Safety-Critical Java Level 2 Programs: Application, Modelling, and Verification

Matt Luckcuck

Supervisors:
Ana Cavalcanti and Andy Wellings

4th of August 2016

Outline

Outline

Java in Safety-Critical Systems

Java

- Java not traditionally associated with safety-critical programs
- More abstraction, less control...
  - Garbage collection
  - Poor scheduling control

"The intrinsic safety of the standard language is irrelevant, **it is how safe the use of the language can be made that matters**" – Hatton *Safer C (1995)*

Java in Safety-Critical Systems

Java

- Interesting for safety-critical systems:
    - Strong typing
    - Precise definition
    - Widely understood
    - Language features e.g. exception handling
- Long standing effort to improve Java. . .
    - Java Community Process's Java Specification Requests (JSR)

Java in Safety-Critical Systems

Real-Time Specification for Java (RTSJ)

- Java Community Process: JSR 1
- RTSJ addresses some of the Java's problems...
  - Region-based memory
  - Control memory usage
  - Better scheduling control
- Complex for safety-critical programs

Safety-Critical Java

SCJ Overview

- International effort lead by The Open Group
- Java Community Process: JSR 302
- Builds on RTSJ
- Aimed at applications that must be certified
- Embeds a new, simpler programming paradigm
- $\sim$ 112 pages of language specification...
  - $\sim$ 36 classes and interfaces
  - Does not cover verification

Safety-Critical Java

SCJ Overview

- Requires a real-time virtual machine
- Real-time abstractions from the RTSJ
- Restricted hierarchical programming structure
- Region-based hierarchical memory
- Fixed priority scheduler with Priority Ceiling Emulation

Safety-Critical Java

Tools

- SCJ has specific tools for...
  - Memory Safety
  - Worst-Case Memory Consumption
  - Worst-Case Execution Time
  - Schedulability
  - Program Verification

Safety-Critical Java

Compliance Levels

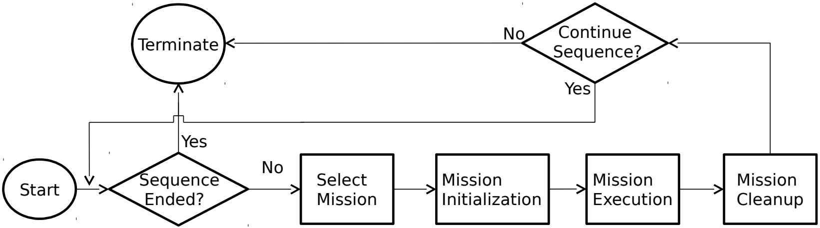
- Level 0:
  - Single processor
  - Cyclic executive
- Level 1:
  - Introduce concurrency
  - More release patterns
- Level 2:
  - Highly concurrent
  - Multi-processor
  - Complicated release patterns
  - Suspension

Safety-Critical Java

SCJ API

- `Safelet`: controls the program and starts the Mission Sequencer
- `MissionSequencer`: instantiates and starts a sequence of Missions
- `Mission`: controls a set of tasks, represented by subclasses of Managed Schedulable
- `ManagedSchedulable`: super-type of all four tasks:
  - `PeriodicEventHandler`
  - `AperiodicEventHandler`
  - `OneShotEventHandler`
  - `ManagedThread`

Safety-Critical Java



Mission Phases

1. Initialize: creates and registers schedulables
2. Execute: simultaneously activate mission's schedulables
3. Cleanup: reset data structures

SCJ Level 2

SCJ Level 2 Features

- Access to suspension features
- Access to all Managed Schedulables. . .
  - Uniquely: `ManagedThread` and `MissionSequencer`
- Schedulable Mission Sequencers allow multiple Missions to be active. . .
  - One active Mission per Mission Sequencer
  - Schedulables from any active Mission may preempt, based on their priorities
  - No assumption of schedulable from a particular mission having priority

Thesis Statement

The paradigm embedded in Safety-Critical Java (SCJ) Level 2 provides features that have useful applications that Levels 0 and 1 are not capable of programming. Further, the Level 2 paradigm can be formally modelled using a language that captures state and behaviour, to show that neither the SCJ infrastructure nor a valid SCJ program present undesirable program states such as deadlock, divergence, or nondeterminism.

Safety-Critical Java Level 2 Utility

Overview

- When to use Level 2 was not obvious:
  - Level 0: Cyclic-Executive and Periodic Tasks
  - Levels 1 and 2: Concurrency and Fixed-Priority Scheduling
- Managed Threads provide a new release pattern...
- But what are Managed Threads Useful for?
- What are the other unique features useful for?

Safety-Critical Java Level 2 Utility

Schedulable Mission Sequencers

- Complex Program Architectures:
- Multiple-Mode Applications...
  - Allows application to change behaviour to suit context
- Independently-Developed Subsystems...
  - Composes programs of subsystems
- Better encapsulation and more control than Level 1

Managed Threads and Suspension

- Producer-Consumer Systems
- Extend SCJ features:
  - E.g extended release patterns
- Level 1 cannot be capture any of these features

Safety-Critical Java Level 2 Utility

Level 2 Problems. . .

- Investigation also uncovered some Level 2 problems
- Proposed solutions to these
- Model used to propose simpler termination protocol
  - Accepted into the specification
- Termination in Level 2. . .

Safety-Critical Java Level 2 Utility

Termination in Level 2

- According to the language specification, during Mission termination the infrastructure will...
  - '... wait for all the Managed Schedulable Objects associated with this Mission to terminate'
- If a schedulable is blocked at this point, it will never terminate
- Language specification now contains specific guidance for termination in Level 2:
  - Manually interrupt potentially blocked threads, using the `signalTermination()` method (called on each schedulable during termination)
  - Does not automatically solve the problem, but provides a uniform way of handling custom termination behaviour

Modelling Approach

This work...

- Models the SCJ Level 2 paradigm using *Circus*
- Agnostic of Java
- Limited treatment of some Exceptions
- First formal semantics of SCJ Level 2
- Builds on a model of SCJ Level 1[1]...
  - Level 2 features
  - API changes
- Model ignores...
  - Scheduling
  - Resources (E.g. Memory)

[1] Zeyda et al. *Circus Models for Safety-Critical Java Programs*. Computer Journal 2014

Model Benefits

Top-Down

Target for refinement-based development of SCJ programs[2]

- Refinement from abstract to concrete specifications...
  - Concrete specifications that capture the SCJ paradigm
- Correctness by construction

Bottom-Up

Translation from SCJ code to model

- Catches certain program errors...
  - Deadlock
  - Divergence
  - Exceptions

[2] Cavalcanti, Sampaio, and Woodcock. *A Refinement Strategy for Circus*. Formal Aspects of Computing. Nov 2003

Modelling Approach

Circus Language

- Combination of **Z** and **CSP**
  - Captures both State and Behaviour
- Organised around Processes
  - State component (**Z**) to hold variables
  - Actions (**Z** and **CSP**) to perform behaviours
  - Main action specifies overall behaviour
- Communication through **CSP** channels

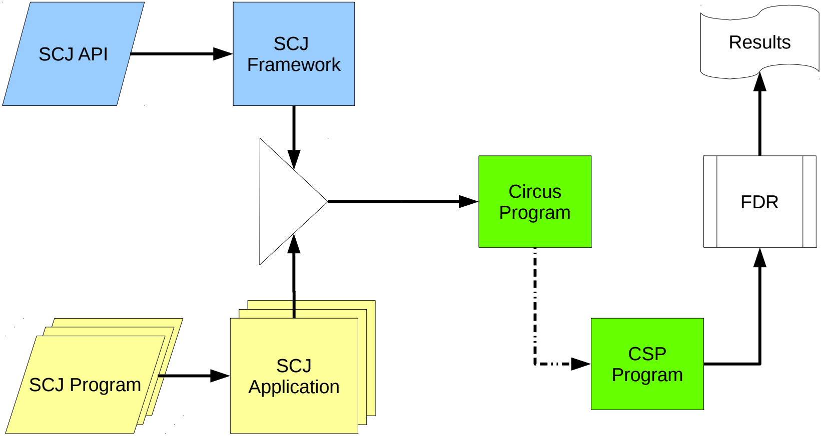
Circus Family

Circus Variants

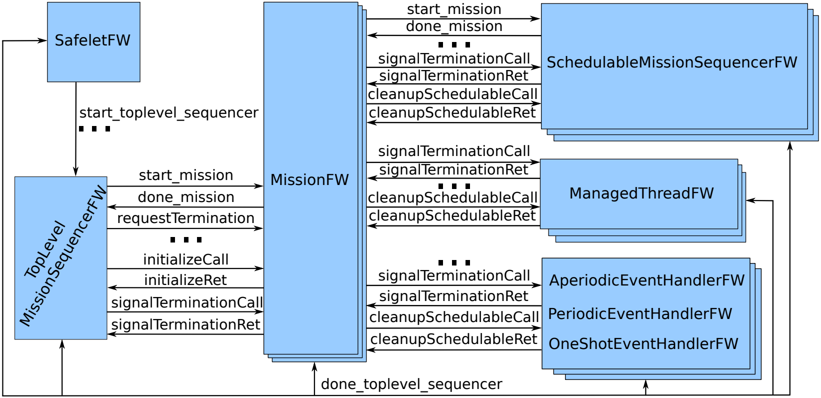
Our model also uses features from other members of the *Circus* family

- Oh*Circus*. . .
  - Classes based on Java classes
  - Inheritance
  - Used to model simple data objects
- *Circus* Time
  - Notion of (relative) time
  - Used for delays and deadlines

Modelling Approach



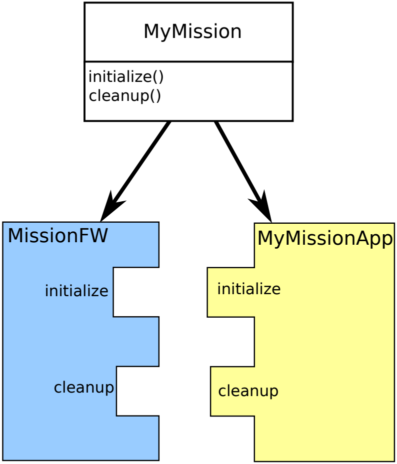
Modelling Approach



Modelling Approach

Framework:

- Generic
- API classes



Application:

- Specific
- Program behaviour

Modelling Extras

Exceptions

- Modelled by an event followed by **Chaos**
    - Built-in process that diverges
- Only for paradigm misuse by an application
- API Coverage:
    - Thread interrupt
    - Incorrect method parameter
    - Suspension without a lock
    - Locking an object with a lower priority
    - Registering schedulable twice

Modelling Extras

Synchronisation and Suspension

- Extra processes in the Framework model capture the synchronisation and suspension behaviour
- Separate because this behaviour is not needed for all objects

Inheritance

- The Method Call Binder process interfaces between the processes in the Application model
- It binds the events representing method calls to the location of the method
- Handles inherited methods and simplifies translation

Translation

Overview

- Informal translation strategy
  - Captures SCJ programs
  - Validation of model
- Formalisation of core elements in Z
- TightRope automatic generation of models from code
- Occurs in Three Phases:
  - Analyse
  - Build
  - Generate

Analyse

Analyse

- Identifies all the components of the program
- Records things like variable types
- TightRope compiles the program to generate ASTs
  - No annotations needed, unlike Level 1 tool

Build

Build

- Build an environment for each class by extracting details from the program and translating them to *Circus*
- Safelet Environment...
  - Name
  - `initialize()`
  - `getSequencer()`
  - (Application-Specific) Fields and Methods

Generate

Generate

- Generates the output files using the environments
- Each component has a template that its application models conform to
- We drop the information from the environments into the gaps in the template
- TightRope uses the FreeMarker template engine to automate this

Program Analysis

Overview

- We want to be able to use this model to verify programs. . .
- But there is no model checker for *Circus*
- So, we use industry-proven CSP model checker FDR3. . .
- But, this requires another translation.

Why Use *Circus*?

Useful integration of state and behaviour

Program Analysis

Circus to `CSPm`

- `CSPm` is the machine-readable version of CSP, used by FDR
- Informal translation from *Circus* to `CSPm`
    - State in *Circus* processes becomes state process in `CSPm`
    - Most behaviour in *Circus* translates straight into `CSPm`
- However, treatment of state in initial translations produced intractable models. . .

Program Analysis

Hi Matthew,

You've currently got three large processes running on csresearch0:

PID	Command	Memory	Started
6880	refines NMSTModuleAssertion.csp	132001.36 MB	Nov11
11766	fdr3 NestedMissionSequencerTest.csp	66205.74 MB	Nov04
35014	fdr3 NestedMissionSequencerTest.csp	32800.37 MB	Nov10

(see also attached).

These have exhausted available memory (128GB) and swap space (100GB) so are impacting on the general availability of the server.

Can I kill any of these processes, or are they likely to complete any time soon?

Program Analysis

Hi Matthew,

Looks like your processes are at it again!

PID	Command	Memory
27411	refines Application.csp	225997.07 MB

If it's likely to complete soon then absolutely leave it running.

However, I might have to configure a limit to per-process memory use as the disk swapping means the server gets pretty sluggish! Will your processes still run with e.g. a 64GB memory limit?

Program Analysis

Circus to `CSPm`

- Improved the `CSPm` model with the help of Tom Gibson-Robinson (FDR3 maintainer) at Oxford University

Animation and Model Checking in FDR3

- Animate the Framework to compare to SCJ API and running programs
- Model Check the program specifications to ensure deadlock- and divergence-freedom
  - We can also construct custom checks: exceptions, particular program behaviours, etc
- Model validity

Evaluation

Model

- Close correspondence with the SCJ API
- Builds on the Level 1 model...
    - Level 1 model has been validated against the API
- Our modelling effort simplified SCJ's termination protocol...
    - Adopted in v0.96

Evaluation

Translation

- Informal translation strategy, which provides semantics to our model
- 10 hand-translated examples covering different release patterns, synchronisation, and schedulable mission sequencers
- TightRope, produces models from code:
  - Producers–Consumers 6 classes ∼1.2 seconds
  - Aircraft 25 classes ∼2.3 seconds
- . . . Some expression rewriting required
- Core elements of translation formalised in Z
- Further work on both of these is engineering

Evaluation

Program Analysis

- Informal translation to CSP, for FDR3
  - *Circus* is close to CSP
  - Scope for automation
- Recent tool that automates *Circus* to `CSPm` translation[3]
  - But, this may have similar problems with scope, when tested on models with larger state

[3] Beg and Butterfield. *Development of a Prototype Translator from Circus to CSPM*. ICOSST 2015

Summary and Further Work

Contributions

1. First examination of the utility of the features of SCJ Level 2,
2. First formal model of the SCJ Level 2 API, and
3. A strategy to translate SCJ Level 2 programs into our model.

Future Work

- Fully formalise translation strategy
- Reduce restrictions on TightRope input
- Automate *Circus* to `CSPm` translation

SCJ Level 2 Utility . . .

- Luckcuck, Wellings, and Cavalcanti. *'Safety-Critical Java: Level 2 in Practice'*. Concurrency and Computation: Practice and Experience, [Accepted].
- Wellings, Luckcuck, and Cavalcanti. *'Safety-Critical Java Level 2: Motivations, Example Applications, and Issues'*. JTRES 2013

Thank you for listening

Model. . .

- Luckcuck. *'A Formal Model for the SCJ Level 2 Paradigm'*. Doctoral Symposium of Formal Methods 2015
- Luckcuck, Cavalcanti, and Wellings. *'A Formal Model of the Safety-Critical Java Level 2 Paradigm'*. iFM 2016