Programming with Non-Heap Memory in the Real Time Specification for Java

Daniel Dvorak, Ph.D.
Information Technologies and Software Systems Division
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California
Daniel.Dvorak@jpl.nasa.gov

Outline

- Background on real-time applications
- Real-Time Specification for Java (RTSJ)
- Effect of RTSJ on normal Java style
- Scoped Memory Scratchpads
- Ending thoughts
Most real-time systems are a mixture of hard, soft, and non-real-time

- Non real-time: No temporal predictability required
- Soft real-time: Medium temporal predictability required
- Hard real-time: High temporal predictability required

A single RTSJ-compliant VM supports all three kinds of activities

Control of a Mars rover involves:
- coordinating 6 driving motors and 2 steering motors
- reading from a 3-axis accelerometer and a 1-axis gyroscope
- taking pictures with 3 stereo cameras
- controlling a camera mast
- controlling the instrument arm

Rocky 7 rover in the JPL Mars Yard with camera mast raised
Java for Real-Time ???

Java Technology was not intended for real-time systems because:

- Requires run-time garbage collection
- Supports threads, but no scheduling control
- Synchronization delays unpredictable
- Very coarse timer support
- No event processing
- No safe asynchronous transfer of control

RTSJ Guiding Principles (partial)

- Predictable Execution
  - "hold predictable execution as first priority in all tradeoffs"
- Backward Compatibility
  - Existing Java programs run on RTSJ implementations
- No Syntactic Extensions
  - No new keywords or other extensions to Java language
- Leading-Edge Scheduling
  - Priorities considered harmful
Definitions

Real-time ≠ “real fast”

- **Real-time** (system or code): A system (or code) which requires that computation have temporal correctness criteria in addition to functional correctness criteria
- **Hard real-time**: A real-time system which requires that the temporal correctness criteria are *always* met (often, incorrectly defined as ‘less than n time-units latency’)
- **Soft real-time**: A real-time system which requires that the temporal correctness criteria are *almost always* met (often, incorrectly defined as ‘more than n time-units latency’)

RTSJ Features

- Full Java capability
- Threads (RealtimeThread, NoHeapRealtimeThread)
- Asynchronous Event handling
- High resolution time
- Precise Timer support
- Asynchronous Transfer of Control
- Flexible memory management
  - *Makes using Java heap optional*
  - *Can avoid interference from garbage collector*
- Raw memory access (e.g., memory-mapped I/O)
**Memory Areas**

- Normal Java heap
- Accessible by all threads
- Subject to garbage collecting
- Prone to execution latencies due to GC

- Fixed size memory area that lives until end of app
- Accessible by all threads
- Immortal objects continue to exist even when there are no references
- Never subject to garbage collecting

**Threads**

- Normal Java thread
- No scheduling control
- Cannot be asynchronously interrupted

- Temporal demands specified (deadline, period)
- Processor demands specified (CPU & memory costs)
- Can run as a periodic, aperiodic, or sporadic
- Can allow for asynchronous interrupts

- Cannot allocate or reference objects in heap memory
- Can *always* execute in preference to GC
Java Programming Style

- Use 'new' to allocate objects (in heap)
- Freely pass references around
- Let GC reclaim unused objects
  - Don't worry about memory management

RTSJ Changes to Style

- Do worry about memory management
  - NHRT cannot allocate or access heap objects
  - java.lang.Thread cannot allocate Scoped Memory objects
  - Must obey memory area assignment rules
Memory Area Assignment Rules

Heap Memory

Scoped Memory

Local Variable

inner scope

Legend: $X \rightarrow Y$ means that object in $X$ can NOT be assigned a reference to an object in $Y$ because $Y$ can disappear before $X$

Real-Time Control Loop

State Estimator

Controller

Sensor Hardware Adapter

Actuator Hardware Adapter

Measurements
cyclic execution

commands

Software Hardware

state estimates

froms hardware
to hardware

Oct. 29, 2003

Dwork
Each component must:
- manage its mutable state in a persistent manner
- manage transient objects allocated during its execution

- How do you manage memory in a way that ...
  - supports hard real-time?
  - is natural for Java?
  - minimizes need for programmer discipline?
  - is verifiable w.r.t. memory errors?

---

Scoped Memory Scratchpad

- Component enters a scoped memory area
- Subsequent 'new's allocate from scoped, whether in own code or inside called methods
- Transient objects (for current cycle only) will disappear upon scoped memory exit
- Persistent data (across cycles) must be kept in a private memory pool (requires copying)
- Required programmer discipline:
  - Called components must not hold references to arguments
  - Calling components must not hold references to returned objects
Comparison

- Scoped Memory Scratchpad:
  - Supports natural *new*-based creation of objects
  - Runs hard real-time components in Scoped Memory
  - Scope emptying automatically discards transient data
  - Works with 3rd-party libraries that do *new*
  - Framework software can shield application developers from type of thread and type of memory area

- Shared Memory Pool (an alternative):
  - Get unused objects from a pool in Immortal Memory, release when no longer used (manual memory management)
  - Pass references from one component to another
  - Avoids copying of data (more efficient)
  - Vulnerable to memory leaks and double-usage

Ending Thoughts

- RTSJ supports hard, soft, and non real-time applications on a single VM having *any GC* ...
- ... but users encounter a conflict between Java's OO style and RTSJ's rules
- The design trade-space includes efficiency, programmer discipline, verifiability, and style
- The approach of "scoped memory scratchpads" favors verifiability and style
- Real-time garbage collectors can alleviate the conflict for apps where the latency is acceptable
  - But GC avoidance enables *very* predictable execution
  - And that's important in control systems
Golden Gate Team

- Sun Microsystems
  - Greg Bollella and James Gosling

- Carnegie Mellon University
  - Virgil Champlin and Brian Giovannoni

- Jet Propulsion Laboratory
  - Tim Canham, Vanessa Carson, Dan Dvorak, Mark Indictor, Kenny Meyer, Alex Murray, Kirk Reinholtz