Real-Time Extensions for the Java^{TM} Platform: A Progress Report

Bill Foote

Sun Microsystems

EmbeddedJava^{TM} Group

What is the goal?

- Extensions for the PersonalJava™ and Enterprise Java™ platforms.
- Applies to EmbeddedJava™.
- Enable the construction of soft real-time systems, and certain hard real-time systems.
- Provide RTOS level of abstraction

Many Groups Involved

- NIST-sponsored requirements working group
- RTJWG, backed by HP, Microsoft, NewMonics
- NCITS R1 group. Part of ANSI. Recently voted *against* adopting RTJWG.
- Expert Group under Sun's Community Process

Sun’s Community Process

- Small “core” group of Experts to develop standard in “internet time.”
- Reference implementation required.
- Compatibility test suites are required.
- Specification Lead is free to structure interaction to suit the problem.
- For real-time, we envision interactive participation by wider group of JSPA signatories.

Sun’s Community Process

- Audited public review of specification drafts at specific milestones.
- Hope to complete real-time specification in CY ‘99.
- “Neither open source nor proprietary”

Benefits of Java^{TM}

- Security
- Compatibility (WORA)
- Attractive, intuitive programming model:
  - OO
  - Strongly Typed
  - Garbage Collected
  - Checked Exceptions
  - Packages

Benefits of Java^{TM}

- Dynamic Linking (downloadable code)
- Safety
- Scalability (JavaCard^{TM} to the enterprise)
- Tool Support
- Libraries
- "Culturally" compatible with C/C++
- Large number of developers

What is EmbeddedJava™?

- It is a technology, not a platform.
- Makes most of the platform optional at *deployment* time
- Entire platform must be available at *application binding* time.
- Sun’s product targets small footprint devices (>= 512K RAM, 512K ROM)
- Sun’s product enables storing code and data in ROM

Application Binding Time

"The time at which a decision as to what features are present in the runtime environment are made."

- **Used to discard fields, methods, and classes.**
- **Can happen before application code exists.**
- **Dynamic classloading still possible.**

Elements of the Real-Time Extension

Sun's Opinion

Deadline-based Scheduling

*The **automatic** application of deadlines, “importance,” and other (often application-specific) criteria in an automated fashion by the execution environment.*

- Might enable “real-time components”
- Not widely used commercially
- Conflicts with standard Java threading model
- Good area for research
- Bad candidate for a standard

Negotiating Components

- Inspired by PERC™
- Probably require deadline-based scheduling
- Resource negotiation is difficult, poorly understood
- Good area for research
- Bad candidate for a standard

Stack Allocation

- Avoids non-determinism typically associated with heap allocation
- Fast
- Corrupts fundamental property of Java platform: Safe pointers
- A great idea for a real-time language… Just not one called “Java”

Hardware I/O Standard

- Real-time systems often control hardware
- Hardware-specific code is inherently non-portable
- Hardware evolves quickly
- Adequate standard would be complex
- Good candidate for an orthogonal standard (or standards!)
- Bad candidate for a real-time standard

Asynchronous Termination

- Thread.stop(Throwable) was deprecated.
- Most code does not expect to be asynchronously terminated
- Default must be no termination
- Might require language changes
- Area is still being studied

Wall Clock

- Time is a long
- Nanoseconds since 1/1/1970 leads to rollover in 2272
- Is microsecond precision adequate?
- Ten nanosecond precision?
- What minimum resolution should spec require?

Real-time Threads

- Priority-based
- Fixed-priority (but dynamically settable via an explicit method call)
- At least 30 priority levels required
- More levels helps RMA
- Legislate a large number (128, 256), or allow maximum to be implementation-defined?

Locking and Synchronization

- "Fix" the semantics of synchronized
  - Bounded time overhead
  - Priority inversion avoidance protocol
    - Priority Inheritance
    - Priority Ceiling
- Explicit lock classes
  - Mutex
  - Binary Semaphore
  - Counting Semaphore

External Events

- Much like signal mechanism
- Notification arrives on *system* thread
- Event generator has a priority that maps to RealtimeThread.priority
- Callback framework
- No detailed support for particular hardware
- Timer, file descriptor, and socket built-in

Garbage Collection

- Disruptive
  - Mark Sweep
  - Generational
- Non-disruptive
  - Incremental
    - Imposes heavy overhead on mutator
  - Hardware Assisted
    - Expensive!

Garbage Collection

- GC is fundamental to the Java paradigm
- Most real-time systems that Java targets aren’t executing real-time code 100% of the time
- Non-disruptive behavior is only required for real-time code
- Performance of real-time part of system is the most critical

Summary

- The marriage of real-time and Java will be an important technology.
- Join us!
  - Sign a JSPA, and participate in the community process
  - Read and comment on the public drafts
  - Send me your thoughts!

http://java.sun.com/people/billf/real-time