Circus Models of Safety-Critical Java

Frank Zeyda

University of York, UK

First hiJaC Workshop, 15 Nov 2011

Introduction

The hiJaC Project

- **H**igh-**I**ntegrity **J**ava **A**pplications using **C***ircus*
- 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.
  $\rightarrow$ 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

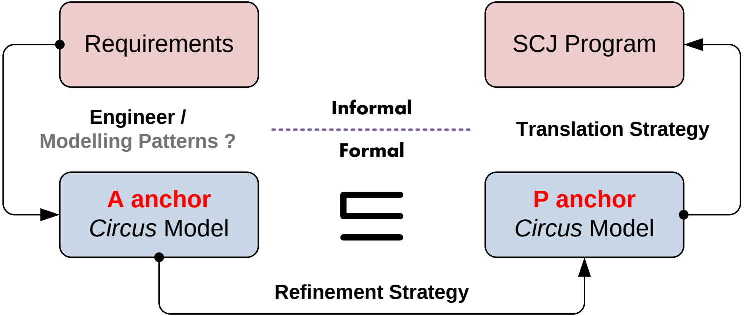
- **H**igh-**I**ntegrity **J**ava **A**pplications using **C***ircus*
- Development of formalisms, techniques and tools to produce formally verified Safety-Critical Java (**SCJ**) programs.
- *Circus* family of languages is used to define a semantic theory.
  $\rightarrow$ Standard *Circus* + *OhCircus* + *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.

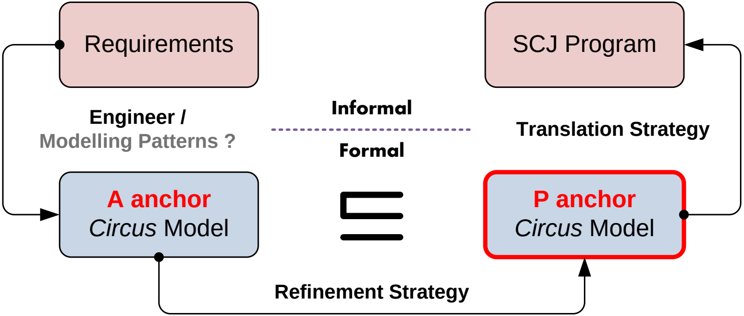
All three possibilities are entailed by our approach.

Top-level Strategy



The model of the program is the subject of this seminar.

Top-level Strategy



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

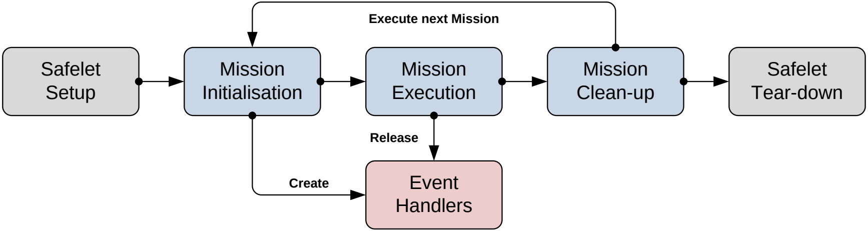
- Specify primary components of SCJ:
  Safelet, **Mission Sequencer**, **Missions**, and **Event Handlers**.
- Encode the mission-based execution paradigm (**Level 1**).
- Capture timing aspects where necessary.
- Be faithful to the SCJ Technology Specification (JSR 302).

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

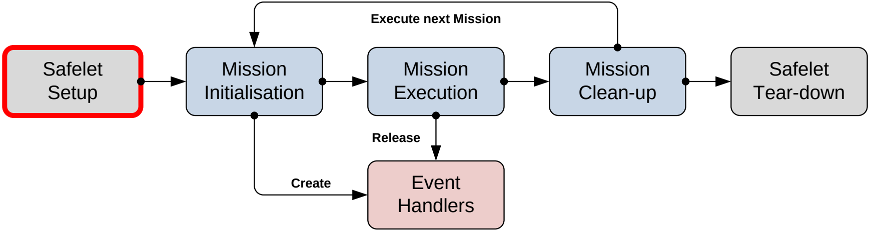
SCJ Programming Model



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.

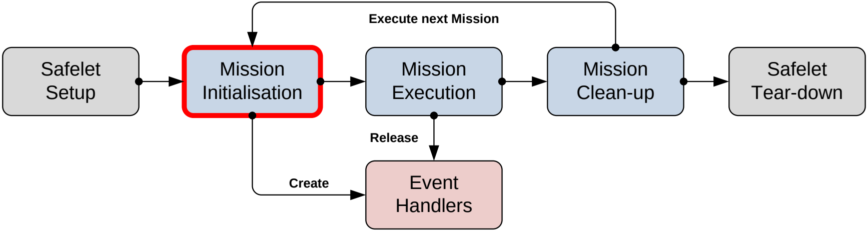
SCJ Programming Model



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.

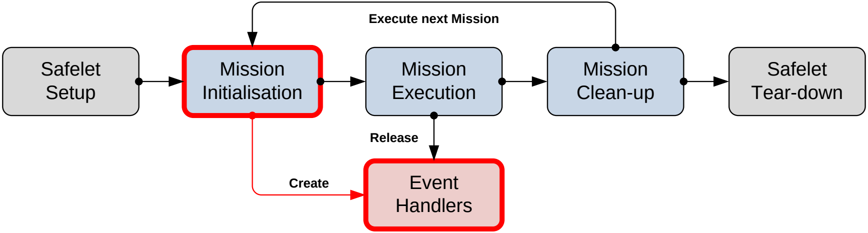
SCJ Programming Model



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.

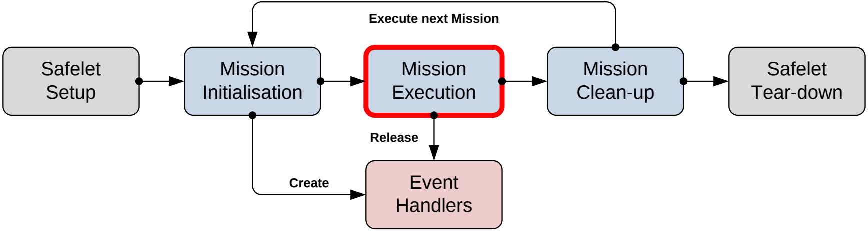
SCJ Programming Model



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.

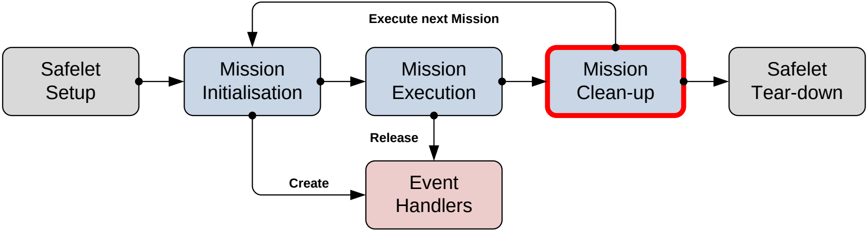
SCJ Programming Model



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.

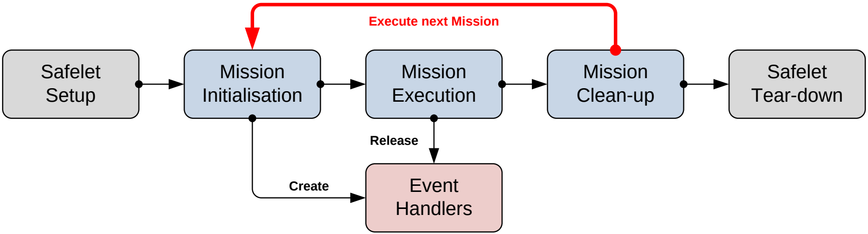
SCJ Programming Model



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.

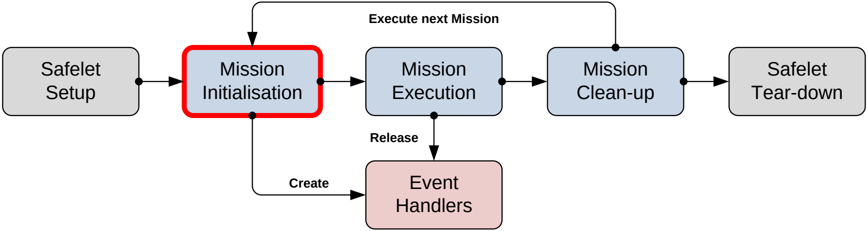
SCJ Programming Model



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.

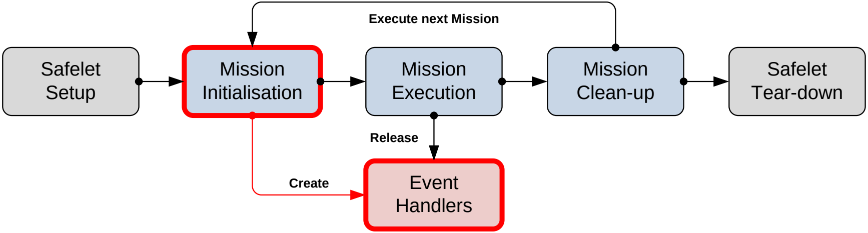
SCJ Programming Model



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.

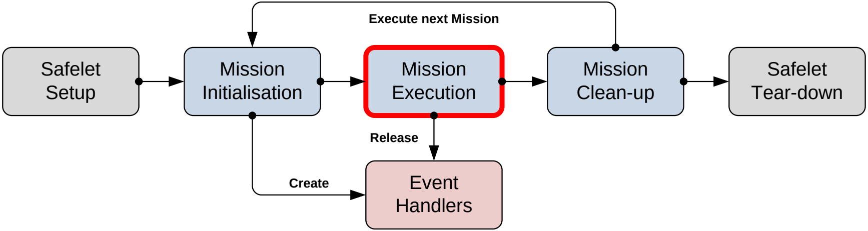
SCJ Programming Model



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.

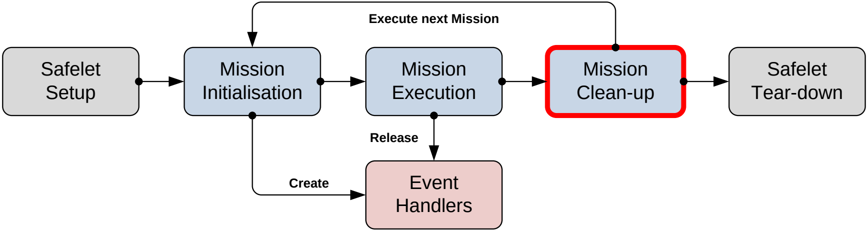
SCJ Programming Model



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.

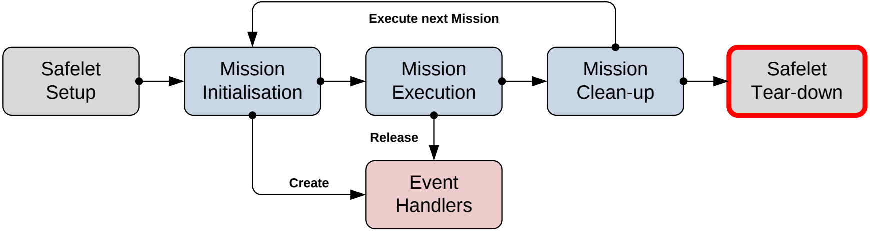
SCJ Programming Model



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.

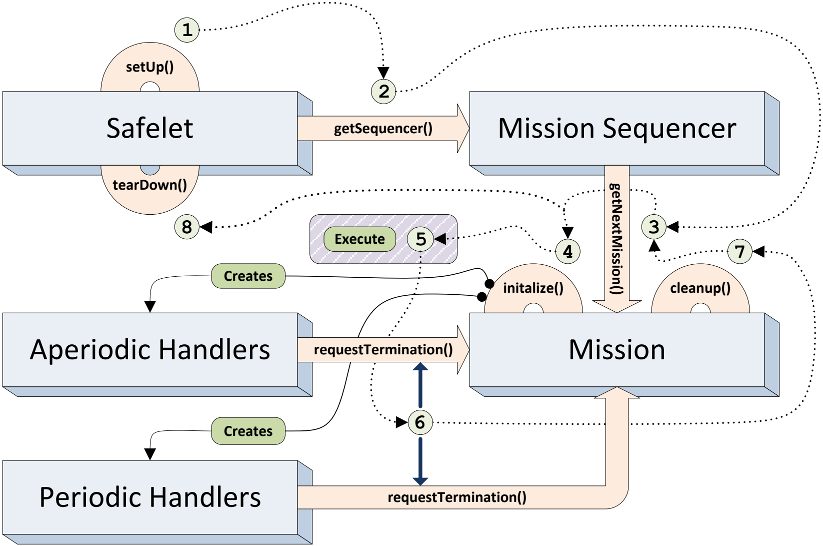
SCJ Programming Model



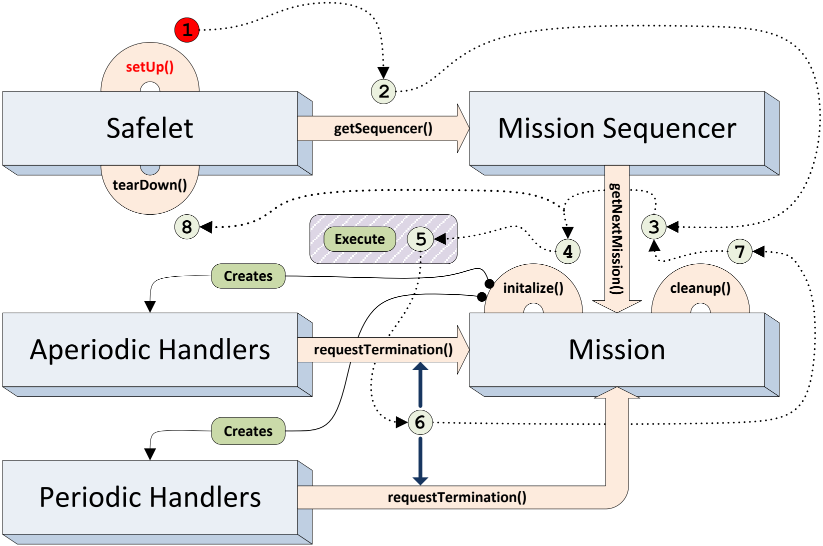
Summary of Level 1

- 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.

The Safelet Life-cycle

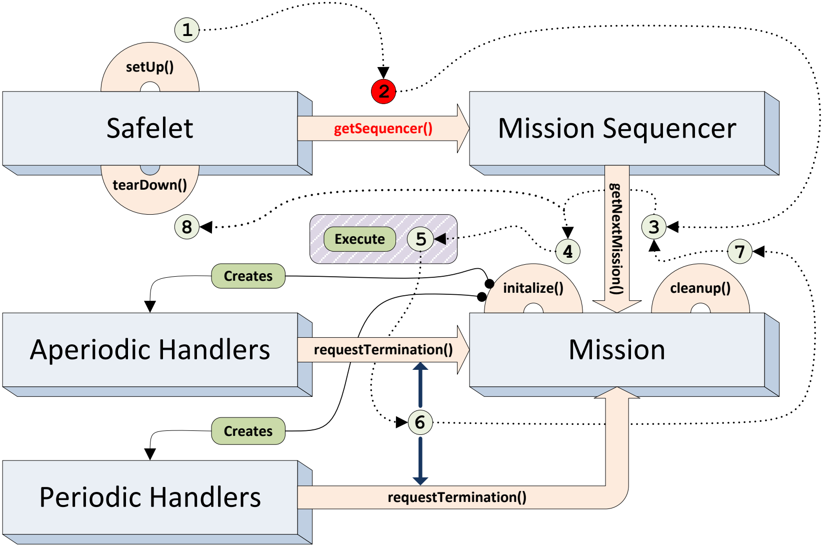


The Safelet Life-cycle



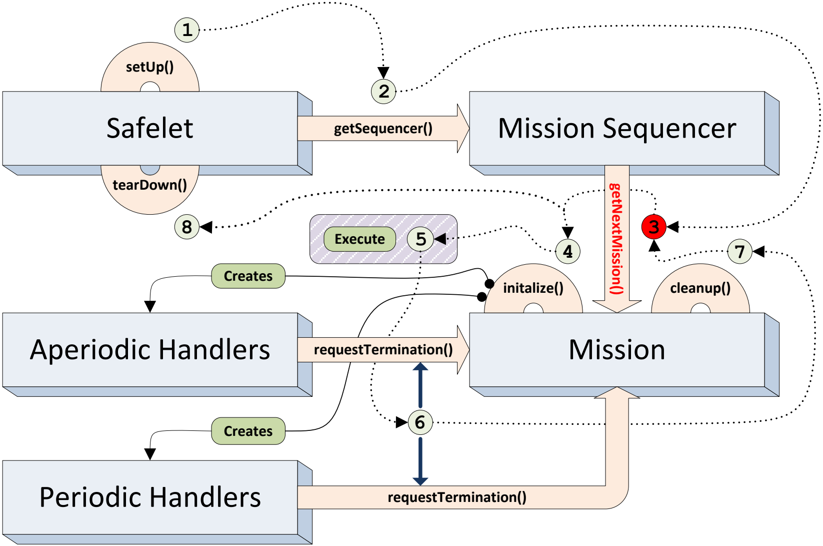
Infrastructure calls `setUp()` to initialise the safelet.

The Safelet Life-cycle



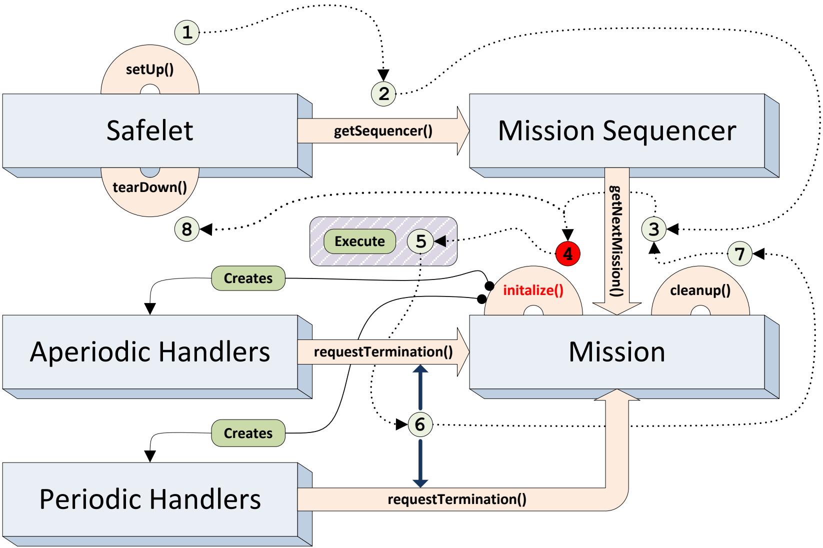
Infrastructure calls `getSequencer()` to obtain the mission sequencer.

The Safelet Life-cycle



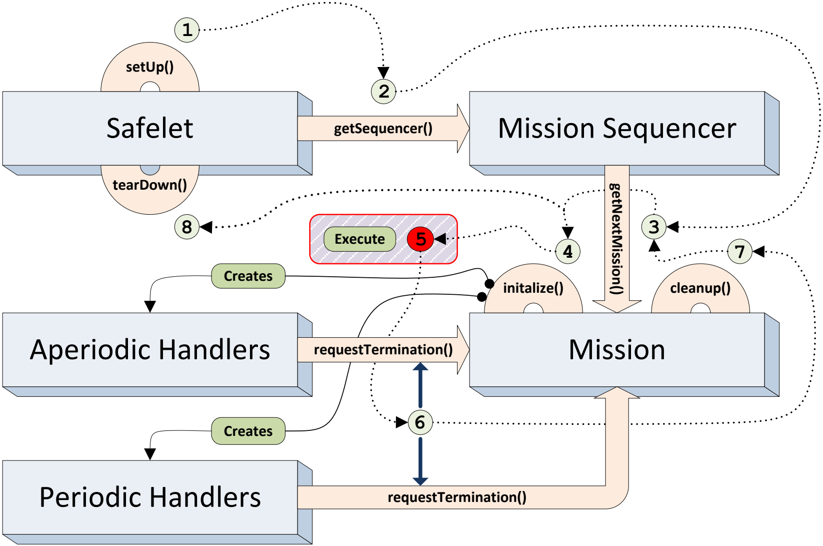
Infrastructure calls `getNextMission()` to obtain the first mission.

The Safelet Life-cycle



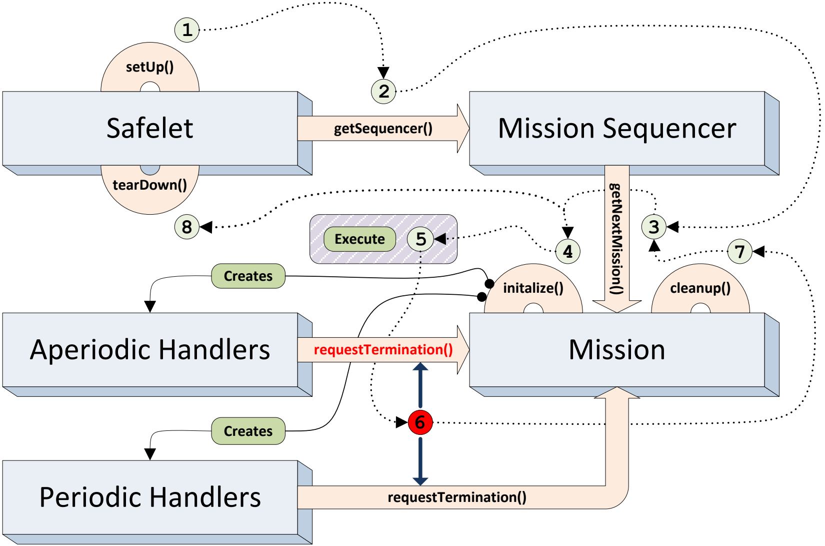
Infrastructure calls `initilize()` to create handlers and other data.

The Safelet Life-cycle



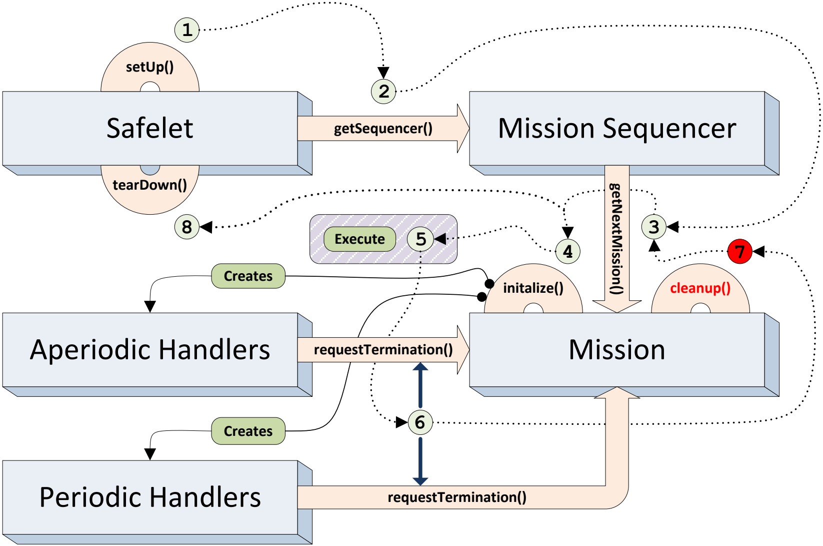
Execution phase of the mission: handlers become active.

The Safelet Life-cycle



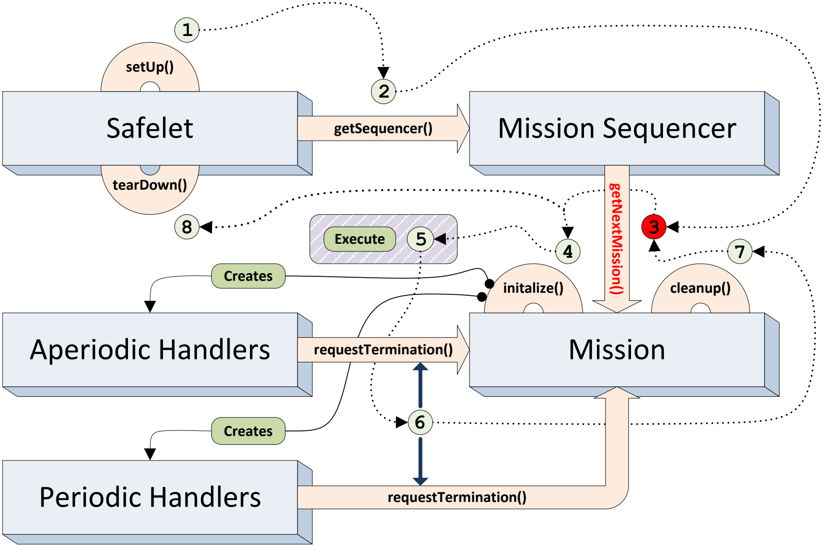
One of the handlers calls `requestTermination()` to initiate

The Safelet Life-cycle



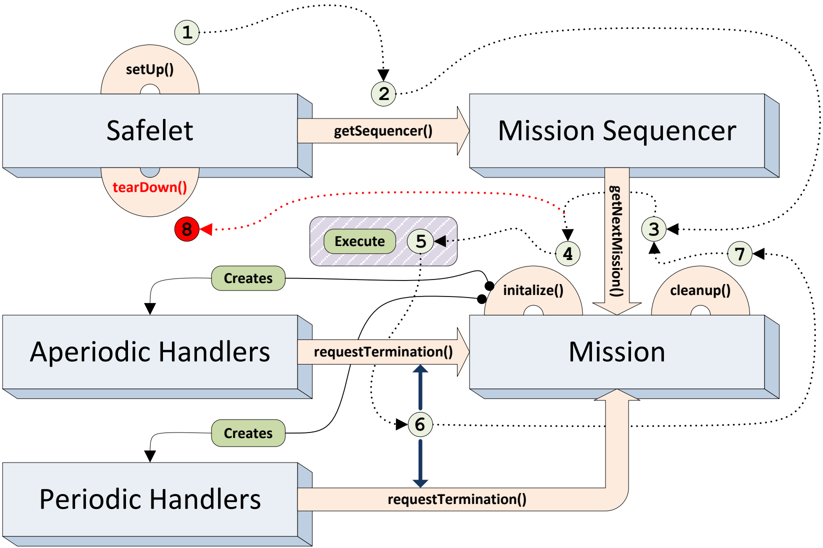
Infrastructure calls `cleanup()` after handlers have been stopped.

The Safelet Life-cycle



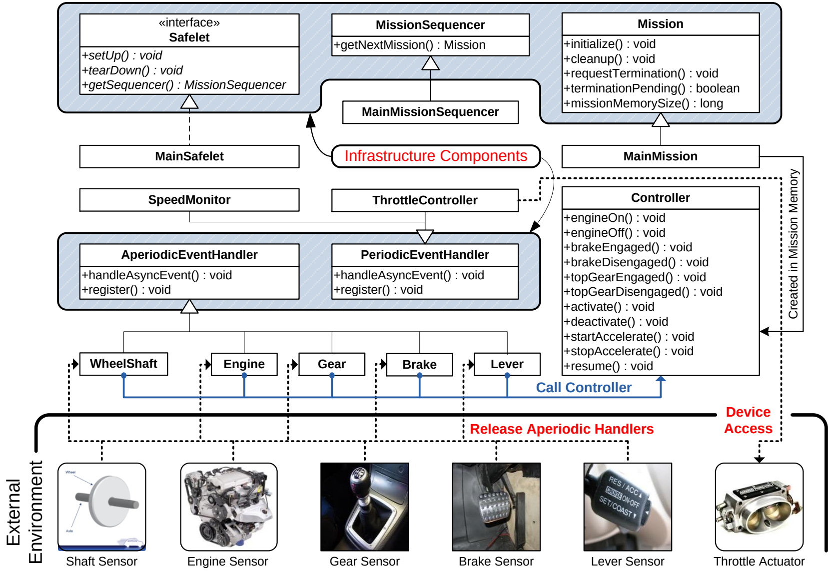
Infrastructure calls `getNextMission()` to obtain the next mission.

The Safelet Life-cycle

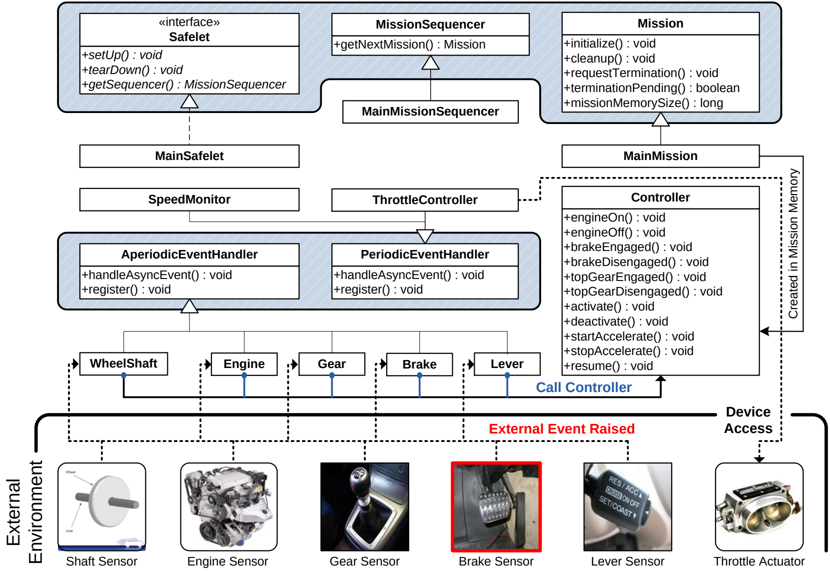


If it returns `null`, `tearDown()` is called and the safelet terminates.

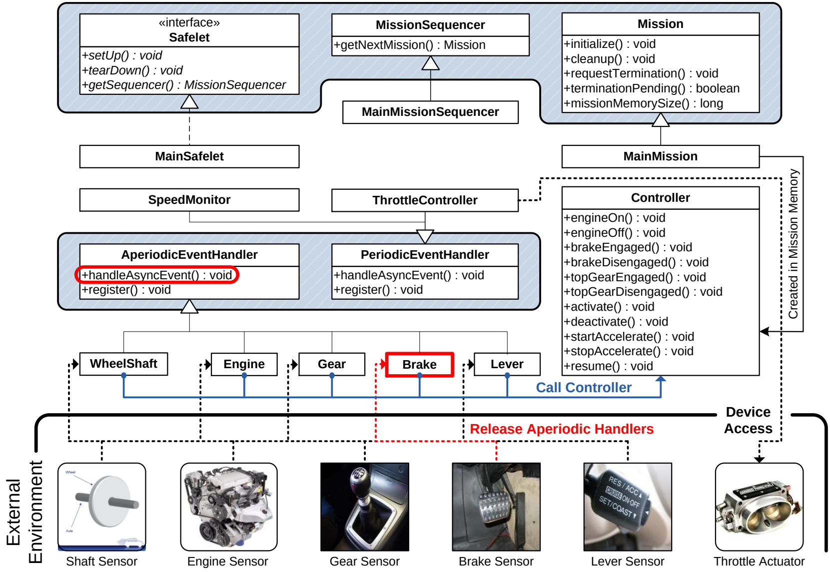
SCJ Program: Automotive Cruise Controller



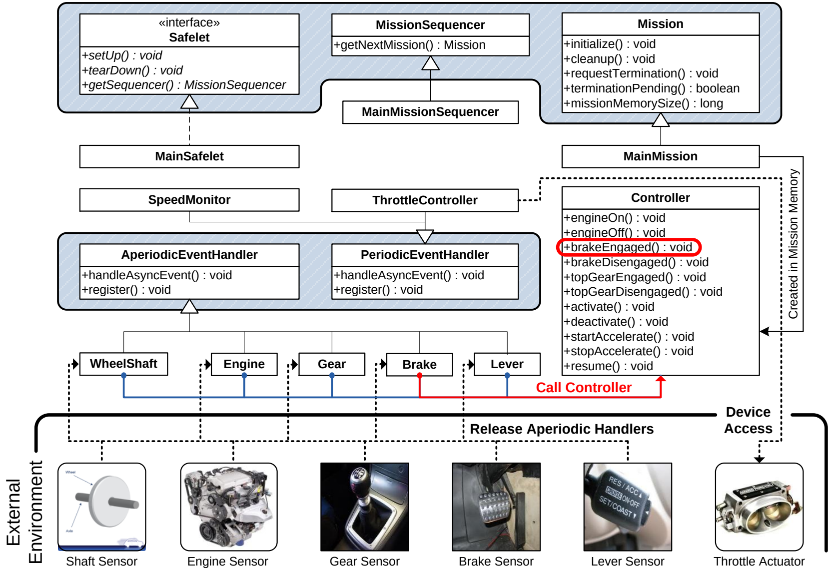
SCJ Program: Automotive Cruise Controller



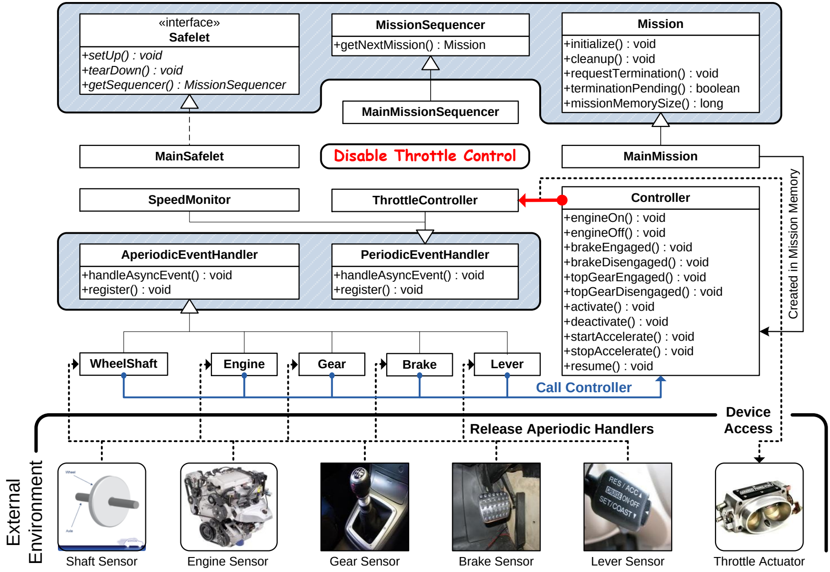
SCJ Program: Automotive Cruise Controller



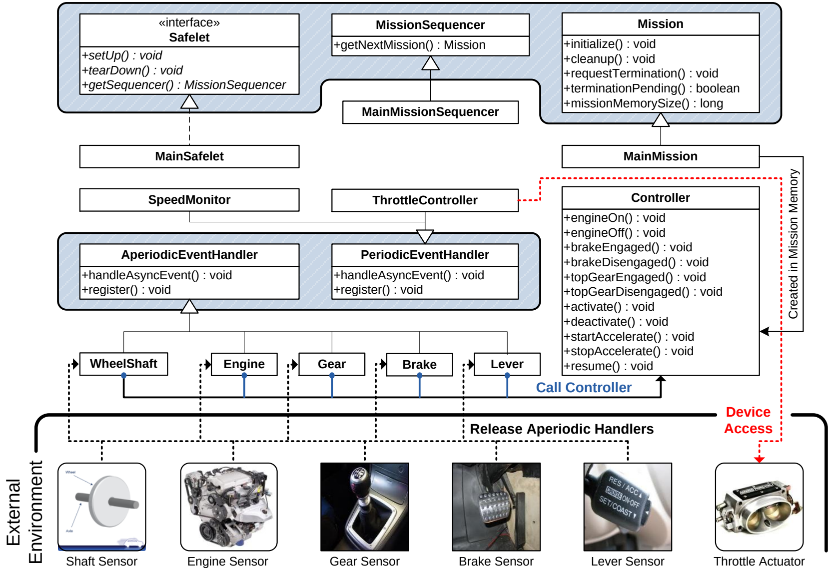
SCJ Program: Automotive Cruise Controller



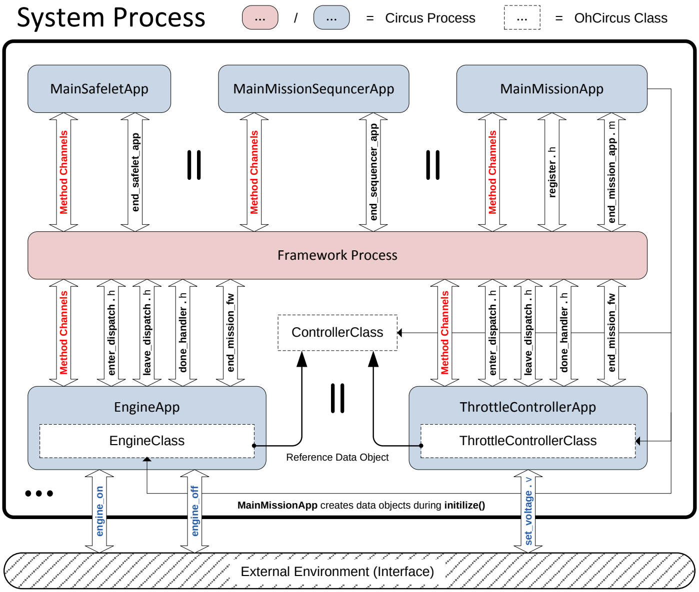
SCJ Program: Automotive Cruise Controller



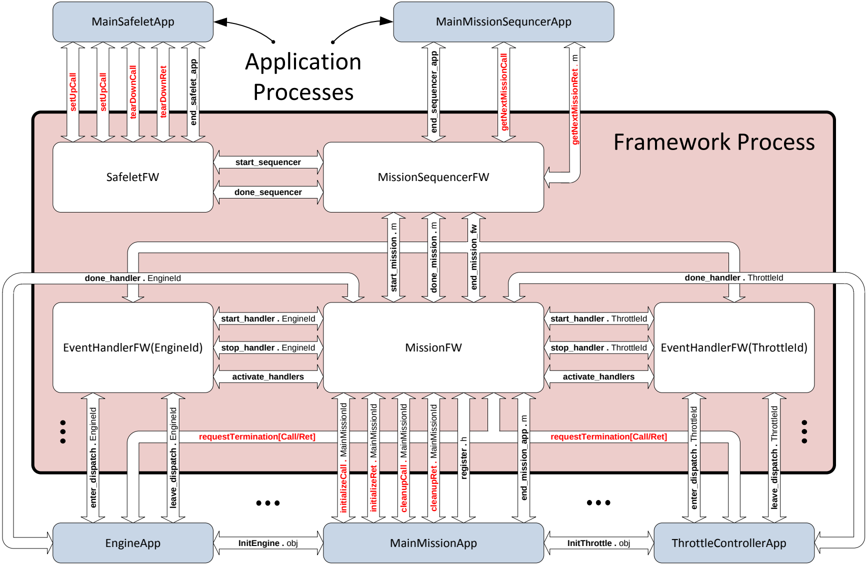
SCJ Program: Automotive Cruise Controller



Structure of the *Circus* Model



Framework Process



Model of the Safelet

Framework Process

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

$SetUp \mathrel{\widehat=} setUpCall \longrightarrow setUpRet \longrightarrow \textbf{skip}$

$Execute \mathrel{\widehat=} start\_sequencer \longrightarrow done\_sequencer \longrightarrow \textbf{skip}$

$TearDown \mathrel{\widehat=} tearDownCall \longrightarrow tearDownRet \longrightarrow \textbf{skip}$

$\bullet\ SetUp \mathbin{;}\ Execute \mathbin{;}\ TearDown \mathbin{;}\ end\_safelet\_app \longrightarrow \textbf{skip}$

$\textbf{end}$

Application Process

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

$setUpMeth \mathrel{\widehat=} setUpCall \longrightarrow \textbf{skip} \mathbin{;}\ setUpRet \longrightarrow \textbf{skip}$

$tearDownMeth \mathrel{\widehat=} tearDownCall \longrightarrow \textbf{skip} \mathbin{;}\ tearDownRet \longrightarrow \textbf{skip}$

$Methods \mathrel{\widehat=} \mu X \bullet (setUpMeth \mathbin{\square} tearDownMeth) \mathbin{;}\ X$

$\bullet\ Methods \mathbin{\triangle} end\_safelet\_app \longrightarrow \textbf{skip}$

$\textbf{end}$

Observation: Application processes exhibit a uniform pattern!

Model of the Safelet

Framework Process

$$
\begin{array}{l}
\textbf{process}\ \mathit{SafeletFW} \mathrel{\widehat=} \textbf{begin} \\
\quad \mathit{SetUp} \mathrel{\widehat=} \mathit{setUpCall} \longrightarrow \mathit{setUpRet} \longrightarrow \textbf{skip} \\
\quad \mathit{Execute} \mathrel{\widehat=} \mathit{start\_sequencer} \longrightarrow \mathit{done\_sequencer} \longrightarrow \textbf{skip} \\
\quad \mathit{TearDown} \mathrel{\widehat=} \mathit{tearDownCall} \longrightarrow \mathit{tearDownRet} \longrightarrow \textbf{skip} \\
\quad \bullet\ \mathit{SetUp} \mathbin{;} \mathit{Execute} \mathbin{;} \mathit{TearDown} \mathbin{;} \mathit{end\_safelet\_app} \longrightarrow \textbf{skip} \\
\textbf{end}
\end{array}
$$

Application Process

$$
\begin{array}{l}
\textbf{process}\ \mathit{MainSafeletApp} \mathrel{\widehat=} \textbf{begin} \\
\quad \mathit{setUpMeth} \mathrel{\widehat=} \mathit{setUpCall} \longrightarrow \textbf{skip} \mathbin{;} \mathit{setUpRet} \longrightarrow \textbf{skip} \\
\quad \mathit{tearDownMeth} \mathrel{\widehat=} \mathit{tearDownCall} \longrightarrow \textbf{skip} \mathbin{;} \mathit{tearDownRet} \longrightarrow \textbf{skip} \\
\quad \mathit{Methods} \mathrel{\widehat=} \mu X \bullet (\mathit{setUpMeth} \mathbin{\Box} \mathit{tearDownMeth}) \mathbin{;} X \\
\quad \bullet\ \mathit{Methods} \mathbin{\triangle} \mathit{end\_safelet\_app} \longrightarrow \textbf{skip} \\
\textbf{end}
\end{array}
$$

Observation: Application processes exhibit a uniform pattern!

Model of Event Handlers

Framework Process (one per handler)

$$
\begin{array}{l}
\mathbf{process}\ EventHandlerFW \mathrel{\widehat=} handler : HandlerId \bullet \mathbf{begin} \\
\quad \mathbf{state}\ State == [active : BOOL] \\
\quad Init == [State' \mid active' = FALSE] \\
\quad StartHandler \mathrel{\widehat=} start\_handler\,.\,handler \longrightarrow active := TRUE \\
\quad ActivateHandlers \mathrel{\widehat=} activate\_handlers \longrightarrow \mathbf{skip} \\
\quad DispatchHandler \mathrel{\widehat=} enter\_dispatch\,.\,handler \longrightarrow \\
\qquad stop\_handler\,.\,handler \longrightarrow leave\_dispatch\,.\,handler \longrightarrow \mathbf{skip} \\
\quad \bullet \left(\mu X \bullet Init \mathbin{;} \left(\begin{array}{l} ((StartHandler \mathbin{;}\ ActivateHandlers) \mathbin{\Box} ActivateHandlers) \mathbin{;} \\ \mathbf{if}\ active = TRUE \longrightarrow DispatchHandler \\ [\!]\ active = FALSE \longrightarrow \mathbf{skip} \\ \mathbf{fi} \end{array} \right) \mathbin{;}\ X \right) \\
\qquad \mathbin{\triangle}\ end\_mission\_fw \longrightarrow \mathbf{skip} \\
\mathbf{end}
\end{array}
$$

Requirement of the Infrastructure

- Asynchronously start handlers (*start_handler*).
- Synchronously activate handlers (*activate_handlers*).

Model of Event Handlers (cont'd)

Engine Handler: Java Code

```
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

$$
\begin{array}{l}
\textbf{class}\ \mathit{EngineClass}\ \widehat{=}\ \textbf{begin} \\
\quad \textbf{state}\ \mathit{EngineState} == [\,\textbf{private}\ \mathit{cruise} : \mathit{CruiseControlClass}\,] \\
\quad \textbf{initial}\ \mathit{EngineInit}\ \widehat{=}\ \textbf{val}\ \mathit{cruise}? : \mathit{CruiseControlClass} \bullet \mathit{cruise} := \mathit{cruise}? \\
\quad \textbf{public}\ \mathit{handleAsyncEvent}\ \widehat{=}\ \textbf{val}\ \mathit{evt} : \mathit{EventId} \bullet \\
\qquad \textbf{if}\ \mathit{evt} = \mathit{EngineOnEvtId} \longrightarrow \mathit{cruise}.\mathit{engineOn}() \\
\qquad [\!]\ \mathit{evt} = \mathit{EngineOffEvtId} \longrightarrow \mathit{cruise}.\mathit{engineOff}() \\
\qquad \textbf{fi} \\
\textbf{end}
\end{array}
$$

Application Process

$$
\begin{array}{l}
\textbf{process}\ \mathit{EngineApp}\ \widehat{=}\ \textbf{begin} \\
\quad \textbf{state}\ \mathit{EngineState} == [\mathit{obj} : \mathit{EngineClass}] \quad \ldots \\
\quad \mathit{Dispatch}\ \widehat{=} \left(\mu X \bullet \left(\begin{array}{l} \mathit{leave\_dispatch}\,.\,\mathit{EngineId} \longrightarrow \textbf{skip} \\ \quad \Box \\ \left(\left(\begin{array}{l} \mathit{engine\_on} \longrightarrow \mathit{obj}.\mathit{handleAsyncEvent}(\mathit{EngineOnEvtId}) \\ \qquad \Box \\ \mathit{engine\_off} \longrightarrow \mathit{obj}.\mathit{handleAsyncEvent}(\mathit{EngineOffEvtId}) \end{array} \right) ; \right. \\ \left. \textbf{wait}\ 0\,..\,\mathit{EngineDeadline} \right) \end{array} \right) ; X \right) \quad \ldots
\end{array}
$$

Aperiodic vs Periodic Handlers

Aperiodic Event Handlers

- Released by external events bound to the handler.
- Modelled by external choice in *Dispatch*:

$$\begin{array}{l} evt1 \longrightarrow handleAsyncEvent(Evt1Id) \;\Box \\ evt2 \longrightarrow handleAsyncEvent(Evt2Id) \;\Box \;\ldots \end{array}$$

Periodic Event Handlers

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

$$Dispatch \;[\![\;\{\!|\; release\_handler \;|\!\}\;]\!]\; Release$$

where the *Release* action is defined as

$$Release \;\widehat{=}\; \mu X \bullet \left(\begin{array}{l} release\_handler \longrightarrow \mathbf{skip} \blacktriangleright 0; \\ \mathbf{wait}\; ThrottleCtrlPeriod \;;\; X \end{array} \right) \triangle \;\ldots$$

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*:

$$\begin{array}{l} evt1 \longrightarrow handleAsyncEvent(Evt1Id) \;\Box \\ evt2 \longrightarrow handleAsyncEvent(Evt2Id) \;\Box \ldots \end{array}$$

Periodic Event Handlers

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

$$Dispatch \;[\![\, \{\!|\, release\_handler \,|\!\} \,]\!]\; Release$$

where the *Release* action is defined as

$$Release \;\widehat{=}\; \mu X \bullet \begin{pmatrix} release\_handler \longrightarrow \mathbf{skip} \blacktriangleright 0; \\ \mathbf{wait}\; ThrottleCtrlPeriod \;;\; X \end{pmatrix} \triangle \ldots$$

But both types of handlers share the same framework model.

Validation of Models

Community Z Tools (CZT)

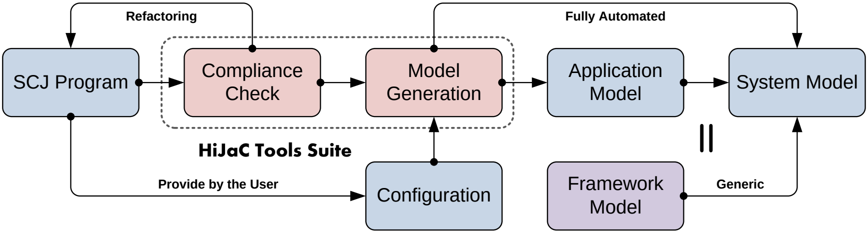
- Open development platform for Z tools.
- Parsing and type-checking support for *Circus*.
- We extended this to include *Circus* Time operators.
- Our models type-check modulo some new notations.

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.

Tool Support and Workflow



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`.

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).
4. Clarify subtleties in the SCJ Technology Specification.
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?