# Circus Model for the SCJ Mission Framework

Frank Zeyda

October 12, 2011

## Contents

1 Notation 3

2 Prelude 3
   2.1 Types 3
   2.2 Functions 3
   2.3 References 3
   2.4 Arrays 4
   2.5 OhCircus 4
   2.6 Circus Time 4

3 Safelet 5
   3.1 Channels 5
   3.2 Framework Process 5
   3.3 Application Process 5
   3.4 Composite Process 5

4 Mission Sequencer 6
   4.1 Channels 6
   4.2 Framework Process 6
   4.3 Application Process 7
   4.4 Composite Process 7

5 Mission 8
   5.1 Channels 8
   5.2 Framework Process 10
   5.3 Application Process 11
   5.4 Composite Process 12

6 Event Handlers 13
   6.1 Channels 13
   6.2 Framework Process 15

7 Aperiodic Event Handlers 16
   7.1 Framework Processes 16
      7.1.1 WheelShaft 16
      7.1.2 Engine 16
      7.1.3 Brake 16
      7.1.4 Gear 16
      7.1.5 Lever 16
7.2 Application Processes .......................................................... 17
  7.2.1 WheelShaft ................................................................. 17
  7.2.2 Engine ................................................................. 18
  7.2.3 Brake ................................................................. 18
  7.2.4 Gear ................................................................. 19
  7.2.5 Lever ................................................................. 19
7.3 Composite Processes .......................................................... 20
  7.3.1 WheelShaft ................................................................. 20
  7.3.2 Engine ................................................................. 20
  7.3.3 Brake ................................................................. 20
  7.3.4 Gear ................................................................. 20
  7.3.5 Lever ................................................................. 20

8 Periodic Event Handlers .......................................................... 21
  8.1 Framework Processes ..................................................... 21
    8.1.1 SpeedMonitor ..................................................... 21
    8.1.2 ThrottleController ................................................ 21
  8.2 Application Processes ..................................................... 21
    8.2.1 SpeedMonitor ..................................................... 21
    8.2.2 ThrottleController ................................................ 21
    8.2.3 ThrottleController – Alternative 1 .................................... 22
    8.2.4 ThrottleController – Alternative 2 .................................... 23
  8.3 Composite Processes ..................................................... 25
    8.3.1 SpeedMonitor ..................................................... 25
    8.3.2 ThrottleController ................................................ 25

9 Cruise Controller ................................................................. 26
  9.1 Constants ................................................................. 26
  9.2 Events ................................................................. 26
  9.3 Mission Identifiers ..................................................... 27
  9.4 Handler Identifiers ..................................................... 27
  9.5 Event Identifiers ..................................................... 27
  9.6 Top-level Model .......................................................... 29

10 Data Objects ................................................................. 30
  10.1 WheelShaft ................................................................. 30
  10.2 Engine ................................................................. 31
  10.3 Brake ................................................................. 32
  10.4 Gear ................................................................. 33
  10.5 Lever ................................................................. 34
  10.6 SpeedMonitor ........................................................... 35
  10.7 ThrottleController .................................................... 36
  10.8 CruiseControl .......................................................... 38
1 Notation

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

\begin{verbatim}
process MainSafelet \equiv (SafeletFW \parallel MainSafeletAppChan) \parallel MainSafeletAppChan
  exposes \{ start\_sequencer, done\_sequencer \}
\end{verbatim}

The \textit{exposes} clause emphasises that the process synchronises on the two channels \textit{start\_sequencer} and \textit{done\_sequencer}. The clause is solely for documentary purposes and does not have any semantic impact.

2 Prelude

Declaration of the \texttt{scj\_prelude} section.

\begin{verbatim}
section scj\_prelude parents circus\_toolkit
\end{verbatim}

2.1 Types

The type \textit{BOOL} is used for boolean values in Z at the level of specification.

\begin{verbatim}
BOOL ::= TRUE | FALSE
\end{verbatim}

2.2 Functions

The unary relation \textit{distinct} captures that the elements of a sequence are distinct.

\begin{verbatim}
relation (distinct \_)

distinct \_: P (iseq X)
\end{verbatim}

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 \textit{REF} that acts as the semantic domain for references over a certain value type. The given type \textit{REF\_TAG} is used as a ‘token type’ in its definition.

\begin{verbatim}
[REF\_TAG]
REF[X] \equiv X \times REF\_TAG
\end{verbatim}

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

\begin{verbatim}
generic (\_)
X \equiv REF[X]
\end{verbatim}

The referencing and dereferencing operators are loosely introduced below.

\begin{verbatim}
[X]
: X \rightarrow (X)
: (X) \rightarrow X
\end{verbatim}
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.

\[
\text{array}[X] \equiv (\lambda T : \mathbb{P} X \bullet (\text{seq } T))
\]

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

### 2.5 OhCircus

Below we provide support for the `new` keyword in OhCircus.

\[
\text{[X] new : \mathbb{P} X \rightarrow X}
\]

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

The OhCircus 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 OhCircus parser and type-checker in CZT.

### 2.6 Circus Time

Here we define types for absolute and relative time.

\[
\text{TIME} \equiv \mathbb{N}
\]

\[
\text{PERIOD} \equiv \mathbb{N}
\]
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 ≡ begin
SetUp ≡ setUpCall → setUpRet → Skip
Execute ≡ start_sequencer → done_sequencer → Skip
TearDown ≡ tearDownCall → tearDownRet → Skip
• SetUp ; Execute ; TearDown ; end_safelet_app → Skip
end

3.3 Application Process

section MainSafeletApp parents SafeletChan, scj_toolkit, scj_library
process MainSafeletApp ≡ begin
setUpMeth ≡ setUpCall → Skip ; setUpRet → Skip
tearDownMeth ≡ tearDownCall → Skip ; tearDownRet → Skip
Methods ≡ μX • (setUpMeth □ tearDownMeth) ; X
• Methods △ end_safelet_app → Skip
end

3.4 Composite Process

section MainSafelet parents SafeletFW, MainSafeletApp
channelset MainSafeletChan == SafeletMethChan ∪ { end_safelet_app }
process MainSafelet ≡
(SafeletFW || MainSafeletChan || MainSafeletApp) \ MainSafeletChan
exposes { start_sequencer, done_sequencer }
4 Mission Sequencer

4.1 Channels

\textbf{section} MissionSequencerChan \textbf{parents} MissionSequencerFWChan, MissionSequencerMethChan

\textbf{Framework channels}

\textbf{section} MissionSequencerFWChan \textbf{parents} scj\_prelude

channel start\_sequencer, done\_sequencer

channel end\_sequencer\_app

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

\textbf{Method channels}

\textbf{section} MissionSequencerMethChan \textbf{parents} MissionId, scj\_prelude

channel getNextMissionCall

channel getNextMissionRet : MissionId

channelset MissionSequencerMethChan == \{ getNextMissionCall, getNextMissionRet \}

4.2 Framework Process

\textbf{section} MissionSequencerFW \textbf{parents} MissionSequencerChan, MissionChan, scj\_prelude

\textbf{process} MissionSequencerFW \equiv \begin{align*}
\text{Start} & \equiv \text{start}\_\text{sequencer} \rightarrow \text{Skip} \\
\text{Execute} & \equiv \mu X \bullet \text{getNextMissionCall} \rightarrow \text{getNextMissionRet} ? \text{next} \rightarrow \\
& \quad \text{if} \next \neq \text{nullMId} \rightarrow \text{start}\_\text{mission} . \text{next} \rightarrow \text{done}\_\text{mission} . \text{next} \rightarrow X \\
& \quad \text{next} = \text{nullMId} \rightarrow \text{Skip} \\
\text{Finish} & \equiv \text{end}\_\text{sequencer}\_\text{app} \rightarrow \text{end}\_\text{mission}\_\text{fw} \rightarrow \text{done}\_\text{sequencer} \rightarrow \text{Skip}
\end{align*}

\bullet Start ; Execute ; Finish

end
4.3 Application Process

section MainMissionSequencerApp parents MissionIds, scj_toolkit, scj_library

process MainMissionSequencerApp ≡ begin
  state MainMissionSequencerState
  mission_done : boolean
  Init ≡ mission_done := jfalse
  getNextMissionMeth ≡ getNextMissionCall →
    if ¬ mission_done = jtrue →
      (mission_done := jtrue ; getNextMissionRet ! MainMissionId → Skip)
    fi
  getNextMissionRet ! MainMissionId → Skip
  Methods ≡ μX • getNextMissionMeth ; X
  • Init ; (Methods △ end_sequencer_app → Skip)
end

4.4 Composite Process

section MainMissionSequencer parents MainMissionSequencerFW, MainMissionSequencerApp

channelset MainMissionSequencerChan == MissionSequencerMethChan ∪ { end_sequencer_app }

process MainMissionSequencer ≡
  (MissionSequencerFW [ MainMissionSequencerChan ] MainMissionSequencerApp) \ MainMissionSequencerChan

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 × 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.

\[
\text{section MissionFW parents MissionId, MissionChan, HandlerChan, scj_prelude}
\]

\[
\text{process MissionFW} \triangleq \begin{array}{c}
\text{state State} \\
\text{mission : MissionId} \\
\text{handlers : \{HandlerId\}} \\
\text{terminating : boolean}
\end{array}
\]

\[
\text{Init} \triangleq \text{Init'}:
\begin{array}{c}
\text{mission'} = \text{nullMId} \\
\text{handlers'} = \emptyset \\
\text{terminating'} = \text{false}
\end{array}
\]

\[
\text{Start} \triangleq \text{Init} ;
\begin{array}{c}
\text{start_mission? } m \rightarrow \\
\text{mission := m}
\end{array}
\]

\[
\text{AddHandler} \triangleq \text{val handler : HandlerId} \bullet \text{handlers := handlers \cup \{handler\}}
\]

\[
\text{Initialize} \triangleq \text{initializeCall . mission} \
\begin{array}{c}
\mu X \bullet (\text{register? } h \rightarrow (\text{AddHandler}(h); X) \Box \text{initializeRet . mission} ightarrow \text{Skip})
\end{array}
\]

\[
\text{StartHandlers} \triangleq \begin{array}{c}
\text{h : handlers} \bullet \text{start_handler . h} \rightarrow \text{Skip}
\end{array}
\]

\[
\text{StopHandlers} \triangleq \begin{array}{c}
\text{h : handlers} \bullet \text{stop_handler . h} \rightarrow \text{done_handler . h} \rightarrow \text{Skip}
\end{array}
\]

\[
\text{Execute} \triangleq (\text{StartHandlers} ; \text{activate_handlers} \
\begin{array}{c}
\Box \text{stop_handlers} \rightarrow \text{StopHandlers} ; \text{done_handlers} \rightarrow \text{Skip}
\end{array}
\]

\[
\Box (\text{Methods} \triangle \text{done_handlers} \rightarrow \text{Skip})
\]

\[
\text{Cleanup} \triangleq \text{cleanupCall . mission} \
\begin{array}{c}
\text{cleanupRet . mission} \rightarrow \text{Skip}
\end{array}
\]

\[
\text{Finish} \triangleq \text{end_mission_app . mission} \
\begin{array}{c}
\text{done_mission . mission} \rightarrow \text{Skip}
\end{array}
\]

\[
\text{requestTerminationMeth} \triangleq \text{requestTerminationCall} \
\begin{array}{c}
\text{if terminating = \text{false} \rightarrow} \\
\text{terminating := \text{true} ; stop_handlers \rightarrow \text{Skip}}
\end{array}
\]

\[
\text{fi} ;
\]

\[
\text{requestTerminationRet} \rightarrow \text{Skip}
\]

\[
\text{terminationPendingMeth} \triangleq \text{terminationPendingCall} \
\begin{array}{c}
\text{terminationPendingRet ! terminating} \rightarrow \text{Skip}
\end{array}
\]

\[
\text{Methods} \triangleq \mu X \bullet (\text{requestTerminationMeth} \Box \text{terminationPendingMeth}) ; X
\]

\[
\bullet (\mu X \bullet \text{Start} ; \text{Initialize} ; \text{Execute} ; \text{Cleanup} ; \text{Finish} ; X) \triangle \text{end_mission_fw} \rightarrow \text{Skip}
\]

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

process MainMissionApp ≡ begin
Init ≡ Skip
initializeMeth ≡
  initializeCall . MainMissionId →
  (var shaft : WheelShaftClass;
   speedo : SpeedMonitorClass;
   throttle : ThrottleControllerClass;
   cruise : CruiseControlClass;
   engine : EngineClass;
   brake : BrakeClass;
   gear : GearClass;
   lever : LeverClass ●
   shaft := new WheelShaftClass;
   WheelShaftInit ! shaft → Skip;
   register . WheelShaftId → Skip;
   speedo := new SpeedMonitorClass(shaft);
   SpeedMonitorInit ! speedo → Skip;
   register . SpeedMonitorId → Skip;
   throttle := new ThrottleControllerClass(shaft, speedo);
   ThrottleControllerInit ! throttle → Skip;
   register . ThrottleControllerId → Skip;
   cruise := new CruiseControlClass(throttle, speedo);
   EngineInit ! engine → Skip;
   register . EngineId → Skip;
   brake := new BrakeClass(cruise);
   BrakeInit ! brake → Skip;
   register . BrakeId → Skip;
   gear := new GearClass(cruise);
   GearInit ! gear → Skip;
   register . GearId → Skip;
   lever := new LeverClass(cruise);
   LeverInit ! lever → Skip;
   register . LeverId → Skip);
initializeRet . MainMissionId → Skip

cleanupMeth ≡
cleanupCall . MainMissionId → Skip;
cleanupRet . MainMissionId → Skip

missionMemorySizeMeth ≡
  missionMemorySizeCall . MainMissionId → Skip;
  missionMemorySizeRet . MainMissionId ! 131072 → Skip

Methods ≡ μX ● (initializeMeth □ cleanupMeth □ missionMemorySizeMeth) ; X
  ● Init ; (Methods △ end_mission_app . MainMissionId → Skip)
end
5.4 Composite Process

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

```
section MainMission parents MissionFW, MainMissionApp

channelset MainMissionChan ==
    { initializeCall, initializeRet, cleanupCall, cleanupRet, register, end_mission_app }

process MainMission ≜
    (MissionFW [[ MainMissionChan || MainMissionApp ]] \ MainMissionChan)

exposes { start_mission, done_mission, activate Handlers, stop Handlers, done Handlers, start Handler, stop Handler, done Handler, requestTerminationCall, requestTerminationRet, terminationPendingCall, terminationPendingRet, missionMemorySizeCall, missionMemorySizeRet, end_mission_fw }
```
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 × 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 ≡ handler : HandlerId ●

begin

state State

active : BOOL

Init

State'

active' = FALSE

StartHandler ≡ start_handler . handler → active := TRUE

ActivateHandler ≡ activate_handlers → Skip

DispatchHandler ≡ enter_dispatch → stop_handler . handler → leave_dispatch → Skip

(μX ● Init ; ((StartHandler ; ActivateHandler) □ ActivateHandler); if active = TRUE → DispatchHandler
  [active = FALSE → Skip
  fin ; X) △ end_mission_fw → 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 illustrate 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

\[
\text{section WheelShaftFW parents EventHandler, HandlerIds}
\]

\[
\text{process WheelShaftFW } \equiv \text{EventHandlerFW(WheelShaftId)}
\]

7.1.2 Engine

\[
\text{section EngineFW parents EventHandler, HandlerIds}
\]

\[
\text{process EngineFW } \equiv \text{EventHandlerFW(EngineId)}
\]

7.1.3 Brake

\[
\text{section BrakeFW parents EventHandler, HandlerIds}
\]

\[
\text{process BrakeFW } \equiv \text{EventHandlerFW(BrakeId)}
\]

7.1.4 Gear

\[
\text{section GearFW parents EventHandler, HandlerIds}
\]

\[
\text{process GearFW } \equiv \text{EventHandlerFW(GearId)}
\]

7.1.5 Lever

\[
\text{section LeverFW parents EventHandler, HandlerIds}
\]

\[
\text{process LeverFW } \equiv \text{EventHandlerFW(LeverId)}
\]
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 ≜ begin
    state WheelShaftState
    this : WheelShaftClass

    Init ≜ WheelShaftInit ? obj → this := obj
    handleAsyncLongEventMeth ≜
        val evt : EventId • this.handleAsyncLongEvent(evt)
    Execute ≜ enter_dispatch → Dispatch
    Dispatch ≜ μX • (leave_dispatch → Skip) □
        (wheel_shaft → handleAsyncLongEventMeth(WheelShaftEvtId);
            wait 0. WheelShaftDeadline ; X)
    Terminate ≜ done_handler . WheelShaftId → Skip
    • (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → 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 ≜ begin

state EngineState

this : EngineClass

Init ≜ EngineInit ? obj → this : obj
handleAsyncLongEventMeth ≜
  val evt : EventId • this.handleAsyncLongEvent(evt)
Execute ≜ enter_dispatch → Dispatch
Dispatch ≜ μX • (leave_dispatch → Skip) □
  ((engine_on → handleAsyncLongEventMeth(EngineOnEvtId)) □
   engine_off → handleAsyncLongEventMeth(EngineOffEvtId));
   wait 0..EngineDeadline ; X)
Terminate ≜ done_handler . EngineId → Skip
• (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → Skip
end

7.2.3 Brake

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

process BrakeApp ≜ begin

state BrakeState

this : BrakeClass

Init ≜ BrakeInit ? obj → this : obj
handleAsyncLongEventMeth ≜
  val evt : EventId • this.handleAsyncLongEvent(evt)
Execute ≜ enter_dispatch → Dispatch
Dispatch ≜ μX • (leave_dispatch → Skip) □
  ((brake_engaged → handleAsyncLongEventMeth(BrakeEngagedEvtId)) □
   brake_disengaged → handleAsyncLongEventMeth(BrakeDisengagedEvtId));
   wait 0..BrakeDeadline ; X)
Terminate ≜ done_handler . BrakeId → Skip
• (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → Skip
end
7.2.4 Gear

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

process GearApp ≜ begin

  state GearState
  this : GearClass

  Init ≜ GearInit ? obj → this := obj
  handleAsyncLongEventMeth ≜
    val evt : EventId • this.handleAsyncLongEvent(evt)
  Execute ≜ enter_dispatch → Dispatch
  Dispatch ≜ μX • (leave_dispatch → Skip) □
    ((top_gear_engaged → handleAsyncLongEventMeth(TopGearEngagedEvtId) □
      top_gear_disengaged → handleAsyncLongEventMeth(TopGearDisengagedEvtId));
    wait 0..GearDeadline ; X)
  Terminate ≜ done_handler . GearId → Skip
  • (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → Skip
end

7.2.5 Lever

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

process LeverApp ≜ begin

  state LeverState
  this : LeverClass

  Init ≜ LeverInit ? obj → this := obj
  handleAsyncLongEventMeth ≜
    val evt : EventId • this.handleAsyncLongEvent(evt)
  Execute ≜ enter_dispatch → Dispatch
  Dispatch ≜ μX • (leave_dispatch → Skip) □
    ((activate → handleAsyncLongEventMeth(ActivateEvtId) □
      deactivate → handleAsyncLongEventMeth(DeactivateEvtId) □
      resume → handleAsyncLongEventMeth(ResumeEvtId) □
      start_acceleration → handleAsyncLongEventMeth(StartAccelerationEvtId) □
      stop_acceleration → handleAsyncLongEventMeth(StopAccelerationEvtId)));
    wait 0..LeverDeadline ; X)
  Terminate ≜ done_handler . LeverId → Skip
  • (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → Skip
end
7.3 Composite Processes

7.3.1 WheelShaft

section WheelShaft parents WheelShaftFW, WheelShaftApp

process WheelShaft $\equiv$

(WheelShaftFW || HandlerAppSyncChan || WheelShaftApp) \ HandlerAppHideChan

exposes $\{ \text{wheel}\_\text{shaft}, \text{start}\_\text{handler}, \text{stop}\_\text{handler}, \text{done}\_\text{handler}, \text{activate}\_\text{handlers}, \text{end}\_\text{mission}\_\text{fw} \}$

7.3.2 Engine

section Engine parents EngineFW, EngineApp

process Engine $\equiv$

(EngineFW || HandlerAppSyncChan || EngineApp) \ HandlerAppHideChan

exposes $\{ \text{engine}\_\text{on}, \text{engine}\_\text{off}, \text{start}\_\text{handler}, \text{stop}\_\text{handler}, \text{done}\_\text{handler}, \text{activate}\_\text{handlers}, \text{end}\_\text{mission}\_\text{fw} \}$

7.3.3 Brake

section Brake parents BrakeFW, BrakeApp

process Brake $\equiv$

(BrakeFW || HandlerAppSyncChan || BrakeApp) \ HandlerAppHideChan

exposes $\{ \text{brake}\_\text{engaged}, \text{brake}\_\text{disengaged}, \text{start}\_\text{handler}, \text{stop}\_\text{handler}, \text{done}\_\text{handler}, \text{activate}\_\text{handlers}, \text{end}\_\text{mission}\_\text{fw} \}$

7.3.4 Gear

section Gear parents GearFW, GearApp

process Gear $\equiv$

(GearFW || HandlerAppSyncChan || GearApp) \ HandlerAppHideChan

exposes $\{ \text{top}\_\text{gear}\_\text{engaged}, \text{top}\_\text{gear}\_\text{disengaged}, \text{start}\_\text{handler}, \text{stop}\_\text{handler}, \text{done}\_\text{handler}, \text{activate}\_\text{handlers}, \text{end}\_\text{mission}\_\text{fw} \}$

7.3.5 Lever

section Lever parents LeverFW, LeverApp

process Lever $\equiv$

(LeverFW || HandlerAppSyncChan || LeverApp) \ HandlerAppHideChan

exposes $\{ \text{activate}, \text{deactivate}, \text{resume}, \text{start}\_\text{acceleration}, \text{stop}\_\text{acceleration}, \text{start}\_\text{handler}, \text{stop}\_\text{handler}, \text{done}\_\text{handler}, \text{activate}\_\text{handlers}, \text{end}\_\text{mission}\_\text{fw} \}$
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

process SpeedMonitorFW ≜ PeriodicEventHandlerFW(SpeedMonitorId)

8.1.2 ThrottleController

section ThrottleControllerFW parents EventHandler, HandlerIds

process ThrottleControllerFW ≜ 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 OhCircus class.

8.2.1 SpeedMonitor

section SpeedMonitorApp parents SpeedMonitorClass, SpeedMonitorMethChan, SpeedMonitorConst, HandlerIds

process SpeedMonitorApp ≜ begin

  _state SpeedMonitorState
  this : SpeedMonitorClass

  Init ≜ SpeedMonitorInit ? obj → this := obj

  handleAsyncEventMeth ≜ this.handleAsyncEvent()

  Execute ≜ enter_dispatch →
         (Dispatch [ {this} | { release_handler } ] & Release) \ { release_handler }

  Dispatch ≜ μX • (leave_dispatch → Skip) □
            (release_handler → handleAsyncEventMeth ; wait 1..SpeedMonitorDeadline ; X)

  Release ≜ (μX • (release_handler → Skip) ◁ 0 ; wait this.period ; X)
            △ leave_dispatch → Skip

  Terminate ≜ done_handler . SpeedMonitorId → Skip

  • (μX • Init ; Execute ; Terminate ; X) △ end_mission-fw → Skip

end
8.2.2 ThrottleController

section ThrottleControllerApp parents ThrottleControllerClass, ThrottleControllerMethChan, ThrottleControllerConst, HandlerIds, Events

process ThrottleControllerApp ≡ begin

| state ThrottleControllerState |
| this : ThrottleControllerClass |

Init ≡ ThrottleControllerInit ? obj → this := obj
handleAsyncEventMeth ≡ this.handleAsyncEvent()

ActuatorEvents ≡
  if (this).schedule = jtrue → set_voltage!((this).voltage) → Skip
  [] (this).schedule = jtrue → Skip fi

Execute ≡ enter_dispatch →
  (Dispatch | {this} | {release_handler} | Release) \ {release_handler}

Dispatch ≡ μX (leave_dispatch → Skip) □
  (release_handler → handleAsyncEventMeth ; wait 1..ThrottleControllerDeadline ;
   ActuatorEvents ; X)

Release ≡ (μX (release_handler → Skip ◁ 0) ; wait this.period ; X)
  □ leave_dispatch → Skip

Terminate ≡ done_handler . ThrottleControllerId → Skip

• (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → Skip

end
8.2.3 ThrottleController – Alternative 1

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

process ThrottleControllerAlt1App ≡ begin

  state ThrottleControllerState
  this : ThrottleControllerClass

  Init ≡ ThrottleControllerInit ? obj → this := obj

  handleAsyncEventMeth ≡
  if this.schedule = jtrue →
    if this.accelerating = jtrue →
      (this.increaseVoltage() ; set_voltage ! this.voltage → Skip)
      ¬ (this.accelerating = jtrue) →
    if this.maintainSpeed = jtrue →
      if this.cruiseSpeed = this.speed.getCurrentSpeed() > 2 →
        (this.increaseVoltage() ; set_voltage ! this.voltage → Skip)
      if this.cruiseSpeed = this.speed.getCurrentSpeed() < 2 →
        (this.decreaseVoltage() ; set_voltage ! this.voltage → Skip)
      ¬ (this.cruiseSpeed = this.speed.getCurrentSpeed() > 2) ∧
      ¬ (this.speed.getCurrentSpeed() = this.cruiseSpeed < 2) →
      var volts : VOLTAGE_RANGE •
      volts := 2 * (this.cruiseSpeed = this.speed.getCurrentSpeed()) + 2;
      if volts > this.voltage →
        (this.increaseVoltage() ; set_voltage ! this.voltage → Skip)
      ¬ (volts > this.voltage) →
        (this.decreaseVoltage() ; set_voltage ! this.voltage → Skip)
    fi
  fi
  ¬ (this.schedule = jtrue) → Skip
  fi

  Execute ≡ enter_dispatch →
  (Dispatch [ {this} | {release_handler} ] | @ Release) \ {release_handler}

  Dispatch ≡ μX • (leave_dispatch → Skip) □
  (release_handler → handleAsyncEventMeth ; wait 1..ThrottleControllerDeadline ; X)

  Release ≡ (μX • (release_handler → Skip ▲ 0) ; wait this.period ; X)
  △ leave_dispatch → Skip

  Terminate ≡ done_handler . ThrottleControllerId → Skip

  • (μX • Init ; Execute ; Terminate ; X) △ end_mission_fw → Skip

end
8.2.4 ThrottleController – Alternative 2

section ThrottleControllerAlt1 App parents ThrottleControllerClass, ThrottleControllerMethChan, ThrottleControllerConst, HandlerIds, Events

process ThrottleControllerApp \(\equiv\) begin

\[\text{state ThrottleControllerState} \]
\[\text{this} : \text{ThrottleControllerClass} \]

\[\text{Init} \equiv \text{ThrottleControllerInit} \ ? \ 	ext{obj} \longrightarrow \text{this} := \text{obj} \]

\[\text{increaseVoltageMeth} \equiv \]
\[\text{if this.voltage} < 80 \longrightarrow \text{this.voltage} := \text{this.voltage} + 1 \]
\[\neg \text{this.voltage} < 80 \longrightarrow \text{Skip fi} ; \]
\[\text{set} \text{voltage} ! (\text{this.voltage}) \longrightarrow \text{Skip} \]

\[\text{decreaseVoltageMeth} \equiv \]
\[\text{if this.voltage} > 0 \longrightarrow \text{this.voltage} := \text{this.voltage} - 1 \]
\[\neg \text{this.voltage} > 0 \longrightarrow \text{Skip fi} ; \]
\[\text{set} \text{voltage} ! (\text{this.voltage}) \longrightarrow \text{Skip} \]

\[\text{handleAsyncEventMeth} \equiv \]
\[\text{if this.schedule} = \text{Jtrue} \longrightarrow \]
\[\text{if this.accelerating} = \text{Jtrue} \longrightarrow \text{increaseVoltageMeth} \]
\[\neg \ (\text{this.accelerating} = \text{Jtrue}) \longrightarrow \]
\[\text{if this.maintainSpeed} = \text{Jtrue} \longrightarrow \]
\[\text{if this.cruiseSpeed} - \text{this.speed.getCurrentSpeed()}) > 2 \longrightarrow \text{increaseVoltageMeth} \]
\[\neg \ (\text{this.speed.getCurrentSpeed()} - \text{this.cruiseSpeed} < 2) \longrightarrow \text{decreaseVoltageMeth} \]
\[\neg \ (\text{this.speed.getCurrentSpeed()} - \text{this.cruiseSpeed} < 2) \longrightarrow \]
\[\text{var} \text{volts} : \text{VOLTAGE\_RANGE} \bullet \]
\[\text{volts} := 2 \ast (\text{this.cruiseSpeed} - \text{this.speed.getCurrentSpeed()}) + 2; \]
\[\text{if volts} > \text{this.voltage} \longrightarrow \text{increaseVoltageMeth} \]
\[\neg \ (\text{volts} > \text{this.voltage}) \longrightarrow \text{decreaseVoltageMeth} \]
\[
\text{fi}
\]
\[\neg \ (\text{this.schedule} = \text{Jtrue}) \longrightarrow \text{Skip} \]
\[\text{fi}
\]
\[\neg \ (\text{this.maintainSpeed} = \text{Jtrue}) \longrightarrow \text{Skip} \]
\[\text{fi}

\[\text{Execute} \equiv \text{enter\_dispatch} \longrightarrow \]
\[\text{(Dispatch} \ [\{\text{this}\} | \{\text{release\_handler}\} | \emptyset] \text{Release}) \\setminus \{\text{release\_handler}\} \]

\[\text{Dispatch} \equiv \mu \text{X} \bullet (\text{leave\_dispatch} \longrightarrow \text{Skip}) \Box \]
\[\text{leave\_dispatch} \longrightarrow \text{AsyncEventMeth} ; \text{wait 1..ThrottleControllerDeadline} ; \text{X} \]

\[\text{Release} \equiv \mu \text{X} \bullet (\text{release\_handler} \longrightarrow \text{Skip} \bullet 0) ; \text{wait this.period} ; \text{X} \]
\[\triangle \text{leave\_dispatch} \longrightarrow \text{Skip} \]

\[\text{Terminate} \equiv \text{done\_handler } \text{.ThrottleControllerId} \longrightarrow \text{Skip} \]
\[\bullet (\mu \text{X} \bullet \text{Init} ; \text{Execute} ; \text{Terminate} ; \text{X}) \triangle \text{end\_mission\_fw} \longrightarrow \text{Skip} \]

end
8.3 Composite Processes

8.3.1 SpeedMonitor

section SpeedMonitor parents SpeedMonitorFW, SpeedMonitorApp

process SpeedMonitor ≡
(SpeedMonitorFW \ HandlerAppSyncChan \ SpeedMonitorApp) \ HandlerAppHideChan

exposes { start_handler, stop_handler, done_handler, activate_handlers, end_mission_FW }

8.3.2 ThrottleController

section ThrottleController parents ThrottleControllerFW, ThrottleControllerApp

process ThrottleController ≡
(ThrottleControllerFW \ HandlerMethChan \ ThrottleControllerApp) \ 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 ∪ ActuatorEvents
```
9.3 Mission Identifiers

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

section MissionIds parents scj_toolkit, scj_library

  MainMissionId : MissionId
  distinct ⟨nullMId, MainMissionId⟩

9.4 Handler Identifiers

Handler identifiers for the cruise controller application.

section HandlerIds parents scj_toolkit, scj_library

  WheelShaftId : HandlerId
  EngineId : HandlerId
  BrakeId : HandlerId
  GearId : HandlerId
  LeverId : HandlerId
  ThrottleControllerId : HandlerId
  SpeedMonitorId : HandlerId
  distinct ⟨WheelShaftId, EngineId, BrakeId, GearId, LeverId, ThrottleControllerId, SpeedMonitorId⟩

9.5 Event Identifiers

Event identifiers for the cruise controller application.
section EventIds parents scj_toolkit, scj_library

WheelShaftEvtId : EventId
EngineOnEvtId : EventId
EngineOffEvtId : EventId
BrakeEngagedEvtId : EventId
BrakeDisengagedEvtId : EventId
TopGearEngagedEvtId : EventId
TopGearDisengagedEvtId : EventId
ActivateEvtId : EventId
DeactivateEvtId : EventId
ResumeEvtId : EventId
StartAccelerationEvtId : EventId
StopAccelerationEvtId : EventId

distinct \{ WheelShaftEvtId,
    EngineOnEvtId,
    EngineOffEvtId,
    BrakeEngagedEvtId,
    BrakeDisengagedEvtId,
    TopGearEngagedEvtId,
    TopGearDisengagedEvtId,
    ActivateEvtId,
    DeactivateEvtId,
    ResumeEvtId,
    StartAccelerationEvtId,
    StopAccelerationEvtId \}
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.

section accs parents MainSafelet, MainMissionSequencer, MainMission, WheelShaft, Engine, Brake, Gear, Lever, SpeedMonitor, ThrottleController, CruiseControl

Channel Sets

channelset MissionSequencerSyncChan ==
{ start_sequencer, done_sequencer } 

channelset MissionSyncChan ==
{ start_mission, done_mission, end_mission/fw } 

channelset MissionHideChan ==
{ start_mission, done_mission } 

channelset HandlersSyncChan ==
{ start_handler, stop_handler, done_handler, activate_handlers, end_mission/fw } 

channelset HandlersHideChan ==
{ start_handler, stop_handler, done_handler, activate_handlers } 

Handlers Process

channelset HandlerSyncChan ==
{ activate_handlers, end_mission/fw } 

process Handlers ≜ ((((((WheelShaft

    ( HandlerSyncChan || Engine )
    ( HandlerSyncChan || Brake )
    ( HandlerSyncChan || Gear )
    ( HandlerSyncChan || Lever )
    ( HandlerSyncChan || SpeedMonitor )
    ( HandlerSyncChan || ThrottleController )})) } ) ) 

System Process

channelset SystemMethChan ==
SafeletMethChan ∪ MissionSequencerMethChan ∪ MissionMethChan ∪ HandlerMethChan 

channelset SystemHideChan == SystemMethChan ∪ { end_mission/fw } 

process accs ≜ (((((MainSafelet

    ( MissionSequencerSyncChan || MainMissionSequencer ) \ MissionSequencerSyncChan )
    ( MissionSyncChan || MainMission ) \ MissionHideChan )
    ( HandlersSyncChan || Handlers ) \ HandlersHideChan ) \ SystemHideChan) ) ) 

29
10 Data Objects

In this section we give the Oh\textit{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\textit{Circus}. The parts that are not subjected to the parser are highlighted in dark red.

10.1 WheelShaft

\texttt{section WheelShaftClass parents scj_toolkit, scj_library}

\begin{verbatim}
Class Definition

class WheelShaftClassDef = begin
  state WheelShaftState
  private count : long

Do the following two methods have to be synchronised?

public getCallibration = res result : \mathbb{N} • result := 100

public getCount = res result : long • result := count

The implementation here executes \texttt{getAndDecrementFireCount()} in order to obtain the increment for \texttt{count}.

public sync handleAsyncEvent = val evt : EventId • count := count + 1

end

Reference Type

WheelShaftClass == WheelShaftState
\end{verbatim}
10.2 Engine

section EngineClass parents scj_toolkit, scj_library, EventIds, CruiseControlClass

Class Definition

class EngineClassDef ≡ begin

  state EngineState

  private cruise : CruiseControlClass

  initial EngineInit

    EngineState'
    cruise? : CruiseControlClass

    cruise' = cruise?

end

Since we pass a parameter below we rather use handleAsyncLongEvent.

public sync handleAsyncLongEvent ≡

  val event : EventId •
    if event = EngineOnEvtId → cruise.engineOn()
    ▷ event = EngineOffEvtId → cruise.engineOff()
  fi

end

Reference Type

EngineClass == EngineState
10.3 Brake

section BrakeClass parents scj_toolkit, scj_library, EventIds, CruiseControlClass

Class Definition

class BrakeClassDef ≡ begin

  state BrakeState

  private cruise : CruiseControlClass

  initial BrakeInit

  BrakeState’

  cruise? : CruiseControlClass

  cruise’ = cruise?

end

Since we pass a parameter below we rather use handleAsyncLongEvent.

public sync handleAsyncLongEvent ≡

  val event : EventId •
    if event = BrakeEngagedEvtId → cruise.brakeEngaged()
    fi
    event = BrakeDisengagedEvtId → cruise.brakeDisengaged()

end

Reference Type

  BrakeClass == BrakeState
10.4 Gear

section GearClass parents scj_toolkit, scj_library, EventIds, CruiseControlClass

Class Definition

class GearClassDef ≡ begin

  state GearState

  private cruise : CruiseControlClass

  initial GearInit
  GearState'
  cruise? : CruiseControlClass
  cruise' = cruise?

Since we pass a parameter below we rather use handleAsyncLongEvent.

  public sync handleAsyncLongEvent ≡
  val event : EventId •
  if event = TopGearEngagedEvtId →→ cruise.topGearEngaged()
  □ event = TopGearDisengagedEvtId →→ cruise.topGearDisengaged()
  fi

end

Reference Type

  GearClass == GearState
10.5 Lever

section LeverClass parents scj_toolkit, scj_library, EventIds, CruiseControlClass

Class Definition

class LeverClassDef ≡ begin
  _state LeverState
  _initial LeverInit
    LeverState′
    cruise? : CruiseControlClass
    cruise′ = cruise?
end

Since we pass a parameter below we rather use handleAsyncLongEvent.

public sync handleAsyncLongEvent ≡
  val event : EventId •
    if event = ActivateEvtId → cruise.activate()
    event = DeactivateEvtId → cruise.deactivate()
    event = ResumeEvtId → cruise.resume()
    event = StartAccelerationEvtId → cruise.startAcceleration()
    event = StopAccelerationEvtId → cruise.stopAcceleration()
  fi
end

Reference Type

LeverClass == LeverState
10.6 SpeedMonitor

section SpeedMonitorClass parents scj_toolkit, scj_library, WheelShaftClass, SpeedMonitorConst

Class Definition

class SpeedMonitorClassDef ≡ 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
  iterationsInOneHour' = ((1000 div period.getPeriod().getMilliseconds()) * 3600)
  cmInKilometer' = 100000
  period' = SpeedMonitorPeriod

  public getCurrentSpeed ≡ res result : int • result := currentSpeed

  public sync handleAsyncEvent ≡
  numberRotations := shaft.getCount();
  var difference : long • difference := numberRotations − lastNumberRotations;
  currentSpeed := (difference * calibration * iterationsInOneHour) div cmInKilometer;
  lastNumberRotations := numberRotations

end

Reference Type

  SpeedMonitorClass == SpeedMonitorState
10.7 ThrottleController

section ThrottleControllerClass parents scj_toolkit, scj_library, SpeedMonitorClass

Class Definition

class ThrottleControllerClassDef ≜ begin

  _state ThrottleControllerState_

  private speed : SpeedMonitorClass
  private accelerating : boolean
  private maintainSpeed : boolean
  private cruiseSpeed : 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 ≜
  (val kph : N •
   cruiseSpeed := kph;
   maintainSpeed := jtrue;
   accelerating := jfalse)

public sync accelerate ≜ accelerating := jtrue

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

public sync increaseVoltage ≜
  if voltage < 80 → voltage := voltage + 1
  ¬voltage < 80 → Skip fi

public sync decreaseVoltage ≜
  if voltage > 0 → voltage := voltage - 1
  ¬voltage > 0 → Skip fi

public sync schedulePeriodic ≜ schedule := jtrue

public sync deschedulePeriodic ≜ schedule := jfalse
public sync handleAsyncEvent() {
    if (schedule = jtrue) {
        if (accelerating = jtrue) → increaseVoltage()
        ¬ (accelerating = jtrue) →
            if (maintainSpeed = jtrue) {
                if (cruiseSpeed - speed.getCurrentSpeed() > 2) → increaseVoltage()
                speed.getCurrentSpeed() - cruiseSpeed < 2 → decreaseVoltage()
            } else {
                var volts : VOLTAGE_RANGE •
                volts := 2 * (cruiseSpeed - speed.getCurrentSpeed()) + 2;
                if (volts > voltage) → increaseVoltage()
            } fi
        fi
    } fi

    if (maintainSpeed = jtrue) → Skip
    fi

    ¬ (schedule = jtrue) → Skip
    fi
}
end

Reference Type

ThrottleControllerClass == ThrottleControllerState
10.8 CruiseControl

section CruiseControlClass parents scj_toolkit, scj_library, ThrottleControllerClass, SpeedMonitorClass

Class Definition

class CruiseControlClassDef $\triangleq$ 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 $\triangleq$

  if engineActive = jtrue $\land$ topGear = jtrue $\land$ $\neg$ (braking = jtrue) $\rightarrow$
    cruising := jtrue;
    throttle.setCruiseSpeed(speed.getCurrentSpeed());
  fi

  if $\neg$ throttleStarted = jtrue $\rightarrow$ throttle.schedulePeriodic()
    throttleStarted := jtrue;
    throttle.schedulePeriodic()
  fi

  $\neg$ (engineActive = jtrue $\land$ topGear = jtrue $\land$ $\neg$ (braking = jtrue)) $\rightarrow$ Skip

public sync activate $\triangleq$

  if engineActive = jtrue $\land$ topGear = jtrue $\land$ $\neg$ (braking = jtrue) $\rightarrow$
    cruising := jtrue;
    throttle.setCruiseSpeed(speed.getCurrentSpeed());
  fi

  if throttleStarted = jtrue $\rightarrow$ throttle.schedulePeriodic()
    throttleStarted := jtrue;
    throttle.schedulePeriodic()
  fi

  $\neg$ (engineActive = jtrue $\land$ topGear = jtrue $\land$ $\neg$ (braking = jtrue)) $\rightarrow$ Skip
public sync deactivate \triangleq
  if \texttt{engineActive = jtrue} \land \texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue}) \land
  \neg (\texttt{cruising = jtrue}) \longrightarrow
  \neg (\texttt{cruising = jtrue}) \longrightarrow
  \neg (\texttt{cruising = jtrue}) \longrightarrow \texttt{Skip}
fi

public sync start Acceleration \triangleq
  if \texttt{engineActive = jtrue} \land \texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue}) \longrightarrow
  \texttt{accelerating = jtrue};
  if \texttt{throttleStarted = jtrue} \longrightarrow \texttt{throttle.schedulePeriodic()}
  \neg (\texttt{throttleStarted = jtrue}) \longrightarrow
  \texttt{throttleStarted = jtrue};
  \texttt{throttle.schedulePeriodic()}
fi;
  \texttt{throttle.accelerate()}
\neg (\texttt{engineActive = jtrue} \land \texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue})) \longrightarrow \texttt{Skip}
fi

public sync stop Acceleration \triangleq
  if \texttt{engineActive = jtrue} \land \texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue}) \land
  \texttt{accelerating = jtrue} \longrightarrow
  \texttt{accelerating = jfalse};
  \texttt{cruising = jtrue};
  \texttt{throttle.setCruiseSpeed(speed.getCurrentSpeed())}
\neg (\texttt{engineActive = jtrue} \land \texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue}) \land
  \texttt{accelerating = jtrue}) \longrightarrow \texttt{Skip}
fi

public sync resume \triangleq
  if \texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue}) \land \texttt{throttleStarted = jtrue} \longrightarrow
  \texttt{cruising = jtrue};
  \texttt{throttle.schedulePeriodic()}
\neg (\texttt{topGear = jtrue} \land \neg (\texttt{braking = jtrue}) \land \texttt{throttleStarted = jtrue}) \longrightarrow \texttt{Skip}
fi

public sync engineOn \triangleq
  \texttt{engineActive := jtrue};
  \texttt{braking := jfalse};
  \texttt{topGear := jfalse;}
  \texttt{cruising := jfalse}

public sync engineOff \triangleq
  \texttt{engineActive := jfalse;}
  \texttt{braking := jfalse;}
  \texttt{topGear := jfalse;}
  if \texttt{cruising = jtrue} \longrightarrow
  \texttt{cruising := jfalse};
  \texttt{throttle.deschedulePeriodic()}
\neg (\texttt{cruising = jtrue}) \longrightarrow \texttt{Skip}
fi

39
public sync topGearEngaged ≜
  if engineActive = jtrue →
    topGear := jtrue
  ↓ ¬ (engineActive = jtrue) → Skip
fi

public sync topGearDisengaged ≜
  if engineActive = jtrue →
    topGear := jfalse;
    if cruising = jtrue →
      cruising := jfalse;
      throttle.deschedulePeriodic()
      ↓ ¬ (cruising = jtrue) → Skip
    fi
  ↓ ¬ (engineActive = jtrue) → Skip
fi

public sync brakeEngaged ≜
  if engineActive = jtrue →
    if cruising = jtrue →
      cruising := jfalse;
      throttle.deschedulePeriodic()
      ↓ ¬ (cruising = jtrue) → Skip
    fi;
    braking := jtrue
  ↓ ¬ (engineActive = jtrue) → Skip
fi

public sync brakeDisengaged ≜
  if engineActive = jtrue →
    braking := jfalse
  ↓ ¬ (engineActive = jtrue) → Skip
fi

end

Due to the lack of support for OhCircus in CZT the above OhCircus 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()