Improving Abstraction, Encapsulation, and Performance within Mixed-Mode Real-Time Java Applications

Kelvin Nilsen

Presenter: Thomas Bøghholm

15. November 2007
# The Author

**Kelvin Nilsen**

- Ph.D., University of Arizona, 1988
- Founder of NewMonics
  - Bought by Aonix 2003
- Chief Technology Officer at Aonix
### The Article

#### Audience
- Workshop on Java Technologies For Real-Time and Embedded systems
  - Vienna, Australia, Sep. 2007

#### Topic
- Mixed-Mode Real-Time Systems in Java
- Reliability
The Article

Audience

- Workshop on Java Technologies For Real-Time and Embedded systems
  - Vienna, Australia, Sep. 2007

Topic

- Mixed-Mode Real-Time Systems in Java
- Reliability
Java for real-time systems

Benefits
- Portability
- Emulation of embedded systems
- Development speedup
Outline

1. RTSJ Overview
2. Soft Real-Time systems
3. Integration with Hard Real-Time Systems
4. Summary
Real-Time Specification for Java

RTSJ

- Real-Time Specification for Java
- Possible to mix Soft, Hard and Non Real-Time systems
- Too extensive and error prone
## Extensions in RTSJ

### Execution modes in RTSJ
- `java.lang.Thread`
- `javax.realtime.RealtimeThread`
- `javax.realtime.NoHeapRealtimeThread`

### Memory Areas
- Heap Memory
- Immortal Memory
- Scoped Memory
Using RTSJ

Challenges

- Execution mode specific operations
- Correct memory protocols
- Shared memory between different execution modes
Using RTSJ

Result

- Performance penalty
- Blocking of real-time threads by non-real-time threads
- Run-time checks before invoking certain operations
- Complexity
RTSJ Alternative

Solution
- Real-Time garbage Collector
- New Libraries compatible with memory protocols
- Use java.lang.Thread for soft real-time systems
- Annotations
Soft real-time

Soft real-time virtual machine

- PERC (Ultra) VM
- Java Standard Edition implementation
- API for querying the VM
  - CPU consumption, Single thread
  - CPU consumption, Priority level
  - Memory allocation rates
  - Amount of memory consumed
Garbage Collection

The PERC(Ultra) garbage collector

- Real-Time garbage collection
- Incremental
- Predictable
- Can be preempted
- Controlled by the *Pacing agent*
Example
Pacing agent

- Knows CPU time required for garbage collection
- Rate monotonic analysis
- Schedules tasks together and small increments of garbage collection
- Errors are reported and can be handled
- Monitors idle CPU cycles
Integration

Hard real-time

- No garbage collection
- Different virtual namespace for hard real-time components
- Separate run-time environments
  - Sharing data through proxies
- Traditional java threads can blocked by hard real-time
- Introduces *annotations* to clarify programmer intentions
Interfaces

Available interfaces

- `com.percpico.util.mc.TraditionalJava`
  - Sub-interfacing makes methods available to traditional java only

- `com.percpico.util.mc.TraditionalJavaShared`
  - Methods are available to both

- Violations are prohibited by byte-code verifier
Example

```java
public interface TraditionalVisible
extends com.percpico.util.mc.TraditionalJava
{
    @ScopedThis
    public byte[] getReadBuffer() throws InterruptedException;
    @ScopedThis
    public int readBufferLength();
}
```
Blocking

Traditional java thread

- Must be transformed into real-time thread
- Transfigurable interface
- `run` method, as `Runnable`
  - Expects variable length argument list
  - Returns Object
- Implements logic for blocking operation
public class SharedDevice extends Device implements TraditionalVisible{
    class SynchronizableLogic implements Transfigurable{
        @Scoped SynchronizableLogic synchronizer;

        @Scoped @ScopedThis
        public final byte [] getReadBuffer() throws InterruptedException {
            return (byte []) NoHeapRealtimeThread.transfigure( new ThreadStack(SynchronizableLogic.class), synchronizer, this);
        }
    }
}
Annotations

Purpose

- Impose restrictions on traditional java
- Improves integration between environments
- Verification by byte-code verifier

Annotations

- @Scoped
  - variables: May be in scoped memory
  - method: May return objects allocated in scoped memory
- @ScopedThis
  - this within a method is in scoped memory
- @ScopedPure
  - Both for all reference arguments
- @CallerAllocatedResult
  - Callers of a method must specify scope the result is placed
Restrictions on Traditional Java

- Methods cannot return results in caller’s scope
  - No @CallerAllocatedResult
- Cannot use immortal memory
  - No @ImmortalAllocation
- Must treat arguments as scoped memory allocated
  - @ScopedPure (or appropriate @Scoped and @ScopedThis)
Sharing Objects between environments

From hard real-time to traditional
- Hard real-time objects are *published* through a proxy-object
  - com.percpico.util.mc.Registry.publish()
- Obtained by traditional java
  - Registry.lookup() passing as argument the publish-name

From traditional to hard real-time
- Traditional java thread serving as communication channel
- Blocks until request is received from hard real-time thread
- Communicates back result through a new traditional method call
Contributions

- Illustrates weaknesses and alternatives to RTSJ
- Introduces reliable real-time garbage collector for soft real-time systems
- Illustrates integration of mixed real-time systems
- Provides means of “defining intentions” through annotations
### The Article

- **Unstructured**
  - Benchmarks
  - Comparison of languages
  - More chapters?

- **Unclear**
  - Short or missing explanation of examples
  - Missing pieces
Questions?