engineering and Software Technology undergraduate students [3]. The module aims to make use of software engineering and object-oriented programming knowledge acquired earlier during the degree to allow students to design and implement concurrent programming applications and to analyse the time based properties and the general predictability of those applications. Three years ago our School moved from C++ to Java as the predominant programming language that is taught to students from level one upwards. We are therefore now attempting to teach advanced, but low level, undergraduate topics, such as game design, adaptive systems, embedded applications, and real-time software engineering to students who have little or no knowledge of C/C++, or of machine or assembly languages.

[†] See module homepage at http://www.soc.napier.ac.uk/module/op/onemodule/moduleid/CO42018

Our RTSE module is administered by a Software Engineering teaching group within the School of Computing. Typically the modules delivered by our group run practical sessions on general purpose desktop machines. In other words, we deliver practical support material that enables students to design, develop and run applications under the standard Microsoft Windows XP desktop on moderately specified Pentium PCs. We use both the Sun Microsystems Java Standard Edition (J2SE) SDK and Borland JBuilder as our Java development environments. Such an environment is, generally, suitable for teaching foundational Levels 1 and 2 programming modules but lacks credibility for delivery of specialist modules. For several years we have been watching with envy as specialist labs are developed for our rival Multimedia Systems teaching group which, together with software licenses, are now costing upwards of $5,000 per seat. We would like to be able to offer a rich learning experience for our software engineers who study our final year Software Engineering modules including RTSE.

This paper describes our recent efforts in the justification, design and development of a real-time systems undergraduate teaching laboratory. We describe the options available to us, our decision to retain Java as the primary programming language despite its inherent lack of real time functionality, and the benefits offered by our chosen final solution.

2. Java as a real-time language for undergraduate teaching

The debate over the choice of programming language for teaching undergraduate computing continues unabated (see [2,3] for instance). Like many university computing departments we have, for better or worse, chosen Java as our introductory language. This choice has many perceived pedagogical benefits though poses some problems when, at later stages during undergraduate study, we encounter some specialist computing topics. In particular, Java as a language for programming real-time systems has come under a great deal of scrutiny. A key issue is that standard Java uses garbage collection (GC) which is inherently non-deterministic. A further problem is low level system or hardware access. In real-time embedded applications we often need to access the underlying hardware directly for tasks such as implementing device drivers, interrupt handling or simply accessing physical memory. The platform independence that Java offers means that the language cannot directly support calls to access the underlying hardware or even system memory. The Java Native Interface (JNI) allows the combination of Java and native code (such as compiled C/C++) that can get around such problems but JNI solutions are typically awkward and non-portable. Other problems include limited thread management, lack of priority inversion detection and restricted priority levels.

Despite all of these problems, partial solutions, workarounds and modified JVM proposals abound all of which attempt to turn the Java language into something more suitable for programming real time systems. The most well known of these is The Real-Time Specification for Java (RTSJ) created by the Real-Time for Java Expert Group (RTJEG) under Sun's Java Community Process [4]. The RTSJ proposes the creation of a new Java package, `javax.realtime`, comprising over fifty new classes and addressing issues such as more deterministic real-time thread support, timers, memory management, and physical memory access. Any RTSJ compliant JVM is expected to support this rather complex functionality and, unsurprisingly, some independent groups have proposed alternative solutions. Recently Kwon et al [5], for instance, describe a

solution that features a subset of Java and RTSJ, whilst Nilsson et al [6] describe a non RTSJ solution altogether.

The conclusions an educator can derive from the plethora of claims and counter claims against the suitability of Java, or modified, enhanced, or restricted versions of Java, is that, generally as a language for teaching basic time-dependent techniques to undergraduates on a general purpose operating system (GPOS), it is probably no worse than other higher level languages such as C and C++. The fact that there is so much interest shown by real time hardware and software vendors in the future potential of Java suggests that we are not pursuing a dead-end in using standard Java as a tool to teach real time development. The most recent edition of Burns and Wellings' well regarded undergraduate text on real time systems [7] also features Java code alongside equivalent implementations in Ada and C. The growing importance of programmable consumer devices with restricted resource profiles, such as mobile phones, PDAs and handheld games consoles, many of which support restricted variations of Java runtime environments, also serves to reinforce the future role for Java in embedded systems programming. Of course, if we were attempting to teach development of hard real time applications using genuine Real-Time Operating Systems (RTOS's) such as VxWorks or QNX then we would have to re-evaluate this stance and consider the retraining of our students in a language such as C, or even Ada. Comprehensive retraining in an alternative language, although recognised as being desirable [3], is often practically impossible given the already compressed nature of most computing degrees.

3. Possible directions in using Java to teach RTSE

To cover the main issues of real-time software engineering we would wish our students to gain practice in designing, developing, and evaluating software which demonstrates: multi-tasking using processes or threads, inter-process communication and synchronisation, pre-emptive priority scheduling policies, and dealing with deadlock and priority inversion. All of these aspects, can in fact, be covered using Java as the programming language. We now outline the different vehicles upon which this may be demonstrated.

3.1. Prioritising threads on a standard JVM running on a GPOS

A large part of real-time systems design and implementation methodology centres on concurrent programming. Solutions to anything other than trivial real-time systems problems are usually by approached by developing multiple processes, or threads, to service the multiple demands of a system. Our approach until recently has been to teach practical concurrent program issues using standard Java threads in a laboratory environment and deal with more theoretical real-time aspects, such as hard real time scheduling, in a lecture environment with paper-based work-through of examples. This approach has advantages in that (1) Java has long been recognised [8] as an excellent language for teaching thread-based concurrent programming issues, (2) we can use existing general purpose OSs to do our practical work, and (3) it is conceptually a clean solution – we can easily differentiate between concurrent programming issues whilst not dirtying our hands with hard time-critical code and peculiar pieces of hardware. Our Java based lab structure firstly teaches basic concurrency using multiple threads, covers prioritisation and the non-deterministic properties of the JVM, and then moves on to the problems of sharing data, and investigates solutions to inter-process communication and

synchronisation (including semaphores). Finally, we also compare the threads programming experience in Java with that in MS Visual C++. A typical coursework assignment has required that the students analyse the time-based properties of some 'black-box' processes (Java threads) and write applications that schedule such processes using an approximation of a real time scheduling approach, dealt with theoretically in the lectures, such as Rate Monotonic Analysis.

Our approach, of course, also has disadvantages: (1) our lectures contradict the theme of the practical sessions in that it is proven in the classroom that neither standard Java nor MS Windows XP are deterministic and therefore provide poor support for hard real-time systems, (2) students are only exposed to one aspect (concurrency) of real time systems programming and many others (for instance device drivers, interrupt programming, high resolution timers, sophisticated time-based schedulers) are only dealt with theoretically, and (3) potential employers maybe sceptical of a student who, at interview for instance, claims real-time systems knowledge but can only program in Java on a Windows XP machine. Our RTSE module practical sessions are also currently delivered in a very large open access teaching facility at Napier (the Jack Kilby Computer Centre, JKCC). This facility, whilst visual impressive and favoured by students for casual computer access and project work has numerous pedagogical limitations [9]. We argue the case therefore that we can improve our current real-time teaching provision by moving practical sessions to a specialist laboratory, and inserting a hardware element into the module whilst still retaining our concurrent Java content. It has been argued, of course, that a realistic lab experience is essential in backing up theory covered in lectures and a number of recent descriptions of real-time/embedded systems teaching labs and teaching philosophies have appeared ([10][11][12]).

3.2. Running the JVM on a processor with an underlying RTOS

Development environments consisting of general-purpose microprocessors running the JVM on top of an existing RTOS are available (e.g.[13]). The existing real-time community can migrate to the Java programming language while making using of their well-proven RTOSs. Using this architecture the JVM runs on a target processor development system and interfaces to one of a number of possible RTOSs. This is attractive for developers with legacy systems who can take advantage of the Java programming language, without sacrificing the real-time benefits of the RTOS and well-proven RTOS/ target processor combinations. Performance improvements are achieved through "ahead of time" and "just in time" (JIT) compilation.

A more cost effective solution for new developers might be a special purpose microcontroller and a Java programmable run time environment (e.g.[14]). Chipsets may include processing, control, device-level communication and networking capabilities and the underlying hardware can be manipulated by the software developer through Java application programming interfaces. The code section of memory contains a run time environment with a special purpose mini operating system and a native methods layer.

3.3. The Java Microprocessor Option

Recently launched processors now offer direct JVM bytecode execution, whereby over 99% of Java bytecodes are micro-programmed in hardware. There is no need for

an extra RTOS layer as on-chip real-time thread managers perform priority-based preemptive scheduling with thread to thread yield in under 1us. These processors support the RTSJ and provide deterministic behaviour. Cost effective development kits exist [15] and can be used in practical work interfacing to a range of devices. Other special purpose microcontrollers have been developed for the mobile technology and embedded systems market and generally support the J2ME on special purpose microcontrollers [16]. These can be packaged with rich development environments offering, for example, on-circuit debugging and performance analysis.

4. Evaluation of student performance and destinations

The first option, using Java on a general purpose operating system, has been in use since 1999. Students submit practical work based on thread prioritisation. A statistical analysis (termed a *Dbar*) is used to give an indication of how well students perform in comparison with other modules they study. A positive Dbar figure indicates a module on which students are generally performing better. In June 2003 the students taking our RTSE module generated the figures shown in Table 1.

Table 1. – Performance of students taking RTSE module compared against their performance in other modules that same semester (Spring 2003).

	Real-time Software Engineering (RTSE)	**Safety-Critical Systems**	**Languages and Algorithms**
Mean (%)	61.6	49.4	49.4
Standard deviation	13.9	13.3	14.1
Pass rate	88.5	81.5	83.3
Dbar	11.1	-4.9	-5.6

The figures given in Table 1 suggest that students perform better in our RTSE module than they do in the other modules that they take in the same semester. Questionnaire data also indicated that students enjoyed the module and engaged well with the practical work. But did their work on this module help them find employment in the sector? And are they any good at it? First destination employment information gathered by the indicates that over 50% of students who took the RTSE module found work in the telecommunications, embedded systems and defence sectors. A number of these students are still in contact with the University and have confirmed that they are working on real-time or embedded systems. Interestingly none of the students contacted are working on the same technologies. Some have had additional company-specific training and some have had to learn new languages and OSs on-the-job. This indicates that it is not necessarily the specific technological skills which are being sought after by companies recruiting new graduates, but that range and depth of subject coverage is a more important factor.

Evidence of student performance and marketability suggests that the module as it stands is successful. However, we believe that a laboratory environment, which offers a solution based on the Java microprocessor option (as described in 3.3), will improve student confidence in developing lower level real-time solutions. This laboratory has been costed at $700 per seat using the hardware system supplied by Imsys [16] but will required more technician support than our existing JVM on a GPOS environment. Less tangible benefits include encouraging peer support through collaborative work (or pair-

programming), which is more difficult to achieve in a large general purpose computing facility such as the JKCC. We will be able to gain useful feedback from student results following the delivery of the module in the new facility by comparing statistical metrics such as our in-house Dbar figures with previous years' delivery.

5. Conclusions

Students can learn most effectively when their practical work reinforces lecture delivered theory. For real-time systems modules where students have only previously learned Java, this is not as impractical as it once was. We have explored the options available to enhance the learning experience and are developing a real-time systems laboratory based on our findings. Less than 20% of our students ultimately work for companies developing hard real-time systems, and it is a typical feature of such employment that graduates are given in-house training for specialist programming languages (such as ADA 95), RTOSs and other software technologies and procedures. A larger percentage of our students find employment in the embedded systems or telecommunications sector. For these students, our chosen, and developing, learning environments fulfill employer and student expectations.

6. References

[1] Sobel, A.E.K. (2003) (ed.), Software Engineering Education Knowledge (SEEK): Software Engineering Volume, Joint IEEE Computer Society/ACM Task Force on the "Model Curricula for Computing, April 2003.

[2] Cecchi, L., Crescenzie, P., and Innocenti, G. (2003) C : C++ = JavaMM : Java. Proceedings of PPPJ2003 International Conference on the Principles and Practice of Programming in Java, June 2003, Kilkenny, Ireland.

[3] de Raadt, M., Watson, R. & Toleman, M. (2003) Language Tug-Of-War: Industry Demand and Academic Choice. Proceedings of the Fifth Australasian Computing Education Conference (ACE2003), Adelaide, Australia, 20:137-142, Greening, T. and Lister, R. (eds). Conferences in Research and Practice in Information Technology.

[4] Bollella, G., Brosgol, B. Dribble, P. et. al (2000). *The RealTime Specification for Java*. The Java Series, Addison-Wesley, Reading, MA, 2000.

[5] Kwon, J., Wellings, A.J., and King, S. (2002). Ravenscar-Java: A High Integrity Profile for Real-Time Java. Proceedings of the Joint ACM Java Grande - ISCOPE 2002 Conference.

[6] Nilsson, A., Ekman, T., and Nilsson, K. (2002). Real Java for Real Time -- Gain and Pain. Proceedings of CASES-2002 (ACM), October 8–11, 2002, Grenoble, France.

[7] Burns, A. and Wellings (2001), A., Real-time systems and programm ing languages, Addison-Wesley.

[8] King, K. N. (1997), The Case for Java as a First Language, Proc. of 35th Annual ACM Southeast Conference, Murfreesboro, Tenn., April 2-4, 1997, pp. 124-131.

[9] Buckner, K. and Davenport, E. (2002). Teaching and learning in the VLCC: actions, reactions and emerging practice in a very large computing centre. In S. Bagnara, S.Pozzi, A.Rizzo and P.Wright (Eds), *11th European Conference on Cognitive Ergonomics* pp 355-360.

[10] Skavhaug, A., Lunheim T., and Skinnemoen, H. (2003), "EmbLab - A Laboratory for Teaching Embedded Systems", ERCIM News No. 52, Jan 2003, http://www.ercim.org/publication/Ercim_News/enw52/skavhaug.html

[11] Neilsen, M.L., Lenhert, D.H., Mizuno, M., Singh G., Zhang, N. and Gross, A.B. (2002), "An Interdisciplinary Curriculum on Real-Time Embedded Systems", Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition.

[12] Dannelly R.S. and Steidley, C. "A Student Laboratory Environment for Real-Time Software Systems Development", *The Journal of Computing in Small Colleges*, March 2001.

[13] The PERC™ JVM www.aonix.com (Visited 9/3/2003).

[14] Introducing TINI: Tiny InterNet Interface http://www.ibutton.com/TINI/index.html (Visited 9/3/2003).

[15] Embedded Low Power Java Microprocessors www.aJile.com (Visited 9/3/2003).

[16] The Cjip java microprocessor. http://www.imsys.se (Visited 9/3/2003).