Safety-Critical Java Level 2: Motivations, Example Applications and Issues

Andy Wellings    Matthew Luckcuck    Ana Cavalcanti

University of York

October 4, 2013
Outline

- Unique Features of Safety-Critical Java Level 2
- Uses of Level 2
- Issues with Level 2
- Conclusions
Unique Features of Level 2

Unique Features

- Nested Mission Sequencers
- ManagedThreads
- `Object.wait()` and `Object.notify()`

However...

- Level 2 has received little public attention
  - No Level 2 implementation
  - Very few example applications
Nested Mission Sequencers

- Mission Sequencers can be nested inside a Mission
- This nesting allows multiple Missions to run at once
  - One per Mission Sequencer
- Allows more complicated program architectures
  - Multi-Mode Applications
  - Independent Subsystems
Multi-Mode Applications

Overview

▶ Allows an application to change its functionality to suit the context

Components

▶ Modes: encapsulate all the concurrent activities needed to control the system during that mode
▶ Mode Changer: switches between different modes
Multi-Mode Applications – Architecture

Mode

- Mode: marker interface used to identify a Mode
- Modes are represented by Mission objects implementing Mode

Mode Changer

- Mode Changer: interface used to identify a Mode Changer
  - changeTo(Mode newMode)
  - advanceMode()
  - modeChangePending()
- Mode Changers are represented by MissionSequencers implementing ModeChanger
- Because the Mode Changer is a Mission Sequencer
  - Other Schedulable Objects may run during all modes
  - Mode Changes are handled automatically by the infrastructure, once a Mission is terminated
Spacecraft

- Three modes: Launch, Cruise, Land
- Each has its own specific concurrent activities
- There are also activities which run throughout all the modes:
  - Monitoring the craft’s environment
  - Handling the craft’s controls
Figure 1: Object Diagram showing the structure of the Spacecraft example application
Could Level 1 Do This?

- Yes, but...
  - Any concurrent activities that run over all Modes would require duplication...
    - ...which would require their state to be stored in Mission Memory
  - Could not be included in more complex systems
- If Level 2 is available it provides more flexibility
Independent Subsystems

Overview

- Allows an application to encapsulate and control disparate concerns into subsystems
- Especially useful if they are developed independently of each other

Components

- A Subsystem Module encapsulates the Schedulable Objects required by a subsystem
Independent Subsystems

Architecture

- A Subsystem Module is represented by:
  - A Mission Sequencer, which controls...
  - A Mission, which controls...
  - The Schedulable Objects for that subsystem

- Multiple Subsystem Modules are controlled by a Mission representing the application
Independent Subsystems – Example Application

Train Control (Hunt and Nilsen, 2012 [1])

- Rail network is divided into segments
- Train has to communicate with central authority to request authorisation to enter track segments
- Application contains four subsystems:
  - Communications
  - Navigation
  - Time
  - Train Controls
- Each of these is a Subsystem Module
- Communications and Navigation have their own subsystems
Independent Subsystems – Example Application Structure

Figure 2: Object Diagram showing the structure of the Train Control example application
Independent Subsystems – Could Level 1 Do This?

Could Level 1 Do This?

- Yes, but...
  - Schedulable Objects would all be contained by one Mission
- If Level 2 is available, it provides better encapsulation and control
Managed Threads and Wait/Notify

Overview

- ManagedThread has no release parameters, only a priority
- Object.wait() and Object.notify() provide simple suspension
- Allows the programming of paradigms unique, within SCJ, to Level 2
  - Unusual Release Patterns
  - Encapsulation of State
Unusual Release Patterns

- Adapting Managed Threads allows release patterns not available in Levels 0 or 1
  - Periodic Thread initially released by a software event
  - Producer-Consumer Threads
  - Run-as-Fast-as-Possible Threads
Periodic Thread

Overview

- Class PeriodicThread inherits from ManagedThread
- Modifies the run method:
  1. Blocks when run is entered
  2. Waits until its first release
  3. Then enters a loop which calls work()
  4. When work() returns then thread delays for its period
  5. Then the loop begins again, calling work
- work() now performs the function of the run() method in a standard thread
- Not available at Levels 0 or 1 due to lack of Object.wait() and Object.notify()
Periodic Thread

Wait Until First Release

```java
private synchronized boolean waitFirstRelease() {
    try {
        wait();
    }
    catch (InterruptedException ie) {
        return false;
    }
    return true;
}
```

Listing 1: The waitFirstRelease method of Periodic Thread
First Release

```java
public synchronized void firstRelease() {
    nextRelease = Clock.getRealtimeClock().getTime(nextRelease);
    nextDeadline.set(nextRelease.getTime(nextRelease) + deadline);
    deadlineMissDetection.scheduleNextReleaseTime(nextDeadline);
    notify();
}
```

Listing 2: The firstRelease method of Periodic Thread
public final void run()
{
    if (waitFirstRelease())
    {
        while (!myMission.terminationPending())
        {
            nextRelease.add(periodMilis, periodNanos);

            work();

            nextDeadline.add(periodMilis, periodNanos);
            deadlineMissDetection.
                scheduleNextReleaseTime(nextDeadline);
            // waitForNextPeriod
            Services.delay(nextRelease);
        }
    }
}
Producer-Consumer Threads

Overview

- Producers and Consumers which communicate via a bounded buffer
- Requires blocking
  - Producers block when the buffer is full
  - Consumers block when the buffer is empty

SCJ Level 2

- This cannot be done at Levels 0 or 1
  - `Object.wait()` and `Object.notify()` only available at Level 2
  - SCJ does not support a queue of outstanding release events for AperiodicEventHandlers
Run as Fast as Possible

Overview

- Low priority background activities
- No pattern of release
- Thread is descheduled and rescheduled as required
- Runs as fast as possible when it does have the processor

Example

- A Logging Thread
- Runs as fast as possible to log system activity in the background
Encapsulation of State

Overview

- Schedulable objects enter their memory area during their release and exit when they return from...
  - Handlers: `handleEvent()`
  - Threads: `run()`

- Managed Thread memory area is active for the length of `run()`

- Can be extended to suit the program’s needs:
  - Loop constructs
  - Blocking
Encapsulation of State

Managed Thread

- Managed Threads can be used to perform activities requiring state
- Handlers would require an outer memory area to be used
  - More visible than needed
- Managed Threads can store this state locally
  - Better encapsulation
Encapsulation of State

Temporary Private Memory Area

► If the Managed Thread allocates large amounts of memory
► We can make these allocations in a Temporary Private Memory Area
► Data needed for the next iteration must be allocated in the Managed Thread’s memory area
Encapsulation of State

```java
public final void run()
{
    if (waitFirstRelease()) {
        while (!myMission.terminationPending()) {
            nextRelease.add(periodMilis, periodNanos);
            ManagedMemory.enterPrivateMemory(
                getPrivateMemorySize(), runnableThatCallsWork);
            nextDeadline.add(periodMilis, periodNanos);
            deadlineMissDetection.scheduleNextReleaseTime(
                nextDeadline);
            // waitForNextPeriod
            Services.delay(nextRelease);
        }
    }
}
```

Listing 4: Periodic Thread run method with Private Memory
Encapsulation of State

Dealing With Scope

- The `work()` method is now executed in Private Memory
- Data that is not temporary must be explicitly allocated in the thread’s memory area:

```
1 PersistentData data = (PersistentData) threadMemory.newInstance(PersistentData.class);
```

Listing 5: Allocation in the Thread’s Memory Area
SCJ Level 2 Issues

- Schedulable Objects and Mission Termination
- Mission Sequencer Deadlines
- Further Support for Subsystems
Managed Schedulable Object Termination

- According to the SCJ language specification, during Mission termination the infrastructure will...
  - “... wait for all the Managed Schedulable Objects associated with this Mission to terminate”
- If a Managed Schedulable Object is blocked at this point, it will never terminate
SCJ Level 2 Issues

Mission Sequencer Deadlines

- Mode changes often have associated deadlines
- We suggest adding three methods to MissionSequencer
  - requestTerminationOfCurrentMission(AbsoluteTime deadline, AperiodicEventHandler deadlinMissHandler)
  - requestMissionChange(AbsoluteTime deadline, AperiodicEventHandler deadlinMissHandler)
  - getCurrentSequencer()
SCJ Level 2 Issues

Further Support for Subsystems

- Support for composing the timing constraints of Subsystems
- Two aspects of hierarchical scheduling needed:
  - Multi-level priorities
  - CPU budgets
Further Support for Subsystems

Priorities

- Desired outcome: when a Subsystem has the highest priority, all of the Schedulable Objects of that Subsystem will run
- In SCJ a two-level priority scheme is needed
  - A Mission Sequencer is given a priority
  - Each Managed Schedulable Object is given a priority
- Ensure all Managed Schedulable Objects have a priority...
  - Greater than or equal to the priority of their Mission Sequencer and...
  - Less than the priority of the Mission Sequencer with the next highest priority
Further Support for Subsystems

Budgets

▶ Desired Outcome: Managed Schedulable Objects to run only when their Subsystem has remaining budget
▶ RTSJ can support this with Processing Group Parameters (PGP)
  ▶ If all Schedulable Objects are running on one processor
▶ Support in SCJ could come from an extension implementing...

```
1 public class ProcessingGroupParameters {
2
3     public ProcessingGroupParameters (  
4         HighResolutionTime start,  
5         RelativeTime period,  
6         RelativeTime budget)  
7     ...  }
```

Listing 6: Proposed Processing Group Parameters Object
Further Support for Subsystems

- ... allowing SCJ to track simple budgets
- Constructors could be added to the Mission Sequencer to accept PGP and an integer to bound the priority range of the Managed Schedulable Objects
- But...
  - Requires SCJ to be extended to honour these budgets
  - Still requires Managed Schedulable Objects to run on a single processor
Conclusion

- SCJ Level 2 has received little public attention
- Clear from the SCJ specification what constitutes a Level 2 application
- Far from clear when SCJ Level 2 should be used
Conclusion

Ups

► We have examined the unique features of Level 2 and found them to be useful
  ► Control
  ► Complexity Management
  ► Encapsulation

Downs

► Deficiencies in Level 2 features
  ► Termination of blocked Schedulable Objects during the termination of Missions
  ► Deadlines on Mission transition
  ► Further Support for Subsystems
Questions?
Hunt, J., and Nilsen, K.
Safety-critical java: The mission approach.