Ravenscar-Java: A High Integrity Profile for Real-Time Java

Jagun Kwon Andy Wellings Steve King

Presenter: Petur Olsen

September 20, 2007

High Integrity Real-Time Systems

What are they and why do we care?

- Expensive failures
- Loss of lives
- Environmental damages
- Financial loss

Real-Time

- Predictable and reliable to external events
- Adhere to deadlines
- Does not mean really fast

High Integrity Real-Time Systems

Traditionally

- Implemented on hardware
- Embedded systems
- Customized components
- Hardware specific software
- Poor reusability

High Integrity Real-Time Systems

Increase in use of software

- Increased flexibility
- Reduced production cost
- Enhanced complexity management
- Improved functionality
- Improved reusability

Java for Real-Time systems

The Java Programming Language

- Easy to learn
- Early (first) programming language
- Object oriented
- Industrial strength
- Platform independent
- Concurrent

Java for Real-Time systems

Disadvantages of Java

- Unpredictable performance
  - Scheduling
  - Memory
  - Control and data flow
- Automatic garbage collection
- Dynamic class loading

Contributions from Sun

Real-Time Specification for Java (RTSJ)

- Predictable execution
- Expressive Real-Time environment
- Complex virtual machine
- Difficult to analyze software

Java 2 Micro Edition (J2ME)

- Simple
- Runs on limited hardware
- Too restricted for RTSJ

The Solution

Ravenscar-Java

Ravenscar-Java

Key features

- Based on Ravenscar for Ada
- Reliable and predictable programming environment
- Analyzable and dependable systems
- Suitable for embedded systems

Ravenscar-Java

Development

- RTSJ
- Computational model
- Memory management
- Scheduling
- Control and data flow

Computational model

Focus on reliability

- No garbage collection
- Well defined scheduling
  - Threads and event handlers
  - Periodic and sporadic

Two phases

- Initialization phase
- Mission phase

Phases

Initialization phase

- Threads
- Memory areas and objects
- Event handlers
- Events
- Scheduling parameters
- (Compiling all load classes)

Mission phase

- Threads run and events are fired

Memory management

Memory types

- Immortal memory
  - Lives throughout the lifespan of the application
  - Allocation only in the initialization phase
- Linear time scoped memory
  - Limited lifetime
  - Allocation during the mission phase
  - Fixed maximum size
  - Not sharable

Scheduling

Threads

- java.lang.Thread is disallowed
- Periodic thread
- Sporadic event handler
- Static allocation

Restrictions

- Only fixed priority based scheduling
- No missed deadline handling

Control and data flow

Restrictions

- Ease the static analysis
- No *break* and *continue*
- One return statement
- No asynchronous transfer of control
- No *wait*, *notify* and *notifyall*

Example Program

Traction Controller

- Monitor wheels on car
- Cut power when wheels spin

Example Program

Traction Controller

- SporadicEventHandler powerCutHandler
- SporadicEvent powerCutEvent
- PeriodicThread spinMonitor
- Initializer TractionController
- Main

Ravenscar-Java

Advantages

- Real-time systems
- Static analysis
- Embedded systems
- On the paper a good profile

Conclusion

Disadvantages

- Class inheritance
- Analysis is seen as a separate process
- Parameters mixed with application logic

Thank you