A New I/O Model for the Real-Time Specification for Java

Dr. James J. Hunt
CEO, aicas GmbH
JSR-282 Spec Lead
JTRES 2012
New RTSJ I/O Model

Agenda

The Old Model
Goals and Challenges of Redesign
The Proposed Model
  • Raw Memory
  • Asynchronous Events and their Managers
  • Asynchronous Event Handlers
  • Interrupt Service Routines
  • Sequence diagrams
Problems with RTSJ 1.0.2

Device memory access inefficient

- Single class for accessing all precision for each of integral and floating types:
  - RawMemoryAccess
  - RawMemoryFloatAccess
- Unclear how I/O memory is address: filters under defined
- Access testing needed on every access
- Cannot type restrict memory access
- Hard to inline
Problems with RTSJ 1.0.2

Happening ill defined

- No standard way to add new happenings
- Happenings not objects
- Inconsistent with other event types
  - POSIXSignalHandler
  - Timer
  - User triggered events
- Too many levels of indirection:
  Happening $\rightarrow$ AsyncEvent $\rightarrow$ AsyncEventHandler
- No support for first level interrupt handlers
Goals of New Model

Unify all input types
Reduce input overhead
Provide user defined Happenings and Clocks
Support interrupt handlers in Java
  • bare metal JVM
  • JVM for user space device driver
Support typing, e.g., ensure LSP compatibility
Add long and Object payloads to Events
Challenges for Restructuring

Maintain backward compatibility
- Easy for output types
- Harder for input types
  - Happening
  - AsyncEvent
  - Clock/Timer
  - POSIXSignalHandler

Preserve ability to multiplex server threads

Not require changes to compiler
New Raw Memory Architecture

FactoryBased

- RawMemory class for registration
- RawMemoryFactory for implementation

Interfaces for each access type: RawInt, RawShort, RawByte, RawFloat, etc.

Concrete classes for

- Memory mapped devices,
- I/O mapped devices, and
- Generic mapped devices.
RawShort Architecture

Java.io.Closable

RawMemory

RawShortRead
getShort(): short
getShort(int): short
get(int, short()): int
get(int, short(), int, int): int

RawShortWriter
setShort(short)
setShort(int, short)
set(int, short()): int
set(int, short(), int, int): int

RawShort
Raw Memory Architecture
RawMemoryFactory

<table>
<thead>
<tr>
<th>javax.realtime::RawMemoryName</th>
</tr>
</thead>
<tbody>
<tr>
<td>RawMemoryName(String name)</td>
</tr>
<tr>
<td>toString(): String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>javax.realtime::RawMemoryCreator</th>
</tr>
</thead>
<tbody>
<tr>
<td>«interface»</td>
</tr>
<tr>
<td>+createRawLongInstance(long base, int size): RawLong</td>
</tr>
<tr>
<td>+createRawIntInstance(long base, int size): RawInt</td>
</tr>
<tr>
<td>+createRawShortInstance(long base, int size): RawShort</td>
</tr>
<tr>
<td>+createRawByteInstance(long base, int size): RawByte</td>
</tr>
<tr>
<td>+createRawFloatInstance(long base, int size): RawFloat</td>
</tr>
<tr>
<td>+createRawDoubleInstance(long base, int size): RawDouble</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>javax.realtime::RawMemoryFactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>+getDefaultRawMemoryFactory(): RawMemoryFactory</td>
</tr>
<tr>
<td>+register(RawMemoryName typeName, RawMemoryCreator creator)</td>
</tr>
<tr>
<td>+createRawLongInstance(RawMemoryName type, long base, int size): RawLong</td>
</tr>
<tr>
<td>+createRawIntInstance(RawMemoryName type, long base, int size): RawInt</td>
</tr>
<tr>
<td>+createRawShortInstance(RawMemoryName type, long base, int size): RawShort</td>
</tr>
<tr>
<td>+createRawByteInstance(RawMemoryName type, long base, int size): RawByte</td>
</tr>
<tr>
<td>+createRawFloatInstance(RawMemoryName type, long base, int size): RawFloat</td>
</tr>
<tr>
<td>+createRawDoubleInstance(RawMemoryName type, long base, int size): RawDouble</td>
</tr>
</tbody>
</table>

New RTSJ I/O Model
Example

Public class IOBusController
{
    private MemoryRawByte command_;
    private MemoryRawByte flag_;  
    private MemoryRawShort data_;  
    private MemoryRawInt address_;  
    private long base_;  

    public short get(int offset)
    {
        address_.put((int)base_ + offset);
        command_.put(READ);
        while (flag_.get() != DONE);
        return data_.get();
    }
}
Happening: Kind of AsyncEvent

AsyncEvent
- Passive (fire mechanism)
- Runs all associated event handlers
- User definable

ActiveEvent: Happening, Timer, POSIX*Signal
- supports active behavior too (trigger mechanism)
- Can have (needs) dispatcher to manage activity
- Can be triggered from outside the VM
Happening Architecture

AbstractAsyncEvent

AsyncObjectEvent

AsyncLongEvent

AsyncEvent

POSIXRealtimeSignal

POSIXSignal

ActiveEvent

ISR

Happening

New RTSJ I/O Model
Event Architecture

Visibility
+= public
# = protected
~= package

javax.realtime::AbstractEvent
«abstract»
+get(String name): AbstractEvent
+getName(AbstractEvent event, String name)
+unsubscribe(String name)
+unsubscribe(AbstractEvent event)
+addHandler(AbstractEventHandler)
+setHandler(AbstractEventHandler)
+removeHandler(AbstractEventHandler)
+handledBy(AbstractEventHandler): boolean

javax.realtime::ActiveEvent
«interface»
+start()
+stop(): boolean
+deregister()
+getName(): String

javax.realtime::AsyncObjectEvent
+fire(Object value)

javax.realtime::AsyncLongEvent
+fire(long value)

javax.realtime::AsyncEvent
+fire()
+attach(String happeningName)

javax.realtime::POSIXSignal
+get(String name): POSIXSignal
+register(POSIXSignalDispatcher dispatcher)

javax.realtime::POSIXRealtimeSignal
+get(String name): POSIXRealtimeSignal
+register(POSIXRealtimeSignalDispatcher dispatcher)

javax.realtime::Happening
+Happening(String name)
+getReference(String name): int
+get(String name): Happening
+getId(): Happening
+register(HappeningDispatcher dispatcher)
+getId(): int
~attach(Happening happening)

javax.realtime::OneShotTimer

javax.realtime::PeriodicTimer

javax.realtime::Timer
+start(boolean disabled)
+register(Clock)
+destroy()
+enable()
+disable()
Dispatchers

One for each ActiveEvent type

- HappeningDispatcher for Happening
- POSIXSignalDispatcher for POSIXSignal
- POSIXRealtimeSignalDispatcher for POSIXRealtimeSignal
- TimerDispatcher for Timer (Clock is ISR for Timer)

Provide contains thread to run dispatch loop

- Dispatcher can dispatch several active events
- User can scheduling parameters for thread
Dispatcher Architecture

javax.realtime::InterruptServiceRoutine
+handleInterrupt(): boolean
#releaseInterrupt()

javax.realtime::Dispatcher
  «abstract»
  ~Dispatcher(int priority)

javax.realtime::Clock
  «abstract»
  +register(TimerDispatcher)
  +unregister()
  #getDispatcher(): TimerDispatcher
  #setTime(AbsoluteTime time)

javax.realtime::POSIXDispatcher
+POSIXDispatcher(int priority, int queueSize)
~trigger(int id)
+overrunHandler(AbstractEventHandler)
~trigger(POSIXSignal target)

javax.realtime::POSIXRealtimeDispatcher
+POSIXRealtimeDispatcher(int priority, int queueSize)
~trigger(int id, long value)
+overrunHandler(AbstractEventHandler)
~trigger(POSIXRealtimeSignal target, long value)

javax.realtime::TimerDispatcher
+TimerDispatcher(int priority, int QueueSize)
~trigger(Timer target)
+atTime()
+discontinuity(RelativeTime delta)
+overrunHandler(AbstractEventHandler)

javax.realtime::HappeningDispatcher
+HappeningDispatcher(int priority, int queueSize)
+trigger(int id)
+overrunHandler(AbstractEventHandler)
~trigger(Happening target)

New RTSJ I/O Model
AsyncEventHandler

Extended to support Long and Object Events

- AsyncEventHandler
- AsyncLongEventHandler
- AsyncObjectEventHandler

New base type for compatibility

- AbstractEventHandler

Not tied to corresponding event type

- Can use with and AsyncEvent
## Mix and Match

<table>
<thead>
<tr>
<th>Types</th>
<th>AsyncEvent</th>
<th>AsyncLongEvent</th>
<th>AsyncObjectEvent</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsyncEventHandler</td>
<td>No payload</td>
<td>No payload</td>
<td>No payload</td>
</tr>
<tr>
<td>AsyncLongEventHandler</td>
<td>Event ID</td>
<td>Payload (int)</td>
<td>Event ID</td>
</tr>
<tr>
<td>AsyncObjectEventHandler</td>
<td>Event Object</td>
<td>Event Object</td>
<td>Payload (Object)</td>
</tr>
</tbody>
</table>
AsyncEventHandlers
InterruptServiceRoutine

Provide first level interrupt handing in Java

- Intended to fire a Happening

Limited Java context

- Requires priority ceiling emulation
- Limited memory allocation

Only usable in JVM interrupt capable context

- JVM on bare metal
- JVM bound in kernel
- JVM bount in user space device driver.
Dispatching Happenings

New RTSJ I/O Model
Dispatching Timers

[Diagram showing the new RTSJ I/O model with Dispatching Timers]
Supporting Binding a Happening
Conclusion

New Happening and Raw Memory models

- Improve performance
- Enable extension by programmer
- Make interrupt controlled I/O possible
- Make device drivers in Java possible
- Can be used to implement higher level interface
  - Real-Time Data Access
  - Hardware Objects
Questions?