Real-Time Java

Martin Schöberl
Overview

- What are real-time systems
- Real-time specification for Java
- RTSJ issues, subset
- Real-time profile
- Open question - GC
History of (Real-Time) Java

- 1996 Nilsen: First Paper on Real-Time Java
- 1997 picoJava, PersonalJava
- 1992 Oak for *7
- 1995 Java for the Internet
- 1998 EmbeddedJava, RTSJ Start
- 2000 J2EE: Server Applications
- 2002 RTSJ Final Release
- 2000 J2ME: Java for Mobile Phones
- 2003 JTime
- Java for Desktop Applications
- Embedded Systems?
Real-Time Systems

A definition by John A. Stankovic:

In real-time computing the correctness of the system depends not only on the logical result of the computation but also on the time at which the result is produced.
Real-Time Threads

- In real-time systems there are:
  - Periodic threads
  - Event threads/handler
- No continuous running threads
- Fixed Priority driven scheduler
- Threads can starve!
Priority

- Kind of *importance*
- Scheduling theory:
  - Shorter periods – higher priorities
  - RMA: Rate Monotonic Analysis
  - Assignment is optimal
- In (hard) RT forbidden
  - sleep()
  - wait(), notify()
Real-Time Specification for Java

- RTSJ for short
- First JSR request
- Still not completely finished
- Implementations
  - Timesys RI
  - Purdue OVM
RTSJ Guiding Principles

- Backward compatibility to standard Java
- Write Once, Run Anywhere
- Current real-time practice
- Predictable execution
- No Syntactic extension
- Allow implementation flexibility
RTSJ Overview

- Clear definition of scheduler
- Priority inheritance protocol
- NoHeapRealtimeThread
- Scoped memory to avoid GC
- Low-level access through raw memory
- High resolution time and timer
RTSJ: Scheduling

- Standard Java offers no guarantee
  - Even non preemptive JVM possible
- Fixed priority
- FIFO within priorities
- Minimum of 28 unique priority levels
- GC priority level not defined
RTSJ: Memory Areas

- GC collected Heap
- Immortal memory
- Scoped memory
  - LTMemory
  - VTMemory
- Physical memory
  - Different time properties
  - Access to HW devices!
RTSJ: Thread Types

- Extensions to `java.lang.Thread`
  - `RealTimeThread`
  - `NoHeapRealTimeThread`
  - `AsyncEventHandler`

- Scoped and immortal memory for NHRTT
  - Strict assignment rules
  - Not easy to use
RTSJ : Synchronization

- Use synchronized
- Priority inversion possible in standard Java
- Priority inheritance protocol
- Priority ceiling emulation protocol
RTSJ: Scoped Memory

- Cumbersome programming style
- New class for each code part

```java
class UseMem implements Runnable {
    public void run() {
        // inside scoped memory
        Integer[] = new Integer[100];
        ...
    }
}

// outside of scoped memory
LTMemory mem = new LTMemory(1024, 1024);
UseMem um = new UseMem();

// usage
computation() {
    mem enter(um);
}
```
Asynchronous Event Handler

- Difference between bound and unbound
  - Implementation *hint* at application level
  - No functional difference for the application

- Better: only one type
  - Specify a minimum latency at creation
  - Runtime system decides about implementation
RTSJ Issues

- J2SE library:
  - Heap usage not documented
  - OS functions can cause blocking

- On small systems:
  - Large and complex specification
  - Expensive longs (64 bit) for time values
RTSJ Subset

- Ravenscar Java
  - Name from Ravenscar Ada
  - Based in Puschner & Wellings paper
- Profile for high integrity applications
- RTSJ compatible
- No dynamic thread creation
- Only NHRTT
- Simplified scoped memory
- Implementation?
Real-Time Profile

- Hard real-time profile
  - See Puschner paper
- Easy to implement
- Low runtime overhead
- No RTSJ compatibility
Real-Time Profile

- Schedulable objects:
  - Periodic activities
  - Asynchronous sporadic activities
    - Hardware interrupt or software event
    - Bound to a thread

- Application:
  - Initialization
  - Mission
Application Structure

- Initialization phase
  - Fixed number of threads
  - Thread creation
  - Shared objects in immortal memory

- Mission
  - Runs forever
  - Communication via shared objects
  - Scoped memory for temporary data
Schedulable Objects

- Three types:
  - RtThread, HwEvent and SwEvent
- Fixed priority
- Period or minimum interarrival time
- Scoped memory per thread
- Dispatched after mission start

```java
public class RtThread {
    public RtThread(int priority, int period) {
        // Constructor...
    }
    public RtThread(int priority, int period, int offset, Memory mem) {
        // Constructor...
    }
    public void enterMemory() {
        // Method...
    }
    public void exitMemory() {
        // Method...
    }
    public void run() {
        // Method...
    }
    public boolean waitForNextPeriod() {
        // Method...
    }
    public static void startMission() {
        // Method...
    }
}

public class HwEvent extends RtThread {
    public HwEvent(int priority, int minTime, Memory mem, int number) {
        // Constructor...
    }
    public void handle() {
        // Method...
    }
}

public class SwEvent extends RtThread {
    public SwEvent(int priority, int minTime, Memory mem) {
        // Constructor...
    }
    public final void fire() {
        // Method...
    }
    public void handle() {
        // Method...
    }
```
Scheduling

- Fixed priority with strict monotonic order
- Priority ceiling emulation protocol
  - Top priority for unassigned objects
- Interrupts under scheduler control
  - Priority for device drivers
  - No additional blocking time
  - Integration in schedulability analysis
Memory

- No GC: Heap becomes immortal memory
- Scoped memory
  - Bound to one thread at creation
  - Constant allocation time
    - Cleared on creation and on exit
  - Simple enter/exit syntax
Restrictions of Java

- Only WCET analyzable language constructs
- No static class initializer
  - Use a static init() function
- No finalization
  - Objects in immortal memory live *forever*
  - Finalization complicates WCET analysis of exit from scoped memory
- No dynamic class loading
public class Worker extends RtThread {
    private SwEvent event;
    public Worker(int p, int t, SwEvent ev) {
        super(p, t,
            // create a scoped
            // memory area
            new Memory(10000)
        );
        event = ev;
        init();
    }
    private void init() {
        // all initialization stuff
        // has to be placed here
    }
    public void run() {
        for (;;) {
            work(); // do work
            event.fire(); // and fire
                // an event
            // some work in
            // scoped memory
            enterMemory();
            workWithMem();
            exitMemory();
            // wait for next period
            if (!waitForNextPeriod()) {
                missedDeadline();
            }
        }
    }
}
// create an Event
Handler h = new Handler(3, 1000);

// create two worker threads with
// priorities according to their periods
FastWorker fw = new FastWorker(2, 2000);
Worker w = new Worker(1, 10000, h);

// change to mission phase for all
// periodic threads and event handler
RtThread.startMission();

// do some non real-time work
// and invoke sleep() or yield()
for (;;) {
    watchdogBlink();
    Thread.sleep(500);
}
Garbage Collection?

- An essential part of Java
- Without GC it is a different computing model
- RTSJ does not believe in real-time GC
- Real-time collectors evolve
- Active research area
  - For You?
Summary

- Real-time Java is emerging
- RTSJ defined by Sun
- Subsets: RJ, J OP-RT
- Real-time GC missing