Safety-Critical Java in *Circus*

Ana Cavalcanti
Frank Zeyda, Andy Wellings, Jim Woodcock, Kun Wei

University of York, hiJaC project

FME AGM, March 2013

Overview

- Safety-Critical Java
- *Circus* family
- Refinement strategy
- Applications
- Current and future work

Safety-Critical Java

- International effort lead by the Open Group
- Performed under the Java Community Process
- Based on the Real-Time Specification for Java
  - A Safety-Critical Java Specification
  - A reference implementation
  - A technology compatibility kit
- Goal: certification
- Levels: 0, 1, 2

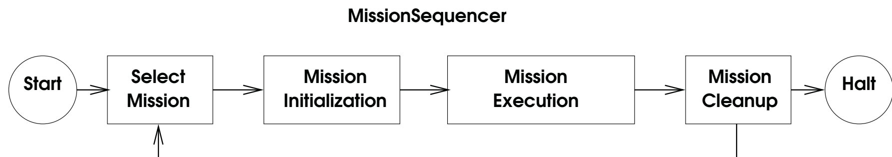
Nothing about design techniques

Our goals

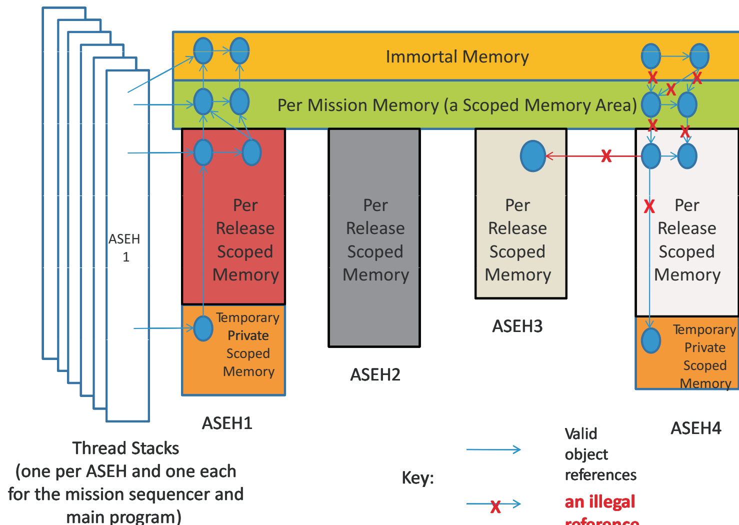
Development technique for SCJ

- Issues involved in the formal verification of an SCJ program
- Guidelines when adopting a refinement-based approach
- Based on the *Circus* family: Z, CSP, Timed CSP, object-orientation
- Timing requirements and their decomposition
- Value-based specification and class-based designs
- SCJ memory model

Application structure



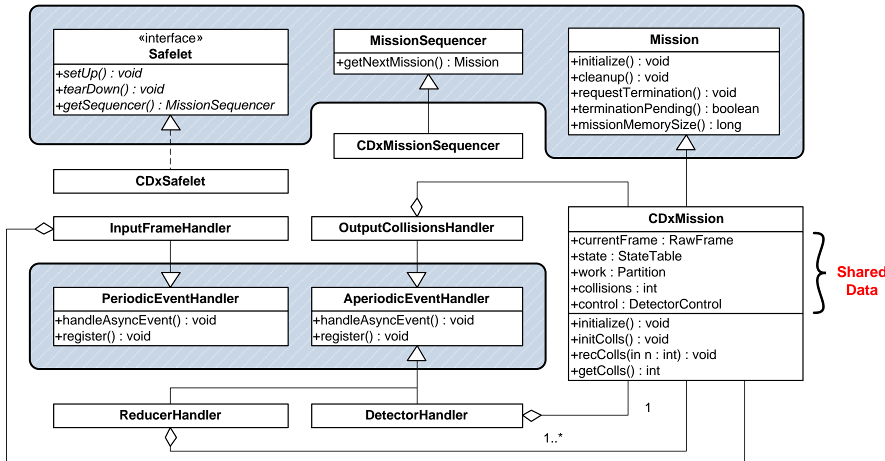
Scoped memory area



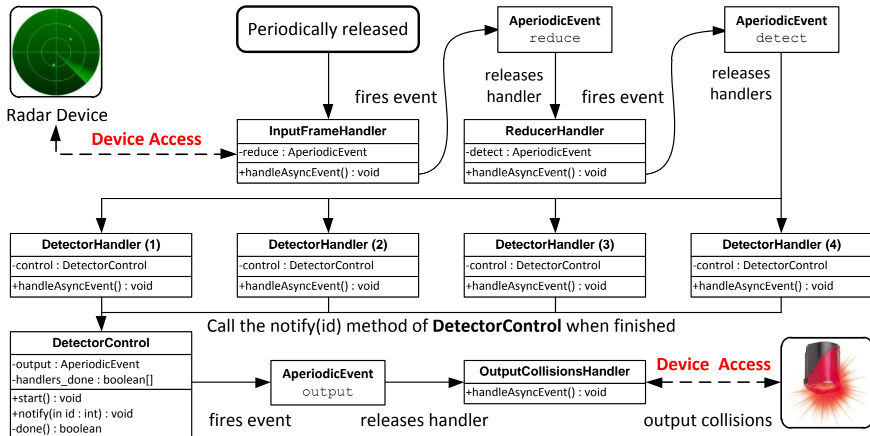
An example: collision detector

- detects potential collision of aircraft in a radar
- benchmark in the SCJ community
- Our implementation: Level 1
- 27 classes
- 3000 lines of code
- Seven handlers
- Shared data
- Barrier mechanism

An example: collision detector



An example: collision detector



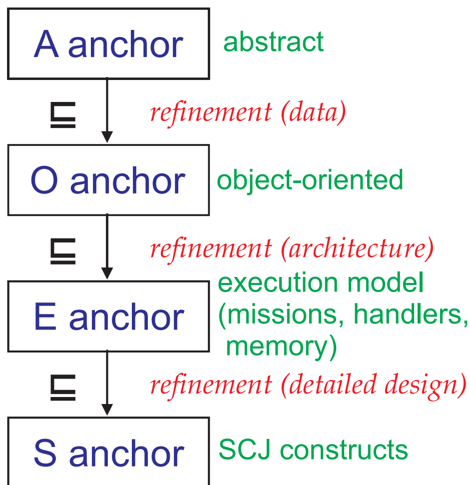
Circus Family

- *Circus*: Z + CSP + ZRC
- Language for refinement
- Target programming languages: occam, Handel-C, SPARK Ada
- Processes: encapsulate state + behaviour
  - State: Z
  - Actions: CSP + Z + guarded command language
  - Communication: through channels
- Semantic model: Unifying Theories of Programming

Circus variants

- *Circus Time*
- *OhCircus*
- ...

Development of SCJ programs: our approach



Development of SCJ programs: our approach

- Each anchor: different member of the *Circus* family
- *SCJ-Circus*
  - Automatic code generation
  - The SCJ programming paradigm
  - SCJ infrastructure and applications
- Modelling and refinement patterns
- Timing
  - deadlines and budgets enforced by the components
  - machine independent
  - schedulability analysis

Development of SCJ programs: our approach

Structure

- Three steps
- Each step is divided in phases
- Each phase: stages captured by refinement laws
    - Novel specific laws
    - Choice depending on target design

A Anchor

- *Circus* and *Circus Time*
- No classes, objects, or references
- Patterns for timing requirements

$$\textbf{system } CDx \mathrel{\widehat{=}} ABReqsCDx \llbracket \{\!| \ldots |\!\} \rrbracket ATReqsCDx$$

- Periodic and sporadic tasks, and input and output jitters

$$TReq \mathrel{\widehat{=}} (A(\textbf{wait}\, 0 \mathrel{..} b) \blacktriangleright d \mathrel{|||} \textbf{wait}\, p) \mathbin{;} TReq$$

where $b \leq d \leq p$

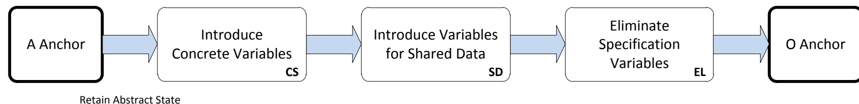
O Anchor: structure and step

Structure

- Language: *OhCircus* with references + *Circus Time*
- Classes and objects
- Same of the A anchor

Step

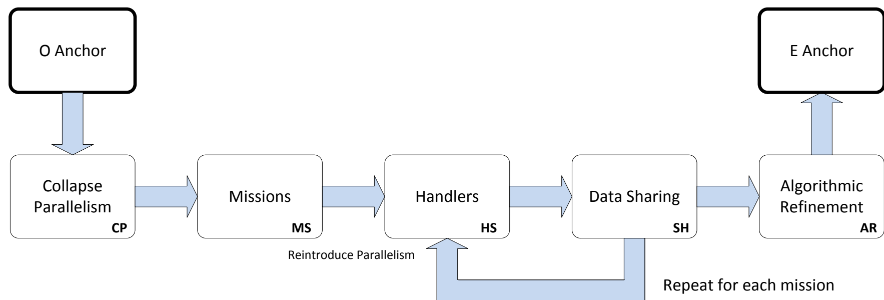
- Data refinement
- Concrete and shared data



E Anchor: structure

$$
\begin{array}{l}
\textbf{process } \mathit{SCJsystem} \mathrel{\widehat=} \textbf{begin} \\
\quad \textbf{state } \mathit{SCJstate} == [x, y, z : \ldots \mid \ldots] \\
\quad \mathit{Init} \mathrel{\widehat=} \ldots \\
\quad \mathit{Handler1} \mathrel{\widehat=} \ldots \textbf{var } a, b, c \bullet \ldots \\
\quad \mathit{Handler2} \mathrel{\widehat=} \ldots \\
\quad \ldots \\
\quad \mathit{InitM1} \mathrel{\widehat=} \ldots \\
\quad \mathit{HandlersM1} \mathrel{\widehat=} (\mathit{Handler1} \parallel \mathit{Handler2} \parallel \ldots) \setminus \mathit{swevts} \\
\quad \mathit{MArea1} \mathrel{\widehat=} \textbf{var } l, m, n \ldots \\
\quad \mathit{Mission1} \mathrel{\widehat=} \\
\qquad \mathit{InitM1} \; ; \; (\mathit{HandlersM1} \llbracket \mathit{ns} \mid \mathit{mcs} \mid \{\} \rrbracket \mathit{MArea1}) \setminus \mathit{mcs} \\
\quad \ldots \\
\quad \mathit{System} \mathrel{\widehat=} \mathit{Mission1} \; ; \; \mathit{Mission2} \; ; \; \ldots \\
\quad \bullet \mathit{Init}; \; \mathit{System} \\
\textbf{end}
\end{array}
$$

E Anchor: step



E anchor: HS phase

Target: for each mission, one parallel action

- the handler actions for its handlers;
- *HandlerController* to control their interaction;
- *Cycle* to capture timing requirements for a cyclic mission.

$$Mission \mathrel{\widehat=} Handlers \parallel HandlerController \parallel Cycle$$

E anchor: HS phase

1. Definition of cycle timings.
2. Decomposition of data operations that are implemented across different handlers.
3. Distribution of time budget to sequential components.
4. Transformation of sequential data operations into parallel handler actions.
5. Transformation of parallel data operations into parallel handler actions.
6. Extraction of the handlers.

E anchor: HS phase

1. patterns: Definition of cycle timings.
2. Decomposition of data operations that are implemented across different handlers.
3. Distribution of time budget to sequential components.
4. Transformation of sequential data operations into parallel handler actions.
5. Transformation of parallel data operations into parallel handler actions.
6. Extraction of the handlers.

E anchor: SH Phase

Target

- Immortal memory: stay where they are
- Mission memory: become local to a new *MArea* action
- Per-release and temporary areas: remain or become local to the handler action

Stages

1. Encapsulate shared data of sequential handlers
2. Encapsulate shared data of concurrent handlers
3. Introduce data to realise control mechanisms
4. Collect specification of the memory area data

E anchor: SH Phase

Target

- Immortal memory: stay where they are
- Mission memory: become local to a new *MArea* action
- Per-release and temporary areas: remain or become local to the handler action

Stages

1. Encapsulate shared data of sequential handlers
2. patterns: Encapsulate shared data of concurrent handlers
3. patterns: Introduce data to realise control mechanisms
4. Collect specification of the memory area data

Example law seq-share-1: for refinement in Stage (1)

$$\left(\begin{array}{l}(\mu X \bullet A_1\ ;\ c\,!\,e \longrightarrow end \longrightarrow X)\\ [\![ns_1 \mid cs \mid ns_2]\!]\\ (\mu X \bullet c\,?\,x \longrightarrow A_2\ ;\ end \longrightarrow X)\end{array}\right) \setminus \{\!|\ c\ |\!\}$$

$$=$$

$$\left(\begin{array}{l}\left(\begin{array}{l}(\mu X \bullet A_1\ ;\ c_1\,!\,e \longrightarrow c_3 \longrightarrow end \longrightarrow X)\\ [\![ns_1 \mid (cs \setminus \{\!|\ c\ |\!\}) \cup \{\!|\ c_3\ |\!\} \mid ns_2]\!]\\ (\mu X \bullet c_3 \longrightarrow c_2\,?\,x \longrightarrow A_2\ ;\ end \longrightarrow X)\end{array}\right) \setminus \{\!|\ c_3\ |\!\}\\ [\![ns_1 \cup ns_2 \mid \{\!|\ c_1, c_2\ |\!\} \mid \varnothing]\!]\\ \mathbf{var}\ v : T \bullet \mu X \bullet (c_1\,?\,x \longrightarrow v := x \;\Box\; c_2\,!\,v \longrightarrow \mathbf{skip})\ ;\ X\end{array}\right) \setminus \{\!|\ c_1, c_2\ |\!\}$$

provided $\{\!|\ c, end\ |\!\} \subseteq cs$; $c \notin usedC(A_1) \cup usedC(A_2)$; and c_1, c_2 and c_3 are fresh channels.

S Anchor

SCJ framework

- Language: *SCJ-Circus*
- Abbreviations
- Underlying: same language + SCJ memory model
- Refinement laws for new constructs

New paragraphs

- safelet
- mission sequencer
- mission
- handler

S Anchor: step

Split the E anchor: examples

- state components ⟶ **safelet** paragraph
- *Init* action ⟶ safelet **setUp** paragraph
- sequences of missions ⟶ **sequencer** paragraph
- each *Mission* action ⟶ a **mission** paragraph
- each *Handler* action ⟶ a **periodic** / **aperiodic** paragraph

Refinement laws of *SCJ-Circus*

- safe use of memory
- but no concern for resources

S Anchor: example

$$
\begin{array}{l}
\textbf{sequencer}\ \mathit{MainMissionSequencer} \mathrel{\widehat=} \textbf{begin} \\
\textbf{state}\ \ \mathit{MainMissionSequencerState} == [\mathit{mission_done} : \mathit{bool}] \\
\textbf{initial} \mathrel{\widehat=} \mathit{mission_done} := \mathit{false} \\
\textbf{getNextMission} \mathrel{\widehat=} \\
\quad \textbf{if}\ \ \mathit{mission_done} = \mathit{false} \longrightarrow \\
\quad \mathit{mission_done} := \mathit{true};\ \textbf{ret} := \mathit{MissionCDx} \\
\quad [\!]\ \mathit{mission_done} = \mathit{true} \longrightarrow \textbf{ret} := \mathit{null} \\
\quad \textbf{fi} \\
\textbf{end}
\end{array}
$$

S Anchor: example

$\textbf{mission}\ \mathit{MissionCDx} \mathrel{\widehat=} \textbf{begin}$

$\textbf{state}\ \mathit{MissionCDxState}$

$\mathit{currentFrame} : \textbf{ref}\ \mathit{RawFrame}$

$\ldots$

$\textbf{initial} \mathrel{\widehat=}$

$\mathit{currentFrame} := \textbf{newM}\ \mathit{RawFrame}\ ;\ \ldots$

$\textbf{initialize} \mathrel{\widehat=}$

$\textbf{cleanup} \mathrel{\widehat=} \textbf{skip}$

$\mathit{MArea} \mathrel{\widehat=} \ldots$

$\textbf{end}$

S Anchor: example

$\mathbf{periodic}(FRAME_PERIOD)\ \mathbf{handler}\ InputFrameHandler \mathrel{\widehat=}$

$\mathbf{begin}$

state *InputFrameHandlerState*

$mission : Mission$

$reduce : AperiodicEvent$

$\mathbf{initial}\ \ InputFrameHandlerInit(m : Mission, \ldots) \mathrel{\widehat=} \ldots$

$\mathbf{handleAsyncEvent} \mathrel{\widehat=} \ldots$

$StoreFrame \mathrel{\widehat=} \ldots$ "emerges from handler refinement."

$\mathbf{dispatch\ handleAsyncEvent}$

$\mathbf{end}$

Automation

Challenges

- Data refinement
- Catalogue of useful patterns and associated laws
- Detailed algorithms

Required information: parameters of the design

- Number of missions
- For each mission
  - Number of handlers
  - Events and data operations controlled by each of them
- Allocation of data in memory areas

S Anchor: applications

- Use *Circus* and the UTP for reasoning
- Automatic generation of SCJ programs
- Conversely: automatic generation of S models
  - Programming patterns
  - Refactoring
  - Examples?
- Identification of good programming practices?
- Basis to verify an SCJ implementation

Final considerations

- SCJ: a programming paradigm and a Java implementation
- *SCJ-Circus*: the novel paradigm
- Theory of programming: in development
- Our contribution: general refinement technique
- More specialised techniques are needed
  - Automatic generation of models
  - Patterns
  - Assertions
  - Memory safety
  - ...

Challenges ahead

Theory

- Integration of theories
- Mechanisation
- Modular reasoning about libraries

Practice

- Case studies
- Design patterns

And beyond

- Certification, Resources, ...