Modelling Safety-Critical Java Level 2 in Circus

Matthew Luckcuck

University of York

3rd June 2014
Outline

- Introduction
  - Safety-Critical Java Level 2
  - Circus
  - Context of Work
  - Modelling Challenges
- Developing the Model
- Summary and Further Work
Introduction

Aims
- Produce a model of the SCJ Level 2 paradigm
- Devise a formal translation strategy to convert SCJ Level 2 programs to this model
Modelling SCJ Level 2 in Circus

SCJ Level 2

SCJ Level 2 Features

- Concurrent Missions with concurrent Managed Schedulables
- Level 1 Managed Schedulables: PeriodicEventHandler, AperiodicEventHandler, OneShotEventHandler
- Level 2 Managed Schedulables: ManagedThread, MissionSequencer
- Access to Object.wait and Object.notify
Mission Sequencers as Schedulables

- Mission Sequencers may be nested inside Missions
- Nested Mission Sequencers allow multiple Missions to be active.
  - One active Mission per Mission Sequencer
  - Managed Schedulable Objects from any running Mission may preempt, based on their priorities
  - No assumption of Schedulable Objects from a particular mission having priority
SCJ Level 2

Figure 1: Possible Structure of a Level 2 Program
SCJ Level 2

Spacecraft Example

- Three modes: Launch, Cruise, Land
- Each has its own specific Schedulable Objects
- There are also Schedulable Objects which run throughout all the modes...
  - Monitoring the craft’s environment
  - Handling the craft’s controls
Figure 2: Object Diagram of the Spacecraft example application
Modelling SCJ Level 2 in Circus

Circus Family

**Circus Language**

- Combination of **Z** and **CSP**
  - Captures both State and Behaviour
- Organised around Processes
  - Similar to Java classes
  - State component (**Z**) to hold variables
  - Actions (free combination of **Z** and **CSP**) to perform behaviours
  - Main action to specify the overall behaviour of the process
  - Communication through **CSP** channels
    - `channel → A`
    - `channel.parameter → A`
Our model also uses features from other members of the Circus family

- **OhCircus**...
  - Classes based on Java classes
  - Inheritance
  - Used to model simple data objects in the Application Model

- **Circus Time**
  - Notion of (relative) time
  - `wait t`
  - `channel@time → A`
Modelling SCJ Level 2 in *Circus*

---

### Circus Family

#### However...  
- Model checker for *Circus* still in development  
- Leads to converting *Circus* specifications into **Z** and **CSP**...  
  - using ProB to check the **Z**  
  - using FDR to check the **CSP**  
- Causes obvious overheads

#### Why Use *Circus*?  
- Previous work using *Circus* and Java/SCJ...  
  - Existing model of SCJ Level 1  
- Refinement-based development
Modelling SCJ Level 2 in Circus

Context of Work

- **Abstract Specification**
  - Circus Time
  - A anchor

- **Execution Architecture** (Missions & Handlers)
  - Circus Time + OhCircus
  - O anchor

- **Refinement Laws**
  - Circus Time + OhCircus
  - E anchor

- **Detailed Design**
  - Circus Time + OhCircus
  - S anchor

- **Diagrams DSLs**
  - Provided by the software engineer

- **Data refinement**

- **Verification**

- **SCJ Program Model**
  - SCJCircus

- **SCJ Program**
  - SCJ-compliant JDK library

- **P model**
  - Application Model
  - Framework Model
  - Library Model

- **SCJ Library**
- **SCJ JVM/API**
Modelling SCJ Level 2 in *Circus*

## Context of Work

### Top-Down

Target for refinement-based development of SCJ programs
- Refinement from abstract to concrete specifications...
  - Concrete specifications that capture the SCJ paradigm
- Correctness by construction

### Bottom-Up

Allows translation from SCJ code to model
- Catches certain program errors...
  - Deadlock
  - Divergence
  - Some exceptions
- Will not catch memory errors
Modelling Challenges

SCJ Challenges
- Changing or untested language specification
- Complexity of the unique features of Level 2
- No complete Level 2 implementation...
  - Use RTSJ to simulate SCJ structure
  - ‘Flatten’ programs with nested missions to test them using a Level 1 implementation

Circus Challenges
- Model checker still in development so convert to CSP...
  - Feature set does not match that of Circus
  - Modelling state becomes complicated
    - Large state process to model variables
    - Allows remote access to a process' state
Developing the Model

## Approach

- Based on current Level 1 model
- Separate model of SCJ into...
  - **Framework**, which captures the infrastructure classes
  - **Application**, which captures application-specific code using...
    - *Circus* processes
    - *OhCircus* classes
- Translation strategy to capture the application-specific information and output the Application model
- Tool to automate this translation
Developing the Model

Figure 3: High-Level Framework and Application Models
Developing the Model

Coverage

- Model ignores...
  - Priorities
  - Resources (E.g. Memory)
- Model Captures...
  - Behaviour and State of Objects
  - Limited treatment of some Exceptions
- Exceptions...
  - Causes Chaos in the specification
    - Built-in process that diverges
  - Memory Exceptions not covered
Developing the Model

Object model

- Each object is modelled by up to four components...
  - **Object**: concerns derived from Object
    - If the class is used as a lock
  - **Thread**: identifies the thread of control
    - If the class has a controlling thread
  - **Framework**: concerns of the SCJ class being extended
  - **Application**: program-specific information
- Composed to form one process
  - Parametrised with an ID
- If a class has two instances, each has its own model
Developing the Model

Figure 4: Potential components of an Object’s Model
Developing the Model

**Communication**

- Free communication across each model and between Framework and Application models
- Components communicate with components that are...  
  - Of a different type
    - Querying concrete information
    - E.g. Mission Sequencer Framework communicating with the Mission Sequencer Application to get the next mission
  - Of the same type
    - Objects communicating with other Objects
    - E.g. Mission Sequencer Framework calling the Mission Framework's Initialize action
Developing the Model

Figure 5: Framework Processes structure
Developing the Model

Model

- Framework model in Figure 5 remains the same for each program
- Application model is similar but is generated afresh for each program
- Translation strategy only needs to extract application-specific information from the program
Summary

- Model SCJ Level 2 paradigm as **Framework** and **Application** combination
- Model of SCJ Level 2 contributes to . . .
  - **Top-down** development as a refinement target
  - **Bottom-up** development as verification tool

Further Work

- Devise Application model
- Translation strategy to convert application code to Application model
- Tool to automate translation
Thank you for listening.
Any Questions?