Safety–Critical Java on a Java Processor

Martin Schoeberl and Juan Ricardo Rios
Technical University of Denmark
Outline

- How does a SCJ application look like
- JOP implementation details
- Some wishes for a change
- Conclusion
A Safelet – is an interface
A Mission object
A sequencer
Collection of handlers
  ◦ Periodic
  ◦ Aperiodic

Interface to the world
  ◦ Simple terminal
  ◦ Many interfaces will be memory mapped IO
    • Just heard about it from James
public class HelloSafelet implements Safelet {

    public MissionSequencer getSequencer() {
        return new HelloSequencer(
            new HelloMission());
    }

    public long immortalMemorySize() {
        return 1000;
    }
}
public class HelloSequencer extends MissionSequencer {

    Mission m;

    public HelloSequencer(Mission mission) {
        super(new PriorityParameters(13),
             new StorageParameters(1000000, null));
        m = mission;
    }

    protected Mission getNextMission() {
        return m;
    }
}
public class HelloMission extends Mission {

    protected void initialize() {
        OutputStream os = null;
        try {
            os = Connector.openOutputStream("console:");
        } catch (IOException e) {
            throw new Error("No console available");
        }
        HelloHandler hh = new HelloHandler(new SimplePrintStream(os));
        hh.register();
    }

    public long missionMemorySize() {
        return 100000;
    }
}
public class HelloHandler extends PeriodicEventHandler {

    SimplePrintStream out;
    int cnt;

    public HelloHandler(SimplePrintStream sps) {
        super(new PriorityParameters(11),
              new PeriodicParameters(
                  new RelativeTime(0, 0),
                  new RelativeTime(500, 0)),
              new StorageParameters(10000, null), 500);
        out = sps;
    }

    public void handleAsyncEvent() {
        out.println("Ping " + cnt);
        ++cnt;
    }
}
There is a JOP simulation
JVM implemented in Java
  ◦ Same restrictions as JOP ;-)
  ◦ Reads and execute JOP ‘binaries’
Use System.currentTimeMillis() for scheduler
  ◦ Time checked during bytecode interpretation
  ◦ Slow, but ok
  ◦ No real-time guarantees
Simulation about as fast as a 1 MHz JOP
Good for system code debugging
Java Optimized Processor
- A JVM in hardware (FPGA)

Optimized for time-predictability

Comes with a WCET analysis tool

Has its ‘own’ restricted real-time Java classes
- RtThread and SwEvent
- No scopes, just IM (and RT GC)

In use in academia and industry

Open-source
SCJ on JOP

- Add scope support
  - With a single Memory class
  - Presented at JTRES 2011 in York

- Scheduling – two options
  - On top of RtThread
  - Restructure to SCJ handlers

- RtThread
  - Used in some examples and industrial applications
  - Don’t want to drop the support
  - Don’t want to change industrial applications
  - Handler on RtThread has overheads
Threads and Handlers

- Move to SCJ handlers
  - More efficient
  - Is a ‘standard’
- Does this restrict the JOP ecosystem?
  - Can we still have plain single threaded Java apps?
- What about RTS with GC?
- Keep it all configurable
  - Nice concept, but might end up in a nightmare

- Current solution: on top of RtThread
SCJ Scheduler

- Priority preemptive
  - Standard RTOS scheduler
- JOP scheduler = SCJ scheduler
- Interrupt handler
  - Timer interrupt
  - Plain Runnable
- Looks ease – right?
Scheduler Implications

- IH created and registered at system start
  - No mission memory
  - Is in immortal
  - Static fields to find the thread list

- SCJ handlers
  - Event handlers created by a mission in mission memory
  - => How to point to those handlers?
  - Assignment issue

- The scheduler shall live in mission memory
Asynchronous Events

- From RTSJ: events and handlers
  - n:m mapping
  - Needs references in both directions
    - => in same scope
- Maybe too general for SCJ
- Simplify to a single class
  - Just the handler
  - Drop inheritance from BoundAsncEventHandler
  - Drop AperiodicEvent and AsyncEvent
  - Add release() to AperiodicEventHandler
Immortal Memory

- Convenient place for shared data
  - E.g. the console connection

- Cumbersome to allocate objects there
  - Might use class initializers
  - Executed at JVM/SCJ start
  - Need to find an order

- Add a method initialize() to Safelet
  - Executed in immortal before getNextMission()
A SCJ application is a Safelet
Start is vendor specific
  ◦ The SCJ implementation needs to create an object of a class that implements Safelet
  ◦ How is this info communicated?
  ◦ We need reflection for the creation
  ◦ The constructor needs to be no-arg

Why all this hassle?
Why not a plain static method (main())?
SCJ and a main() Method

```java
public static void main(String[] args) {
    JopSystem.startMission(new HelloSafelet());
}
```

- Initial thread is in immortal
- SCJ app object allocated in immortal
- RI on RTSJ initial thread is in heap
  - No issue for the start
    - Provide the RI main method as part of the implementation
    - Enter IM
    - Call the application/user main
Status, Source

- Implementation ok for examples
  - Example app in next session
  - Some parts are still missing
- Open Source
- Can be used without JOP
  - Run the JOP SW simulator (in Java)
  - No timing guarantees, but easy access
- Try it out and submit bug fixes ;-}
Conclusion

- Safety–Critical Java is here
- Prototype implementations emerge
  - SCJ on HVM
  - SCJ on JOP
- First test applications emerge
- Time to explore SCJ
  - Expressiveness
  - Easy to use – libraries
- Will it be a business?
  - Not yet fully commitment form commercial vendors
  - Is it just an academic toy?