

# Real-Time Java

Andy Wellings and Marisol Garcia Valls

## 1 Introduction

Since its inception in the early 1990s, there is little doubt that Java has been a great success. However, the language does have serious weaknesses both in its overall model of concurrency and in its support for real-time systems. Consequently, it was initially treated with disdain by much of the real-time community. Nevertheless, the perceived advantages of Java (from a software engineering perspective) over languages like C and C++, coupled with the failure of Ada to make strong inroads into the broad real-time and embedded system markets, resulted in several attempts to extend the language so that it is more appropriate for a wide range of real-time systems. The simplest extension was Real-Time Java Threads (Miyoshi et al., 1997), which provided rudimentary real-time support. Another proposal was Portable Executive for Reliable Control (PERC) (Nilsen, 1998) that provided both high-level abstractions for real-time systems and low level abstractions to access the hardware (included is real-time garbage collection). In Communication Threads for Java (Hilderink, 1998), an approach was proposed that was based on the CSP algebra, the Occam2 language and the Transputer microprocessor. Others attempted to integrate the Java Virtual Machine (JVM) into the operating system (for example GVM (Back et al., 1998)) or to provide a hardware implementation of the JVM ( e.g., picoJava-I (McGhan and O'Connor, 1998)).

Unfortunately, much of this early work was fragmented and lacked clear direction. In the late 1990s, under the auspices of the US National Institute of Standards and Technology (NIST), approximately 50 companies and organizations pooled their resources and generated several guiding principles and a set of requirements (Carnahan and Ruark, 1999) for real-time extensions to the Java platform. Among the guiding principles was that Real-Time Java should take into account current real-time practices and facilitate advances in the state of the art of real-time systems implementation technology. There were two competing responses to the NIST requirements.

- An initiative backed by Sun Microsystems and IBM, the Real-Time Specification for Java (Bollella et al., 2000) or RTSJ. RTSJ required no changes to the Java language but did require a modified JVM and a standard package of classes.
- An initiative backed by the J Consortium (HP, Siemens, Newmonics): the Real-Time Core Extension for the Java Platform (RT Core)(J Consortium, 2000). Real-time core was based on the PERC system. It consisted of two separate APIs: the Baseline Java API for non real-time Java threads, and the Real-Time Core API for real-time tasks. Some language changes to Java were proposed.

Both of these approaches have their advantages and disadvantages. However, the language changes proposed by the J Consortium failed to find favor with the wider Java community and the work has dropped away. In contrast, the RTSJ has continued to evolve (the initial version, 1.0, has been rewritten, version 1.0.1, to make the designers' intention clearer and to tighten up the semantics) (Dibble et al., 2006), implementations have become available (for example, the Reference Implementation by TimeSys – <http://www.timesys.com>, the Jamaica VM from aicas – <http://www.aicas.com>, OVM (an open source version) – <http://www.ovmj.org>, and Mackinac from Sun – <http://research.sun.com/projects/mackinac/>) and supporting tutorials have appeared (Dibble, 2002; Wellings, 2004).

## 2 Description

The RTSJ has several “guiding principles” that have shaped and constrained the development of the specification. These include requirements to

- be backward compatible with non real-time Java programs,
- support the principle of “Write Once, Run Anywhere” but not at the expense of predictability,
- address current real-time system practices and allow future implementations to include advanced features,
- give priority to predictable execution in all design trade-offs,
- require no syntactic extensions to the Java language,
- allow implementers flexibility.

The requirement for no syntactic enhancements to Java has had a strong impact on the manner in which the real-time facilities can be provided. In particular, all facilities have to be provided by an augmented virtual machine and a library of classes (and interfaces). In places, this has had a marked effect on the readability of real-time applications.

The RTSJ enhances the Java platform in the following areas:

- schedulable objects and scheduling – pre-emptive priority based scheduling of real-time threads and asynchronous event handlers all within a framework allowing on-line feasibility analysis;
- synchronization and resource sharing – support for the well-known priority inversion minimization algorithms;
- time values and clocks – high resolution time values and a real-time clock;
- memory management – a complementary model based on the concept of memory regions (Tofte and Talpin, 1997; Tofte et al., 2004) (called scoped memory areas) allow the reclamation of objects without the vagaries of garbage collection;
- asynchronous transfer of control – extensions to the Java thread interrupt mechanism allowing the controlled delivery and handling of asynchronous exceptions (see (Brosgol et al., 2002; Wellings, 2004));
- physical and raw memory access – allowing more control over allocation of objects in different types of memory and interfacing to device drivers (see (Wellings, 2004)).

The intention is that RTSJ should be capable of supporting both hard and soft real-time components. The hard real-time components should never access the Java heap but use the alternative memory management support. This way, they will never be delayed by garbage collection activities.

### 3 Conclusions and Future

The RTSJ is an evolving specification. Within the Java Community Process, JSR 282 (<http://jcp.org/en/jsr/detail?id=282>) is extending the semantics to address, amongst other things, multiprocessor issues. However, the current advances in the distributed version of RTSJ (DRTSJ - JSR 50) specification (<http://jcp.org/en/jsr/detail?id=50>) are far behind those of the centralized version mainly due to the small number of groups that are currently actively and constantly working in it within the main group-core. However, we ought not to forget that, even from its origin, Java language is very much network oriented. Therefore, its distributed real-time version is a very important one and will help Java penetrate, to start with, in the large market of the soft real-time systems. Moreover, some work on the distributed version even feeds back the centralized version as proven by the work of (Bas08ThreadModel, 2008), (Bas07SyncJava, 2007), (Bas06RTSJRules, 2006), (Bas05ScopedMem, 2005), citeDreq04.

Profiles for mission and safety critical applications are being developed under JSR 302 (<http://jcp.org/en/jsr/detail?id=302>). A new JSR will start soon on a soft real-time profile that focuses on support real-time garbage collection technology (Bacon et al., 2003; Chang and Wellings, 2005; Henriksson, 1998; Kim et al., 1999; Siebert, 1999).

## 4 Links

The definition of the current version along with the reference implementation and other resources can be found at <http://www.rtsj.org/>

## References

- G. Back, P. Tullmann, L. Stoller, W.C. Hsieh, and J. Lepreau. Java operating systems: Design and implementation, uucs-98-015. Technical report, Department of Computer Science, University of Utah, 1998.
- D.F Bacon, P. Cheng, and V.T. Rajan. A real-time garbage collector with low overhead and consistent utilization. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 285–298, 2003.
- P. Basanta-Val, M. Garcia-Valls, I. Estevez-Ayres. Simplifying the Dualized Threading Model of RTSJ In *11th IEEE International Symposium on Object-Oriented Real-Time Computing (ISORC 2008)*, pages 265 – 272, 2008.
- P. Basanta-Val, L. Almeida, M. Garcia-Valls, and I. Estevez-Ayres. Towards a Synchronous Scheduling Service on Top of a Unicast Distributed Real-Time Java In *13th IEEE Real Time and Embedded Technology and Applications Symposium* , pages 123 – 132, 2007. isbn issn 1080-1812 , 0-7695-2800-7
- P. Basanta-Val, M. Garcia-Valls, I. Estevez-Ayres, and Carlos Delgado-Kloos. Extended portal: violating the assignment rule and enforcing the single parent rule In *4th International Workshop on Java technologies for real-time and embedded systems* , pages 30 – 37, 2006. 0-7695-2356-0
- P. Basanta-Val, M. Garcia-Valls, I. Estevez-Ayres. Towards the Integration of Scoped Memory in Distributed Real-Time Java In *8th IEEE International Symposium on Object-Oriented Real-Time Computing (ISORC 2005)*, pages 382 – 389, 2005.
- P. Basanta-Val, M. Garcia-Valls, and I. Estevez-Ayres. No Heap Remote Objects: Leaving Out Garbage Collection at the Server Side In *2th International Workshop on Java technologies for real-time and embedded systems* , pages 359 – 370, 2004.
- G. Bollella, B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin, and M. Turnbull. *The Real-Time Specification for Java*. Addison-Wesley, 2000.
- B. M. Brosgol, R.J. Hassan, and S. Robbins. Asynchronous transfer of control in the real-time specification for java. In *Proceedings of Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 101–108, 2002.
- L. Carnahan and M. Ruark. Requirements for real-time extensions for the Java platform. NIST, <http://www.nist.gov/rt-java>, US National Institute of Standards and Technology, 1999.
- Y. Chang and A. J. Wellings. Integrating hybrid garbage collection with dual priority scheduling. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 185–188. IEEE, IEEE Computer Society Press, aug 2005.
- P. Dibble, R. Belliardi, B. Brosgol, D. Holmes, and A.J. Wellings. *The Real-Time Specification for Java: Second Edition*. 2006.
- P.C. Dibble. *Real-time Java platform programming*. Sun Microsystems Press, 2002.
- J. Gosling, B. Joy, G. Steele, and B. Bracha. *The Java Language Specification: 2nd Edition*. Addison-Wesley, 2000.
- R. Henriksson. *Scheduling Garbage Collection in Embedded Systems*. PhD thesis, Department of Computer Science, Lund University, Sweden, 1998.

- G. Hilderink. A new Java thread model for concurrent programming of real-time systems. *Real-Time Magazine*, 1, 1998.
- J Consortium. Real-time core extensions for the Java platform. Revision 1.0.10, J Consortium, 2000.
- T. Kim, N. Chang, N. Kim, and H. Shin. Scheduling garbage collector for embedded real-time systems. In *Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'99), ACM SIGPLAN Notices*, pages 55–64, 1999.
- H. McGhan and M. O'Connor. picoJava: a direct execution engine for Java bytecode. *IEEE Computer*, 1998.
- A. Miyoshi, H. Tokuda, and Kitayama T. Implementation and evaluation of real-time Java threads. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–175, 1997.
- K. Nilsen. Adding real-time capabilities to the Java programming language. *Communications of the ACM*, pages 44–57, 1998.
- F. Pizlo, J. Fox, D. Holmes, and J. Vitek. Real-time Java scoped memory: Design patterns and semantics. In *Proceedings of the Seventh IEEE Symposium on Object-Oriented Real-Time Distributed Computing, ISORC*, pages 101–110, 2004.
- F. Siebert. Real-time garbage collection in multi-threaded systems on a single microprocessor. In *IEEE Real-Time Systems Symposium*, pages 277–278, 1999.
- M. Tofte and J-P. Talpin. Region-Based Memory Management. *Information and Computation*, 132(2): 109–176, 1997.
- M. Tofte, L. Birkedal, M. Elsmann, and N. Hallenberg. A Retrospective on Region-Based Memory Management. *Higher-Order and Symbolic Computation*, 17(3):245–265, 2004. ISSN 1388-3690.
- A. J. Wellings. *Concurrent and Real-Time Programming in Java*. Wiley, 2004.