Safety-Critical Java in *Circus*

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

University of York

JTRES, September 2011

Overview

Refinement technique for SCJ

- 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

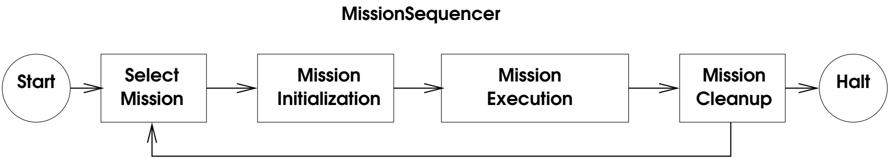
hiJaC project

Safety-Critical Java

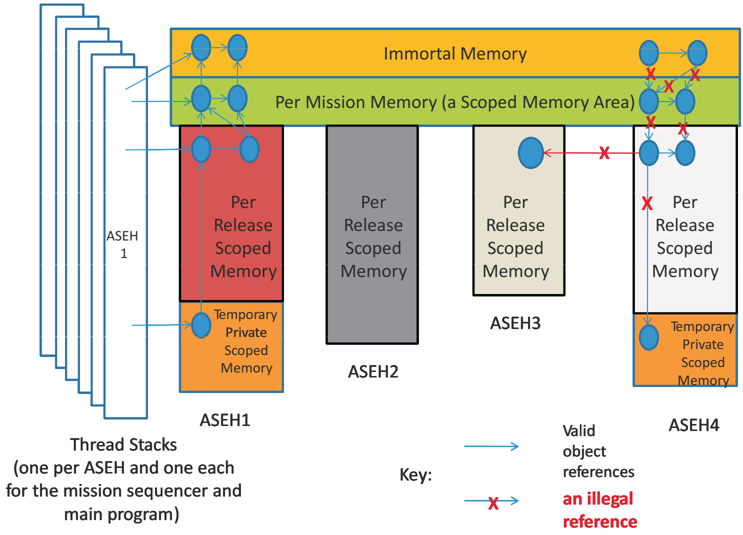
- International effort lead by the Open Group
- Performed under the Java Community Process
- Based on the Real-Time Specification for Java
  - A Safety-Critical Java Specification
  - A reference implementation
  - A technology compatibility kit
- Goal: certification
- Levels: 0, 1, 2

Nothing about design techniques

Application structure



Scoped memory area



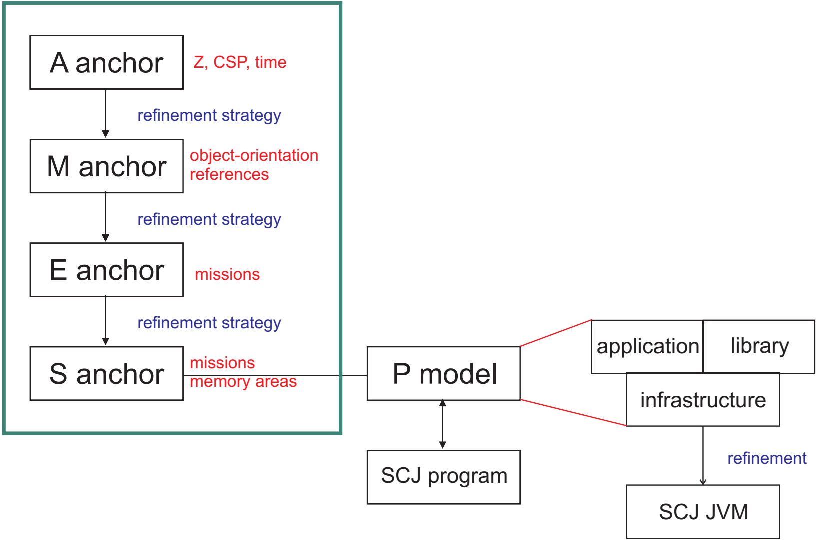
Circus Family

- *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

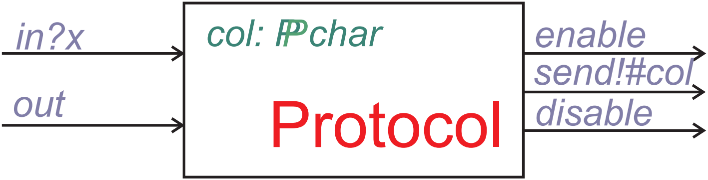
Circus variants

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

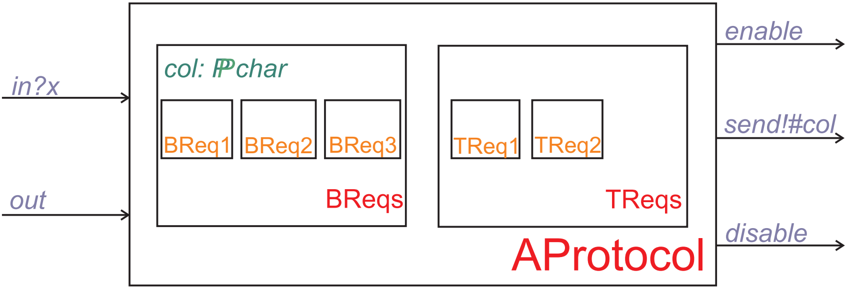
Development of SCJ programs: our approach



Example: simple protocol



Example: A Anchor



Example: A Anchor

$\textbf{process}\ \ BReqs \mathrel{\widehat=} \textbf{begin}$

$\textbf{state}\ \ APState == [col : \mathbb{P}\,\mathrm{char}]$

$Init == [APState' \mid col' = \varnothing]$

$Insert == [\Delta APState;\ x? : \mathrm{char} \mid col' = col \cup \{x?\}]$

$InsS(w) \mathrel{\widehat=} (\textbf{wait}\ 0..w\ ;\ \ Insert)\ \Box\ (send!(\#\,col)@t \longrightarrow InsS(w - t))$

$BReq1 \mathrel{\widehat=} (in?x@t \longrightarrow InsS(100 - t)\ \Box\ \ send!(\#\,col) \longrightarrow \textbf{Skip})\ ;\ \ BReq1$

$BReq2 \mathrel{\widehat=} out \longrightarrow enable \longrightarrow send?x \longrightarrow BReq2$

$BReq3 \mathrel{\widehat=} send?x \longrightarrow disable \longrightarrow BReq3$

$\bullet\ \ \textbf{wait}\ 0\,..\,3\ ;\ \ Init;$
$(\ BReq1\ \ \llbracket\ \{col\} \mid \{\!|\ send\ |\!\} \mid \{\}\ \rrbracket\ \ (\ BReq2\ \ \llbracket\ \{\!|\ send\ |\!\}\ \rrbracket\ \ BReq3\ )\ )$

$\textbf{end}$

Example: A Anchor

$$\textbf{process}\ \ TReqs \mathrel{\widehat=} \textbf{begin}$$

$$TReq1 \mathrel{\widehat=} ((in?x \longrightarrow \textbf{Skip}) \blacktriangleright 5 \mathbin{|||} \textbf{wait}\ 100)\ ;\ \ TReq1$$

$$TReq2 \mathrel{\widehat=}\ \ out \longrightarrow \textbf{wait}\ 0\,..\,7\ ;\ \ enable \longrightarrow (disable \longrightarrow \textbf{Skip}) \blacktriangleright 15;\ TReq2$$

$$\bullet\ \ TReq1 \mathbin{|||} TReq2$$

$$\textbf{end}$$

$$\textbf{system}\ \ AProtocol \mathrel{\widehat=} BReqs \llbracket \{\!|\ in, out, enable, disable\ |\!\} \rrbracket\ TReqs$$

M Anchor

Memory allocation

- Java memory model
- Language: *OhCircus* with references
- Data refinement
- Automation: not possible in general

Example: M Anchor

$\textbf{class}\ \mathit{List} \mathrel{\widehat{=}}$

$\textbf{state}\ \ \mathit{LState} == [\mathit{val} : \mathrm{char};\ \mathit{next} : \mathit{List};\ \mathit{empty} : \mathit{Bool} \mid \ldots]$

$\textbf{initial}\ \mathit{Init} == [\mathit{LState}' \mid \mathit{empty}' = \mathit{true}]$

synchronized public *insert*

$\Delta \mathit{LState};\ x? : \mathrm{char}$

$\textbf{let}\ \mathit{col} == \textbf{self}.\mathit{elems}();\ \ \mathit{col}' == \textbf{self}'.\mathit{elems}() \bullet \mathit{col}' = \mathit{col} \cup \{x?\}$

$\textbf{logical}\ \mathit{elems} \mathrel{\widehat{=}} \textbf{res}\ \ \mathit{col} : \mathbb{P}\,\mathrm{char} \bullet$

$\quad \textbf{if}\ \ \mathit{empty} = \mathit{true} \longrightarrow \mathit{col} := \varnothing$

$\quad [\!]\ \mathit{empty} = \mathit{false} \longrightarrow \mathit{col} := \mathit{next}.\mathit{elems}() \cup \{\mathit{val}\}$

$\quad \textbf{fi}$

$\textbf{synchronized public}\ \mathit{size} == [\,\Xi \mathit{LState};\ s! : \mathbb{Z} \mid \ldots\,]$

end

Example: M Anchor

$\textbf{process}\ \ \mathit{MBReqs} \mathrel{\widehat=} \textbf{begin}$

$\textbf{state}\ \ \mathit{MPState} == [\mathit{l} : \mathit{List}]$

$\mathit{Init} \mathrel{\widehat=} (\mathit{l} := \textbf{new}\ \mathit{List})$

$\mathit{InsS}(w) \mathrel{\widehat=} (\textbf{wait}\ 0..w\ ;\ \ \mathit{l.insert}(x))\ \ \Box\ \ (\mathit{send}!(\mathit{l.size}())@t \longrightarrow \mathit{InsS}(w - t))$

$\mathit{BReq1} \mathrel{\widehat=} (\mathit{in}?x@t \longrightarrow \mathit{InsS}(100 - t)\ \ \Box\ \ \mathit{send}!(\mathit{l.size}()) \longrightarrow \textbf{Skip})\ ;\ \ \mathit{BReq1}$

$\ldots$

E Anchor

Design of missions and handlers

- Language: no change
- Four phases of refinement
  - CP: collapse parallelism
  - SH: sharing
  - MH: missions and handlers
  - AR: algorithmic refinement
- Automation
  - Interface of the handlers?
  - Sharing among handlers?

Example: CP phase

Example: CP phase

system *EProtocol* $\widehat{=}$ **begin**

state $MPState == [l : List]$

$Init \mathrel{\widehat{=}} (l := \mathbf{new}\ List)$

$$InPending(t, d) \mathrel{\widehat{=}} \begin{array}{l} (in?x@u \longrightarrow AfterInPinsert(t + u, 100 - (t + u), x)) \blacktriangleleft d \\ \Box \\ out@u \longrightarrow InAfterOut(t + u, d - u, 7) \end{array}$$

$AfterInPinsert(t, wins, x) \mathrel{\widehat{=}}$

$\quad \bigsqcap d : 0 \ldots wins \bullet (out@u \longrightarrow \ldots$

$\ldots$

$\bullet$ **wait** $0 \ldots 3\ ;\ Init\ ;\ InPending(0, 5)$

end

Example: CP phase

system *EProtocol* $\widehat{=}$ **begin**

state *MPState* $== [l : List]$

$Init \mathrel{\widehat{=}} (l := \textbf{new}\ List)$

$$InPending(t, d) \mathrel{\widehat{=}} \begin{array}{l} (in?x@u \longrightarrow AfterInPinsert(t+u, 100-(t+u), x)) \blacktriangleleft d \\ \Box \\ out@u \longrightarrow InAfterOut(t+u, d-u, 7) \end{array}$$

$AfterInPinsert(t, wins, x) \mathrel{\widehat{=}}$

$\quad \sqcap d : 0 \mathrel{..} wins \bullet (out@u \longrightarrow \ldots$

$\ldots$

$\bullet\ \textbf{wait}\ 0 \mathrel{..} 3\ ;\ \boxed{Init\ ;\ InPending(0, 5)}$

end

SH Phase

Splitting the state

Components in

- Immortal memory: stay where they are
- Per-release and temporary areas: become local to the main action
- Mission memory: become local to a new separate parallel action

Example: SH phase

$$\textbf{system}\ EProtocol \mathrel{\widehat=} \textbf{begin}$$

$$\ldots$$

$$InPending(t, d) \mathrel{\widehat=} \ldots$$

$$\ldots$$

$$System \mathrel{\widehat=} InPending(0, 5)$$

$$MArea \mathrel{\widehat=} \left(\begin{array}{l} \textbf{var}\ l : List \bullet Init; \\ \left(\mu\ X \bullet \left(\begin{array}{l} insertLC?x \longrightarrow l.insert(x)\ ;\ insertLR \longrightarrow X \\ \Box \\ sizeLC \longrightarrow sizeLR!(l.size(x)) \longrightarrow X \end{array}\right)\right)\end{array}\right)$$

$$\bullet\ \ \textbf{wait}\ 0\,..\,3;$$
$$(System\ [\![\ \{\!|\ insertLC, insertLR, \ldots\ |\!\}\ ]\!]\ MArea) \setminus \{\!|\ insertLC, insertLR, \ldots\ |\!\}$$

$$\textbf{end}$$

Example: SH phase

$$\textbf{system}\ EProtocol \mathrel{\widehat=} \textbf{begin}$$

$\ldots$

$$InPending(t, d) \mathrel{\widehat=} \ldots$$

$\ldots$

$$System \mathrel{\widehat=} \boxed{InPending(0, 5)}$$

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

$$\left(\begin{array}{l} \textbf{var}\ l : List \bullet Init; \\ \left(\mu\ X \bullet \left(\begin{array}{l} insertLC?x \longrightarrow l.insert(x)\ ;\ insertLR \longrightarrow X \\ \Box \\ sizeLC \longrightarrow sizeLR!(l.size(x)) \longrightarrow X \end{array}\right)\right) \end{array}\right)$$

$$\bullet\ \ \textbf{wait}\ 0 \mathrel{..} 3;$$

$$(System \llbracket \{\!|\ insertLC, insertLR, \ldots\ |\!\} \rrbracket\ MArea) \setminus \{\!|\ insertLC, insertLR, \ldots\ |\!\}$$

end

Example: MH phase

$\mathbf{system}\ \mathit{EProtocol} \mathrel{\widehat=} \mathbf{begin}$

$\ldots$

$\mathit{Handler1} \mathrel{\widehat=}$

$\quad ((\mathit{in}?x@t \longrightarrow \mathbf{wait}\ 0..(100 - t) ;\ \mathit{insertLC}!x \ldots$

$\mathit{Handler2} \mathrel{\widehat=}$

$\quad \mathit{out} \longrightarrow \mathit{sizeLC} \longrightarrow \mathit{sizeLR}?x \longrightarrow \mathbf{wait}\ 0 \mathinner{..} 7;$

$\quad \mathit{enable} \longrightarrow (\mathit{send}!x \longrightarrow \mathit{disable} \longrightarrow \mathbf{Skip}) \blacktriangleright 15 ;\ \mathit{Handler2}$

$\mathit{Mission} \mathrel{\widehat=} (\mathit{Handler1} \interleave \mathit{Handler2})$

$\mathit{System} \mathrel{\widehat=} \mathit{Mission}$

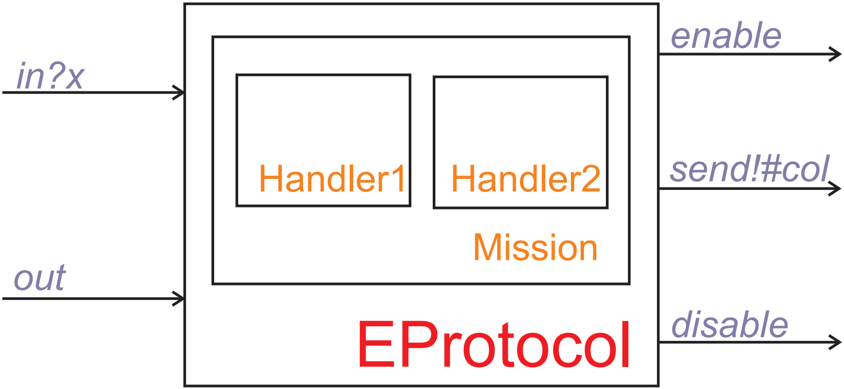
$\mathit{MArea} \mathrel{\widehat=} \ldots$

$\bullet\ \mathbf{wait}\ 0 \mathinner{..} 3;$

$\quad (\mathit{System} \llbracket \{\!| \mathit{insertLC}, \mathit{insertLR}, \ldots |\!\} \rrbracket \mathit{MArea}) \setminus \{\!| \mathit{insertLC}, \mathit{insertLR}, \ldots |\!\}$

$\mathbf{end}$

Example: MH phase



S Anchor

SCJ framework

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

Example: S Anchor

sequencer *MainMissionSequencer* $\widehat{=}$ **begin**
state $MainMissionSequencerState == [mission\_done : Bool]$
initial $\widehat{=}$ $mission\_done := false$
getNextMission $\widehat{=}$
  if $mission\_done = false \longrightarrow$
    $mission\_done := true;$ **ret** $:= ProtocolMission$
  $[\!]$ $mission\_done = true \longrightarrow$ **ret** $:= null$
  fi
end

mission *ProtocolMission* $\widehat{=}$ **begin**
state $MState == [l : List]$
initialize $\widehat{=}$
  $l :=$ **new** $List$; (**newHandler** $Handler1(l)$) ; (**newHandler** $Handler2(l)$)
cleanup $\widehat{=}$ **Skip**
end

Example: S Anchor

$\mathbf{periodic}(100)\ \mathbf{handler}\ \mathit{Handler1} \mathrel{\widehat=} \mathbf{begin}$
$\mathbf{state}\ \mathit{Handler1\_State} == [l : \mathit{List}]$
$\mathbf{initial}\ \mathit{Handler1\_Init} \mathrel{\widehat=} \mathbf{val}\ \mathit{list?} : \mathit{List} \bullet l := \mathit{list?}$
$\mathbf{handleAsyncEvent}(x, w) \mathrel{\widehat=} \mathbf{wait}\ 0..w\ ;\ l.\mathit{insert}(x)$
$\mathbf{dispatch} \mathrel{\widehat=} (\mathit{in?x@t} \longrightarrow \mathbf{handleAsyncEvent}(x, 100 - t)) \blacktriangleleft 5$
$\mathbf{end}$

$\mathbf{aperiodic\ handler}\ \mathit{Handler2} \mathrel{\widehat=} \mathbf{begin}$
$\mathbf{state}\ \mathit{Handler2\_State} == [l : \mathit{List}]$
$\mathbf{initial}\ \mathit{Handler2\_Init} \mathrel{\widehat=} \mathbf{val}\ \mathit{list?} : \mathit{List} \bullet l := \mathit{list?}$
$\mathbf{handleAsyncEvent} \mathrel{\widehat=}$
$\quad \mathbf{var}\ \mathit{size} : \mathbb{N} \bullet \mathit{size} := l.\mathit{size}()\ ;\ \mathbf{wait}\ 0\ ..\ 7;$
$\quad\quad \mathit{enable} \longrightarrow (\mathit{send}\,!\,\mathit{size} \longrightarrow \mathit{disable} \longrightarrow \mathit{Skip}) \blacktriangleright 15$
$\mathbf{dispatch} \mathrel{\widehat=} (\mathit{out} \longrightarrow \mathbf{handleAsyncEvent}())$
$\mathbf{end}$

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

Challenges ahead

Theory

- Integration of languages and theories
- Mechanisation
- Refinement laws and detailed strategies
- Modular reasoning about libraries

Practice

- Case studies
- Design patterns
- Generation of abstract models
- Automation

And beyond

- Certification, Resources, ...