Circus Models of Safety-Critical Java

Frank Zeyda

University of York, UK

First hiJaC Workshop, 15 Nov 2011
Introduction

The hiJaC Project

- **High-Integrity Java Applications using Circus**
- Development of formalisms, techniques and tools to produce formally verified Safety-Critical Java (SCJ) programs.
- *Circus* family of languages is used to define a semantic theory.
  → Standard *Circus* + *OhCircus* + *Circus* Time
- Pursue refinement-based techniques. (*Correct-by-Construction*)

From a practical perspective...

- Construct a correct SCJ program for a given specification.
- Verify that a given SCJ program is correct.
- Make use of design patterns, tools, and automation.

All three possibilities are entailed by our approach.
Introduction

The hiJaC Project

- High-Integrity Java Applications using Circus
- Development of formalisms, techniques and tools to produce formally verified Safety-Critical Java (SCJ) programs.
- Circus family of languages is used to define a semantic theory.
  → Standard Circus + OhCircus + Circus Time
- Pursue refinement-based techniques. (Correct-by-Construction)

From a practical perspective...

- Construct a correct SCJ program for a given specification.
- Verify that a given SCJ program is correct.
- Make use of design patterns, tools, and automation.

All three possibilities are entailed by our approach.
The model of the program is the subject of this seminar.
The model of the program is the subject of this seminar.
Objectives of this Work Stage

Specification of the Program Model (*P model*)
- Define a semantics for the infrastructure (*framework model*).
- Define a semantics for SCJ programs (*application model*).
- Determine the *Circus* variant needed to express those models.

Requirements
- Specify primary components of SCJ: *Safelet*, *Mission Sequencer*, *Missions*, and *Event Handlers*.
- Encode the mission-based execution paradigm (*Level 1*).
- Capture timing aspects where necessary.
- Be faithful to the *SCJ Technology Specification* (JSR 302).

**Design goal**: Make the framework model independent of the application.
Objectives of this Work Stage

Specification of the Program Model (**P model**)

- Define a semantics for the infrastructure (**framework model**).
- Define a semantics for SCJ programs (**application model**).
- Determine the **Circus** variant needed to express those models.

Requirements

- Specify primary components of SCJ: **Safelet**, **Mission Sequencer**, **Missions**, and **Event Handlers**.
- Encode the mission-based execution paradigm (**Level 1**).
- Capture timing aspects where necessary.
- Be faithful to the **SCJ Technology Specification** (**JSR 302**).

**Design goal**: Make the framework model independent of the application.
Remainder of the Presentation

1. SCJ Paradigm
2. P Model: Overview
3. P Model: Details
4. Validation and Tools
5. Conclusions and Future work
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
**Summary of Level 1**

- An application consists of a number of **missions**.
- Missions are executed sequentially.
- Missions create **event handlers** during initialisation.
- SCJ supports **periodic** and **aperiodic** event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
SCJ Programming Model

Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
An application consists of a number of missions.
Missions are executed sequentially.
Missions create event handlers during initialisation.
SCJ supports periodic and aperiodic event handlers.
Termination can be initiated by a handler at any time.
Summary of Level 1

- An application consists of a number of missions.
- Missions are executed sequentially.
- Missions create event handlers during initialisation.
- SCJ supports periodic and aperiodic event handlers.
- Termination can be initiated by a handler at any time.
The Safelet Life-cycle

1. setUp()
2. getSequencer()
3. getNextMission()
4. initialize()
5. Execute
6. requestTermination()
7. cleanup()
8. tearDown()

Safelet
- Creates Aperiodic Handlers
- Creates Periodic Handlers

Mission Sequencer
- Mission
- Aperiodic Handlers
- Periodic Handlers

Circus Models of SCJ
The Safelet Life-cycle

Infrastructure calls `setUp()` to initialise the safelet.
The Safelet Life-cycle

Infrastructure calls `getSequencer()` to obtain the mission sequencer.
The Safelet Life-cycle

1. setUp()
2. getSequencer()
3. getNextMission()
4. initialize()
5. Execute
6. requestTermination()
7. cleanup()
8. tearDown()

Infrastructure calls getNextMission() to obtain the first mission.

Circus Models of SCJ
The Safelet Life-cycle

1. setUp()
2. getSequencer()
3. getNextMission()
4. initialize()
5. Execute
6. requestTermination()
7. cleanup()
8. tearDown()

Aperiodic Handlers
Periodic Handlers

Infrastructure calls initialize() to create handlers and other data.
The Safelet Life-cycle

Execution phase of the mission: handlers become active.
The Safelet Life-cycle

One of the handlers calls `requestTermination()` to initiate
The Safelet Life-cycle

Infrastructure calls `cleanup()` after handlers have been stopped.
The Safelet Life-cycle

Infrastructure calls `getNextMission()` to obtain the next mission.
The Safelet Life-cycle

If it returns `null`, `tearDown()` is called and the safelet terminates.
SCJ Program: Automotive Cruise Controller

«interface»
Safelet

MissionSequencer
+setUp() : void
+tearDown() : void
+getSequencer() : MissionSequencer

MainMissionSequencer

Mission
+initialize() : void
+fini() : void
+requestTermination() : void
+terminationPending() : boolean
+missionMemorySize() : long

MainMission

Controller
+engineOn() : void
+engineOff() : void
+brakeEngaged() : void
+brakeDisengaged() : void
+topGearEngaged() : void
+topGearDisengaged() : void
+activate() : void
+deactivate() : void
+startAccelerate() : void
+stopAccelerate() : void
+resume() : void

AperiodicEventHandler
+handleAsyncEvent() : void
+register() : void

PeriodicEventHandler
+handleAsyncEvent() : void
+register() : void

WheelShaft
Engine
Gear
Brake
Lever

SpeedMonitor

ThrottleController

Circus Models of SCJ 8/20
**SCJ Program: Automotive Cruise Controller**

**MissionSequencer**

- `+setUp() : void`
- `+tearDown() : void`
- `+getSequencer() : MissionSequencer`

**Mission**

- `+initialize() : void`
- `+cleanup() : void`
- `+requestTermination() : void`
- `+terminationPending() : boolean`
- `+missionMemorySize() : long`

**MainMissionSequencer**

**MainMission**

**Controller**

- `+engineOn() : void`
- `+engineOff() : void`
- `+brakeEngaged() : void`
- `+brakeDisengaged() : void`
- `+topGearEngaged() : void`
- `+topGearDisengaged() : void`
- `+activate() : void`
- `+deactivate() : void`
- `+startAccelerate() : void`
- `+stopAccelerate() : void`
- `+resume() : void`

**Mission**

**Safelet**

- `{interface} Safelet`
  - `+setUp() : void`
  - `+tearDown() : void`
  - `+getSequencer() : MissionSequencer`

**MainSafelet**

**SpeedMonitor**

**ThrottleController**

**AperiodicEventHandler**

- `+handleAsyncEvent() : void`
- `+register() : void`

**PeriodicEventHandler**

- `+handleAsyncEvent() : void`
- `+register() : void`

**WheelShaft**

**Engine**

**Gear**

**Brake**

**Lever**

**Device Access**

**External Environment**

- Shaft Sensor
- Engine Sensor
- Gear Sensor
- Brake Sensor
- Lever Sensor
- Throttle Actuator

**Call Controller**

**External Event Raised**

**Circus Models of SCJ**
SCJ Program: Automotive Cruise Controller

«interface»
Safelet

+setUp : void
+tearDown : void
+getSequencer() : MissionSequencer

MissionSequencer

+getNextMission() : Mission

MainMissionSequencer

Mission

+initialize() : void
+cleanup() : void
+requestTermination() : void
+terminationPending() : boolean
+missionMemorySize() : long

ThrottleController

+engineOn() : void
+engineOff() : void
+brakeEngaged() : void
+brakeDisengaged() : void
+topGearEngaged() : void
+topGearDisengaged() : void
+activate() : void
+deactivate() : void
+startAccelerate() : void
+stopAccelerate() : void
+resume() : void

Controller

AperiodicEventHandler

+handleAsyncEvent() : void
+register() : void

PeriodicEventHandler

+handleAsyncEvent() : void
+register() : void

WheelShaft Engine Gear Brake Lever

SpeedMonitor

MainSafelet

External Environment

Shaft Sensor Engine Sensor Gear Sensor Brake Sensor Lever Sensor Throttle Actuator

Device Access

Circus Models of SCJ
SCJ Program: Automotive Cruise Controller

- **Safelet**
  - `setUp() : void`
  - `tearDown() : void`
  - `getSequencer() : MissionSequencer`

- **MissionSequencer**
  - `getNextMission() : Mission`

- **Mission**
  - `initialize() : void`
  - `cleanup() : void`
  - `requestTermination() : void`
  - `terminationPending() : boolean`
  - `missionMemorySize() : long`

- **MainMissionSequencer**

- **MainSafelet**

- **SpeedMonitor**

- **ThrottleController**
  - `engineOn() : void`
  - `engineOff() : void`
  - `brakeEngaged() : void`
  - `brakeDisengaged() : void`
  - `topGearEngaged() : void`
  - `topGearDisengaged() : void`
  - `activate() : void`
  - `deactivate() : void`
  - `startAccelerate() : void`
  - `stopAccelerate() : void`
  - `resume() : void`

- **AperiodicEventHandler**
  - `handleAsyncEvent() : void`
  - `register() : void`

- **PeriodicEventHandler**
  - `handleAsyncEvent() : void`
  - `register() : void`

- **Controller**
  - Created in Mission Memory

- **Release Aperiodic Handlers**

- **Device Access**

- **External Environment**
  - Shaft Sensor
  - Engine Sensor
  - Gear Sensor
  - Brake Sensor
  - Lever Sensor
  - Throttle Actuator
SCJ Program: Automotive Cruise Controller

Discharge Throttle Control

+setUp() : void
+tearDown() : void
+getSequencer() : MissionSequencer
+getNextMission() : Mission
+initialize() : void
+cleanup() : void
+requestTermination() : void
+terminationPending() : boolean
+missionMemorySize() : long
+engineOn() : void
+engineOff() : void
+brakeEngaged() : void
+brakeDisengaged() : void
+topGearEngaged() : void
+topGearDisengaged() : void
+activate() : void
+deactivate() : void
+startAccelerate() : void
+stopAccelerate() : void
+resume() : void

AperiodicEventHandler

ThrottleController

SpeedMonitor

MainMissionSequencer

MainSafelet

Controller

Mission

Disable Throttle Control

Created in Mission Memory

Release Aperiodic Handlers

Device Access

External Environment

Shaft Sensor

Engine Sensor

Gear Sensor

Brake Sensor

Lever Sensor

Throttle Actuator

Circus Models of SCJ
SCJ Program: Automotive Cruise Controller

- **Safelet**
  - `setUp() : void`
  - `tearDown() : void`
  - `getSequencer() : MissionSequencer`

- **MissionSequencer**
  - `getNextMission() : Mission`

- **MainMissionSequencer**

- **Mission**
  - `initialize() : void`
  - `cleanup() : void`
  - `requestTermination() : void`
  - `terminationPending() : boolean`
  - `missionMemorySize() : long`

- **MainMission**

- **Controller**
  - `engineOn() : void`
  - `engineOff() : void`
  - `brakeEngaged() : void`
  - `brakeDisengaged() : void`
  - `topGearEngaged() : void`
  - `topGearDisengaged() : void`
  - `activate() : void`
  - `deactivate() : void`
  - `startAccelerate() : void`
  - `stopAccelerate() : void`
  - `resume() : void`

- **AperiodicEventHandler**
  - `handleAsyncEvent() : void`
  - `register() : void`

- **PeriodicEventHandler**
  - `handleAsyncEvent() : void`
  - `register() : void`

- **WheelShaft**
  - **Engine**
  - **Gear**
  - **Brake**
  - **Lever**

- **SpeedMonitor**

- **ThrottleController**

- **Call Controller**

- **Release Aperiodic Handlers**

- **Device Access**

- **External Environment**
  - Shaft Sensor
  - Engine Sensor
  - Gear Sensor
  - Brake Sensor
  - Lever Sensor
  - Throttle Actuator

- **Created in Mission Memory**
Structure of the *Circus* Model

System Process

| ... | / | ... | = Circus Process | ... | = OhCircus Class |

![Diagram of Circus Model]

- **MainSafeletApp**
- **MainMissionSequncerApp**
- **MainMissionApp**

**Method Channels**
- `enter dispatch.h`
- `leave dispatch.h`
- `done handler.h`
- `end sequential.app`

**ControllerClass**

**EngineApp**

**EngineClass**

- `engine on`
- `engine off`

- **ThrottleControllerApp**
  - **ThrottleControllerClass**
  - `set voltage`

**External Environment (Interface)**

**EngineApp on MainMissionApp**

- creates data objects during `initialize()`
Framework Process

Application Processes

Circus Models of SCJ
Model of the Safelet

Framework Process

process SafeletFW ≜ begin
    SetUp ≜ setUpCall → setUpRet → skip
    Execute ≜ start_sequencer → done_sequencer → skip
    TearDown ≜ tearDownCall → tearDownRet → skip
    • SetUp; Execute; TearDown; end_safelet_app → skip
end

Application Process

process MainSafeletApp ≜ begin
    setUpMeth ≜ setUpCall → skip; setUpRet → skip
    tearDownMeth ≜ tearDownCall → skip; tearDownRet → skip
    Methods ≜ μX • (setUpMeth □ tearDownMeth); X
    • Methods △ end_safelet_app → skip
end

Observation: Application processes exhibit a uniform pattern!
Model of the Safelet

Framework Process

process SafeletFW ≜ begin
  SetUp ≜ setUpCall → setUpRet → skip
  Execute ≜ start_sequencer → done_sequencer → skip
  TearDown ≜ tearDownCall → tearDownRet → skip
  • SetUp ; Execute ; TearDown ; end_safelet_app → skip
end

Application Process

process MainSafeletApp ≜ begin
  setUpMeth ≜ setUpCall → skip ; setUpRet → skip
  tearDownMeth ≜ tearDownCall → skip ; tearDownRet → skip
  Methods ≜ µX • (setUpMeth □ tearDownMeth) ; X
  • Methods △ end_safelet_app → skip
end

Observation: Application processes exhibit a uniform pattern!
Model of Event Handlers

Framework Process (one per handler)

\[
\text{process } \text{EventHandlerFW} \triangleq \text{handler : HandlerId} \bullet \text{begin}
\]
\[
\text{state } \text{State} \triangleq [\text{active : BOOL}]
\]
\[
\text{Init} \triangleq [\text{State}' \mid \text{active}' = \text{FALSE}]
\]
\[
\text{StartHandler} \triangleq \text{start\_handler \_handler} \rightarrow \text{active} := \text{TRUE}
\]
\[
\text{ActivateHandlers} \triangleq \text{activate\_handlers} \rightarrow \text{skip}
\]
\[
\text{DispatchHandler} \triangleq \text{enter\_dispatch \_handler} \rightarrow
\]
\[
\text{stop\_handler \_handler} \rightarrow \text{leave\_dispatch \_handler} \rightarrow \text{skip}
\]
\[
\bullet \left( \mu X \bullet \text{Init} ; \right) \left( ((\text{StartHandler} ; \text{ActivateHandlers}) \Box \text{ActivateHandlers}) ;
\]
\[
\text{if active} = \text{TRUE} \rightarrow \text{DispatchHandler}
\]
\[
\text{fi}
\]
\[
\text{active} = \text{FALSE} \rightarrow \text{skip}
\]
\[
\triangle \text{end\_mission\_fw} \rightarrow \text{skip}
\]
end

Requirement of the Infrastructure

- Asynchronously start handlers (\textit{start\_handler}).
- Synchronously activate handlers (\textit{activate\_handlers}).
public class Engine extends AperiodicEventHandler {
    private CruiseControl cruise;

    public Engine(CruiseControl cruise, ...) {
        super(...);
        this.cruise = cruise;
    }

    public void handleAsyncLongEvent(long evt) {
        switch (evt) {
            case ENGINE_ON:
                cruise.engineOn();
                break;

            case ENGINE_OFF:
                cruise.engineOff();
                break;
        }
    }
}
Model of Event Handlers (cont’d)

Application Class

class EngineClass ≜ begin
  state EngineState ≜ [private cruise : CruiseControlClass]
  initial EngineInit ≜ val cruise? : CruiseControlClass •
  public handleAsyncEvent ≜ val evt : EventId •
    if evt = EngineOnEvtId −→ cruise.engineOn()
    if evt = EngineOffEvtId −→ cruise.engineOff()
  fi
end

Application Process

process EngineApp ≜ begin
  state EngineState ≜ [obj : EngineClass] . . .
  Dispatch ≜ μX •
    leave_dispatch, EngineId −→ skip
    □ engine_on −→ obj.handleAsyncEvent(EngineOnEvtId)
    □ engine_off −→ obj.handleAsyncEvent(EngineOffEvtId)
    wait 0..EngineDeadline
    ; X
  )
)
Aperiodic vs Periodic Handlers

Aperiodic Event Handlers

- Released by external events bound to the handler.
- Modelled by external choice in Dispatch:
  \[ \text{evt}1 \rightarrow \text{handleAsyncEvent}(	ext{Evt1Id}) \]
  \[ \text{evt}2 \rightarrow \text{handleAsyncEvent}(	ext{Evt2Id}) \]

Periodic Event Handlers

- Released by an internal framework event (release_handler).
- Modelled by a parallel action:
  \[ \text{Dispatch} \left[ \{ \text{release}_\text{handler} \} \right] \text{Release} \]

where the \text{Release} action is defined as

\[ \text{Release} \triangleq \mu X \bullet \left( \text{release}_\text{handler} \rightarrow \text{skip} \triangleright 0; \right. \]  \[ \left. \text{wait ThrottleCtrlPeriod} ; X \right) \triangle \ldots \]

But both types of handlers share the same framework model.
Aperiodic vs Periodic Handlers

Aperiodic Event Handlers

- Released by external events bound to the handler.
- Modelled by external choice in *Dispatch*:
  
  \[
  \text{evt1} \rightarrow \text{handleAsyncEvent}(\text{Evt1Id}) \quad \square
  \]
  
  \[
  \text{evt2} \rightarrow \text{handleAsyncEvent}(\text{Evt2Id}) \quad \square \quad \ldots
  \]

Periodic Event Handlers

- Released by an internal framework event (*release_handler*).
- Modelled by a parallel action:

  \[
  \text{Dispatch} \left[ \{ \text{release\_handler} \} \right] \text{ Release}
  \]

  where the *Release* action is defined as

  \[
  \text{Release} \triangleq \mu X \bullet \left( \text{release\_handler} \rightarrow \text{skip} \triangleright 0; \quad \text{wait} \ ThrottleCtrlPeriod \ ; \ X \right) \triangle \ldots
  \]

But both types of handlers share the same framework model.
Validation of Models

Community Z Tools (CZT)
- Open development platform for Z tools.
- Parsing and type-checking support for Circus.
- We extended this to include Circus Time operators.
- Our models type-check modulo some new notations.

Model-checking in FDR
- Part of the model was translated into CSP.
- Circus State is encapsulated in parametrised processes.
- Established absence of deadlocks.
- Established process termination (helped us fix initial glitches).

Current validation aims at applying the approach to larger case studies.
The hiJaC Tools

- General-purpose tool components for SCJ.
- Include parsing and analysis facilities for SCJ programs.
- A *proto-type translator* is under development.
- Will support *fully automatic* generation of the P model.
- We also include various case studies and their models.

Open access at [https://svn.cs.york.ac.uk/anonsvn/hijactools](https://svn.cs.york.ac.uk/anonsvn/hijactools).
Conclusion

Primary Achievements

- We formalised the mission execution model of Level 1 SCJ.
- Clean separation between framework and application models.
- We capture timing as well as object-oriented designs.

Our models...

1. Enforce clean and sound design principles (!!)
2. Highlight the **SCJ Programming Paradigm**.
3. Evoke a new language **SCJ-Circus** (*work in progress*).
5. Open pathways for formal development techniques.

We did not consider yet...

- Semantic foundations of an integrated *Circus* language for SCJ.
Future Work

Three main areas of future work are.

1. Formalisation of the Translation Strategy
   - Define the precise subset of Java/SCJ that we admit.
   - Compositional translation rules for all elements of the syntax.

2. Automatic Tool for Translation
   - The hiJaC Tools suite provides the framework.
   - But more work to do before we have a robust translator.
   - Potential opportunities for student projects.

3. Industrial-size Case Studies
   - CDx, jPapaBench, . . . → Input from industry would be very useful.

→ What other features of the infrastructure do we need to support?
Future Work

Three main areas of future work are.

1. Formalisation of the Translation Strategy
   - Define the precise subset of Java/SCJ that we admit.
   - Compositional translation rules for all elements of the syntax.

2. Automatic Tool for Translation
   - The hiJaC Tools suite provides the framework.
   - But more work to do before we have a robust translator.
   - Potential opportunities for student projects.

3. Industrial-size Case Studies
   - CDx, jPapaBench, ... → Input from industry would be very useful.
   → What other features of the infrastructure do we need to support?
Thanks for your attention.

Questions?