

# **Requirements For Real-time Extensions For the Java™ Platform**

September 1999

U.S. DEPARTMENT OF COMMERCE  
Technology Administration  
National Institute of Standards and Technology

(Excerpted by E. Douglas Jensen)

(24 January 2002)

## “Java” is not a noun

- In this document, “RTJ” is used to identify the sum of the functionality and services that would be provided through real-time extensions for the Java platform

# Guiding Principles

- The design of RTJ may involve compromises that improve ease of use at the cost of less than optimal efficiency or performance
- RTJ should support the creation of software with useful lifetimes that span multiple decades, maybe even centuries
- RTJ requirements are intended both to be pragmatic, by taking into account current real-time practice, and visionary, by providing a roadmap and direction to advance the state of the art

# Core Functionality and Profiles

- Because of the wide variety of applications that need to be served by these requirements, they should be partitioned into a small set of profiles built on top of a common RTJ core
- This partitioning should result in the smallest set of profiles possible to meet the varying needs of users
- Specifications should provide a framework for these profiles
- All implementations must implement the core except as defined by profiles that restrict the core

# Example Profiles

- Safety critical
- High availability and fault tolerant
- Low latency
- Deadline-based scheduling, or priority, or round-robin, or none
- No dynamic loading
- Bare bones
- Distributed real-time

# Core Requirement 1

- The specification must include a framework for the lookup and discovery of available profiles
  - e.g., existence of a profile; enumeration of profiles; availability of profiles for loading; and (potentially) versioning

## Core Requirement 2

- Any garbage collection that is provided shall have a bounded preemption latency
- The specification must clearly define the restrictions that are imposed by preemption
  - e.g., can preempting tasks allocate memory? Can they refer to existing objects?

## Derived Core Requirement 2.1

- For the functionality (language and libraries) that are expected to be used by real-time threads, a tight upper bound on the memory resources required by that functionality must be quantifiable prior to runtime
  - For example, how much memory is required to represent objects of a particular type? How much memory will be allocated by invoking a particular method?
  - See Goal 8 (real-time Java should allow resource reservations and should enforce resource budgets)

## Derived Core Requirement 2.2

- The specification must identify the set of functionality that may be used by real-time threads
- Possible restrictions on these threads are not necessarily the result of only memory allocation (e.g., pointer assignments that change the liveness of an object)
  - The behavior of real-time threads may need to be restricted so as not to interfere with the preempted system state
  - For example, timely preemption of a garbage collector may put the heap under its management in an inconsistent state, so it may not be possible for a preempting real-time thread to act upon that heap

## Derived Core Requirement 2.3

- RTJ must not require bounds on when or whether an object is finalized or reclaimed
  - Such bounds would conflict with the Java Language Specification

## Derived Core Requirement 2.4

- Within RTJ, the GC overhead, if any, on the application must be quantified
  - Garbage collection activity must be scheduled like any other activity in a real-time system
  - If the GC overhead is not quantifiable, a valid real-time schedule cannot be constructed

## Core Requirement 3

- The RTJ specification must define the relationships among real-time Java threads at the same level of detail as is currently available in existing standards documents
  - An example of the minimal requirements for specification would be POSIX 1003.1B
  - Examples of these relationships include thread scheduling, wait queue ordering, and priority inversion avoidance policies

## Core Requirement 4

- The RTJ specification must include APIs to allow communication and synchronization between Java and non-Java tasks

## Core Requirement 5

- The RTJ specification must include handling of both internal and external asynchronous events
- The model must support a mechanism for executing Java code in response to such events
- This mechanism should fit in well with existing Java mechanisms (such as wait/notify)
- (There was no agreement whether general asynchronous events that target individual threads are required)

# Core Requirement 6

- The RTJ specification must include some form of asynchronous thread termination that must have the following properties
  - By default a thread cannot be aborted
  - The target code determines where it can be aborted
  - When a thread is aborted all locks are released, and finally clauses execute on the way out when the thread is being aborted
  - No facility for aborting uncooperative code need be provided
  - The termination shall be deferred if the code currently executing has not indicated a willingness to be terminated
  - Mechanisms must be provided that allow the programmer to insure data integrity

# Core Requirement 7

- The RTJ core must provide mechanisms for enforcing mutual exclusion without blocking
- This requirement does not imply that a real-time thread should be allowed to disable/enable interrupts to achieve these results
- The specification must require that the mechanisms do not allow a non-real-time thread to gain complete control of the machine
- Specifically, the scheduler will continue to dispatch threads and interrupt handlers, other than the one possibly attached to the thread using the non-blocking mutex, which will continue to execute
- The specification should take care to minimize the risks to the system integrity, possibly by integrating with Java's existing security manager
- The following requirement was removed by consensus: The RTJ specification must provide mechanisms for ensuring data integrity in the presence of asynchronous event handling which may result in changes in the control sequence

## Core Requirement 8

- The RTJ specification must provide a mechanism to allow code to query whether it is running under a real-time Java thread or a non-real-time Java thread

## Core Requirement 9

- The RTJ specification must define the relationships that exist between real-time Java and non-real-time Java threads
- It must provide mechanisms for communication and sharing of information between real-time Java and non-real-time Java threads

## Derived Core Requirement 9.1

- Traditional Java software must run as non-real-time tasks

## Derived Core Requirement 9.2

- The sharing and communications protocols must have known tight upper bounds or some other form of predictability on blocking delays

## Derived Core Requirement 9.3

- The relationships between RTJ threads and the other three possible types of threads (non-realtime Java, non-Java real-time, and non-Java non-real-time) need to be defined
- These relationships include
  - Priorities and other scheduling relationships
  - Sharing resources (memory, devices)
  - Other processes
  - Synchronization
  - Budgets (memory, CPU, other resources)
  - Protections and privileges

# Goals

- The Requirements Group defined thirteen goals that should be addressed, where appropriate, in the base real-time Java specification, future profiles, or a future base specification.
- Some of the goals are accompanied by derived requirements, many of which are intended for future use
- The Requirements Group purposely added requirements that would create discussion and generate future innovation

# Goal 1

- RTJ should allow any desired degree of real-time resource management for the purpose of the system operating in real-time to any desired degree (e.g., hard real-time, and soft real-time with any time constraints, collective timeliness optimization criteria, and optimality/predictability tradeoffs)

## Goal 2

- Support for RTJ specification should be possible on any implementation of the complete Java programming language

## Goal 2 Derived Requirements

- 1. RTJ programming techniques should scale to large or small-memory systems, to fast or slow computers, to single CPU architectures and to SMP machines
- 2. RTJ should support the creation of both small, simple systems and large, complex systems (possibly using different "profiles")
- 3. Standard subsets of RTJ and RTJVM specifications should be created as necessary to support improved efficiency and/or reliability for particular specialized domains

## Goal 3

- Subject to resource availability and performance characteristics, it should be possible to write RTJ programs and components that are fully portable regardless of the underlying platform

## Goal 3 Derived Requirements

- 1. Minimal human intervention should be required when the software is "ported" to new hardware platforms or combined with new software components
- 2. RTJ should abstract operating system and hardware dependencies
- 3. RTJ must support standard Java semantics
- 4. The RTJ technologies should maximize the use of non-RTJ technologies (e.g., development tools and libraries)
- 5. The RTJ API must be well-defined with guarantees on all language features

## Goal 4

- RTJ should support workloads comprised of the combination of real-time tasks and non-real-time tasks

## Goal 5

- RTJ should allow real-time application developers to separate concerns between negotiating components

## Goal 6

- RTJ should allow real-time application developers to automate resource requirements analysis either at runtime or off-line

## Goal 7

- RTJ should allow real-time application developers to write real-time constraints into their software

# Goal 7 Derived Requirements

- 1. RTJ should provide application developers with the option of using conservative or aggressive resource allocation [*Open — no consensus*]
- 2. The same RTJVM should support combined workloads in which some activities budget aggressively and other conservatively [*Open — no consensus*]
- 3. RTJ infrastructure should allow negotiating components to take responsibility for assessing and managing risks associated with resource budgeting and contention
- 4. RTJ should allow application developers to specify real-time requirements without understanding "global concerns" -- for example, a negotiating component should speak in terms of deadlines and periods rather than priorities

- 5. RTJ must provide a mechanism to discover the relationship between available priorities for Java threads and the set of all priorities available in the system -- in addition, a mechanism must be provided to allow the relationships between Java priorities and system priorities to be determined

## Goal 8

- RTJ should allow resource reservations and should enforce resource budgets
- The following resources should be budgeted: CPU time, memory, and memory allocation rate

# Goal 8 Derived Requirement 1

- At least support strict priority-based scheduling, queuing, and lock contention, applying to existing language features as well
- At least support some kind of priority "boosting" (either priority inheritance or priority ceilings, applying to existing language features as well
- Support dynamic priority changes
- Support the ability to propagate a local priority and changes to remote servers -- not just in support of RMI but also in support of user-written communication mechanisms
- Support the ability to defer asynchronous suspension or disruption when manipulating a data structure
- Support the ability to build deadline-based scheduler on top
- Support the ability to query to find out the underlying resource availability (non-Java) and handle asynchronous changes to it

## Goal 8 Derived Requirements 2-6

- 2. Language and libraries must be clearly understood in terms of memory usage
- 3. RTJ shall provide support for a guaranteed allocation rate
- 4. RTJ must not require bounds on when an object is finalized or reclaimed
- 5. RTJ should provide for specifying memory
- 6. The priority mechanism must take into consideration the existing security protocols related to setting priorities to high levels

## Goal 9

- RTJ should support the use of components as "black boxes" including such use on the same thread

## Goal 9 Derived Requirements

- 1. RTJ should support dynamic loading and integration of negotiating components
- 2. RTJ should support a mechanism for negotiating components whereby the behavior of critical sections of code is locally analyzable
- 3. RTJ should support the ability to enforce (with notification, event handling and accounting) space/time limits, in a scoped manner, from the outside (on "standard" Java features as well)
- 4. In a real-time context, existing Java features should "work right", including synchronized (bounded priority inversion) and wait/notify (priority queuing)

## Goal 10

- RTJ must provide real-time garbage collection when garbage collection is necessary
- GC implementation information must be visible to the RTJ application

## Goal 10 Derived Requirements

- 1. RTJ must define “garbage”
- 2. RTJ should provide “hint handling” information regarding the GC (e.g., accurate vs. conservative? Does it defragment?). [*Open — no consensus*]
- 3. RTJ must not restrict nor specify the garbage collection technique; rather, it should be capable of supporting all appropriate techniques for real-time GC
- 4. The GC must make forward progress at some rate, which must be “queryable” and configurable
- 5. Within RTJ, the GC overhead, if any, on the application must be quantified

# Goal 11

- RTJ should support straightforward and reliable integration of independently developed software components (including changing hardware)

## Goal 12

- RTJ should be specified in sufficient detail to support (and with particular consideration for) targeting by other languages, such as Ada

## Goal 13

- RTJ should be implementable on operating systems that support real-time behavior