Java doing Hard Time

Practical Real Time Java Experiences

March 15, 2004
THRijswijk

by

Jandit van Doorn &
Jan Edelbroek

TURNKIEK

Real-Time Java

Experiences in high speed & performance embedded style application

Jan Edelbroek, Jandit van Doorn
Turnkiek Technical Systems for:

TU Delft

Your 'Entertainers'

- **Jan** Edelbroek - System Architect Embedded software
- **Jandit** van Doorn - Chief Technology Officer @ Imtech ICT
  &
  Lector Software Engineering for
  Real Time & Embedded Systems
  @ Saxion Universities
  of
  Professional Education

Contents

Part 1: Background & motivation by Jandit
- Introduction to Imtech ICT Technical Systems
- Java in technical environments
- Why our Real Time Java efforts

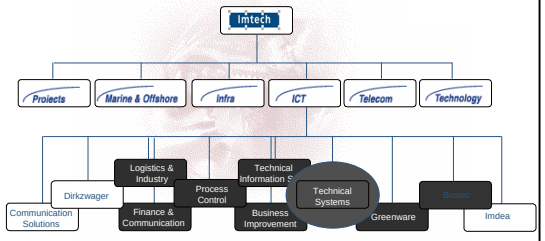
Part 2: Demonstrator implementation by Jan
- Laser Projector idea
- Demonstrator solution
- Lessons learned and future work (Jandit again)

Contents

Part 1: Background & motivation by Jandit
- **Introduction to Imtech ICT Technical Systems**
- Java in technical environments
- Why our Real Time Java efforts

Part 2: Demonstrator implementation by Jan
- Laser Projector idea
- Demonstrator solution
- Lessons learned and future work

Organisation



Information Technology

Mission statement:

- **Early adopter**
- Quality solutions
- Cost effective and future aimed
- In technical industrial & scientific market
- Reliable and valued partner
  - Give the customer peace of mind
  - Commitment = commitment
  - Small things matter
  - Treat and respect people as people

Platforms
- Java
- Embedded
- Microsoft
- Unix /Linux

Service areas.
- Inproduct sw
- V&S systemen
- Prod. Autom
- Network Comp

Market segments
- R &D
- Traffic & Transport
- Manufacturing
- Defense
- Advanced machines
- Utilities
- Telecom

Mission statement:

- **Early adopter**
- Quality solutions
- Cost effective and future aimed
- In technical industrial & scientific market
- Reliable and valued partner
  - Give the customer peace of mind
  - Commitment = commitment
  - Small things matter
  - Treat and respect people as people

Platforms
- Java
- Embedded
- Microsoft
- Unix /Linux

Service areas.
- Inproduct sw
- V&S systemen
- Prod. Autom
- Network Comp

Market segments
- R &D
- Traffic & Transport
- Manufacturing
- Defense
- Advanced machines
- Utilities
- Telecom

Contents

Why Java in Embedded ?

- More reliable
- More productive
  - Object oriented
  - High level
  - Component model
- Inherent networking
- Less porting effort (if any)
- Machine code skills deteriorating

Java Misconceptions ? !

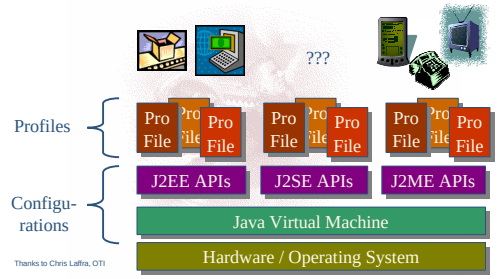
- "Too big, too slow, too unpredictable"
- Not deterministic
  - Just-In-Time compilation
  - Garbage collector
- Not Real-Time capable
  - Weak scheduling and synchronization semantics
  - No way to express or enforce timeliness constraints
- Too high-level
  - No direct access to memory or registers
  - No interrupt handling

Contents

Why our Real Time Java efforts ?

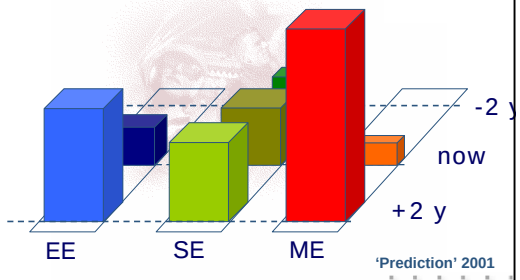
- Active in Java / Network computing 'long time'
- 35% Turnover from 'Embedded'
  Growing market !
- High expectations for Java in Embedded
- BUT… is it viable, does it work ?
- If yes: just show, we CAN go the last mile too !
  (most work will be embedded, not Real Time)
- Early adopter
- We just **need** the productivity gain

Java™ 2 Platform

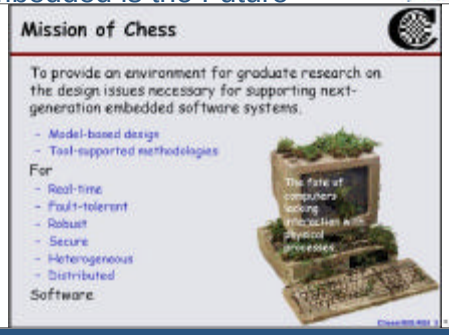


Thanks to Chris Laffra, OTI

Evolution -> Revolution

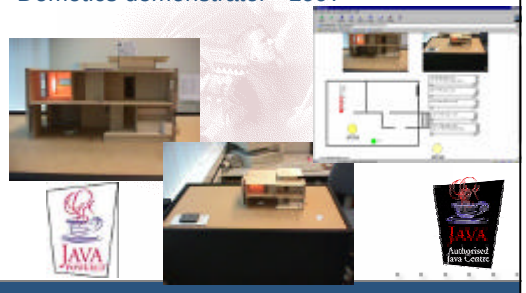


Embedded is the Future



Early Adopter ?

- Domotics demonstrator - 1997



The critics:

Yes,

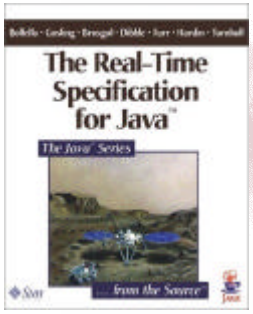
but……

What about Real Time?

Specification(s) on RT Java

- J-Consortium, RTJWG
  - NewMonics, HP, Microsoft, Aonix, Honeywell,...
- **JCP, JSR-0000001**
  - Sun, IBM, Mitre, Windriver, QNX, Motorola, Rockwell,..
  - Delivered Q4-2001 (app. 1 year delay)
    - Specs
    - Reference implementation
    - Test suite

Interested in Specs ?



Amazon:

Product Details

Paperback: 195 pages ; Dimensions (in inches): 0.50 x 9.00 x 7.25

Publisher: Addison-Wesley Pub Co; 1st edition (January 15, 2000)

ASIN: 0201703238

Average Customer Review: ****• Based on 5 reviews.

Amazon.com Sales Rank: 179,013

RTSJ ambitions

Guiding Principles

- Applicability to particular Java environments
- Backward compatibility
- Write once **carefully**, run anywhere **conditionally** (WORA → WOCRAC)
- "Toys to Cruise Missiles"
- Current practice versus advanced future features
- Predictable execution
- No syntactic extension

Current status of RT-Java

- Specification 1.0 ready on the November 12th , 2001
- OTI (IBM) had virtual machine with preliminary implementation end 2000. Turnkiek teamed…
- Timesys Linux has reference implementation ready since march 2002
- Difficult economic times, no progress @ OTI & others
- Ongoing work on Distributed Real Time (JSR-000050 : www.drtsj.org)

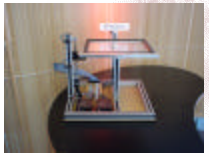
RTSJ: 8 modification areas

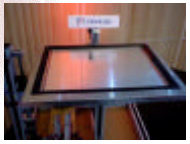
1. Thread scheduling and dispatching
2. Memory management
3. Synchronization and resource sharing
4. Physical memory access
5. Asynchronous event handling
6. Asynchronous transfer of control
7. Asynchronous thread termination
8. Exceptions

Read the Spec's (RTFM)

OK, we took the RT challenge

- Labyrinth – 2000 (finished 2001)
- Based on preliminary RTSJ specs and product





- And by now: **remotely controlled** (O2 XDA, Java midle

And again, the critics:

Nice, your RT demo…

But not very speedy !

☹

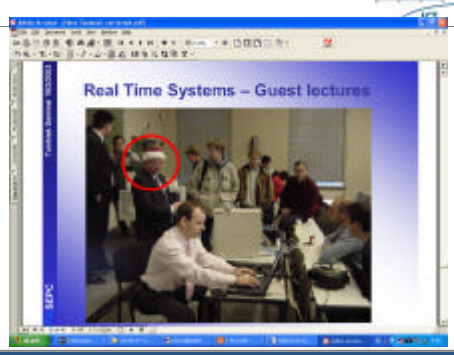
So,

we looked for the sound barrier …

And present it to you now

☺

Coffee Break



Time for the REAL action ?

The
Real Time
Java
Laser
Demon-
strator

Contents

Laser Demonstrator

Issues covered:

- The objectives of the demonstrator
- Why laser projector?
- System overview
- Exploring RT-Java
- Conclusions
- What's next?
- Additional information

Laser Demonstrator

The objectives of the demonstrator

- Get experience with Java in Real Time, high performance
- Examine possibilities of RT-Java for embedded applications
- To boldly go where Java has never gone before explore the limits (and we found them)
- Finally have a nice demonstrator

Laser Demonstrator

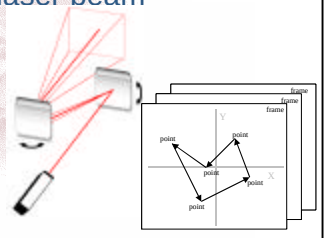
Why laser Projector ?

- Real time aspect is important for both drawing a figure and playing an animation
- Very often real time, really high speed Up to 30.000 points / sec
- Laser is attractive to see

Laser Demonstrator

Controlling the laser beam

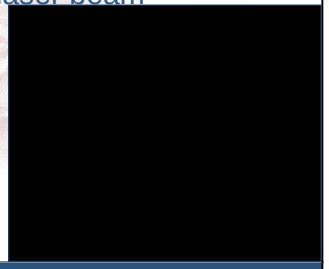
- Laser beam is projected on surface via 2 mirrors
- One mirror moves beam in **horizontal** direction
- Other mirror moves beam in **vertical** direction



Laser Demonstrator

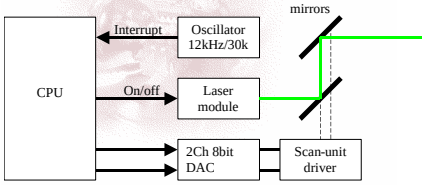
Controlling the laser beam

- Laser beam is projected on surface via 2 mirrors
- One mirror moves beam in **horizontal** direction
- Other mirror moves beam in **vertical** direction



Laser Demonstrator

Hardware Overview



Laser Demonstrator Components

Hardware

- PC compatible
  Pentium II 350 MHz initially
  Now demo's on 300 MHz, laptop
  Also ran on 700 MHz SBC
- 2 channel digital to analog converter with oscillator 12kHz/30kHz and digital output (***home brew***)
- Closed loop scan-unit driver
- Scanner safety board
- Power supply

Mechanical / optical

- Diode pumped solid state laser 10 mW with modulation input
- Scan-unit with XY-galvos, 30 kpps

Software

- QNX real time platform 6.1
- J9 virtual machine 1.5 with real time extensions
- Websphere Studio Device Developer 4.0 based on Eclipse, successor of Visual Age Micro Edition 1.4

Laser projector standards

ILDA, International Laser Display Association, defined a standard laser projector

- Signals specification
- Image data transfer format
- Test patterns for speeds 12k and 30k

INTERNATIONAL LASER DISPLAY ASSOCIATION

Contents

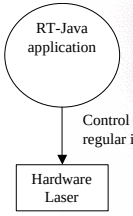
Part 1: Background & motivation by Jandit

- Introduction to Imtech ICT Technical Systems
- Java in technical environments
- Why our Real Time Java efforts

Part 2: Demonstrator implementation by Jan

- Laser Projector idea
- **Demonstrator solution**
- Lessons learned and future work

Exploring RT-Java

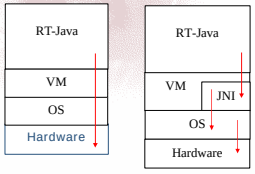


The challenges:

- How to control the hardware?
- How to achieve intervals of 30.000 times/second?

How to control the hardware?

- `RawMemoryAccess` gives direct access to memory-mapped hardware (i.e. device registers)
- Java Native Interface (JNI)



Finally, some code

RawMemoryAccess

Java code

```
Public void init()
{
  RawMemoryAccess eDvc = new RawMemoryAccess(type, 512);
  try
  {
    // read from device
    int reg = eDvc.getInt(CTRLREG);
    // write to device
    eDvc.setInt(CTRLREG, 0);
  }
  catch (Exception e)
  {
    // Do error handling here
  }
}
```


Code, continued

JNI

Java code

```
package natives;
public class ParallelPort8bits {
  private static native boolean moveTo(int x,int y,boolean blanking);
  static
  {
    System.loadLibrary("lpt8bits");
  }
}
```

C code in JNI layer

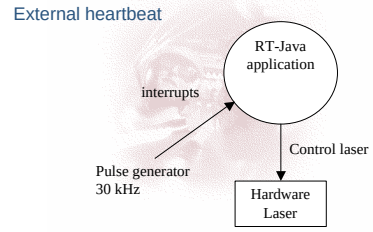
```
JNIEXPORT jboolean JNICALL Java_natives_ParallelPort8bits_1moveTo
  (JNIEnv * jenv, jclass jcls, jint x, jint y, jboolean blanking)
{
   ... Write data to I/O port here, using conventional in8 and out8 functions ...

   return JNI_TRUE;
}
```

How to achieve 30k points/sec?

- RT-Java contains timer (`PeriodicTimer`) and thread (`RealtimeThread`) classes
- Granularity in nanoseconds, accuracy usually in milliseconds
  (related to the Operating System time-base)
  Insufficient for our purpose !
- Therefore we have to make use of (hardware generated) interrupts

The next challenge

Hardware interrupts

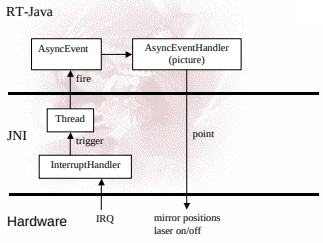


How to handle external interrupts?

Three options investigated

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`
- Call Java method directly from native interface (aka. Callback)
- Handle interrupt in native interface and provide data to native interface.

Attach interrupt to AsyncEvent



Attach interrupt to AsyncEvent

C Code in JNI layer

```
void * int_thread (void *arg) {
        ...
        (*theVM)->AttachCurrentThread(theVM, (void **)&env, NULL);
        mid = (* env)->GetMethodID(env, cls, "fire","()V");
        id = InterruptAttach (irq_number, interrupthandler, NULL, 0, 0);
        while (1) {
                InterruptWait(NULL,NULL);
                (* env)->CallVoidMethod(env, obj, mid);
        }
}
const struct sigevent * interrupthandler (void *arg, int id) {
        SIGEV_INTR_INIT(&event);
        return &event;
}
```


Attach interrupt to AsyncEvent

Java code

```
AsyncEvent event = new AsyncEvent();

AsyncEventHandler evHandler = new
        AsyncEventHandler(sp,null,null,ImmortalMemory.instance(),null)
{
        public void handleAsyncEvent()
        {
                // put your code here to handle the event
        }
};

event.addHandler(evHandler);
```

Attach interrupt to AsyncEvent

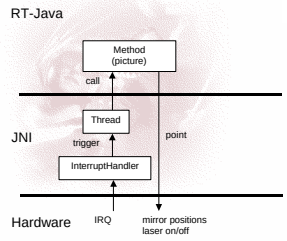
Advantages

- High use of available classes of RT-Java
- Very flexible

Disadvantages

- Delay between interrupt and new point to output can be up to 0,5 msec
- Delay varies a lot
  Not deterministic, so **not** RT
- Not useful for this application

Call directly from native interface



Call directly from native interface

C Code in JNI layer

```
void * int_thread (void *arg) {
        ...
        (*theVM)->AttachCurrentThread(theVM, (void **)&env, NULL);
        mid = (* env)->GetMethodID(env, cls, "point","()V");
        id = InterruptAttach (irq_number, interrupthandler, NULL, 0, 0);
        while (1) {
                InterruptWait(NULL,NULL);
                (* env)->CallVoidMethod(env, obj, mid);
        }
}
const struct sigevent * interrupthandler (void *arg, int id) {
        SIGEV_INTR_INIT(&event);
        return &event;
}
```


Call directly from native interface

Java code ;-)

```
public void point()
{
        // put your code here to handle the event
}
```


Call directly from native interface

Advantages

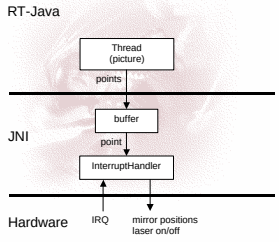
- Faster than AsyncEvent / AsyncEventHandler
- Delay approximately 20 microseconds & constant

Disadvantages

- Less flexible
- High CPU load

Can be used for this application, but looking further

Handle int'rupt in native interface



Handle int'rupt in native interface

C Code in JNI layer

```
JNIEXPORT jboolean JNICALL Java_natives_ParallelPort8bits_native_1moveTo
  (JNIEnv * jenv, jclass jcls, jint x, jint y, jboolean blanking)
{
        // store point in buffer
        return JNI_TRUE;
}

const struct sigevent * interrupthandler (void *arg, int id)
{
        // get point from buffer and control laser
        return NULL;
}
```


Handle int'rupt in native interface

Java code

```
private class FramePlayer extends RealtimeThread {
public void run() {
  while (running) {
    while (result) {
      result = beamer.moveTo(x, y, blanking);
      ...
    }
    Thread.sleep(1, 0);
  }
}

player = new FramePlayer( ... );
player.start();
```


Handle int'rupt in native interface

Advantages

- Java application is less time critical, but still uses RT-Java for animation
- CPU load decreases drastically

Disadvantages

- Some of the real time control is moved to the JNI
- Still rely on C through JNI, not fully Java functionally

External interrupts wrap-up

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`
- Call Java method directly from native interface (callback)
- Handle interrupt in native interface and provide data to native interface.

External interrupts wrap-up

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`

Delays 0,5 msec UNACCEPTABLE

- Call Java method directly from native interface (callback)
- Handle interrupt in native interface and provide data to native interface.

External interrupts wrap-up

- Attach interrupt handler to ~~`AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`~~

Delays 0,5 msec UNACCEPTABLE

- Call Java method directly from native interface (callback)
- Handle interrupt in native interface and provide data to native interface.

External interrupts wrap-up

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`
  Delays 0,5 msec UNACCEPTABLE
- Call Java method directly from native interface (callback) **Delays 20 usec, code not nice**
  Acceptable but a lot of overhead
- Handle interrupt in native interface and provide data to native interface.

External interrupts wrap-up

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`
  Delays 0,5 msec UNACCEPTABLE
- Call Java method directly from native interface (callback) **Delays 20 usec, code not nice**
  Acceptable but a lot of overhead
- Handle interrupt in native interface and provide data to native interface.

External interrupts wrap-up

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`
  Delays 0,5 msec UNACCEPTABLE
- Call Java method directly from native interface (callback) **Delays 20 usec, code not nice**
  Acceptable but a lot of overhead
- Handle interrupt in native interface and provide data to native interface.

The solution we used

External interrupts wrap-up

- Attach interrupt handler to `AsyncEvent` and use `AsyncEvent` and `AsyncEventHandler`
  Delays 0,5 msec UNACCEPTABLE
- Call Java method directly from native interface (callback) **Delays 20 usec, code not nice**
  Acceptable but a lot of overhead
- Handle interrupt in native interface and provide data to native interface.

The solution we used

Threading issue, which one to use

`RealtimeThread` versus `NoHeapRealtimeThread`
➔ Hard vs. soft Real Time trade off
Latter also introduces necessity to use `ScopedMemory`

- `RealtimeThread` cannot preempt GC
- When new objects are created, an On-demand garbage collection can delay a `RealtimeThread` from execution
- `NoHeapRealtimeThread` can preempt GC.
- `NoHeapRealtimeThread` has to be used in combination with `ScopedMemory` (doesn't work in current VM: J9)

Conclusions

- RT-Java can handle interrupts
- Not to the full extend ☹, still C down there
- `RealtimeThreads` are indeed real time, except for On-demand garbage collection (but that is what **you** do)
- JNI calls are not efficient (in J9 VM ?)
- `AsyncEvents` and `AsyncEventHandlers` seem to cost a lot of time
- Virtual machine (J9) currently has several shortcomings (not conforming to current specs)

Demo (now the real work)

Questions so far …

?

Contents

Java **Mis**conceptions!

- "Too big, too slow, too unpredictable"
- Not deterministic
  - Just-In-Time compilation
  - Garbage collector
- Not Real-Time capable
  - Weak scheduling and synchronization semantics
  - No way to express or enforce timeliness constraints
- Too high-level
  - No direct access to memory or registers
  - No interrupt handling

Conclusions revisited

- RT-Java can handle interrupts
- Not to the full extend ☹, still C down there
- `RealtimeThreads` are indeed real time, except for On-demand garbage collection
- JNI calls are not efficient (in J9 VM ?)
- `AsyncEvents` and `AsyncEventHandlers` seem to cost a lot of time
- Virtual machine (J9) currently has several shortcomings (not conforming to current specs)

Conclusions

- RT-Java can handle interrupts
- Not to the full extend ☹, still C down there
- `RealtimeThreads` are indeed real time, except for On-demand garbage collection
- JNI calls are not efficient (in J9 VM ?)
- `AsyncEvents` and `AsyncEventHandlers` seem to cost a lot of time **Why ?**
- Virtual machine (J9) currently has several shortcomings (not conforming to current specs)

Conclusions

- RT-Java can handle interrupts
- Not to the full extend ☹, still C down there
- `RealtimeThreads` are indeed real time, except for On-demand garbage collection
- JNI calls are not efficient (in J9 VM ?)
- `AsyncEvents` and `AsyncEventHandlers` seem to cost a lot of time
- Virtual machine (J9) currently has several shortcomings (not conforming to current specs)

Why ?
Howcome ?

Conclusions

- RT-Java can handle interrupts
- Not to the full extend ☹, still C down there
- `RealtimeThreads` are indeed real time, except for On-demand garbage collection
- JNI calls are not efficient (in J9 VM ?)
- `AsyncEvents` and `AsyncEventHandlers` seem to cost a lot of time
- Virtual machine (J9) currently has several shortcomings (not conforming to current

Why ?
Howcome ?
Others ?

Conclusions

- RT-Java can handle interrupts
- Not to the full extend ☹, still C down there
- `RealtimeThreads` are indeed real time, except for On-demand garbage collection
- JNI calls are not efficient (in J9 VM ?)
- `AsyncEvents` and `AsyncEventHandlers` seem to cost a lot of time
- Virtual machine (J9) currently has several shortcomings (not conforming to current specs)

Why ?
Howcome ?
Others ?

Further investigations

- How to bind External events to `AsyncEvents`?
- How to use `RawMemoryAccess` for access to I/O-ports such as parallel port or dedicated I/O hardware?
- Will `NoHeapRealtimeThreads`, in combination with `ScopedMemory`, be totally independent of all kinds of garbage collections? (needs improved VM) ?
- How about other VM and/or OS ?
  We're doing MontaVista (HardHat) Linux right now…

  No longer any more,
  MontaVista stopped Java activities… Saxion however

RTSJ Implementations

- TimeSyS: ***Reference Implementation JSR-1***
- OTI/IBM: J9 (VAME → WSDD)
- Esmertec: JBed (moved focus to phones)
- Insignia Solutions: Jeode (also on VxWorks)
- aJile Systems: <Silicon>; "Physical" VM
- … and others

More info

- java.sun.com — Java
- www.rtj.org — Realtime Java Specification
- www.drtsj.org — Distributed Real Time (still thinking)
- www.embedded.oti.com — Websphere Studio Device Developer, J9
- www.timesys.com — Reference implementation RTJ
- www.qnx.com — QNX Realtime Operating System
- www.laserist.org/ilda, www.laserdisplay.org, www.medialas.com — Information on lasers, laser-components and ~shows
- Real-Time Java Platform Programming, Peter C. Dibble (ISBN 0-13-028261-8)
- jandit@turnkiek.nl, java@turnkiek.nl,
- edelbroek@turnkiek.nl

Questions / discussion

?

Email: ***Jandit.vanDoorn@imtech.nl***
Jan.Edelbroek@imtech.nl
Handouts: Will be provided in pdf through Jan Dirk

Thank you

For your time and attention