Developing Safety Critical Applications in Java with oSCJ/L0

Ales Plsek, Lei Zhao, Veysel H. Sahin, Daniel Tang, Tomas Kalibera, Jan Vitek

www.omvj.net/oscj
oSCJ Overview

SCJ Library
- Level 0 support

SCJ-compliant VM
- RTEMS/LEON3 and x86 platforms

Tools
- Static Checker
- Technology Compatibility Kit (TCK)

Benchmark
- miniCDj benchmark
Safety Critical Systems

• is a system whose failure or malfunction may result in: death or serious injury to people, or loss or severe damage to equipment.

Software engineering challenge

• productivity, reusability, and availability of trained personnel

Certification standards

• DO-178 A, B, C and D
The Mission Concept

MissionSequencer
- getNextMission()
- creates MissionMemory
- runs in
- startAll()
- waitAll()

MissionManager
- setup
- teardown
- initialization
- execution
- cleanup
- run in
- startAll()
- waitAll()

MissionMemory
- setup
- teardown
- initialization
- execution
- cleanup
- run in

Mission
- setup
- teardown
- execution
- cleanup
- run in

Monday, August 16, 2010
Memory Model

- enterPrivateMemory()
- executeInArea()
- enterPrivateMemory()
- enterPrivateMemory()
- enter()

Mission Memory
- Mission Object
- PEH 1

Immortal Memory
Compliance Levels

Level 0

- single-threaded, Periodic Event Handlers, single Mission
- frame-based cyclic-execution model
- no synchronization support required

Level 1

- Aperiodic Event handlers, Fixed-Priority Preemptive Scheduler Periodic and Aperiodic Event Handlers

Level 2

- sub-missions, ManagedThreads
VM Interface

interface VM_Interface {
    Opaque makeExplicitArea (long size);
    Opaque makeArea (MemoryArea ma, long size);
    Opaque setCurrentArea(Opaque scope);
    Opaque getCurrentArea( );
    ...
    Opaque getCurrentTime{};
    long getClockResolution();
    ...
    int delayCurrentThreadAbsolute(long nanos);
}

- Library designed independently on the VM
- a dedicated interface for communication with the VM
- tested with 2 VMs - Ovm and FijiVM
SCJ VM Design

- based on OVM
  - a **metacircular** Virtual Machine
  - similarly as J9, FijiVM, Squawk VM, etc.
  - requires a bootstrap JVM to run upon to create a boot image.

- a small C loader is responsible for loading the boot image at runtime.

- Java code compiled down to C

- supported platforms: RTEMS/LEON3, x86
Memory Model Implementation

- **IM**
- **M1**
- P1
- P2
- P3

**Scope level**

**Top level**

**BS level**
Optimizations

Synchronization Support

- Level 0 - single threaded

Object Model

- optimized fields
- hash-code, GC information
- hash-code
- physical address of the object

Primordial Thread

- modified to be RealtimeThread
Evaluation

• Benchmark
  • miniCDj - periodic real-time task

• Hardware Platform
  • Xilinx FPGA GR-XC3S-1500 development board
  • 40 MHz, 8MB flash PROM, 64MB SDRAM, no FPU
  • RTEMS 4.9 OS

• SCJ configuration
  • no scope checks, miniCDj statically analyzed by SCJ Checker
Results - LEON3 board

Workload:
- 6 planes
- 120 milliseconds period
- 10 000 iterations

Results Summary

<table>
<thead>
<tr>
<th>Workload</th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDc</td>
<td>52.8</td>
<td>67.9</td>
</tr>
<tr>
<td>miniCDj</td>
<td>69.6</td>
<td>91.2</td>
</tr>
</tbody>
</table>

Overhead: 31% 34%
Results - x86

Workload:
60 planes
120 milliseconds period
10 000 iterations

Results Summary

<table>
<thead>
<tr>
<th></th>
<th>avg</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDc</td>
<td>0.39</td>
<td>0.49</td>
</tr>
<tr>
<td>miniCDj</td>
<td>0.34</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Overhead: -15% -4%
Conclusion

Implementation Summary

• SCJ library - 3 months
• SCJ VM
  • Memory model - 3 weeks
  • VM Optimizations - 2 weeks
  • porting VM to RTEMS - 2 months

oSCJ Distribution

• library, VM, tools and benchmark

• open-source at www.omvj.net/oscj

Monday, August 16, 2010