Real-time Programming in Java

12.11.2008
Czech Java User Group
Prague, Czech Republic

Aleš Plšek
ales.plsek@inria.fr
INRIA

- The French National Institute for Research in Computer Science and Control
- 8 research centers
- 1800 scientists, 1000 PhD students, 100 post-docs
- 150 joint research project-teams
- 186 million Euros budget, 20% from research contracts
- Industrial Relations
  - 790 active research contracts
  - 89 companies
INRIA Nord Europe, team ADAM

Adaptive Distributed Applications and Middleware
- Component Oriented Programming
  - Fractal Component Model
- Model-Driven Engineering
- Service Oriented Architectures
  - WebServices
- Ubiquitous computing
  - Mobile Computing
  - Context Oriented Programming

The team
- 3 Profs, 2 Asist.Prof
- 2 Post-Doc
- 7 PhD students
- 6 R&D engineers
Me…

Past (2001-2006)
• Master Studies, DSRG, Charles University in Prague
• Model Checking of Software Components

Present – Since 2006
• 3rd year PhD Student, INRIA ADAM
• Research Interests
  – Component-Oriented Programming
  – Real-time Java Programming
  – Model Checking
Outline

Realtime Programming
• What is real-time?
• RT OS
• Developing RT Applications

Realtime Programming in Java
• RTSJ - Realtime Java Specification
• Realtime Garbage Collection
• Future of RT Java
Outline

Realtime Programming

• What is real-time?
• RT OS
• Developing RT Applications

Realtime Programming in Java

• RTSJ - Realtime Java Specification
• Realtime Garbage Collection
• Future of RT Java
What is Real-time?

“Real-time is not real-fast!”

Real-time is any information processing system which has to respond to externally generated input stimuli within a finite and specified period.

- the correctness depends not only on the logical result but also the time it was delivered
- “Realtime is Determinism.”

Real-time is ubiquitous

- Tiny (sensors), Small (pacemakers), Medium (phone), Large (avionics)
- Single processor vs. Multi-core vs. Distributed
What is Real-time?

“Real-time is not real-fast!”

Realtime is any information processing system which has to respond to externally generated input stimuli within a finite and specified period.

- the correctness depends not only on the logical result but also the time it was delivered
- “Realtime is Determinism.”

Real-time is ubiquitous

- Tiny (sensors), Small (pacemakers), Medium (phone), Large (avionics)
- Single processor vs. Multi-core vs. Distributed
Why Real-Time?

Real-time Programming

• A little interest in Real-time from the mainstream software engineering community
  – Deadlines, interruption handling, too low-level…

Real-Time Systems Trends

• Large-scale, heterogeneous systems
• Dynamically highly adaptable systems
• Systems composed from hard-, soft-, and non-real-time units

Many software engineering techniques can be applied in real-time domain
  – Component oriented programming, Code generation, Model Driven Engineering, Formal Verification, etc.
Successful Stories

Shipboard computing
- US navy Zumwalt-class Destroyer
- 5mio lines of Java code
- Red Hat Linux, RT GC the key part

Avionics
- 787 Dreamliner saves 900kg weight
- A380 saves a half of the processing units

Financial Information Systems
- 100ms deadlines
Time-Scale Pyramid

- Business and commercial real-time systems: 100 milliseconds
- Distributed real-time systems: 10 milliseconds
- General real-time systems: milliseconds
- Tight real-time code: 100 microseconds
- Drivers: 10 microseconds
Hard-, Soft- and Non-Real-time

Non Real-time

Soft Real-time

Hard Real-time

“Hard real-time is hard but soft real-time is harder.”
RT System Categories

Hard vs Soft real-time

Embedded systems (cars, trains, sensors)

Event-triggered vs. Time-triggered

Safety-critical vs. Mission-critical Systems
Let’s get Real

Systems are designed for **throughput**, not for **determinism**.

Problems causing non-determinism
- Garbage collection
- Failure to respect application priorities
- Priority inversions
- Uncontrolled load

Methods
- Normal program optimization is not helpful.

What we need:
- RT OS
- Deterministic programming model
  - Assembler, Ada, C, … ?
Real-time OS

RT OS features
- Fully preemptible kernel
- Fixed priority pre-emptive scheduling
  - FIFO scheduling
  - Different priority levels (at least 38)
  - Priority Inheritance
- High-Resolution timers

Real-life Example
- RT Linux
- IBM Blades
  - hardware patch needed
Developing Real-time Systems

Schedulability Analysis
1. List all the tasks the system must perform:
   - When they will be released for execution
   - The deadline
   - The cost (CPU time)
   - The resources
     (including locks) that they need

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Period</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>100</td>
<td>?</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>300</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td>350</td>
<td>?</td>
</tr>
</tbody>
</table>
Rate Monotonic Analysis

Schedulability Analysis

2. Analyze the system to find the priority assignment (if any) that will let the system meet all the deadlines.

- Rate Monotonic Analysis

\[ U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(\sqrt{2} - 1) \]

\[
\lim_{n \to \infty} n(\sqrt{2} - 1) = \ln 2 \approx 0.693147 \ldots
\]

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution Time</th>
<th>Period</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>100</td>
<td>High</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
<td>300</td>
<td>Medium</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td>350</td>
<td>Low</td>
</tr>
</tbody>
</table>
Developing Real-time Systems II.

Programming Realtime Systems
- Worst Case Execution Analysis
- E.g. quicksort vs. mergesort

Deterministic Programming model
- Garbage collection
- Thread model for preemptive scheduling
- C, C++, AdA
RT Systems Intricacy

ABS-Systems

What really happened on Mars?
- Mars, Pathfinder, July 4, 1997
- Lost contact due to embedded software failure

Ariane 5
- French Guyana, June 4, 1996
- $800 million embedded software failure
Real-time Summary

Realtime
- Predictability
- Performance is not important
- Real-time is everywhere

RealTime OS
- Priority Fixed Preemptive Scheduling
- Priority Inversion

Developing Realtime Systems
- Schedulability Analysis
- Worst Case Execution (WCE) analysis
- Deterministic programming model
  - C, C++, Ada
Outline

Realtime Programming
• What is real-time?
• RT OS
• Developing RT Applications

Realtime Programming in Java
• RTSJ - Realtime Java Specification
• Realtime Garbage Collection
• Future of RT Java
Why Java?

Java

- Easy to use
- Familiar and widely used language.
- Libraries
- Well-defined semantics.
- Portable across platforms.
- Efficient development oriented.
Predictable?

Not really.

Hard real-time often requires microsecond accuracy.
Making Java Predictable…?

- Real-time Specification for Java (RTSJ, 2000)
- Real-time Garbage Collection (RTGC, 2006?)
Real-time Specification for Java

Java-like programming model
   • Compliant to Java SE

RTSJ features:
   • Fixed priority and round robin scheduling
   • Realtime Threads and inter-thread communication (e.g. semaphores)
   • User-defined interrupt handlers and device drivers — including the ability to manage interrupts (e.g., enabling and disabling)
   • Timeouts and aborts on running threads

However,
   • Feasibility analysis: The RTSJ does not mandate any particular schedulability analysis technique
Realtime Threads

Real-Time threads
• 28 Real-time priorities
• Can start at a specific or relative time / run on specified intervals or periods
• Runaway Threads
• Asynchronous event handlers

```java
class RTHelloWorld extends RealtimeThread {
    RTHelloWorld(PriorityParameters pp,
                  PeriodicParameters p)
        super(pp, p);
}

public void run(){
    boolean noProblems = true;
    while(noProblems) {
        // code to be run each period
        System.out.println("RT HelloWorld");
        noProblems = waitForNextPeriod();
    }
    ...
}
```
Memory Management

Memory Areas
- Heap memory
- Immortal Memory
- Scoped Memory

Immortal Memory
- Never released during the lifetime of the application

Scoped Memory
- a well-defined life time
- enter and leave a scoped memory area
- all memory allocations performed from the scoped memory
- the allocated memory is reclaimed when no thread in scope
Scoped Memory - Example

Application Logic Runnable

Cross-Scope Communication

```java
public class RTHelloWorld {
    LTMemory myMem = new LTMemory(1000, 5000);
    public int match(final int[][] with) {
        .......
        for (int i = 0; i < table.length; i++) {
            myMem.enter(new Runnable() {
                public void run() {
                    doStuff();
                }
            });
        }
        return result;
    }
    public void run() {
        doStuff();
    }
}
```
Non-Heap Real-time Thread (NHRT)

NoheapRealtime threads
- Can not be preempted by Garbage Collector
- No heap memory access
- Immortal or Scope Memory

```java
public class PeriodicThread extends NoHeapRealtimeThread {
    public PeriodicThread(...) {
        super(pp, p, ImmortalMemory.instance());
        applicationLogic = logic;
    }

    public void run() {
        boolean noProblems = true;
        while (noProblems) {
            applicationLogic.run();
            noProblems = waitForNextPeriod();
        }
        // Deadline missed. If allowed, a recovery routine here
    }

    public void start() {
        super.start();
    }
}
```
Benefits of Real-Time Java

Rule of thumb: 1% hard real-time, 9% soft real-time, 90% non-real-time

Standard Java Advantages

The main reason of RTSJ’s success is that it allows mixing freely real-time and timing-oblivious code in the same platform.
RTSJ Limitations

Complexities
- Error-prone process
- Non-intuitive rules and restrictions
- Introducing a new programming style
- Standard Java Libraries can not be used in RT

Software Engineering Aspect
- Ad-hoc approach
- No reuse, verification, formalization, etc.
- No adaptability, distribution support
RTSJ Implementations

SUN RTSJ
- Solaris 10 or Linux with RT-Preempt patch
- Java RTS

IBM
- IBM Blades
- Hardware supported
  - IBM BladeCenter
  - Red Hat Enterprise Linux, Version 4 (POSIX conformant)

TimeSys, Aonix, JamaicaVM, Ovm, …
- Usually not fully compliant to RTSJ
RT Garbage Collection

Don’t fear the garbage.

- Automatic defragmentation for long lived apps.

Avoid manual memory management unless absolutely necessary.
RT Garbage Collection

Traditional Garbage Collection

Real-time Garbage Collection

Time-based collectors can easily bound pauses to one millisecond.

Slack-based collectors (SUN’s) ensure RT code never preempted by GC.
RT Garbage Collection

Metronome
- Maximum pause times down to 250 microseconds

Blue Wonder
- 150 microseconds worst case latency
- 15 microseconds WCL for NonHeap
- Solaris OS
RTGC Limitations

• Ensuring that the GC can keep up requires some understanding of whole-program allocation behavior.
Future of Real-time Java

RTSJ
- JSR-282
  - Fixing the problems of RTSJ
- JSR-302
  - Safety-Critical Java
  - A strict subset of RTSJ
  - Certified for Safety-Critical systems

RT Garbage Collection
- The most practical solution
- However, mind the limitations
  - Overhead
  - Allocation behavior of applications
Conclusion

Real-time
- Determinism
- RT OS
- Programming Realtime
  - Scheduability Analysis
  - Worst Case Execution Analysis

Real-time Programming in Java
- RTSJ
  - Manual memory management
  - Real-time Threads
- RT Garbage Collection
  - Overhead, limitations
Further reading….


Acknowledgements

- Slides based on material by Jan Vitek and Andy Wellings

- Further information based on [1,2,3,4,5] and many others…

- Last but not least, the author acknowledges the following colleagues for their comments and consultations:
  - Frederic Loiret, Michal Malohlava, Lionel Seinturier, Philippe Merle, Carlos Noguera, Martin Decky and Marek Prochazka