Circus Model for the SCJ Mission Framework

Frank Zeyda

October 12, 2011

Contents

1 Notation

To highlight channels that are exposed in process definitions, we introduce a new documentary clause **exposes** as part of a process definition. An example is given below.

$$
\begin{array}{l}
\textbf{process}\ \mathit{MainSafelet} \mathrel{\widehat=} \\
\quad (\mathit{SafeletFW} \llbracket \mathit{MainSafeletAppChan} \rrbracket \mathit{MainSafeletApp}) \setminus \mathit{MainSafeletAppChan} \\
\textbf{exposes}\ \{\!| \mathit{start\_sequencer}, \mathit{done\_sequencer} |\!\}
\end{array}
$$

The **exposes** clause emphasises that the process synchronises on the two channels *start_sequencer* and *done_sequencer*. The clause is solely for documentary purposes and does not have any semantic impact.

2 Prelude

Declaration of the *scj_prelude* section.

$\textbf{section}\ \mathit{scj\_prelude}\ \textbf{parents}\ \mathit{circus\_toolkit}$

2.1 Types

The type *BOOL* is used for boolean values in Z at the level of specification.

$\mathit{BOOL} ::= \mathit{TRUE} \mid \mathit{FALSE}$

2.2 Functions

The unary relation *distinct* captures that the elements of a sequence are distinct.

$\textbf{relation}\ (\mathit{distinct}\ \_)$

$$
\begin{array}{l}
[X] \\
\hline
\mathit{distinct}\ \_ : \mathbb{P}(\mathrm{iseq}\ X)
\end{array}
$$

We primarily require this function to formulate that the various ids introduced are unique.

2.3 References

We do not actually define a semantics for references here. Instead, we provide loose definitions of operators that allow us to some extent to type-check specifications that make use of the reference constructs.

We first introduce a generic type *REF* that acts as the semantic domain for references over a certain value type. The given type *REF_TAG* is used as a 'token type' in its definition.

$[\mathit{REF\_TAG}]$

$\mathit{REF}[X] == X \times \mathit{REF\_TAG}$

With it we next define a generic function that serves as the type constructor for references.

$\textbf{generic}\ (\_)$

$X == \mathit{REF}[X]$

The referencing and dereferencing operators are loosely introduced below.

$$
\begin{array}{l}
[X] \\
\hline
: X \to (X) \\
: (X) \to X
\end{array}
$$

As can be seen they do not actually constrain the operators but leave their semantics implicit.

The one operator which we cannot introduce in this way is destructive (reference) assignment. This can only be done at the language level, namely the *Circus* parser, because the operator corresponds to an action rather than a mathematical function.

2.4 Arrays

Arrays are modelled by sequences. For convenience, we introduce a type constructor for them.

$$\mathrm{array}[X] == (\lambda\, T : \mathbb{P}\, X \bullet (\mathrm{seq}\, T))$$

Note that the result is not the sequence type but a reference to it.

2.5 Oh*Circus*

Below we provide support for the **new** keyword in Oh*Circus*.

$$[X]$$
$$\mathbf{new} : \mathbb{P}\, X \rightarrow X$$

Again the specification is loose in that we do not define a semantics for the operator.

The Oh*Circus* keyword **class** is used synonymously for **process**. The keywords **state**, **initial**, **public**, **protected**, **private**, **logical** are ignored (treated as white spaces) for the time being. This is until we have a working Oh*Circus* parser and type-checker in CZT.

2.6 *Circus* Time

Here we define types for absolute and relative time.

$$TIME == \mathbb{N}$$
$$PERIOD == \mathbb{N}$$

3 Safelet

3.1 Channels

section *SafeletChan* **parents** *SafeletMethChan*

channel *end_safelet_app*

Method channels

section *SafeletMethChan* **parents** *scj_prelude*

channel *setUpCall*, *setUpRet*

channel *tearDownCall*, *tearDownRet*

channelset $SafeletMethChan == \{\!| setUpCall, setUpRet, tearDownCall, tearDownRet |\!\}$

3.2 Framework Process

section *SafeletFW* **parents** *SafeletChan*, *MissionSequencerChan*, *scj_prelude*

process $SafeletFW \widehat{=}$ **begin**

$SetUp \widehat{=} setUpCall \longrightarrow setUpRet \longrightarrow \mathbf{Skip}$

$Execute \widehat{=} start\_sequencer \longrightarrow done\_sequencer \longrightarrow \mathbf{Skip}$

$TearDown \widehat{=} tearDownCall \longrightarrow tearDownRet \longrightarrow \mathbf{Skip}$

$\bullet\ SetUp\ ;\ Execute\ ;\ TearDown\ ;\ end\_safelet\_app \longrightarrow \mathbf{Skip}$

end

3.3 Application Process

section *MainSafeletApp* **parents** *SafeletChan*, *scj_toolkit*, *scj_library*

process $MainSafeletApp \widehat{=}$ **begin**

$setUpMeth \widehat{=} setUpCall \longrightarrow \mathbf{Skip}\ ;\ setUpRet \longrightarrow \mathbf{Skip}$

$tearDownMeth \widehat{=} tearDownCall \longrightarrow \mathbf{Skip}\ ;\ tearDownRet \longrightarrow \mathbf{Skip}$

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

$\bullet\ Methods \triangle end\_safelet\_app \longrightarrow \mathbf{Skip}$

end

3.4 Composite Process

section *MainSafelet* **parents** *SafeletFW*, *MainSafeletApp*

channelset $MainSafeletChan == SafeletMethChan \cup \{\!| end\_safelet\_app |\!\}$

process $MainSafelet \widehat{=}$
$(SafeletFW \llbracket MainSafeletChan \rrbracket MainSafeletApp) \setminus MainSafeletChan$

exposes $\{\!| start\_sequencer, done\_sequencer |\!\}$

4 Mission Sequencer

4.1 Channels

section $MissionSequencerChan$ **parents** $MissionSequencerFWChan, MissionSequencerMethChan$

Framework channels

section $MissionSequencerFWChan$ **parents** $scj\_prelude$

channel $start\_sequencer, done\_sequencer$

channel $end\_sequencer\_app$

channelset $MissionSequencerFWChan == \{\!| start\_sequencer, done\_sequencer, end\_sequencer\_app |\!\}$

Method channels

section $MissionSequencerMethChan$ **parents** $MissionId, scj\_prelude$

channel $getNextMissionCall$

channel $getNextMissionRet : MissionId$

channelset $MissionSequencerMethChan == \{\!| getNextMissionCall, getNextMissionRet |\!\}$

4.2 Framework Process

section $MissionSequencerFW$ **parents** $MissionSequencerChan, MissionChan, scj\_prelude$

process $MissionSequencerFW \mathrel{\widehat=}$ **begin**

$Start \mathrel{\widehat=} start\_sequencer \longrightarrow \mathbf{Skip}$

$$
\begin{array}{l}
Execute \mathrel{\widehat=} \mu X \bullet getNextMissionCall \longrightarrow getNextMissionRet\,?\,next \longrightarrow \\
\quad \mathbf{if}\ next \neq nullMId \longrightarrow start\_mission\,.\,next \longrightarrow done\_mission\,.\,next \longrightarrow X \\
\quad [\!]\ next = nullMId \longrightarrow \mathbf{Skip} \\
\quad \mathbf{fi}
\end{array}
$$

$Finish \mathrel{\widehat=} end\_sequencer\_app \longrightarrow end\_mission\_fw \longrightarrow done\_sequencer \longrightarrow \mathbf{Skip}$

$\bullet\ Start\ ;\ Execute\ ;\ Finish$

end

4.3 Application Process

section *MainMissionSequencerApp* **parents** *MissionIds*, *scj_toolkit*, *scj_library*

process *MainMissionSequencerApp* $\widehat{=}$ **begin**

state *MainMissionSequencerState*

$mission\_done : boolean$

$Init \mathrel{\widehat=} mission\_done := jfalse$

$getNextMissionMeth \mathrel{\widehat=} getNextMissionCall \longrightarrow$
$\quad \mathbf{if}\ \neg\ mission\_done = jtrue \longrightarrow$
$\qquad (mission\_done := jtrue\ ;\ \ getNextMissionRet\,!\,MainMissionId \longrightarrow \mathbf{Skip})$
$\quad [\!]\ \neg\ (\neg\ mission\_done = jtrue) \longrightarrow getNextMissionRet\,!\,nullMId \longrightarrow \mathbf{Skip}$
$\quad \mathbf{fi}$

$Methods \mathrel{\widehat=} \mu X \bullet getNextMissionMeth\ ;\ \ X$

$\bullet\ Init\ ;\ \ (Methods \ \triangle\ end\_sequencer\_app \longrightarrow \mathbf{Skip})$

end

4.4 Composite Process

section *MainMissionSequencer* **parents** *MissionSequencerFW*, *MainMissionSequencerApp*

$\mathbf{channelset}\ MainMissionSequencerChan == MissionSequencerMethChan \cup \{\!|\ end\_sequencer\_app\ |\!\}$

$\mathbf{process}\ MainMissionSequencer \mathrel{\widehat=}$
$\quad (MissionSequencerFW \ [\![\ MainMissionSequencerChan\ ]\!]\ MainMissionSequencerApp) \setminus$
$\qquad MainMissionSequencerChan$

$\mathbf{exposes}\ \{\!|\ start\_sequencer, done\_sequencer, start\_mission, done\_mission, end\_mission\_fw\ |\!\}$

5 Mission

The life-cycle of a mission consists of first executing its `initialize()` method. It creates the event handlers as well as other data objects that are shared by the mission's handlers. The mission then enters its execution phase in which the event handlers become active, and are released in response to events being fired that are associated with the handlers. This continues until one of the handlers requests termination of the mission by calling `requestTermination()` on the current `Mission` object. The handlers are then stopped and the mission subsequently enters the cleanup phase by calling `cleanup()`. The mission subsequently terminates.

5.1 Channels

section *MissionChan* **parents** *MissionFWChan*, *MissionMethChan*

Framework channels

section *MissionFWChan* **parents** *MissionId*, *HandlerId*, *scj_prelude*

The events *start_mission* . m and *done_mission* . m are used to start mission m and wait for its termination.

channel *start_mission* : *MissionId*

channel *done_mission* : *MissionId*

The next channel is used to register an handler to be executed as part of the current mission. This takes place during execution of the `initialize()` method of the mission application process. Synchronisation on this channel corresponds to calling the `register()` method on a handler object.

channel *register* : *HandlerId*

The following two channels control handler execution. They are broadcast events, meaning that all handlers synchronise on them. First, *activate_handlers* is used to synchronously activate all handlers that have been started as part of the current mission. Next, *stop_handlers* is used to synchronously stop all executing handlers, namely as a consequence of calling the `requestTermination()` method of the mission class.

channel *activate_handlers*

channel *stop_handlers*

The channel below is used internally by the mission framework process. The *done_handlers* event is raised when all handlers have acknowledged termination, and hence no more calls to infrastructure methods have to be serviced and the execution phase thus is able to complete.

channel *done_handlers*

The last two channels are used to terminate the mission framework and application process, respectively.

channel *end_mission_fw*

channel *end_mission_app* : *MissionId*

Method channels

section *MissionMethChan* **parents** *scj_library*, *MissionId*

The following channels are declared for infrastructure methods of the `Mission` class.

channel *initializeCall*, *initializeRet* : *MissionId*

channel *cleanupCall*, *cleanupRet* : *MissionId*

channel *missionMemorySizeCall* : *MissionId*

channel *missionMemorySizeRet* : $MissionId \times \mathbb{N}$

channel *requestTerminationCall*

channel *requestTerminationRet*

channel *terminationPendingCall*

channel *terminationPendingRet* : *boolean*

For convenience, we introduce a channel set that includes all method channels.

channelset *MissionMethChan* ==
$\{\!|$ *initializeCall*, *initializeRet*, *cleanupCall*, *cleanupRet*,
missionMemorySizeCall, *missionMemorySizeRet*,
requestTerminationCall, *requestTerminationRet*,
terminationPendingCall, *terminationPendingRet* $|\!\}$

5.2 Framework Process

The framework process for mission execution is given below.

section $MissionFW$ **parents** $MissionId, MissionChan, HandlerChan, scj\_prelude$

process $MissionFW \mathrel{\widehat=}$ **begin**

state $State$

$mission : MissionId$
$handlers : \mathbb{F}\ HandlerId$
$terminating : boolean$

$Init$

$State'$

$mission' = nullMId$
$handlers' = \varnothing$
$terminating' = jfalse$

$Start \mathrel{\widehat=} Init\ ;\ start\_mission\,?\,m \longrightarrow mission := m$

$AddHandler \mathrel{\widehat=} \mathbf{val}\ handler : HandlerId \bullet handlers := handlers \cup \{handler\}$

$Initialize \mathrel{\widehat=} initializeCall\,.\,mission \longrightarrow$
$\quad \mu X \bullet (register\,?\,h \longrightarrow (AddHandler(h)\ ;\ X) \mathrel{\Box} initializeRet\,.\,mission \longrightarrow \mathbf{Skip})$

$StartHandlers \mathrel{\widehat=} \mathop{|||}\ h : handlers \bullet start\_handler\,.\,h \longrightarrow \mathbf{Skip}$

$StopHandlers \mathrel{\widehat=} \mathop{|||}\ h : handlers \bullet stop\_handler\,.\,h \longrightarrow done\_handler\,.\,h \longrightarrow \mathbf{Skip}$

$Execute \mathrel{\widehat=} (StartHandlers\ ;\ activate\_handlers \longrightarrow$
$\qquad stop\_handlers \longrightarrow StopHandlers\ ;\ done\_handlers \longrightarrow \mathbf{Skip})$
$\qquad [\![\varnothing \mid \{\!|\ stop\_handlers, done\_handlers\ |\!\} \mid \{terminating\}]\!]$
$\quad (Methods \mathbin{\triangle} done\_handlers \longrightarrow \mathbf{Skip})$

$Cleanup \mathrel{\widehat=} cleanupCall\,.\,mission \longrightarrow cleanupRet\,.\,mission \longrightarrow \mathbf{Skip}$

$Finish \mathrel{\widehat=} end\_mission\_app\,.\,mission \longrightarrow done\_mission\,.\,mission \longrightarrow \mathbf{Skip}$

$requestTerminationMeth \mathrel{\widehat=}$
$\quad requestTerminationCall \longrightarrow$
$\qquad \mathbf{if}\ terminating = jfalse \longrightarrow$
$\qquad\quad terminating := jtrue\ ;\ stop\_handlers \longrightarrow \mathbf{Skip}$
$\qquad [\!]\ terminating = jtrue \longrightarrow \mathbf{Skip}$
$\qquad \mathbf{fi}\ ;$
$\quad requestTerminationRet \longrightarrow \mathbf{Skip}$

$terminationPendingMeth \mathrel{\widehat=}$
$\quad terminationPendingCall \longrightarrow$
$\quad terminationPendingRet\,!\,terminating \longrightarrow \mathbf{Skip}$

$Methods \mathrel{\widehat=} \mu X \bullet (requestTerminationMeth \mathrel{\Box} terminationPendingMeth)\ ;\ X$

$\bullet\ (\mu X \bullet Start\ ;\ Initialize\ ;\ Execute\ ;\ Cleanup\ ;\ Finish\ ;\ X) \mathbin{\triangle} end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

5.3 Application Process

The application process for the `MainMission` class of the cruise controller is given below.

section *MainMissionApp* **parents** *MissionIds*, *HandlerIds*, *DataObjects*,
WheelShaft, *Engine*, *Brake*, *Gear*, *Lever*, *SpeedMonitor*, *ThrottleController*

$$
\begin{array}{l}
\mathbf{process}\ MainMissionApp \mathrel{\widehat=} \mathbf{begin} \\[4pt]
Init \mathrel{\widehat=} \mathbf{Skip} \\[4pt]
initializeMeth \mathrel{\widehat=} \\
\quad initializeCall\,.\,MainMissionId \longrightarrow \\
\quad (\mathbf{var}\ shaft : WheelShaftClass; \\
\qquad speedo : SpeedMonitorClass; \\
\qquad throttle : ThrottleControllerClass; \\
\qquad cruise : CruiseControlClass; \\
\qquad engine : EngineClass; \\
\qquad brake : BrakeClass; \\
\qquad gear : GearClass; \\
\qquad lever : LeverClass \bullet \\
\qquad shaft := \mathbf{new}\ WheelShaftClass; \\
\qquad WheelShaftInit\,!\,shaft \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,WheelShaftId \longrightarrow \mathbf{Skip}; \\
\qquad speedo := \mathbf{new}\ SpeedMonitorClass(shaft); \\
\qquad SpeedMonitorInit\,!\,speedo \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,SpeedMonitorId \longrightarrow \mathbf{Skip}; \\
\qquad throttle := \mathbf{new}\ ThrottleControllerClass(shaft, speedo); \\
\qquad ThrottleControllerInit\,!\,throttle \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,ThrottleControllerId \longrightarrow \mathbf{Skip}; \\
\qquad cruise := \mathbf{new}\ CruiseControlClass(throttle, speedo); \\
\qquad engine := \mathbf{new}\ EngineClass(cruise); \\
\qquad EngineInit\,!\,engine \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,EngineId \longrightarrow \mathbf{Skip}; \\
\qquad brake := \mathbf{new}\ BrakeClass(cruise); \\
\qquad BrakeInit\,!\,brake \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,BrakeId \longrightarrow \mathbf{Skip}; \\
\qquad gear := \mathbf{new}\ GearClass(cruise); \\
\qquad GearInit\,!\,gear \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,GearId \longrightarrow \mathbf{Skip}; \\
\qquad lever := \mathbf{new}\ LeverClass(cruise); \\
\qquad LeverInit\,!\,lever \longrightarrow \mathbf{Skip}; \\
\qquad register\,.\,LeverId \longrightarrow \mathbf{Skip}); \\
\quad initializeRet\,.\,MainMissionId \longrightarrow \mathbf{Skip} \\[4pt]
cleanupMeth \mathrel{\widehat=} \\
\quad cleanupCall\,.\,MainMissionId \longrightarrow \mathbf{Skip}; \\
\quad cleanupRet\,.\,MainMissionId \longrightarrow \mathbf{Skip} \\[4pt]
missionMemorySizeMeth \mathrel{\widehat=} \\
\quad missionMemorySizeCall\,.\,MainMissionId \longrightarrow \mathbf{Skip}; \\
\quad missionMemorySizeRet\,.\,MainMissionId\,!\,131072 \longrightarrow \mathbf{Skip} \\[4pt]
Methods \mathrel{\widehat=} \mu X \bullet (initializeMeth \mathrel{\Box} cleanupMeth \mathrel{\Box} missionMemorySizeMeth)\ ;\ X \\[4pt]
\bullet\ Init\ ;\ (Methods \mathrel{\triangle} end\_mission\_app\,.\,MainMissionId \longrightarrow \mathbf{Skip}) \\[4pt]
\mathbf{end}
\end{array}
$$

5.4 Composite Process

The composite model for the `MainMission` class of the cruise controller is given below.

$\textbf{section}\ \mathit{MainMission}\ \textbf{parents}\ \mathit{MissionFW}, \mathit{MainMissionApp}$

$$\begin{array}{l}\textbf{channelset}\ \mathit{MainMissionChan} == \\ \quad \{\!| \mathit{initializeCall}, \mathit{initializeRet}, \mathit{cleanupCall}, \mathit{cleanupRet}, \mathit{register}, \mathit{end\_mission\_app} |\!\}\end{array}$$

$$\begin{array}{l}\textbf{process}\ \mathit{MainMission} \mathrel{\widehat=} \\ \quad (\mathit{MissionFW} \llbracket \mathit{MainMissionChan} \rrbracket \mathit{MainMissionApp}) \setminus \mathit{MainMissionChan}\end{array}$$

$$\begin{array}{l}\textbf{exposes}\ \{\!| \mathit{start\_mission}, \mathit{done\_mission}, \\ \quad \mathit{activate\_handlers}, \mathit{stop\_handlers}, \mathit{done\_handlers}, \\ \quad \mathit{start\_handler}, \mathit{stop\_handler}, \mathit{done\_handler}, \\ \quad \mathit{requestTerminationCall}, \mathit{requestTerminationRet}, \\ \quad \mathit{terminationPendingCall}, \mathit{terminationPendingRet}, \\ \quad \mathit{missionMemorySizeCall}, \mathit{missionMemorySizeRet}, \\ \quad \mathit{end\_mission\_fw} |\!\}\end{array}$$

6 Event Handlers

In this section we present the framework model for both aperiodic and periodic event handlers.

6.1 Channels

We separately discuss the framework and method channels for handlers.

Framework channels

section $HandlerFWChan$ **parents** $HandlerId, scj\_prelude$

The $start\_handler$, $stop\_handler$ and $done\_handler$ channels are used by the mission to control handler execution. They are all parametrised in terms of a handler id. We thus require particular handler to synchronise on instance of those events for their underlying id, so that handlers can be independently controlled.

channel $start\_handler : HandlerId$

channel $stop\_handler : HandlerId$

channel $done\_handler : HandlerId$

At first, $start\_handler$ communicates to a handler that it should be started as part of the current mission. Note that the handler does not immediately start execution after synchronising on $start\_handler$ but waits for the $activate\_handlers$ broadcasting event. Secondly, $stop\_handler$ communicates that a handler should terminate as soon as possible. The $done\_handler$ event is subsequently raised by the handler to acknowledge to the mission that the handler has terminated.

The $enter\_dispatch$ and $leave\_dispatch$ events are used by the handler framework process to control the respective application process. They determine when a handler enters and leave its dispatch loop.

channel $enter\_dispatch$

channel $leave\_dispatch$

The $release\_handler$ event is generated internally for periodic event handlers to cause a release.

channel $release\_handler$

Method channels

section *HandlerMethChan* **parents** *HandlerId*, *EventId*, *scj_prelude*

We introduce a channels pair to model calls to `handleAsyncEvent()` in the `AperiodicEventHandler` class.

channel $handleAsyncEventCall : HandlerId$

channel $handleAsyncEventRet : HandlerId$

We moreover introduce a channel pair to model calls to `handleAsyncLongEvent(int value)` in the `AperiodicLongEventHandler` class.

channel $handleAsyncLongEventCall : HandlerId \times EventId$

channel $handleAsyncLongEventRet : HandlerId$

For convenience, we introduce a channel set that includes all method channels.

channelset $HandlerMethChan ==$
$\{\!|\ handleAsyncEventCall, handleAsyncEventRet,$
$handleAsyncLongEventCall, handleAsyncLongEventRet\ |\!\}$

Handler channels

The channel sets below are introduced to facilitate the definition of composite processes for handlers.

section *HandlerChan* **parents** *HandlerFWChan*, *HandlerMethChan*, *MissionChan*

channelset $HandlerAppSyncChan == \{\!|\ enter\_dispatch, leave\_dispatch, end\_mission\_fw\ |\!\}$

channelset $HandlerAppHideChan == \{\!|\ enter\_dispatch, leave\_dispatch\ |\!\}$

6.2 Framework Process

section $EventHandlerFW$ **parents** $MissionChan, HandlerChan$

process $EventHandlerFW \mathrel{\widehat=} handler : HandlerId \bullet$

begin

state $State$

$active : BOOL$

$Init$

$State'$

$active' = FALSE$

$StartHandler \mathrel{\widehat=} start\_handler\,.\,handler \longrightarrow active := TRUE$

$ActivateHandler \mathrel{\widehat=} activate\_handlers \longrightarrow \mathbf{Skip}$

$DispatchHandler \mathrel{\widehat=} enter\_dispatch \longrightarrow stop\_handler\,.\,handler \longrightarrow leave\_dispatch \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet Init\ ;\ ((StartHandler\ ;\ ActivateHandler) \mathrel{\Box} ActivateHandler);$
$\quad \mathbf{if}\ active = TRUE \longrightarrow DispatchHandler$
$\quad [\!]\ active = FALSE \longrightarrow \mathbf{Skip}$
$\quad \mathbf{fi}\ ;\ X) \bigtriangleup end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

7 Aperiodic Event Handlers

In this section we present the framework and application models for aperiodic event handlers of the cruise controller application.

7.1 Framework Processes

In this section we illustrates the instantiation of the framework process for the aperiodic event handlers of the cruise controller in order to obtain the models for particular aperiodic handlers.

7.1.1 WheelShaft

$\mathbf{section}\ \mathit{WheelShaftFW}\ \mathbf{parents}\ \mathit{EventHandler}, \mathit{HandlerIds}$

$\mathbf{process}\ \mathit{WheelShaftFW} \mathrel{\widehat=} \mathit{EventHandlerFW}(\mathit{WheelShaftId})$

7.1.2 Engine

$\mathbf{section}\ \mathit{EngineFW}\ \mathbf{parents}\ \mathit{EventHandler}, \mathit{HandlerIds}$

$\mathbf{process}\ \mathit{EngineFW} \mathrel{\widehat=} \mathit{EventHandlerFW}(\mathit{EngineId})$

7.1.3 Brake

$\mathbf{section}\ \mathit{BrakeFW}\ \mathbf{parents}\ \mathit{EventHandler}, \mathit{HandlerIds}$

$\mathbf{process}\ \mathit{BrakeFW} \mathrel{\widehat=} \mathit{EventHandlerFW}(\mathit{BrakeId})$

7.1.4 Gear

$\mathbf{section}\ \mathit{GearFW}\ \mathbf{parents}\ \mathit{EventHandler}, \mathit{HandlerIds}$

$\mathbf{process}\ \mathit{GearFW} \mathrel{\widehat=} \mathit{EventHandlerFW}(\mathit{GearId})$

7.1.5 Lever

$\mathbf{section}\ \mathit{LeverFW}\ \mathbf{parents}\ \mathit{EventHandler}, \mathit{HandlerIds}$

$\mathbf{process}\ \mathit{LeverFW} \mathrel{\widehat=} \mathit{EventHandlerFW}(\mathit{LeverId})$

7.2 Application Processes

In this section we illustrate the application model for aperiodic event handlers by defining a process for each aperiodic handler of the cruise controller. They all have very similar shapes.

7.2.1 WheelShaft

section *WheelShaftApp* **parents** *WheelShaftClass*, *WheelShaftMethChan*, *WheelShaftConst*, *HandlerIds*, *Events*

process $WheelShaftApp \mathrel{\widehat=}$ **begin**

state *WheelShaftState*
$this : WheelShaftClass$

$Init \mathrel{\widehat=} WheelShaftInit\,?\,obj \longrightarrow this :- obj$

$handleAsyncLongEventMeth \mathrel{\widehat=}$
$\quad \mathbf{val}\ evt : EventId \bullet this.handleAsyncLongEvent(evt)$

$Execute \mathrel{\widehat=} enter\_dispatch \longrightarrow Dispatch$

$Dispatch \mathrel{\widehat=} \mu X \bullet (leave\_dispatch \longrightarrow \mathbf{Skip}) \Box$
$\quad (wheel\_shaft \longrightarrow handleAsyncLongEventMeth(WheelShaftEvtId);$
$\qquad \mathbf{wait}\ 0..WheelShaftDeadline\ ;\ X)$

$Terminate \mathrel{\widehat=} done\_handler\,.\,WheelShaftId \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet Init\ ;\ Execute\ ;\ Terminate\ ;\ X) \triangle\ end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

Note The channels for methods calls to `void handleAsyncEvent()` and `void handleAsyncLongEvent(int value)` seem redundant with the most recent revision of the model. Furthermore, I have not encoded the *handlerAsyncLongEvent* method as an action here but data operations. What policy we adopt here is still subject to discussion. Clearly, however, since there no outputs a data operations is sufficient here.

7.2.2 Engine

section *EngineApp* **parents** *EngineClass*, *EngineMethChan*, *EngineConst*, *HandlerIds*, *Events*

process *EngineApp* $\widehat{=}$ **begin**

$$\begin{array}{|l}\textbf{state}\ \mathit{EngineState} \\ \hline \mathit{this} : \mathit{EngineClass} \\ \hline \end{array}$$

$\mathit{Init} \mathrel{\widehat{=}} \mathit{EngineInit} \mathbin{?} \mathit{obj} \longrightarrow \mathit{this} := \mathit{obj}$

$\mathit{handleAsyncLongEventMeth} \mathrel{\widehat{=}}$
$\quad \textbf{val}\ \mathit{evt} : \mathit{EventId} \bullet \mathit{this}.\mathit{handleAsyncLongEvent}(\mathit{evt})$

$\mathit{Execute} \mathrel{\widehat{=}} \mathit{enter\_dispatch} \longrightarrow \mathit{Dispatch}$

$$\begin{array}{l}\mathit{Dispatch} \mathrel{\widehat{=}} \mu X \bullet (\mathit{leave\_dispatch} \longrightarrow \textbf{Skip}) \mathrel{\Box} \\ \quad ((\mathit{engine\_on} \longrightarrow \mathit{handleAsyncLongEventMeth}(\mathit{EngineOnEvtId}) \mathrel{\Box} \\ \quad\ \ \mathit{engine\_off} \longrightarrow \mathit{handleAsyncLongEventMeth}(\mathit{EngineOffEvtId})); \\ \quad \textbf{wait}\ 0..\mathit{EngineDeadline} \mathbin{;}\ X)\end{array}$$

$\mathit{Terminate} \mathrel{\widehat{=}} \mathit{done\_handler} . \mathit{EngineId} \longrightarrow \textbf{Skip}$

$\bullet\ (\mu X \bullet \mathit{Init} \mathbin{;}\ \mathit{Execute} \mathbin{;}\ \mathit{Terminate} \mathbin{;}\ X) \mathbin{\triangle} \mathit{end\_mission\_fw} \longrightarrow \textbf{Skip}$

end

7.2.3 Brake

section *BrakeApp* **parents** *BrakeClass*, *BrakeMethChan*, *BrakeConst*, *HandlerIds*, *Events*

process *BrakeApp* $\widehat{=}$ **begin**

$$\begin{array}{|l}\textbf{state}\ \mathit{BrakeState} \\ \hline \mathit{this} : \mathit{BrakeClass} \\ \hline \end{array}$$

$\mathit{Init} \mathrel{\widehat{=}} \mathit{BrakeInit} \mathbin{?} \mathit{obj} \longrightarrow \mathit{this} := \mathit{obj}$

$\mathit{handleAsyncLongEventMeth} \mathrel{\widehat{=}}$
$\quad \textbf{val}\ \mathit{evt} : \mathit{EventId} \bullet \mathit{this}.\mathit{handleAsyncLongEvent}(\mathit{evt})$

$\mathit{Execute} \mathrel{\widehat{=}} \mathit{enter\_dispatch} \longrightarrow \mathit{Dispatch}$

$$\begin{array}{l}\mathit{Dispatch} \mathrel{\widehat{=}} \mu X \bullet (\mathit{leave\_dispatch} \longrightarrow \textbf{Skip}) \mathrel{\Box} \\ \quad ((\mathit{brake\_engaged} \longrightarrow \mathit{handleAsyncLongEventMeth}(\mathit{BrakeEngagedEvtId}) \mathrel{\Box} \\ \quad\ \ \mathit{brake\_disengaged} \longrightarrow \mathit{handleAsyncLongEventMeth}(\mathit{BrakeDisengagedEvtId})); \\ \quad \textbf{wait}\ 0..\mathit{BrakeDeadline} \mathbin{;}\ X)\end{array}$$

$\mathit{Terminate} \mathrel{\widehat{=}} \mathit{done\_handler} . \mathit{BrakeId} \longrightarrow \textbf{Skip}$

$\bullet\ (\mu X \bullet \mathit{Init} \mathbin{;}\ \mathit{Execute} \mathbin{;}\ \mathit{Terminate} \mathbin{;}\ X) \mathbin{\triangle} \mathit{end\_mission\_fw} \longrightarrow \textbf{Skip}$

end

7.2.4 Gear

section *GearApp* **parents** *GearClass*, *GearMethChan*, *GearConst*, *HandlerIds*, *Events*

process *GearApp* $\widehat{=}$ **begin**

state *GearState*
$this : GearClass$

$Init \mathrel{\widehat{=}} GearInit\,?\,obj \longrightarrow this :- obj$

$handleAsyncLongEventMeth \mathrel{\widehat{=}}$
$\quad \mathbf{val}\ evt : EventId \bullet this.handleAsyncLongEvent(evt)$

$Execute \mathrel{\widehat{=}} enter\_dispatch \longrightarrow Dispatch$

$$
\begin{aligned}
&Dispatch \mathrel{\widehat{=}} \mu X \bullet (leave\_dispatch \longrightarrow \mathbf{Skip}) \;\Box \\
&\quad ((top\_gear\_engaged \longrightarrow handleAsyncLongEventMeth(TopGearEngagedEvtId) \;\Box \\
&\quad\;\; top\_gear\_disengaged \longrightarrow handleAsyncLongEventMeth(TopGearDisengagedEvtId)); \\
&\quad \mathbf{wait}\ 0..GearDeadline \,;\; X)
\end{aligned}
$$

$Terminate \mathrel{\widehat{=}} done\_handler\,.\,GearId \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet Init\,;\; Execute\,;\; Terminate\,;\; X) \mathbin{\triangle} end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

7.2.5 Lever

section *LeverApp* **parents** *LeverClass*, *LeverMethChan*, *LeverConst*, *HandlerIds*, *Events*

process *LeverApp* $\widehat{=}$ **begin**

state *LeverState*
$this : LeverClass$

$Init \mathrel{\widehat{=}} LeverInit\,?\,obj \longrightarrow this :- obj$

$handleAsyncLongEventMeth \mathrel{\widehat{=}}$
$\quad \mathbf{val}\ evt : EventId \bullet this.handleAsyncLongEvent(evt)$

$Execute \mathrel{\widehat{=}} enter\_dispatch \longrightarrow Dispatch$

$$
\begin{aligned}
&Dispatch \mathrel{\widehat{=}} \mu X \bullet (leave\_dispatch \longrightarrow \mathbf{Skip}) \;\Box \\
&\quad ((activate \longrightarrow handleAsyncLongEventMeth(ActivateEvtId) \;\Box \\
&\quad\;\; deactivate \longrightarrow handleAsyncLongEventMeth(DeactivateEvtId) \;\Box \\
&\quad\;\; resume \longrightarrow handleAsyncLongEventMeth(ResumeEvtId) \;\Box \\
&\quad\;\; start\_acceleration \longrightarrow handleAsyncLongEventMeth(StartAccelerationEvtId) \;\Box \\
&\quad\;\; stop\_acceleration \longrightarrow handleAsyncLongEventMeth(StopAccelerationEvtId)); \\
&\quad \mathbf{wait}\ 0..LeverDeadline \,;\; X)
\end{aligned}
$$

$Terminate \mathrel{\widehat{=}} done\_handler\,.\,LeverId \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet Init\,;\; Execute\,;\; Terminate\,;\; X) \mathbin{\triangle} end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

7.3 Composite Processes

7.3.1 WheelShaft

section $WheelShaft$ **parents** $WheelShaftFW, WheelShaftApp$

process $WheelShaft \mathrel{\widehat=}$
$(WheelShaftFW \llbracket HandlerAppSyncChan \rrbracket WheelShaftApp) \setminus HandlerAppHideChan$

exposes $\{\!| wheel\_shaft,$
$start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw |\!\}$

7.3.2 Engine

section $Engine$ **parents** $EngineFW, EngineApp$

process $Engine \mathrel{\widehat=}$
$(EngineFW \llbracket HandlerAppSyncChan \rrbracket EngineApp) \setminus HandlerAppHideChan$

exposes $\{\!| engine\_on, engine\_off,$
$start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw |\!\}$

7.3.3 Brake

section $Brake$ **parents** $BrakeFW, BrakeApp$

process $Brake \mathrel{\widehat=}$
$(BrakeFW \llbracket HandlerAppSyncChan \rrbracket BrakeApp) \setminus HandlerAppHideChan$

exposes $\{\!| brake\_engaged, brake\_disengaged,$
$start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw |\!\}$

7.3.4 Gear

section $Gear$ **parents** $GearFW, GearApp$

process $Gear \mathrel{\widehat=}$
$(GearFW \llbracket HandlerAppSyncChan \rrbracket GearApp) \setminus HandlerAppHideChan$

exposes $\{\!| top\_gear\_engaged, top\_gear\_disengaged,$
$start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw |\!\}$

7.3.5 Lever

section $Lever$ **parents** $LeverFW, LeverApp$

process $Lever \mathrel{\widehat=}$
$(LeverFW \llbracket HandlerAppSyncChan \rrbracket LeverApp) \setminus HandlerAppHideChan$

exposes $\{\!| activate, deactivate, resume, start\_acceleration, stop\_acceleration,$
$start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw |\!\}$

8 Periodic Event Handlers

In this section we present the framework and application model for periodic event handlers.

8.1 Framework Processes

8.1.1 SpeedMonitor

section *SpeedMonitorFW* **parents** *EventHandler*, *HandlerIds*

$\mathbf{process}\ SpeedMonitorFW \mathrel{\widehat=} PeriodicEventHandlerFW(SpeedMonitorId)$

8.1.2 ThrottleController

section *ThrottleControllerFW* **parents** *EventHandler*, *HandlerIds*

$\mathbf{process}\ ThrottleControllerFW \mathrel{\widehat=} PeriodicEventHandlerFW(ThrottleControllerId)$

8.2 Application Processes

The application processes for handlers basically all have the same shape. They result from a lifting of the underlying data object. The latter is modelled by an Oh*Circus* class.

8.2.1 SpeedMonitor

section *SpeedMonitorApp* **parents** *SpeedMonitorClass*, *SpeedMonitorMethChan*, *SpeedMonitorConst*, *HandlerIds*

$\mathbf{process}\ SpeedMonitorApp \mathrel{\widehat=} \mathbf{begin}$

state *SpeedMonitorState*

$this : SpeedMonitorClass$

$Init \mathrel{\widehat=} SpeedMonitorInit\,?\,obj \longrightarrow this :- obj$

$handleAsyncEventMeth \mathrel{\widehat=} this.handleAsyncEvent()$

$$Execute \mathrel{\widehat=} enter\_dispatch \longrightarrow \\ (Dispatch \llbracket \{this\} \mid \{\!| release\_handler |\!\} \mid \varnothing \rrbracket\ Release) \setminus \{\!| release\_handler |\!\}$$

$$Dispatch \mathrel{\widehat=} \mu X \bullet (leave\_dispatch \longrightarrow \mathbf{Skip}) \,\Box \\ (release\_handler \longrightarrow handleAsyncEventMeth\ ;\ \mathbf{wait}\ 1..SpeedMonitorDeadline\ ;\ X)$$

$$Release \mathrel{\widehat=} (\mu X \bullet (release\_handler \longrightarrow \mathbf{Skip}) \blacktriangleleft 0\ ;\ \mathbf{wait}\ this.period\ ;\ X) \\ \triangle\ leave\_dispatch \longrightarrow \mathbf{Skip}$$

$Terminate \mathrel{\widehat=} done\_handler\,.\,SpeedMonitorId \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet Init\ ;\ Execute\ ;\ Terminate\ ;\ X) \triangle\ end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

8.2.2 ThrottleController

section $\mathit{ThrottleControllerApp}$ **parents** $\mathit{ThrottleControllerClass}, \mathit{ThrottleControllerMethChan}, \mathit{ThrottleControllerConst}, \mathit{HandlerIds}, \mathit{Events}$

process $\mathit{ThrottleControllerApp} \mathrel{\widehat=}$ **begin**

state $\mathit{ThrottleControllerState}$
$\mathit{this} : \mathit{ThrottleControllerClass}$

$\mathit{Init} \mathrel{\widehat=} \mathit{ThrottleControllerInit}\,?\,\mathit{obj} \longrightarrow \mathit{this} :- \mathit{obj}$

$\mathit{handleAsyncEventMeth} \mathrel{\widehat=} \mathit{this}.\mathit{handleAsyncEvent}()$

$\mathit{ActuatorEvents} \mathrel{\widehat=}$
$\quad \mathbf{if}\ (\mathit{this}).\mathit{schedule} = \mathit{jtrue} \longrightarrow \mathit{set\_voltage}\,!\,((\mathit{this}).\mathit{voltage}) \longrightarrow \mathbf{Skip}$
$\quad [\!]\ \neg\,(\mathit{this}).\mathit{schedule} = \mathit{jtrue} \longrightarrow \mathbf{Skip}\ \mathbf{fi}$

$\mathit{Execute} \mathrel{\widehat=} \mathit{enter\_dispatch} \longrightarrow$
$\quad (\mathit{Dispatch} \llbracket \{\mathit{this}\} \mid \{\!|\ \mathit{release\_handler}\ |\!\} \mid \varnothing \rrbracket\ \mathit{Release}) \setminus \{\!|\ \mathit{release\_handler}\ |\!\}$

$\mathit{Dispatch} \mathrel{\widehat=} \mu X \bullet (\mathit{leave\_dispatch} \longrightarrow \mathbf{Skip}) \Box$
$\quad (\mathit{release\_handler} \longrightarrow \mathit{handleAsyncEventMeth}\ ;\ \mathbf{wait}\ 1..\mathit{ThrottleControllerDeadline};$
$\qquad \mathit{ActuatorEvents}\ ;\ X)$

$\mathit{Release} \mathrel{\widehat=} (\mu X \bullet (\mathit{release\_handler} \longrightarrow \mathbf{Skip} \blacktriangleleft 0)\ ;\ \mathbf{wait}\ \mathit{this}.\mathit{period}\ ;\ X)$
$\quad \triangle\ \mathit{leave\_dispatch} \longrightarrow \mathbf{Skip}$

$\mathit{Terminate} \mathrel{\widehat=} \mathit{done\_handler}\,.\,\mathit{ThrottleControllerId} \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet \mathit{Init}\ ;\ \mathit{Execute}\ ;\ \mathit{Terminate}\ ;\ X) \triangle\ \mathit{end\_mission\_fw} \longrightarrow \mathbf{Skip}$

end

8.2.3 ThrottleController – Alternative 1

$$
\begin{array}{l}
\mathbf{section}\ \mathit{ThrottleControllerAlt1App}\ \mathbf{parents}\ \mathit{ThrottleControllerClass}, \mathit{ThrottleControllerMethChan},\\
\quad \mathit{ThrottleControllerConst}, \mathit{HandlerIds}, \mathit{Events}
\end{array}
$$

$$\mathbf{process}\ \mathit{ThrottleControllerApp} \mathrel{\widehat=} \mathbf{begin}$$

$$
\begin{array}{|l}
\mathbf{state}\ \mathit{ThrottleControllerState}\\
\hline
\mathit{this} : \mathit{ThrottleControllerClass}\\
\hline
\end{array}
$$

$$\mathit{Init} \mathrel{\widehat=} \mathit{ThrottleControllerInit}\,?\,\mathit{obj} \longrightarrow \mathit{this} :- \mathit{obj}$$

$$
\begin{array}{l}
\mathit{handleAsyncEventMeth} \mathrel{\widehat=}\\
\quad \mathbf{if}\ \mathit{this}.\mathit{schedule} = \mathit{jtrue} \longrightarrow\\
\quad\quad \mathbf{if}\ \mathit{this}.\mathit{accelerating} = \mathit{jtrue} \longrightarrow\\
\quad\quad\quad (\mathit{this}.\mathit{increaseVoltage}()\ ;\ \mathit{set\_voltage}\,!\,\mathit{this}.\mathit{voltage} \longrightarrow \mathbf{Skip})\\
\quad\quad [\!]\ \neg\,(\mathit{this}.\mathit{accelerating} = \mathit{jtrue}) \longrightarrow\\
\quad\quad\quad \mathbf{if}\ \mathit{this}.\mathit{maintainSpeed} = \mathit{jtrue} \longrightarrow\\
\quad\quad\quad\quad \mathbf{if}\ \mathit{this}.\mathit{cruiseSpeed} - \mathit{this}.\mathit{speed}.\mathit{getCurrentSpeed}() > 2 \longrightarrow\\
\quad\quad\quad\quad\quad (\mathit{this}.\mathit{increaseVoltage}()\ ;\ \mathit{set\_voltage}\,!\,\mathit{this}.\mathit{voltage} \longrightarrow \mathbf{Skip})\\
\quad\quad\quad\quad [\!]\ \mathit{this}.\mathit{speed}.\mathit{getCurrentSpeed}() - \mathit{this}.\mathit{cruiseSpeed} < 2 \longrightarrow\\
\quad\quad\quad\quad\quad (\mathit{this}.\mathit{decreaseVoltage}()\ ;\ \mathit{set\_voltage}\,!\,\mathit{this}.\mathit{voltage} \longrightarrow \mathbf{Skip})\\
\quad\quad\quad\quad [\!]\ \neg\,(\mathit{this}.\mathit{cruiseSpeed} - \mathit{this}.\mathit{speed}.\mathit{getCurrentSpeed}() > 2) \land\\
\quad\quad\quad\quad\ \ \neg\,(\mathit{this}.\mathit{speed}.\mathit{getCurrentSpeed}() - \mathit{this}.\mathit{cruiseSpeed} < 2) \longrightarrow\\
\quad\quad\quad\quad\quad \mathbf{var}\ \mathit{volts} : \mathit{VOLTAGE\_RANGE} \bullet\\
\quad\quad\quad\quad\quad\quad \mathit{volts} := 2 * (\mathit{this}.\mathit{cruiseSpeed} - \mathit{this}.\mathit{speed}.\mathit{getCurrentSpeed}()) + 2;\\
\quad\quad\quad\quad\quad\quad \mathbf{if}\ \mathit{volts} > \mathit{this}.\mathit{voltage} \longrightarrow\\
\quad\quad\quad\quad\quad\quad\quad (\mathit{this}.\mathit{increaseVoltage}()\ ;\ \mathit{set\_voltage}\,!\,\mathit{this}.\mathit{voltage} \longrightarrow \mathbf{Skip})\\
\quad\quad\quad\quad\quad\quad [\!]\ \neg\,(\mathit{volts} > \mathit{this}.\mathit{voltage}) \longrightarrow\\
\quad\quad\quad\quad\quad\quad\quad (\mathit{this}.\mathit{decreaseVoltage}()\ ;\ \mathit{set\_voltage}\,!\,\mathit{this}.\mathit{voltage} \longrightarrow \mathbf{Skip})\\
\quad\quad\quad\quad\quad\quad \mathbf{fi}\\
\quad\quad\quad\quad \mathbf{fi}\\
\quad\quad\quad [\!]\ (\mathit{this}.\mathit{maintainSpeed} = \mathit{jtrue}) \longrightarrow \mathbf{Skip}\\
\quad\quad\quad \mathbf{fi}\\
\quad\quad \mathbf{fi}\\
\quad [\!]\ \neg\,(\mathit{this}.\mathit{schedule} = \mathit{jtrue}) \longrightarrow \mathbf{Skip}\\
\quad \mathbf{fi}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Execute} \mathrel{\widehat=} \mathit{enter\_dispatch} \longrightarrow\\
\quad (\mathit{Dispatch} \lbrack\!\lbrack\, \{\mathit{this}\} \mid \{\!|\, \mathit{release\_handler} \,|\!\} \mid \varnothing \,\rbrack\!\rbrack\, \mathit{Release}) \setminus \{\!|\, \mathit{release\_handler} \,|\!\}
\end{array}
$$

$$
\begin{array}{l}
\mathit{Dispatch} \mathrel{\widehat=} \mu X \bullet (\mathit{leave\_dispatch} \longrightarrow \mathbf{Skip}) \,\Box\\
\quad (\mathit{release\_handler} \longrightarrow \mathit{handleAsyncEventMeth}\ ;\ \mathbf{wait}\ 1..\mathit{ThrottleControllerDeadline}\ ;\ X)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Release} \mathrel{\widehat=} (\mu X \bullet (\mathit{release\_handler} \longrightarrow \mathbf{Skip} \blacktriangleleft 0)\ ;\ \mathbf{wait}\ \mathit{this}.\mathit{period}\ ;\ X)\\
\quad \triangle\ \mathit{leave\_dispatch} \longrightarrow \mathbf{Skip}
\end{array}
$$

$$\mathit{Terminate} \mathrel{\widehat=} \mathit{done\_handler}\,.\,\mathit{ThrottleControllerId} \longrightarrow \mathbf{Skip}$$

$$\bullet\ (\mu X \bullet \mathit{Init}\ ;\ \mathit{Execute}\ ;\ \mathit{Terminate}\ ;\ X) \triangle\ \mathit{end\_mission\_fw} \longrightarrow \mathbf{Skip}$$

$$\mathbf{end}$$

8.2.4 ThrottleController – Alternative 2

section *ThrottleControllerAlt1App* **parents** *ThrottleControllerClass*, *ThrottleControllerMethChan*, *ThrottleControllerConst*, *HandlerIds*, *Events*

process *ThrottleControllerApp* $\widehat{=}$ **begin**

state *ThrottleControllerState*
$this : ThrottleControllerClass$

$Init \mathrel{\widehat{=}} ThrottleControllerInit\,?\,obj \longrightarrow this :- obj$

$$
\begin{array}{l}
increaseVoltageMeth \mathrel{\widehat{=}} \\
\quad \mathbf{if}\ this.voltage < 80 \longrightarrow this.voltage := this.voltage + 1 \\
\quad [\!]\ \neg\ this.voltage < 80 \longrightarrow \mathbf{Skip}\ \mathbf{fi}\ ; \\
\quad set\_voltage\,!\,(this.voltage) \longrightarrow \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
decreaseVoltageMeth \mathrel{\widehat{=}} \\
\quad \mathbf{if}\ this.voltage > 0 \longrightarrow this.voltage := this.voltage - 1 \\
\quad [\!]\ \neg\ this.voltage > 0 \longrightarrow \mathbf{Skip}\ \mathbf{fi}\ ; \\
\quad set\_voltage\,!\,(this.voltage) \longrightarrow \mathbf{Skip}
\end{array}
$$

$$
\begin{array}{l}
handleAsyncEventMeth \mathrel{\widehat{=}} \\
\quad \mathbf{if}\ this.schedule = jtrue \longrightarrow \\
\quad\quad \mathbf{if}\ this.accelerating = jtrue \longrightarrow increaseVoltageMeth \\
\quad\quad [\!]\ \neg\ (this.accelerating = jtrue) \longrightarrow \\
\quad\quad\quad \mathbf{if}\ this.maintainSpeed = jtrue \longrightarrow \\
\quad\quad\quad\quad \mathbf{if}\ this.cruiseSpeed - this.speed.getCurrentSpeed() > 2 \longrightarrow increaseVoltageMeth \\
\quad\quad\quad\quad [\!]\ this.speed.getCurrentSpeed() - this.cruiseSpeed < 2 \longrightarrow \\
\quad\quad\quad\quad\quad decreaseVoltageMeth \\
\quad\quad\quad\quad [\!]\ \neg\ (this.cruiseSpeed - this.speed.getCurrentSpeed() > 2) \land \\
\quad\quad\quad\quad\ \ \neg\ (this.speed.getCurrentSpeed() - this.cruiseSpeed < 2) \longrightarrow \\
\quad\quad\quad\quad\quad \mathbf{var}\ volts : VOLTAGE\_RANGE \bullet \\
\quad\quad\quad\quad\quad\quad volts := 2 * (this.cruiseSpeed - this.speed.getCurrentSpeed()) + 2; \\
\quad\quad\quad\quad\quad\quad \mathbf{if}\ volts > this.voltage \longrightarrow increaseVoltageMeth \\
\quad\quad\quad\quad\quad\quad [\!]\ \neg\ (volts > this.voltage) \longrightarrow decreaseVoltageMeth \\
\quad\quad\quad\quad\quad\quad \mathbf{fi} \\
\quad\quad\quad\quad \mathbf{fi} \\
\quad\quad\quad [\!]\ (this.maintainSpeed = jtrue) \longrightarrow \mathbf{Skip} \\
\quad\quad\quad \mathbf{fi} \\
\quad\quad \mathbf{fi} \\
\quad [\!]\ \neg\ (this.schedule = jtrue) \longrightarrow \mathbf{Skip} \\
\quad \mathbf{fi}
\end{array}
$$

$$
\begin{array}{l}
Execute \mathrel{\widehat{=}} enter\_dispatch \longrightarrow \\
\quad (Dispatch \llbracket \{this\} \mid \{\!|\ release\_handler\ |\!\} \mid \varnothing \rrbracket\ Release) \setminus \{\!|\ release\_handler\ |\!\}
\end{array}
$$

$$
\begin{array}{l}
Dispatch \mathrel{\widehat{=}} \mu X \bullet (leave\_dispatch \longrightarrow \mathbf{Skip}) \ \Box \\
\quad (release\_handler \longrightarrow handleAsyncEventMeth\ ;\ \mathbf{wait}\ 1..ThrottleControllerDeadline\ ;\ X)
\end{array}
$$

$$
\begin{array}{l}
Release \mathrel{\widehat{=}} (\mu X \bullet (release\_handler \longrightarrow \mathbf{Skip} \blacktriangleleft 0)\ ;\ \mathbf{wait}\ this.period\ ;\ X) \\
\quad \triangle\ leave\_dispatch \longrightarrow \mathbf{Skip}
\end{array}
$$

$Terminate \mathrel{\widehat{=}} done\_handler\,.\,ThrottleControllerId \longrightarrow \mathbf{Skip}$

$\bullet\ (\mu X \bullet Init\ ;\ Execute\ ;\ Terminate\ ;\ X) \triangle\ end\_mission\_fw \longrightarrow \mathbf{Skip}$

end

8.3 Composite Processes

8.3.1 SpeedMonitor

section $SpeedMonitor$ **parents** $SpeedMonitorFW, SpeedMonitorApp$

process $SpeedMonitor \widehat{=}$
$(SpeedMonitorFW \llbracket HandlerAppSyncChan \rrbracket SpeedMonitorApp) \setminus HandlerAppHideChan$

exposes $\{\| start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw \|\}$

8.3.2 ThrottleController

section $ThrottleController$ **parents** $ThrottleControllerFW, ThrottleControllerApp$

process $ThrottleController \widehat{=}$
$(ThrottleControllerFW \llbracket HandlerMethChan \rrbracket ThrottleControllerApp) \setminus HandlerMethChan$

exposes $\{\| start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw \|\}$

9 Cruise Controller

This section contains definitions of constants and channels specific to the cruise controller example.

9.1 Constants

section *Constants* **parents** *scj_toolkit*, *scj_library*

The following constant defines the integer range for voltage outputs on the throttle.

$VOLTAGE\_RANGE == 0..80$

It corresponds to the physical range from 0 volts to 8 volts, hence one unit is 0.1 volts.

9.2 Events

Below we declare a channel for each sensor and actuator event of the cruise controller.

section *Events* **parents** *scj_toolkit*, *scj_library*, *EventIds*, *Constants*

Sensor Events

channel *wheel_shaft*

channel *engine_on*, *engine_off*

channel *brake_engaged*, *brake_disengaged*

channel *top_gear_engaged*, *top_gear_disengaged*

channel *activate*

channel *deactivate*

channel *resume*

channel *start_acceleration*

channel *stop_acceleration*

channelset *SensorEvents* == {|
wheel_shaft,
engine_on, *engine_off*,
brake_engaged, *brake_disengaged*,
top_gear_engaged, *top_gear_disengaged*,
activate, *deactivate*, *resume*, *start_acceleration*, *stop_acceleration* |}

Actuator Events

channel *set_voltage* : *VOLTAGE_RANGE*

channelset *ActuatorEvents* == {| *set_voltage* |}

Cumulative Events

channelset *Events* == *SensorEvents* $\cup$ *ActuatorEvents*

9.3 Mission Identifiers

Mission identifiers for the cruise controller application (we only have one mission).

section *MissionIds* **parents** *scj_toolkit*, *scj_library*

$$
\begin{array}{|l}
MainMissionId : MissionId \\
\hline
distinct \langle nullMId, MainMissionId \rangle
\end{array}
$$

9.4 Handler Identifiers

Handler identifiers for the cruise controller application.

section *HandlerIds* **parents** *scj_toolkit*, *scj_library*

$$
\begin{array}{|l}
WheelShaftId : HandlerId \\
EngineId : HandlerId \\
BrakeId : HandlerId \\
GearId : HandlerId \\
LeverId : HandlerId \\
ThrottleControllerId : HandlerId \\
SpeedMonitorId : HandlerId \\
\hline
distinct \langle WheelShaftId, \\
\quad EngineId, \\
\quad BrakeId, \\
\quad GearId, \\
\quad LeverId, \\
\quad ThrottleControllerId, \\
\quad SpeedMonitorId \rangle
\end{array}
$$

9.5 Event Identifiers

Event identifiers for the cruise controller application.

section *EventIds* **parents** *scj_toolkit*, *scj_library*

$$
\begin{array}{|l}
WheelShaftEvtId : EventId \\
EngineOnEvtId : EventId \\
EngineOffEvtId : EventId \\
BrakeEngagedEvtId : EventId \\
BrakeDisengagedEvtId : EventId \\
TopGearEngagedEvtId : EventId \\
TopGearDisengagedEvtId : EventId \\
ActivateEvtId : EventId \\
DeactivateEvtId : EventId \\
ResumeEvtId : EventId \\
StartAccelerationEvtId : EventId \\
StopAccelerationEvtId : EventId \\
\hline
distinct\ \langle WheelShaftEvtId, \\
\quad EngineOnEvtId, \\
\quad EngineOffEvtId, \\
\quad BrakeEngagedEvtId, \\
\quad BrakeDisengagedEvtId, \\
\quad TopGearEngagedEvtId, \\
\quad TopGearDisengagedEvtId, \\
\quad ActivateEvtId, \\
\quad DeactivateEvtId, \\
\quad ResumeEvtId, \\
\quad StartAccelerationEvtId, \\
\quad StopAccelerationEvtId \rangle
\end{array}
$$

9.6 Top-level Model

In this section we specify the top-level model of the entire cruise controller application. It is the composition of the safelet, mission sequencer, mission, as well as all handler components.

$\textbf{section}\ accs\ \textbf{parents}\ MainSafelet, MainMissionSequencer, MainMission,$
$\quad WheelShaft, Engine, Brake, Gear, Lever, SpeedMonitor, ThrottleController, CruiseControl$

Channel Sets

$\textbf{channelset}\ MissionSequencerSyncChan ==$
$\quad \{\!|\ start\_sequencer, done\_sequencer\ |\!\}$

$\textbf{channelset}\ MissionSyncChan ==$
$\quad \{\!|\ start\_mission, done\_mission, end\_mission\_fw\ |\!\}$

$\textbf{channelset}\ MissionHideChan ==$
$\quad \{\!|\ start\_mission, done\_mission\ |\!\}$

$\textbf{channelset}\ HandlersSyncChan ==$
$\quad \{\!|\ start\_handler, stop\_handler, done\_handler, activate\_handlers, end\_mission\_fw\ |\!\}$

$\textbf{channelset}\ HandlersHideChan ==$
$\quad \{\!|\ start\_handler, stop\_handler, done\_handler, activate\_handlers\ |\!\}$

Handlers Process

$\textbf{channelset}\ HandlerSyncChan ==$
$\quad \{\!|\ activate\_handlers, end\_mission\_fw\ |\!\}$

$$
\begin{array}{l}
\textbf{process}\ Handlers \mathrel{\widehat=} \\
\quad ((((((WheelShaft \\
\qquad [\![HandlerSyncChan]\!]\ Engine) \\
\qquad [\![HandlerSyncChan]\!]\ Brake) \\
\qquad [\![HandlerSyncChan]\!]\ Gear) \\
\qquad [\![HandlerSyncChan]\!]\ Lever) \\
\qquad [\![HandlerSyncChan]\!]\ SpeedMonitor) \\
\qquad [\![HandlerSyncChan]\!]\ ThrottleController)
\end{array}
$$

System Process

$\textbf{channelset}\ SystemMethChan ==$
$\quad SafeletMethChan \cup MissionSequencerMethChan \cup MissionMethChan \cup HandlerMethChan$

$\textbf{channelset}\ SystemHideChan == SystemMethChan \cup \{\!|\ end\_mission\_fw\ |\!\}$

$$
\begin{array}{l}
\textbf{process}\ accs \mathrel{\widehat=} \\
\quad ((((((MainSafelet \\
\qquad [\![MissionSequencerSyncChan]\!]\ MainMissionSequencer) \setminus MissionSequencerSyncChan) \\
\qquad [\![MissionSyncChan]\!]\ MainMission) \setminus MissionHideChan) \\
\qquad [\![HandlersSyncChan]\!]\ Handlers) \setminus HandlersHideChan) \setminus SystemHideChan
\end{array}
$$

10 Data Objects

In this section we give the Oh*Circus* class definitions for all data objects of the Cruise Controller case study. It also illustrates how the respective Java classes are translated into formal models.

Not all of the specification can be parsed at the moment due to limitations of CZT to understand Oh*Circus*. The parts that are not subjected to the parser are highlighted in dark red.

10.1 WheelShaft

$\mathbf{section}\ WheelShaftClass\ \mathbf{parents}\ scj\_toolkit, scj\_library$

Class Definition

$\mathbf{class}\ WheelShaftClassDef \mathrel{\widehat=} \mathbf{begin}$

$$\begin{array}{|l}
\textbf{state}\ WheelShaftState \\
\hline
\mathbf{private}\ count : long
\end{array}$$

Do the following two methods have to be synchronised?

$\mathbf{public}\ getCallibration \mathrel{\widehat=} \mathbf{res}\ result : \mathbb{N} \bullet result := 100$

$\mathbf{public}\ getCount \mathrel{\widehat=} \mathbf{res}\ result : long \bullet result := count$

The implementation here executes `getAndDecrementFireCount()` in order to obtain the increment for `count`.

$\mathbf{public}\,\mathbf{sync}\ handleAsyncEvent \mathrel{\widehat=} \mathbf{val}\ evt : EventId \bullet count := count + 1$

$\mathbf{end}$

Reference Type

$WheelShaftClass == WheelShaftState$

10.2 Engine

section *EngineClass* **parents** *scj_toolkit*, *scj_library*, *EventIds*, *CruiseControlClass*

Class Definition

class *EngineClassDef* $\widehat{=}$ **begin**

state *EngineState*

private *cruise* : *CruiseControlClass*

initial *EngineInit*

$EngineState'$

$cruise? : CruiseControlClass$

$cruise' = cruise?$

Since we pass a parameter below we rather use *handleAsyncLongEvent*.

public sync *handleAsyncLongEvent* $\widehat{=}$

val $event : EventId \bullet$

if $event = EngineOnEvtId \longrightarrow cruise.engineOn()$

$[\!]\, event = EngineOffEvtId \longrightarrow cruise.engineOff()$

fi

end

Reference Type

$EngineClass == EngineState$

10.3 Brake

section *BrakeClass* **parents** *scj_toolkit*, *scj_library*, *EventIds*, *CruiseControlClass*

Class Definition

class *BrakeClassDef* $\widehat{=}$ **begin**

state *BrakeState*

private *cruise* : *CruiseControlClass*

initial *BrakeInit*

$BrakeState'$

$cruise? : CruiseControlClass$

$cruise' = cruise?$

Since we pass a parameter below we rather use *handleAsyncLongEvent*.

public sync *handleAsyncLongEvent* $\widehat{=}$

val *event* : *EventId* $\bullet$

if $event = BrakeEngagedEvtId \longrightarrow cruise.brakeEngaged()$

$[\!]\, event = BrakeDisengagedEvtId \longrightarrow cruise.brakeDisengaged()$

fi

end

Reference Type

$BrakeClass == BrakeState$

10.4 Gear

section *GearClass* **parents** *scj_toolkit*, *scj_library*, *EventIds*, *CruiseControlClass*

Class Definition

class *GearClassDef* $\widehat{=}$ **begin**

state *GearState*

$$\textbf{private}\ cruise : CruiseControlClass$$

initial *GearInit*

$$GearState'$$
$$cruise? : CruiseControlClass$$
$$\rule{3cm}{0.4pt}$$
$$cruise' = cruise?$$

Since we pass a parameter below we rather use *handleAsyncLongEvent*.

$$
\begin{array}{l}
\textbf{public}\,\textbf{sync}\ handleAsyncLongEvent \ \widehat{=} \\
\quad \textbf{val}\ event : EventId \bullet \\
\qquad \textbf{if}\ event = TopGearEngagedEvtId \longrightarrow cruise.topGearEngaged() \\
\qquad [\!]\ event = TopGearDisengagedEvtId \longrightarrow cruise.topGearDisengaged() \\
\qquad \textbf{fi}
\end{array}
$$

end

Reference Type

$$GearClass == GearState$$

10.5 Lever

section *LeverClass* **parents** $scj\_toolkit, scj\_library, EventIds, CruiseControlClass$

Class Definition

class *LeverClassDef* $\widehat{=}$ **begin**

state *LeverState*

$$\textbf{private}\ cruise : CruiseControlClass$$

initial *LeverInit*

$$LeverState'$$
$$cruise? : CruiseControlClass$$
$$cruise' = cruise?$$

Since we pass a parameter below we rather use *handleAsyncLongEvent*.

$$
\begin{array}{l}
\textbf{public}\ \textbf{sync}\ handleAsyncLongEvent\ \widehat{=} \\
\quad \textbf{val}\ event : EventId \bullet \\
\quad\quad \textbf{if}\ event = ActivateEvtId \longrightarrow cruise.activate() \\
\quad\quad [\!]\ event = DeactivateEvtId \longrightarrow cruise.deactivate() \\
\quad\quad [\!]\ event = ResumeEvtId \longrightarrow cruise.resume() \\
\quad\quad [\!]\ event = StartAccelerationEvtId \longrightarrow cruise.startAcceleration() \\
\quad\quad [\!]\ event = StopAccelerationEvtId \longrightarrow cruise.stopAcceleration() \\
\quad\quad \textbf{fi}
\end{array}
$$

end

Reference Type

$$LeverClass == LeverState$$

10.6 SpeedMonitor

section *SpeedMonitorClass* **parents** *scj_toolkit*, *scj_library*, *WheelShaftClass*, *SpeedMonitorConst*

Class Definition

class *SpeedMonitorClassDef* $\widehat{=}$ **begin**

state *SpeedMonitorState*

private $shaft : WheelShaftClass$
private $numberRotations, lastNumberRotations : long$
private $calibration : int$
private $currentSpeed : int$
private $iterationsInOneHour : int$
private $cmInKilometer : int$
private $period : PERIOD$

initial *SpeedMonitorInit*

$SpeedMonitorState\,'$
$shaft? : WheelShaftClass$

$shaft' = shaft$
$lastNumberRotations' = (shaft?).getCount()$
$calibration = (shaft?).getCallibration()$
$currentSpeed' = 0$
$iterationsInOneOur = ((1000 \operatorname{div} period.getPeriod().getMilliseconds()) * 3600)$
$cmInKilometer' = 100000$
$period' = SpeedMonitorPeriod$

public $getCurrentSpeed \;\widehat{=}\;$ **res** $result : int \bullet result := currentSpeed$

public sync $handleAsyncEvent \;\widehat{=}$
$numberRotations := shaft.getCount();$
var $difference : long \bullet difference := numberRotations - lastNumberRotations;$
$currentSpeed := (difference * calibration * iterationsInOneHour) \operatorname{div} cmInKilometer;$
$lastNumberRotations := numberRotations$

end

Reference Type

$SpeedMonitorClass \;==\; SpeedMonitorState$

10.7 ThrottleController

section *ThrottleControllerClass* **parents** *scj_toolkit*, *scj_library*, *SpeedMonitorClass*

Class Definition

class *ThrottleControllerClassDef* $\mathrel{\widehat=}$ **begin**

state *ThrottleControllerState*

private *speed* : *SpeedMonitorClass*
private *accelerating* : *boolean*
private *maintainSpeed* : *boolean*
private *cruiseSpeed* : $\mathbb{N}$
private *voltage* : *int*
private *schedule* : *boolean*

initial *ThrottleControllerInit*

$ThrottleControllerState\,'$
$speed? : SpeedMonitorClass$

$speed' = speed?$
$accelerating' = jfalse$
$maintainSpeed' = jfalse$
$cruiseSpeed' = 0$
$voltage' = 0$
$schedule' = jtrue$

public sync *setCruiseSpeed* $\mathrel{\widehat=}$
(**val** $kph : \mathbb{N}$ $\bullet$
$cruiseSpeed := kph;$
$maintainSpeed := jtrue;$
$accelerating := jfalse)$

public sync *accelerate* $\mathrel{\widehat=}$ $accelerating := jtrue$

In the implementation the following two methods write out the voltage.

public sync *increaseVoltage* $\mathrel{\widehat=}$
if $voltage < 80 \longrightarrow voltage := voltage + 1$
$[\!] \neg\ voltage < 80 \longrightarrow$ **Skip fi**

public sync *decreaseVoltage* $\mathrel{\widehat=}$
if $voltage > 0 \longrightarrow voltage := voltage - 1$
$[\!] \neg\ voltage > 0 \longrightarrow$ **Skip fi**

public sync *schedulePeriodic* $\mathrel{\widehat=}$ $schedule := jtrue$

public sync *deschedulePeriodic* $\mathrel{\widehat=}$ $schedule := jfalse$

$$
\begin{array}{l}
\textbf{public}\,\textbf{sync}\ \mathit{handleAsyncEvent}\ \widehat{=} \\
\quad \textbf{if}\ \mathit{schedule} = \mathit{jtrue} \longrightarrow \\
\quad\quad \textbf{if}\ \mathit{accelerating} = \mathit{jtrue} \longrightarrow \mathit{increaseVoltage}() \\
\quad\quad [\!]\ \neg\ (\mathit{accelerating} = \mathit{jtrue}) \longrightarrow \\
\quad\quad\quad \textbf{if}\ \mathit{maintainSpeed} = \mathit{jtrue} \longrightarrow \\
\quad\quad\quad\quad \textbf{if}\ \mathit{cruiseSpeed} - \mathit{speed}.\mathit{getCurrentSpeed}() > 2 \longrightarrow \mathit{increaseVoltage}() \\
\quad\quad\quad\quad [\!]\ \mathit{speed}.\mathit{getCurrentSpeed}() - \mathit{cruiseSpeed} < 2 \longrightarrow \mathit{decreaseVoltage}() \\
\quad\quad\quad\quad [\!]\ \neg\ (\mathit{cruiseSpeed} - \mathit{speed}.\mathit{getCurrentSpeed}() > 2\ \vee \\
\quad\quad\quad\quad\quad \mathit{speed}.\mathit{getCurrentSpeed}() - \mathit{cruiseSpeed} < 2) \longrightarrow \\
\quad\quad\quad\quad\quad \textbf{var}\ \mathit{volts} : \mathit{VOLTAGE\_RANGE}\ \bullet \\
\quad\quad\quad\quad\quad\quad \mathit{volts} := 2 * (\mathit{cruiseSpeed} - \mathit{speed}.\mathit{getCurrentSpeed}()) + 2; \\
\quad\quad\quad\quad\quad\quad \textbf{if}\ \mathit{volts} > \mathit{voltage} \longrightarrow \mathit{increaseVoltage}() \\
\quad\quad\quad\quad\quad\quad [\!]\ \neg\ (\mathit{volts} > \mathit{voltage}) \longrightarrow \mathit{decreaseVoltage}() \\
\quad\quad\quad\quad\quad\quad \textbf{fi} \\
\quad\quad\quad\quad \textbf{fi} \\
\quad\quad\quad [\!]\ (\mathit{maintainSpeed} = \mathit{jtrue}) \longrightarrow \textbf{Skip} \\
\quad\quad\quad \textbf{fi} \\
\quad\quad \textbf{fi} \\
\quad [\!]\ \neg\ (\mathit{schedule} = \mathit{jtrue}) \longrightarrow \textbf{Skip} \\
\quad \textbf{fi} \\
\textbf{end}
\end{array}
$$

Reference Type

$$\mathit{ThrottleControllerClass} == \mathit{ThrottleControllerState}$$

10.8 CruiseControl

section *CruiseControlClass* **parents** *scj_toolkit*, *scj_library*,
ThrottleControllerClass, *SpeedMonitorClass*

Class Definition

class *CruiseControlClassDef* $\widehat{=}$ **begin**

state *CruiseControlState*

$engineActive : boolean$
$topGear : boolean$
$braking : boolean$
$accelerating : boolean$
$cruising : boolean$
$throttle : ThrottleControllerClass$
$throttleStarted : boolean$
$speed : SpeedMonitorClass$

initial *CruiseControlInit*

$CruiseControlState'$
$throttle? : ThrottleControllerClass$
$speed? : SpeedMonitorClass$

$engineActive' = jfalse$
$topGear' = jfalse$
$braking' = jfalse$
$accelerating' = jfalse$
$cruising' = jfalse$
$throttle' = throttle?$
$speed' = speed?$

public sync *activate* $\widehat{=}$

$\mathbf{if}\ engineActive = jtrue \land topGear = jtrue \land \neg\,(braking = jtrue) \longrightarrow$
$\quad cruising := jtrue;$
$\quad throttle.setCruiseSpeed(speed.getCurrentSpeed());$
$\quad \mathbf{if}\ throttleStarted = jtrue \longrightarrow throttle.schedulePeriodic()$
$\quad [\!]\ \neg\,(throttleStarted = jtrue) \longrightarrow$
$\qquad throttleStarted := jtrue;$
$\qquad throttle.schedulePeriodic()$
$\quad \mathbf{fi}$
$[\!]\ \neg\,(engineActive = jtrue \land topGear = jtrue \land \neg\,(braking = jtrue)) \longrightarrow \mathbf{Skip}$
$\mathbf{fi}$

public sync *activate* $\widehat{=}$

$\mathbf{if}\ engineActive = jtrue \land topGear = jtrue \land \neg\,(braking = jtrue) \longrightarrow$
$\quad cruising := jtrue;$
$\quad throttle.setCruiseSpeed(speed.getCurrentSpeed());$
$\quad \mathbf{if}\ throttleStarted = jtrue \longrightarrow throttle.schedulePeriodic()$
$\quad [\!]\ \neg\,(throttleStarted = jtrue) \longrightarrow$
$\qquad throttleStarted := jtrue;$
$\qquad throttle.schedulePeriodic()$
$\quad \mathbf{fi}$
$[\!]\ \neg\,(engineActive = jtrue \land topGear = jtrue \land \neg\,(braking = jtrue)) \longrightarrow \mathbf{Skip}$
$\mathbf{fi}$

$\textbf{public}\,\textbf{sync}\ \mathit{deactivate}\ \widehat{=}$
$\quad\textbf{if}\ \mathit{engineActive} = \mathit{jtrue} \land \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \land$
$\qquad\qquad \mathit{cruising} = \mathit{jtrue} \longrightarrow$
$\qquad \mathit{cruising} := \mathit{jfalse};$
$\qquad \mathit{throttle}.\mathit{deschedulePeriodic}()$
$\quad [\!]\ \neg\ (\mathit{engineActive} = \mathit{jtrue} \land \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \land$
$\qquad\qquad \mathit{cruising} = \mathit{jtrue}) \longrightarrow \textbf{Skip}$
$\quad\textbf{fi}$

$\textbf{public}\,\textbf{sync}\ \mathit{startAcceleration}\ \widehat{=}$
$\quad\textbf{if}\ \mathit{engineActive} = \mathit{jtrue} \land \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \longrightarrow$
$\qquad \mathit{accelerating} = \mathit{jtrue};$
$\qquad \textbf{if}\ \mathit{throttleStarted} = \mathit{jtrue} \longrightarrow \mathit{throttle}.\mathit{schedulePeriodic}()$
$\qquad [\!]\ \neg\ (\mathit{throttleStarted} = \mathit{jtrue}) \longrightarrow$
$\qquad\quad \mathit{throttleStarted} := \mathit{jtrue};$
$\qquad\quad \mathit{throttle}.\mathit{schedulePeriodic}()$
$\qquad \textbf{fi}\,;$
$\qquad \mathit{throttle}.\mathit{accelerate}()$
$\quad [\!]\ \neg\ (\mathit{engineActive} = \mathit{jtrue} \land \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue})) \longrightarrow \textbf{Skip}$
$\quad\textbf{fi}$

$\textbf{public}\,\textbf{sync}\ \mathit{stopAcceleration}\ \widehat{=}$
$\quad\textbf{if}\ \mathit{engineActive} = \mathit{jtrue} \land \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \land$
$\qquad\qquad \mathit{accelerating} = \mathit{jtrue} \longrightarrow$
$\qquad \mathit{accelerating} := \mathit{jfalse};$
$\qquad \mathit{cruising} := \mathit{jtrue};$
$\qquad \mathit{throttle}.\mathit{setCruiseSpeed}(\mathit{speed}.\mathit{getCurrentSpeed}())$
$\quad [\!]\ \neg\ (\mathit{engineActive} = \mathit{jtrue} \land \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \land$
$\qquad\qquad \mathit{accelerating} = \mathit{jtrue}) \longrightarrow \textbf{Skip}$
$\quad\textbf{fi}$

$\textbf{public}\,\textbf{sync}\ \mathit{resume}\ \widehat{=}$
$\quad\textbf{if}\ \mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \land \mathit{throttleStarted} = \mathit{jtrue} \longrightarrow$
$\qquad \mathit{cruising} := \mathit{jtrue};$
$\qquad \mathit{throttle}.\mathit{schedulePeriodic}()$
$\quad [\!]\neg\ (\mathit{topGear} = \mathit{jtrue} \land \neg\ (\mathit{braking} = \mathit{jtrue}) \land \mathit{throttleStarted} = \mathit{jtrue}) \longrightarrow \textbf{Skip}$
$\quad\textbf{fi}$

$\textbf{public}\,\textbf{sync}\ \mathit{engineOn}\ \widehat{=}$
$\quad \mathit{engineActive} := \mathit{jtrue};$
$\quad \mathit{braking} := \mathit{jfalse};$
$\quad \mathit{topGear} := \mathit{jfalse};$
$\quad \mathit{cruising} := \mathit{jfalse}$

$\textbf{public}\,\textbf{sync}\ \mathit{engineOff}\ \widehat{=}$
$\quad \mathit{engineActive} := \mathit{jfalse};$
$\quad \mathit{braking} := \mathit{jfalse};$
$\quad \mathit{topGear} := \mathit{jfalse};$
$\quad \textbf{if}\ \mathit{cruising} = \mathit{jtrue} \longrightarrow$
$\qquad \mathit{cruising} := \mathit{jfalse};$
$\qquad \mathit{throttle}.\mathit{deschedulePeriodic}()$
$\quad [\!]\neg\ (\mathit{cruising} = \mathit{jtrue}) \longrightarrow \textbf{Skip}$
$\quad\textbf{fi}$

$\mathbf{public}\,\mathbf{sync}\ topGearEngaged\ \widehat{=}$
$\quad \mathbf{if}\ engineActive = jtrue \longrightarrow$
$\quad\quad topGear := jtrue$
$\quad [\!] \neg\,(engineActive = jtrue) \longrightarrow \mathbf{Skip}$
$\quad \mathbf{fi}$

$\mathbf{public}\,\mathbf{sync}\ topGearDisengaged\ \widehat{=}$
$\quad \mathbf{if}\ engineActive = jtrue \longrightarrow$
$\quad\quad topGear := jfalse;$
$\quad\quad \mathbf{if}\ cruising = jtrue \longrightarrow$
$\quad\quad\quad cruising := jfalse;$
$\quad\quad\quad throttle.deschedulePeriodic()$
$\quad\quad [\!] \neg\,(cruising = jtrue) \longrightarrow \mathbf{Skip}$
$\quad\quad \mathbf{fi}$
$\quad [\!] \neg\,(engineActive = jtrue) \longrightarrow \mathbf{Skip}$
$\quad \mathbf{fi}$

$\mathbf{public}\,\mathbf{sync}\ brakeEngaged\ \widehat{=}$
$\quad \mathbf{if}\ engineActive = jtrue \longrightarrow$
$\quad\quad \mathbf{if}\ cruising = jtrue \longrightarrow$
$\quad\quad\quad cruising := jfalse;$
$\quad\quad\quad throttle.deschedulePeriodic()$
$\quad\quad [\!] \neg\,(cruising = jtrue) \longrightarrow \mathbf{Skip}$
$\quad\quad \mathbf{fi}\,;$
$\quad\quad braking := jtrue$
$\quad [\!] \neg\,(engineActive = jtrue) \longrightarrow \mathbf{Skip}$
$\quad \mathbf{fi}$

$\mathbf{public}\,\mathbf{sync}\ brakeDisengaged\ \widehat{=}$
$\quad \mathbf{if}\ engineActive = jtrue \longrightarrow$
$\quad\quad braking := jfalse$
$\quad [\!] \neg\,(engineActive = jtrue) \longrightarrow \mathbf{Skip}$
$\quad \mathbf{fi}$

end

Due to the lack of support for Oh*Circus* in CZT the above Oh*Circus* methods cannot be type-checked.

Reference Type

$$CruiseControlClass\ ==\ CruiseControlState$$

The methods called by this class are:

- `ThrottleController:  void setCruiseSpeed(int speed)`
- `ThrottleController:  void accelerate()`
- `ThrottleController:  void schedulePeriodic()`
- `ThrottleController:  void deschedulePeriodic()`
- `SpeedMonitor:  int getCurrentSpeed()`