RJD

A Ravenscar-Java Profile Implementation

Hans Søndergaard
University College - Vitus Bering Denmark

joint work with
Bent Thomsen and Anders P. Ravn
Aalborg University, Denmark

JTRES 2006, Paris, October 2006

The Ravenscar-Java Development Project

RJD

- *Implement the Ravenscar-Java Profile on an aJ-100 processor*
- *Investigate whether Real-Time UML is useful when designing embedded real-time systems*
- *Investigate through development of an industrial case how useful the Ravenscar-Java Profile is for design and development of industrial embedded systems with real-time requirements*
- *Compare the Ravenscar-Java solution with a C++ solution.*

http://www.cs.aau.dk/ravenscar/

Outline

RJD

- An industrial case from FOSS
  - experiences and observations
- The aJ-100 Java processor
- Implementation of the Ravenscar-Java Profile
- Summary
- Discussion items

Industrial Case from FOSS

RJD

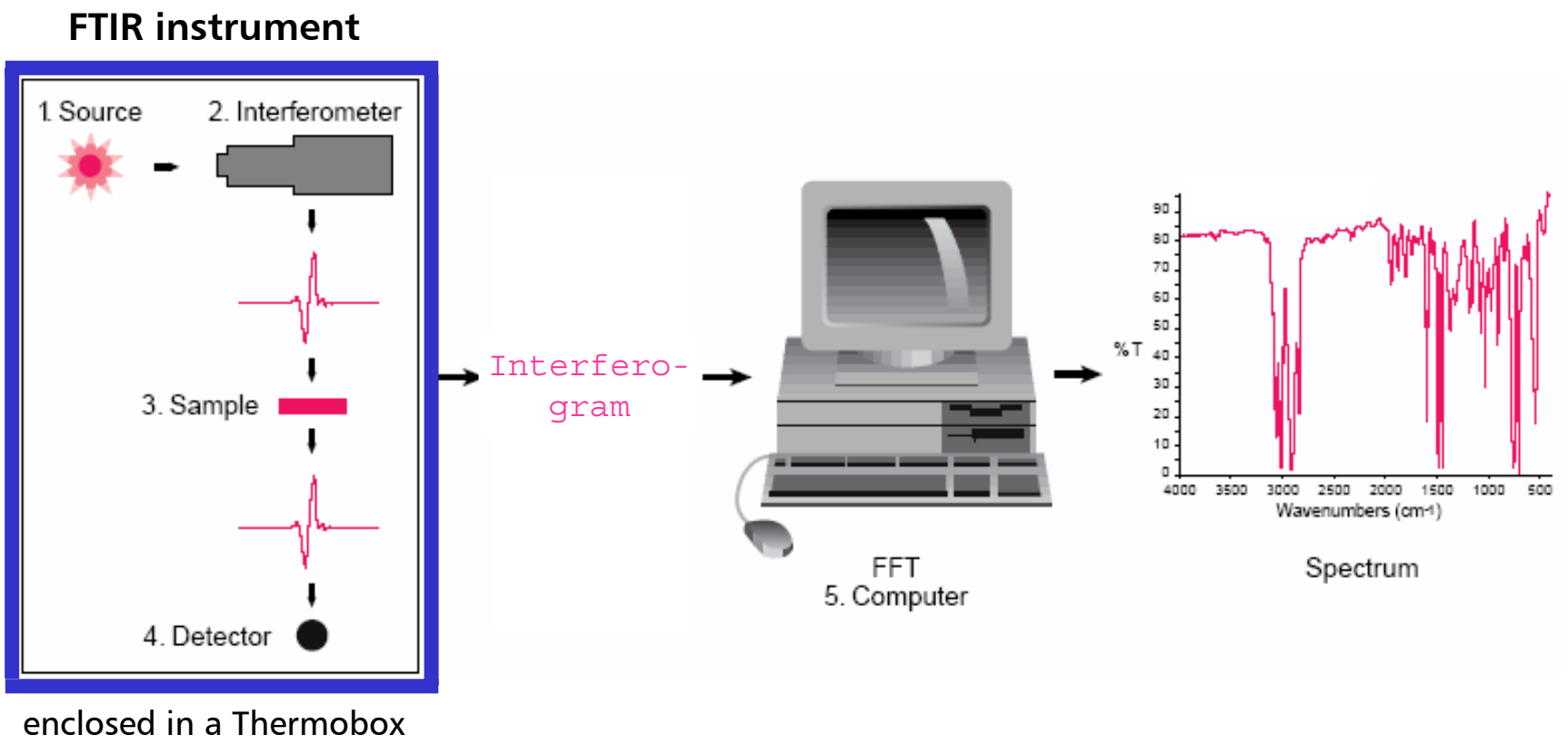
MilkoScan™ FT2

The instrument uses Fourier Transform Infrared Spectroscopy (FTIR) to identify quality parameters in milk

Existing FTIR program: written in C/C++

Purpose: redesign and implement the FTIR program using Ravenscar

The FTIR instrument



Functional requirements

RJD

- Temperature reading and regulation
  - reading
    - 5 times/sec
  - regulation
    - 1 time/sec
- Interferometer measurement
  - reading IR-detector
    - every 333 μs
- Other usual functionalities:
  - Watchdog, Monitoring, Logging, …

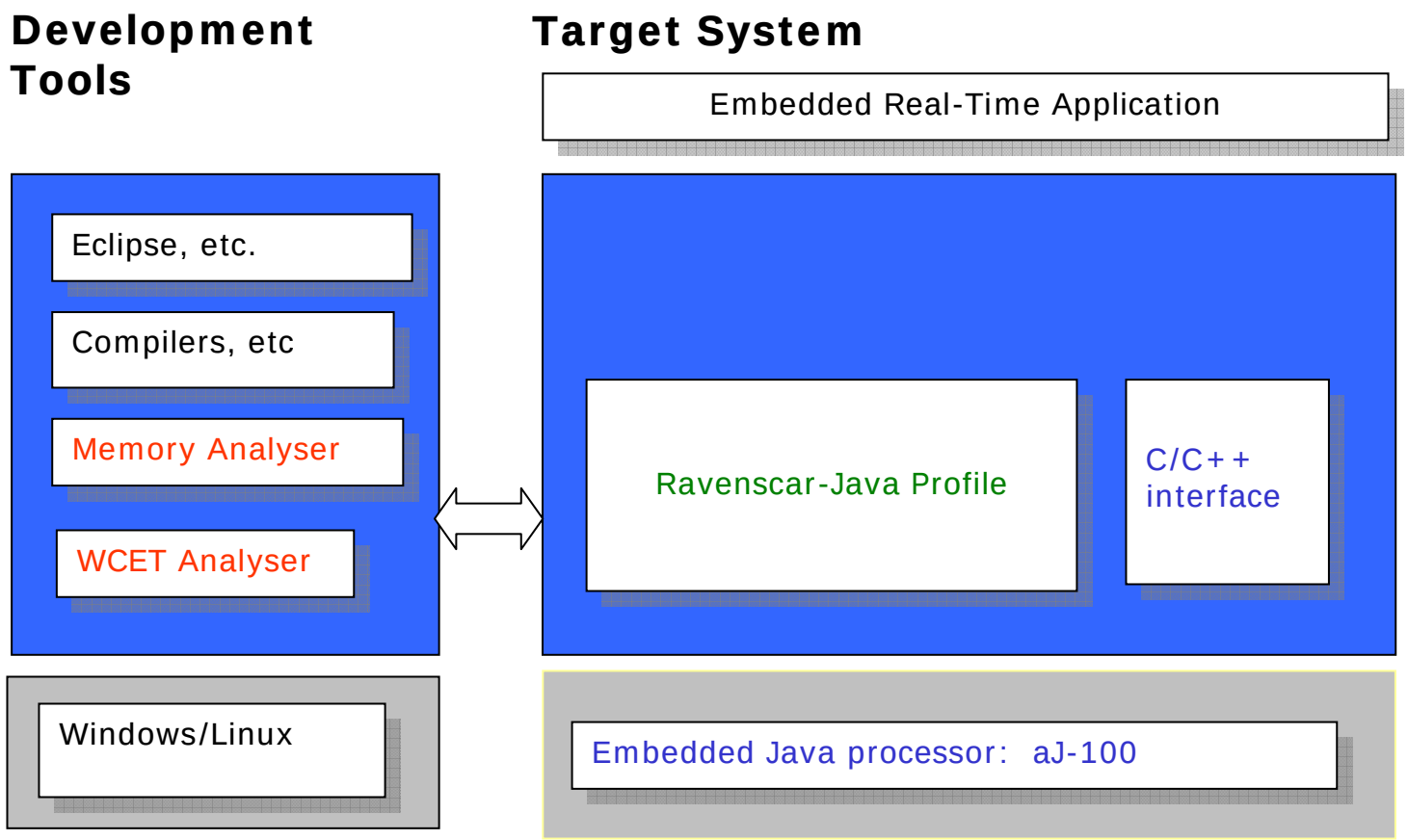
Process requirements

RJD

	Periodic/Sporadic activity	Period/Inter-arrival time (T)	Deadline (D)	Priority (P)
Temp reader	periodic	200 ms	10 ms	2
Temp regulator	periodic	1000 ms	100 ms	3
Interferometer measurement	sporadic	333 μs	200 μs	1 (highest)
Watchdog	periodic	1000 ms	D = T	5
Limits monitor	periodic	333 ms	D = T	4
Logging	periodic	5000 ms	D = T	6 (lowest)
Extern. Comm	periodic	333 ms	D = T	4

Overview of Future Foss system ?

RJD



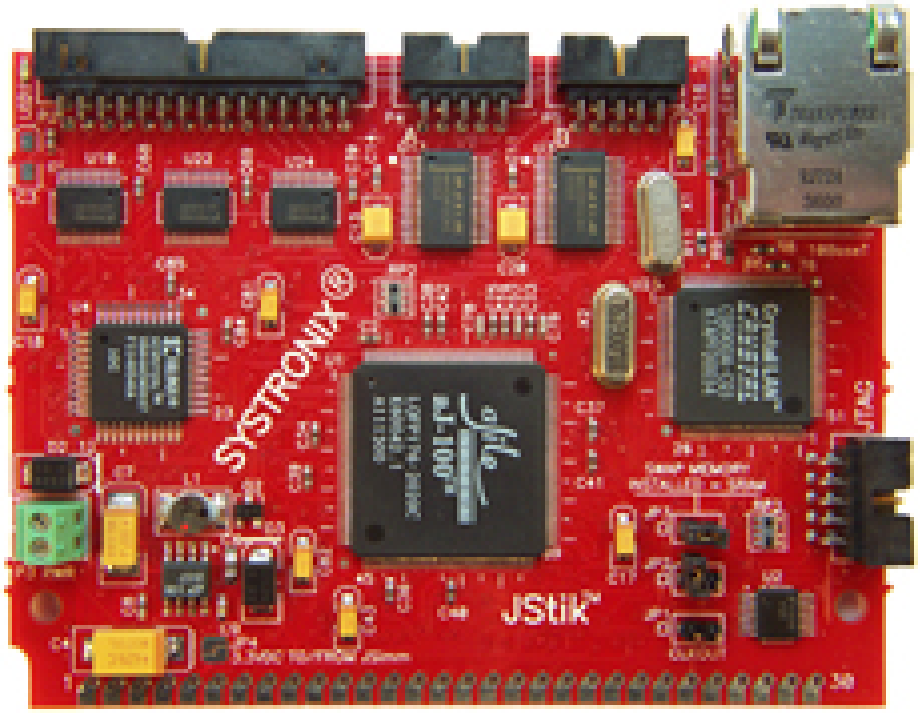
Ravenscar-Java for this application ?

RJD

- For industrial use:
  - Ravenscar-Java is simple and easy to use
  - deadline is missing
  - no pause and termination methods are specified
- Ravenscar-Java is not a subset of RTSJ
  - adds new classes
    - `Initializer, PeriodicThread`
    - `SporadicEvent, SporadicInterrupt`
- Scoped memory is difficult to understand and use
  - Peter Dibble: 40 pages (“hard to use”, p. 318)
  - Andy Wellings: 36 pages (“one of the most complicated areas of the RTSJ”, p. 170)
  - Instead, use: Real-Time garbage collector ?

Our implementation on the aJ-100 processor

RJD



The aJ-100 processor

RJD

- uses Java bytecode as its native instruction set
- embedded real-time multi-threading kernel
  - microcoded in hardware, including:
    - a priority pre-emptive scheduler, 32 priority levels
    - a priority ceiling protocol
    - periodic threads
- has all the common embedded peripherals
  - I/O Ports, Serial Interface, Ethernet, Timers, etc.

Programming aJile

RJD

- A runtime system based upon
  - J2ME (Java 2 Platform Micro Edition)
  - CLDC (Connected Limited Device Configuration 1.0)
- An aJile Java API to access the processor
  - 85 interfaces and classes, e.g.
    - `PianoRoll, PeriodicThread`
    - `rawJEM` (low level access to physical memory)
    - `GpioPin` (controls general purpose IO pins)
- JEM Builder and Charade:
  - tools for static linking and loading, etc.
- Other development tools are general purpose, e.g. Eclipse

Ravenscar-Java classes

RJD

- Real-time threads
  - Initializer thread
  - Periodic thread
- Sporadic event and interrupt
- Sporadic event handler
- Memory
  - Immortal memory
  - Raw memory
  - (Scoped memory)
- Time classes

Real-time threads

RJD

```
java.lang.Object
 |
 +--java.lang.Thread
      |
      +--javax.ravenscar.RealtimeThread
           |
           +--javax.ravenscar.Initializer
           |
           +--javax.ravenscar.NoHeapRealtimeThread
                |
                +--javax.ravenscar.PeriodicThread
```

Periodic thread

RJD

```
public class PeriodicThread extends NoHeapRealtimeThread
{ ...
  private com.ajile.jem.PeriodicThread aJileTh;

  public PeriodicThread (PriorityParameters pp, PeriodicParameters p,
                         Runnable logic)
  {
    super (pp,p,ImmortalMemory.instance(),logic);
    aJileTh = new aJilePeriodicThread();
    ...
  }
  public final void run() {
    info = SetupInfoArray.getInstance().getSetupInfo(this);
    aJileTh.makePeriodic (period, priority, …);
    aJileTh.start();
  }
  static boolean waitForNextPeriod() {
    com.ajile.jem.PeriodicThread.cycle();
    return true;
  }
}
```

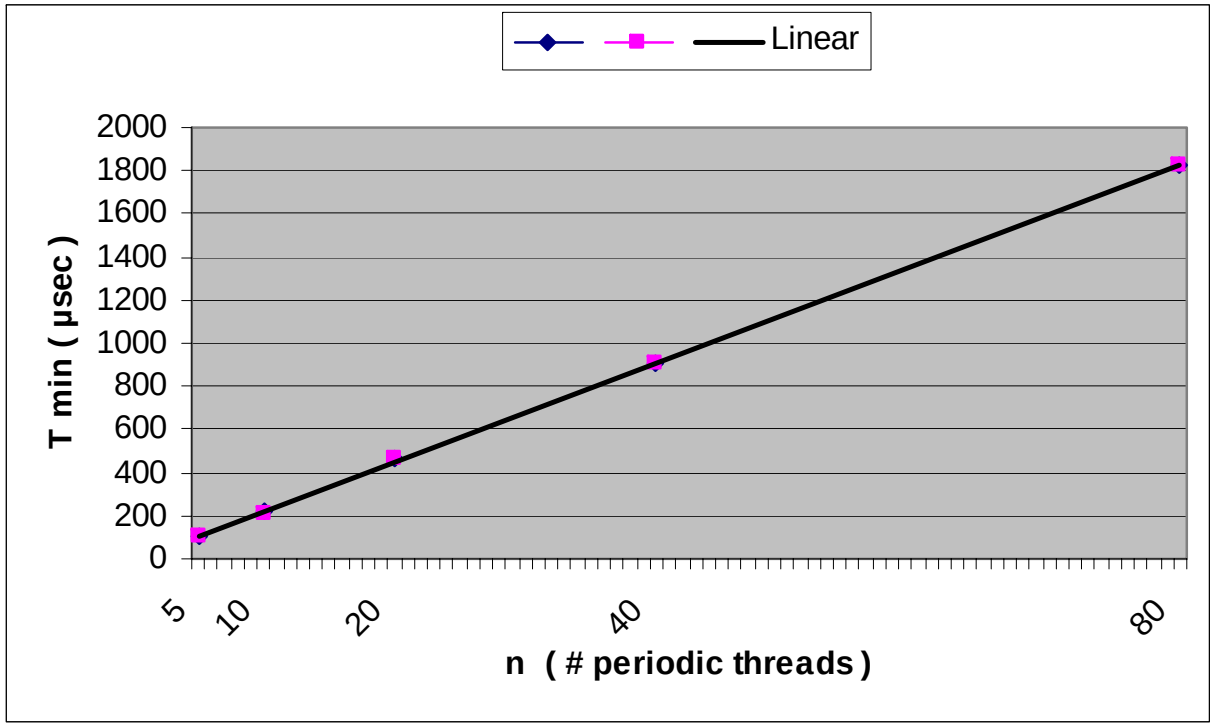
aJilePeriodicThread

RJD

```
private class aJilePeriodicThread extends com.ajile.jem.PeriodicThread
{
  public void run() {
    // run until start time:
    for(long count = info.startTime/info.period; count > 0; count--)
    {
      PeriodicThread.waitForNextPeriod();
    }
    // run from start time:
    for (;;)
    {
      logic.run();
      PeriodicThread.waitForNextPeriod();
    }
  }
}
```

Performance

RJD



- Just as effective as the native implementation
- Changing from thread-to-thread: < 1 μsec
- Execution time nearly the same as JOP, and comparable with C
- Can run up to 500 periodic threads.

Summary

RJD

- The Ravenscar-Java Profile
  - implemented on a native Java processor
    - about 35 classes, and 12 utility classes
  - easier than expected
    - lines of code:

      `PeriodicThread`: 25

      `SporadicEventHandler`: 35

      `SporadicInterrupt`: 20
    - total: 47 * 35 ≈ 1700 lines of code
- Just as efficient as the native implementation

Discussion items

RJD

- A revised profile
  - free the profile from the bindings of RTSJ
  - use the Midlet as model
    - start, pause and termination states
  - use a real-time garbage collector
  - throw away scoped memory