Real-Time Programming in Java

Real-Time Middleware Seminar 2007

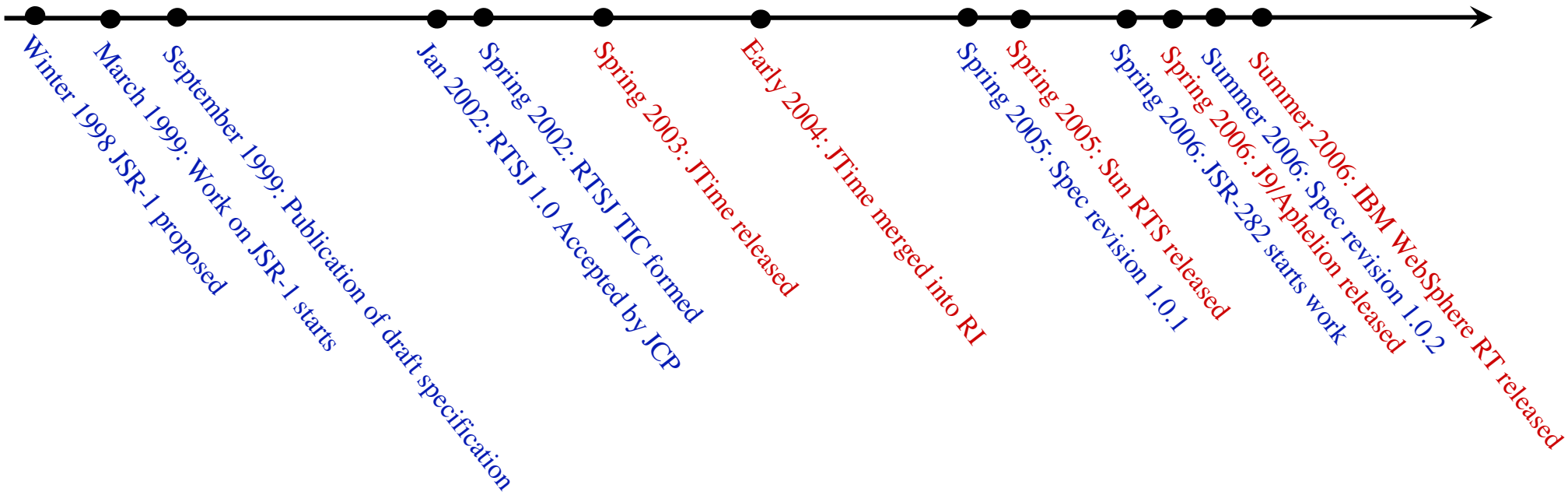
Konstantin Shagin

IBM Labs in Haifa

Motivation and Guiding Principles

- Why Java? Popular, safe, easy-to-use, portable
- Main concerns: garbage collection, no language support for scheduling and asynchrony
- ***Real-Time Specification for Java (RTSJ)***: a set of interfaces and behavioral specs
  - Designed to support both hard and soft real-time applications
  - Guiding principles:
    1. Applicability to any Java environment, not only for Micro Edition or Embedded JAE
    2. Backward compatibility
    3. Recognize the importance of “write once, run anywhere” and the difficulty of achieving it
    4. Address current real-time practices but allow addition of advanced features
    5. Predictability is first priority in all trade-offs
    6. No syntactic extensions
    7. Allow variation in implementation decisions

RTSJ Timeline



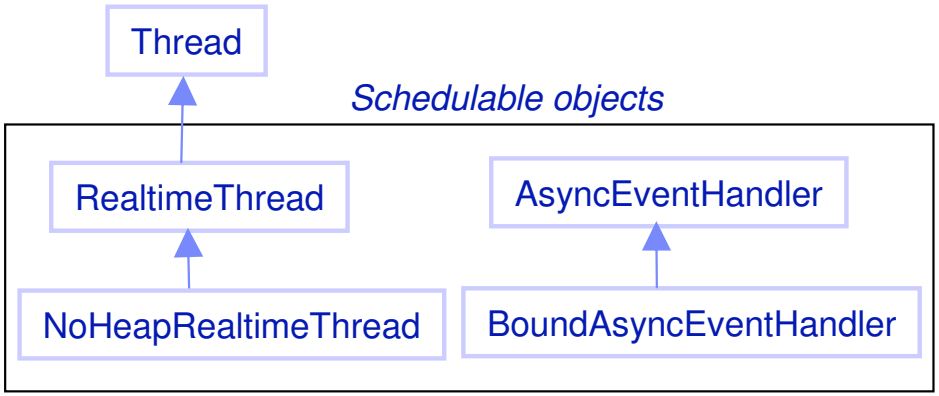
***Latest news**: 3 April 2007, Alpha 2 RI for RTSJ 1.1 is available on Linux 2.6.18*

Enhanced Areas

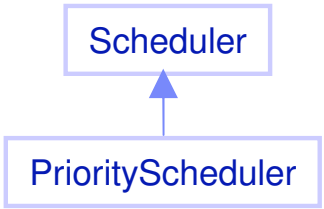
1. Thread scheduling and dispatching
   - Allow arbitrary scheduling policy
   - Support feasibility analysis
   - Require a fixed-priority preemptive scheduler in all implementations
2. Memory management
   - Independent of any particular GC algorithm
   - Allow allocation and reclamation that do not interfere with GC
3. Synchronization and resource sharing
   - Require implementations to include priority inversion mechanisms
4. Asynchronous event handling
   - Allow to define events and logic that triggered by an event
   - Execution of event logic is controlled by an implemented scheduler
5. Asynchronous transfer of control
6. Asynchronous thread termination
7. Physical memory access
   - Allow byte-level access to physical memory
   - Allow construction of objects in physical memory

Scheduler and Schedulable Objects

- A *schedulable* is an object that can be managed by a scheduler. It has:
  - Scheduling parameters (e.g., priority)
  - Release parameters
    - Periodic: start, period, cost, deadline
    - Sporadic: min. interval, cost, deadline
    - Aperiodic: cost, deadline
  - Memory parameters



- The scheduler
  - Manages schedulable objects
  - Implements a feasibility algorithm
    - Can check whether scheduling of tasks is feasible at runtime
    - Default feasibility analysis is trivial: assumes unlimited resources
  - The required default scheduler is preemptive, fixed priority-based
    - Critical for portability
  - Implementation are expected to define other schedulers, *e.g,* EDF

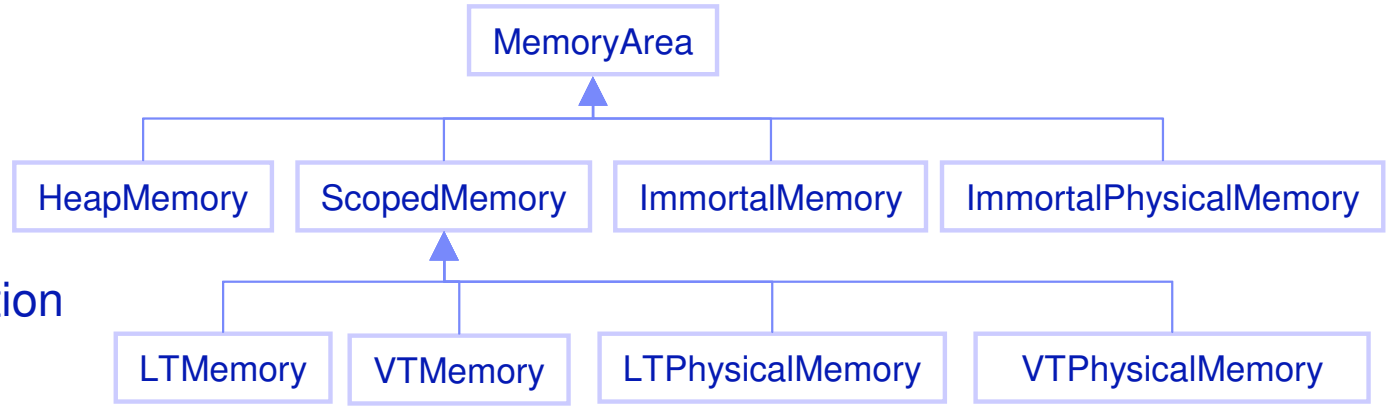


The Priority Scheduler

- Preemptive
- Gives CPU to the runnable schedulable object with the highest priority
- Supports a real-time priority range of at least 28 unique priorities (the larger the value, the higher the priority)
- Priorities can be changed at run time
- Supports priority inheritance or priority ceiling emulation inheritance for synchronized objects (covered later)
- The placement of a pre-empted schedulable is not specified but must me documented
- A blocked schedulable object which becomes runnable, or has its priority changed is placed at the back of the run queue associated with its new priority
- A thread which performs a yield operation is placed at the back of the run queue associated with its (new) priority

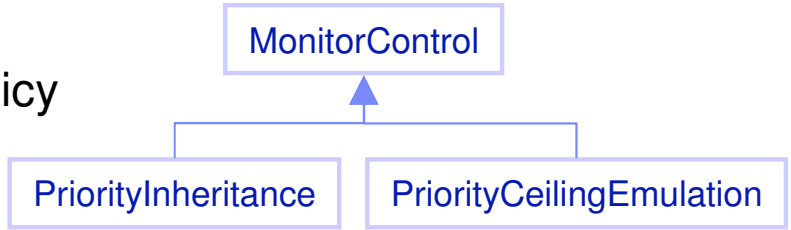
Memory Management

- The goal: avoid interference with the garbage collector
- Memory areas
  - Heap – same as standard heap; lifetime is determined by visibility
  - Immortal - area that is not subject to GC
  - Scoped – object lifetime is limited and depends on the allocation scope
    - Schedulable objects may enter and leave a scoped memory area
    - Whilst they are executing within that area, all memory allocations are performed from the scoped memory
    - When there are no schedulable objects active inside a scoped memory area, the allocated memory is reclaimed
    - Reference assignment rules prevent dangling references to scoped memory
  - Physical – objects are created within physical memory regions
- Budgeted allocation
  - Bounded memory area consumption
  - Allocation rate limit
- Support for limited interaction with the garbage collector



Synchronization

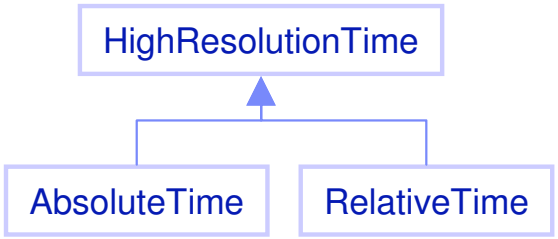
- Threads are given access to monitors according to their execution eligibility order
- Priority Inversion Avoidance
  - Implementation of the synchronized primitive must prevent unbounded priority inversion
  - Specifies two mechanisms:
    - Priority inheritance (required)
    - Priority ceiling emulation (optional)
  - Programmer can override the monitor control policy system-wide or for a particular monitor
  - Each schedulable has object two priorities:
    - Base priority – creation time priority
    - Active priority – derived from the current synchronization operations
  - Future implementations can define new monitor control policies



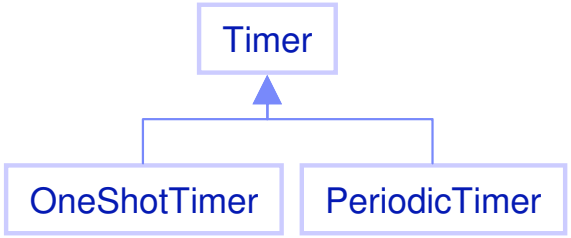
- Wait-free queues – allow shared access by both regular and RT no-heap threads
  - Wait-free read queue, wait-free write queue
  - Prevent dependency of a no-heap thread on GC by synchronizing with heap-using thread

Time, Clocks and Timers

- High resolution time is defined: 64-bit millisecond and 32-bit nanoseconds
  - Absolute time – represents a specific point in time
  - Relative time – represents an interval



- Time is always associated with a *clock* object
- Implementation must provide at least one clock: the system *real-time clock*
- Timer objects (use Asynchronous Event Handlers)
  - One shot timer – an event that fires only once at a specific time
  - Periodic timer – an event that is fired repeatedly at regular intervals



Asynchrony

- Asynchronous Event Handling
  - Specification introduces asynchronous events and event handlers
  - Asynchronous event handler (AEH) is a schedulable object: scheduled and dispatched similarly to a thread
  - Multiple handlers may be attached on an event object
  - An event can be triggered by
    - Application logic
    - Internal RTSJ mechanisms, *e.g.,* miss handler or overrun handler
    - External happening, *e.g., software signal or hardware interrupt*
  - Can bind an AEH to a dedicated real-time thread

Asynchrony (cont.)

- Asynchronous Control Transfer
  - Based on throwing and propagating exceptions
  - An exception is deferred when in synchronized statement or method
  - Interruptible methods are explicitly declared
  - The interrupt is generated when
    - The *interrupt* method of a real-time thread is called (system-wide interrupt)
    - The fire method of a asynchronous interrupted exception is called (specific interrupt)
    - A timer expires (specific interrupt)
  - Application can define a handler that is invoked when a specific interrupt occurs
- Asynchronous Real-Time Thread Termination
  - Allowed through combination of asynchronous event handling and asynchronous control transfer mechanisms

Summary

- The RTSJ originates from desire to write RT applications in Java
- Main problems that include unsuitable memory management (due to GC), insufficient control of thread scheduling and lack of support for asynchrony are addressed
- Usage can be complex (example: scoped memory)
- Still no support for multiprocessor systems

Questions?