SSTC 2009

Safety Critical Java in a Mission and Safety Critical Environment

Johan Olmütz Nielsen

DDC-I

Safety Critical Software Solutions for Mission Critical Systems

Safety Critical Java

- Are you kidding?

- What about GC?
- What about JVM?
- What about JIT?
- What about dynamic class loading?

DDC-I

Safety Critical Java

- Why?

- Mature language
- Full OO
- Precise semantics
- Analysis tools
- Pool of programmers

DDC-I

Safety Critical Java

Objective

- Enable creation of safety-critical applications amenable to certification under DO-178B Level A

DDC-I

Requirements

Mission & Safety Critical Applications need

- Real-time responses
- Known worst-case resource consumption
- Source traceability
- Schedulability analysis
- Analysis tools

DDC-I

Safety Critical Java – How

Foundation

- RTSJ 1.1
  - Real-time response
  - Memory areas
  - PCE
- Java 5.0
  - Annotations for analysis
- Java ME
  - Limited API

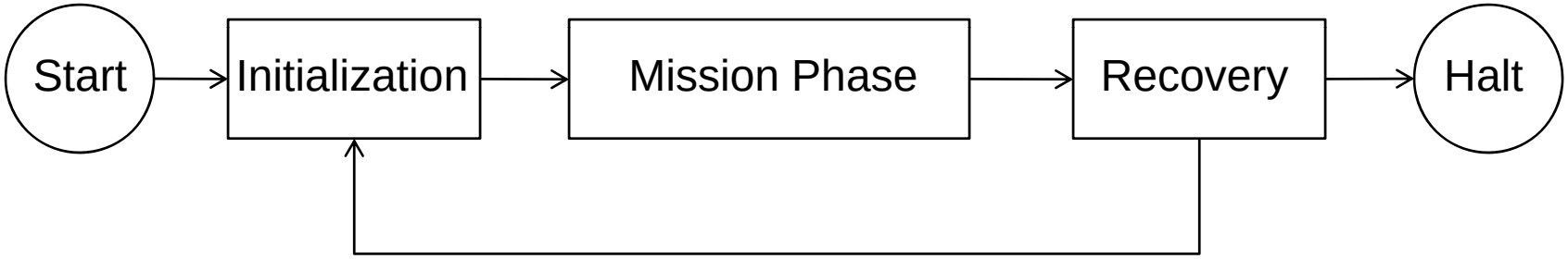
DDC-I

Programming Model

- Safelet runs application
- Mission concept
- Three compliance levels
- Limited API
- Absence of memory reference errors

DDC-I

Mission Concept



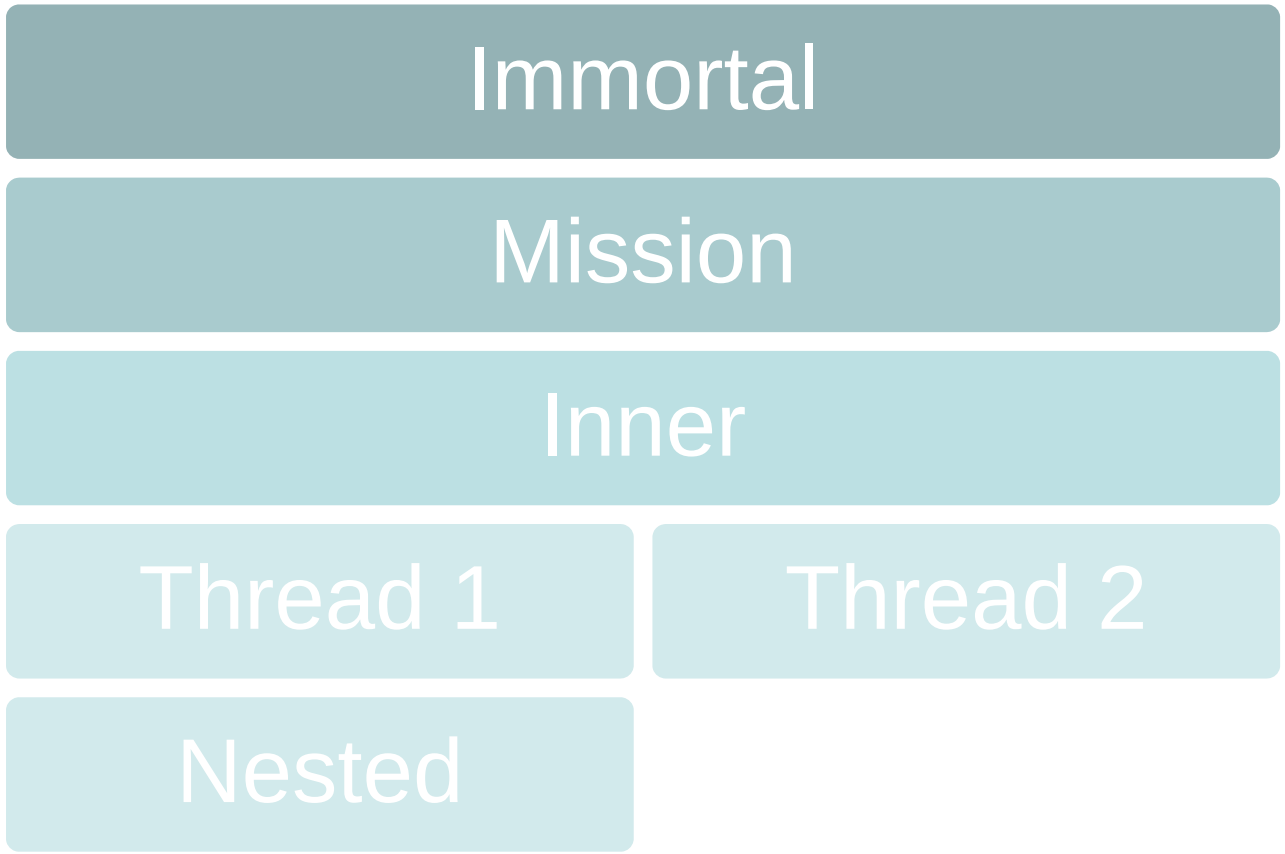
Memory Areas

- Only Immortal & Scoped RTSJ memory permitted
- No heap use
- Fixed reclamation point
- Consequence: GC not required

Scoped Memory

- Simple nesting of Scoped memory
- Only entered from creating area
- Simplifies analysis
  - Worst case memory consumption
  - Freedom from exceptions

Scoped Memory (Level 2)



Scope Annotations

RTSJ relies on dynamic checks

- IllegalAccessException
- ScopedCycleException
- InaccessibleAreaException

SCJ intents to enable static analysis

- @ScopeDef
- @Scope
- @Nested
- @Outer

DDC-I

Scope Example

```
@ScopeDef(name="MyScope", parent="Immortal")
class MyScope extends LTMemory { … }

@Scope("MyScope")
class MyClass { … }

@Outer("Immortal") @Nested("MyScope")
class Runner implements Runnable {
    MyScope scope;
    public void invoke() { scope.executeInArea(this); }
    public void run() { … }
}
```

DDC-I

Compliance Levels

Level 0 Cyclic Executive

- Single thread of execution
- Periodic AEHs each with one memory area

Level 1 Single Mission

- Concurrent periodic & aperiodic AEHs
- Multiple non-shared memory areas per AEH

Level 2 Nested Missions

- Periodic & aperiodic AEHs and NHRTs
- Dynamically created missions

DDC-I

Limited API

SCJ provides

- java.lang
- java.util
- javax.realtime

but only partially:

- many classes have been removed
- methods have been removed
- classes made immutable

DDC-I

Traceability

- Scope annotations to avoid memory exceptions
- Whole program analysis to eliminate
  - Array bounds & divide-by-zero checks
  - Required but unreachable catches
- No reflection
- Limited JNI

DDC-I

Safelet Example

```
public class Level1App implements Safelet {
    public MissionSequencer getSequencer() {
        return new ASequencer(new PriorityParameters(10));
    }
    public void setup() {}
    public void teardown() {}
}
```

DDC-I

MissionSequencer Example

```
public class ASequencer extends MissionSequencer {
    private Mission _initial; private Mission _next;
    ASequencer(PriorityParameters pp) {
        super(pp);
        _initial = new RealMission();
        _next = new CleanupMission();
    }
    protected Mission getInitialMission() { return _initial; }
    protected Mission getNextMission() { return _next; }
}
```

DDC-I

Mission Example

```
public class RealMission extends Mission {
    public void initialize() {
        RelativeTime start = new RelativeTime(0,0);
        new PeriodicEH
            (“AEH A”, start, new RelativeTime(5,0));
        new PeriodicEH
            (“AEH B”, start, new RelativeTime(10,0));
    }
    public long missionMemorySize() { return 1000L; }
}
```

DDC-I

EventHandler Example

```
public class PeriodicEH extends PeriodicEventHandler {
    private int _counter;
    public PeriodicEH(String nm, RelativeTime start,
                      RelativeTime period) {
        super(new PriorityParameters(15),
              new PeriodicParameters(start, period), 100);
        _counter = 0;
    }
    public void handleEvent() { ++_counter; }
    public void cleanup() {}
}
```

DDC-I

Deployment step 1

Prove SCJ compliance

- For given SCJ Level
- Using SCJ API
- Verifying annotations
- No dynamic class loading

DDC-I

Deployment step 2

Compile to native code

- Whole program analysis
- Non-varying code, no JIT
- No JVM

DDC-I

Schedulability

- Periodic and aperiodic AEHs
- No sporadic AEHs
- Priority Ceiling Emulation required
  - Beyond RTSJ
  - Efficient
  - Contributes to make program deadlock free

Safety Critical Java – Results

- No kidding
  - No GC, JIT, JVM, dynamic class loading
- Known resource consumption
- Schedulability analyzable
- Assists
  - Source traceability
  - Tools

DDC-I

Safety Critical Java – When?

- JSR 302 working since 2006
- Finalizing draft specification Summer 2009
  - Meeting in London next week
- Draft becomes available at www.jcp.org
- Stay tuned

DDC-I

Acronyms

AEH	Aperiodic Event Handler
GC	Garbage Collector or Garbage Collection
JCP	Java Community Process
JIT	Just In Time (compiler)
JNI	Java Native Interface
JSR	Java Specification Request
JVM	Java Virtual Machine
NHRT	NoHeapRealtimeThread
PCE	Priority Ceiling Emulation
RTSJ	Real-Time Specification for Java
SCJ	Safety Critical Java