A Simple Protocol
Safety-Critical Java Program and its *Circus* Model

Ana Cavalcanti, Andy Wellings, and Frank Zeyda

October 12, 2011

Contents

1 External Events

section *Events* **parents** *scj_toolkit*, *scj_library*

1.1 External Inputs

The *in* event is used to periodically communicate the next input, and the *out* even to request an output to be sent on the network.

channel $in : \mathbb{Z}$

channel *out*

1.2 External Outputs

The following channels are used to establish a connection to the network and send the data.

channel *enable*

channel $send : \mathbb{Z}$

channel *disable*

2 Framework Model

2.1 Safelet Framework Process

section $SafeletFW$ **parents** $SafeletChan, MissionSequencerChan, scj\_prelude$

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

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

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

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

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

end

2.2 Mission Sequencer Framework Process

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

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

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

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

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

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

end

2.3 Mission Framework Process

section *MissionFW* **parents** *MissionId*, *MissionChan*, *HandlerChan*, *scj_prelude*

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

state *State*

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

Init

$State'$

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

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

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

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

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

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

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

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

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

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

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

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

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

end

2.4 Event Handler Framework Process

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

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

begin

state *State*

$active : BOOL$

Init

$State'$

$active' = FALSE$

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

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

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

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

end

3 MainSafelet

3.1 Application Process

section $MainSafeletApp$ **parents** $scj\_toolkit, scj\_library$

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

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

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

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

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

end

3.2 Composite Process

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

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

process $MainSafelet \mathrel{\widehat=} (SafeletFW \ [\![\ MainSafeletChan\ ]\!]\ MainSafeletApp) \setminus MainSafeletChan$

3.3 Java Code

```
package jtres;

import javax.safetycritical.MissionSequencer;
import javax.safetycritical.Safelet;

public class MainSafelet implements Safelet {
   public void setUp() { }

   public MissionSequencer getSequencer() {
      /* Created in Immortal Memory */
      return new MainMissionSequencer();
   }

   public void tearDown() { }
}
```

4 MainMissionSequencer

4.1 Application Process

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

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

state *MainMissionSequencerState*

$mission\_done : boolean$

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

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

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

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

end

4.2 Composite Process

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

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

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

4.3 Java Code

```
package jtres;

import javax.realtime.PriorityParameters;

import javax.safetycritical.Mission;
import javax.safetycritical.MissionSequencer;
import javax.safetycritical.PriorityScheduler;
import javax.safetycritical.StorageParameters;

public class MainMissionSequencer extends MissionSequencer {
   public boolean mission_done;

   public MainMissionSequencer() {
      super(
         /* Let MainMissionSequencer run at max priority. */
         new PriorityParameters(
            PriorityScheduler.instance().getMaxPriority()),
         new StorageParameters(10000, 10000, 10000));
      mission_done = false;
   }

   public Mission getNextMission() {
      if (!mission_done) {
         mission_done = true;
         /* Created in Immortal Memory */
         return new MainMission();
      }
      else {
         return null;
      }
   }
}
```

5 MainMission

5.1 Application Process

section *MainMissionApp* **parents** *scj_toolkit*, *scj_library*, *Handler*1, *Handler*2, *List*, *MissionIds*, *HandlerIds*

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

state *MainMissionApp_State*

$list : List$

Below we do not explicitly pass the parameter objects of type *PriorityParameters*, *AperiodicParameters*, *PeriodicParameters*, and *StorageParameters*. Also we do not pass the *Aperiodic*[*Long*]*Event* object to which the handler is bound. We assume that all of this is determined by the implementation architecture.

$initializeMeth \mathrel{\widehat=} initializeCall\,.\,MainMissionId \longrightarrow$
  $list :-$ **new** $List$;
  (**var** $handler1 : Handler1Class \bullet$
    $handler1 :-$ **new** $Handler1Class(list)$;
    $Handler1Init\,.\,handler1 \longrightarrow$ **Skip**;
    $register\,.\,Handler1Id \longrightarrow$ **Skip**);
  (**var** $handler2 : Handler2Class \bullet$
    $handler2 :-$ **new** $Handler2Class(list)$;
    $Handler2Init\,.\,handler2 \longrightarrow$ **Skip**;
    $register\,.\,Handler2Id \longrightarrow$ **Skip**);
  $initializeRet\,.\,MainMissionId \longrightarrow$ **Skip**

$cleanupMeth \mathrel{\widehat=}$
  $cleanupCall\,.\,MainMissionId \longrightarrow$ **Skip**;
  $cleanupRet\,.\,MainMissionId \longrightarrow$ **Skip**

$missionMemorySizeMeth \mathrel{\widehat=}$
  $missionMemorySizeCall\,.\,MainMissionId \longrightarrow$ **Skip**;
  $missionMemorySizeRet\,.\,MainMissionId\,!\,131072 \longrightarrow$ **Skip**

$Methods \mathrel{\widehat=} \mu X \bullet (initializeMeth \;\Box\; cleanupMeth \;\Box\; missionMemorySizeMeth)\;;\; X$

$\bullet\ (Methods \;\triangle\; end\_mission\_app\,.\,MainMissionId \longrightarrow$ **Skip**)

end

5.2 Composite Process

section *MainMission* **parents** *MissionFW*, *MainMissionApp*

channelset *MainMissionChan* ==
  $\{\!|\ initializeCall, initializeRet, cleanupCall, cleanupRet, register, end\_mission\_app\ |\!\}$

process $MainMission \mathrel{\widehat=}$
  $(MissionFW \;[\![\ MainMissionChan\ ]\!]\; MainMissionApp) \setminus MainMissionChan$

5.3 Java Code

```
package jtres;

import javax.realtime.AbsoluteTime;
import javax.realtime.RelativeTime;
import javax.realtime.PriorityParameters;
import javax.realtime.PeriodicParameters;

import javax.safetycritical.AperiodicLongEvent;
import javax.safetycritical.Mission;
import javax.safetycritical.PriorityScheduler;
import javax.safetycritical.Services;
import javax.safetycritical.StorageParameters;

/* All objects in this class are created in mission memory. */

public class MainMission extends Mission {
   /* Elements of the Architecture */
   private Events events;
   private Interrupts interrupts;

   /* Mission Memory Variable */
   private List list;

   public void initialize() {
      /* Initialise the Architecture (Not in the Circus Model) */
      initArchitecture();

      /* Create Persistent Objects in MissionMemory */
      list = new List();

      /* Periodic Event Handler: Handler1 */
      Handler1 handler1 = new Handler1(list,
         new PriorityParameters(
            PriorityScheduler.instance().getNormPriority()),
         new PeriodicParameters(
            new AbsoluteTime(0, 0), new RelativeTime(100, 0)),
         new StorageParameters(4096, 4096, 4096));

      /* Register Handler */
      handler1.register();

      /* Aperiodic Event Handler: Handler2 */
      Handler2 handler2 = new Handler2(list,
         new PriorityParameters(
            PriorityScheduler.instance().getMinPriority() + 6),
         new StorageParameters(10000, 10000, 10000),
         events.out);

      /* Register Handler */
      handler2.register();
   }

   public void initArchitecture() {
      /* Create SCJ Events */
```

```
      createEvents();

      /* Create Interrupt Handlers */
      createInterrupts();
   }

   /* Create SCJ Events */
   public void createEvents() {
      events = new Events();
   }

   /* Create Interrupt Handlers */
   public void createInterrupts() {
      interrupts = new Interrupts(events);
      interrupts.register();
      interrupts.setPriorities();
   }

   public void cleanup() { }

   public long missionMemorySize() {
      return 131072;
   }
}
```

6 Handler1

6.1 Framework Process

section $Handler1FW$ **parents** $EventHandler, HandlerIds$

process $Handler1FW \mathrel{\widehat=} EventHandlerFW(Handler1Id)$

6.2 Application Process

section $Handler1App$ **parents** $Handler1Class, Handler1Chan, Handler1Const, HandlerIds, Events$

process $Handler1App \mathrel{\widehat=}$ **begin**

state $Handler1State$

$this : Handler1Class$

$Init \mathrel{\widehat=} Handler1Init\,?\,obj \longrightarrow this :\!- obj$

$handleAsyncEventMeth \mathrel{\widehat=}$
$\quad$ **val** $x : \mathbb{N} \bullet$ **wait** $1..Handler1Deadline\,;\ this.handleAsyncLongEvent(x)$

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

$Dispatch \mathrel{\widehat=} \mu X \bullet (leave\_dispatch \longrightarrow \mathbf{Skip}) \,\Box$
$\quad (release\_handler \longrightarrow (in\,?\,x \longrightarrow handleAsyncEventMeth(x) \blacktriangleleft 5)\,;\ X)$

$Release \mathrel{\widehat=} (\mu X \bullet (release\_handler \longrightarrow \mathbf{Skip} \blacktriangleleft 0)\,;$ **wait** $100\,;\ X) \triangle leave\_dispatch \longrightarrow \mathbf{Skip}$

$Terminate \mathrel{\widehat=} done\_handler\,.\,Handler1Id \longrightarrow \mathbf{Skip}$

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

end

6.3 Composite Process

section $Handler1$ **parents** $Handler1FW, Handler1App$

process $Handler1 \mathrel{\widehat=}$
$\quad (Handler1FW \llbracket HandlerAppSyncChan \rrbracket Handler1App) \setminus HandlerAppHideChan$

6.4 Data Object

section *Handler*1*Class* **parents** *scj_toolkit*, *scj_library*, *List*

class *Handler*1 $\widehat{=}$ **begin**

state *Handler*1_*State*

$list : List$

initial *Handler*1_*Init*

$Handler1\_State'$

$list? : List$

$list' = list?$

Although the program contains code to execute this method in mission memory, by virtue of an inner class that implements `Runnable`, we do not make this explicit in the statement below but use $newM()$ in the *List* class. This slightly reduces the tracability between the *Circus* model and Java code.

public *handleAsyncLongEvent* $\widehat{=}$ **val** $x : \mathbb{Z} \bullet list.insert(x)$

end

6.5 Java Code

```
package jtres;

import javax.realtime.AbsoluteTime;
import javax.realtime.RelativeTime;
import javax.realtime.PeriodicParameters;
import javax.realtime.PriorityParameters;
import javax.realtime.PriorityScheduler;
import javax.realtime.MemoryArea;
import javax.realtime.RawMemory;
import javax.realtime.RawInt;

import javax.safetycritical.PeriodicEventHandler;
import javax.safetycritical.StorageParameters;
import javax.safetycritical.MissionMemory;

public class Handler1 extends PeriodicEventHandler {
   /* Device Register */
   public final static long IN_DATA_REGISTER_ADDRESS = 0;

   private final RawInt in_data_register =
      RawMemory.createRawIntAccessInstance(
      RawMemory.IO_MEM_MAPPED, IN_DATA_REGISTER_ADDRESS);

   /* Shared Data in Mission Memory */
   private List list;

   public Handler1(List list,
      PriorityParameters priority,
      PeriodicParameters period,
      StorageParameters storage)  {
```

```
      super(priority, period, storage, "Handler1");
      this.list = list;
   }

   public void handleAsyncEvent() {
      /* Read input value from hardware here. */
      /* We assume the the reading is asynchronous (non-blocking). */
      int value = in_data_register.get();
      System.out.println("[Handler1] input " + value + " received");
      MissionMemory mission_memory =
         (MissionMemory) MemoryArea.getMemoryArea(this);
      mission_memory.executeInArea(new MissionMemoryEntry(value));
   }

   /* The insert() method allocates data and hence the call it must be carried
    * out in MissionMemory. */

   class MissionMemoryEntry implements Runnable {
      public int value;

      public MissionMemoryEntry(int value) {
         this.value = value;
      }

      public void run() {
         list.insert(value);
      }
   }
}
```

7 Handler2

7.1 Framework Process

section $Handler2FW$ **parents** $EventHandler, HandlerIds$

process $Handler2FW \mathrel{\widehat=} EventHandlerFW(Handler2Id)$

7.2 Application Process

section $Handler2App$ **parents** $Handler2Class, Handler2Chan, Handler2Const, HandlerIds, Events$

process $Handler2App \mathrel{\widehat=}$ **begin**

state $Handler2State$

$this : Handler2Class$

$Init \mathrel{\widehat=} Handler2Init\,?\,obj \longrightarrow this :- obj$

$handleAsyncEventMeth \mathrel{\widehat=}$
var $size : \mathbb{N} \bullet$
$size := this.size()\ ;$ **wait** $0..7;$
$enable \longrightarrow$ **Skip**;
$(send\,!\,size \longrightarrow$ **Skip**;
$disable \longrightarrow$ **Skip**$)\ \blacktriangleright\ 15$

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

$Dispatch \mathrel{\widehat=} \mu X \bullet (leave\_dispatch \longrightarrow$ **Skip**$)\ \Box$
$(out \longrightarrow handleAsyncEventMeth\ ;\ X)$

$Terminate \mathrel{\widehat=} done\_handler\,.\,Handler2Id \longrightarrow$ **Skip**

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

end

7.3 Composite Process

section $Handler2$ **parents** $Handler2FW, Handler2App$

process $Handler2 \mathrel{\widehat=}$
$(Handler2FW \llbracket HandlerAppSyncChan \rrbracket Handler2App) \setminus HandlerAppHideChan$

7.4 Data Object

section *Handler2Class* **parents** *scj_toolkit*, *scj_library*, *List*

class *Handler2* $\widehat{=}$ **begin**

state *Handler2_State*

$list : List$

initial *Handler2_Init*

$Handler2\_State'$
$list? : List$

$list' = list?$

The definition of the *handleAsyncEvent* method as a data operation is trivial here since we do not change the state of the object. As the behaviour of the method involves output communications, *handleAsyncEvent* has to be specified as an action in any case.

public *handleAsyncEvent* $\widehat{=}$ **Skip**

end

7.5 Java Code

```
package jtres;

import javax.realtime.*;

import javax.safetycritical.*;

public class Handler2 extends AperiodicEventHandler {
   /* Object to access the network (Resides in MissionMemory). */
   private Network network;

   /* Shared Data in Mission Memory */
   private List list;

   public Handler2(List list,
      PriorityParameters priority,
      StorageParameters storage,
      AperiodicEvent event) {
      super(priority, storage, event, "Handler2");
      /* Created in Mission Memory */
      Network network = new Network();

      /* Set ceiling to execute network methods at the highest priority. */
      Services.setCeiling(network,
         javax.safetycritical.PriorityScheduler.instance().getMaxPriority());
   }

   public void handleAsyncEvent() {
      int size = list.size();
      /* The following method calls all execute at a high priority. */
      /* We moreover consider their execution as instantaneous. */
      network.enable();
      network.send(size);
      network.disable();
   }
}
```

8 List

In this section I provide a model for the implementation of the List class in the abstract model. We would expect that it data-refines the *List* class specified in the paper. Note that none if it can be type-checked at the moment with the CZT *Circus* tools.

8.1 Data Object

section *List* **parents** *scj_toolkit*, *scj_library*

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

state *LState*

$val : int$
$next : List$
$empty : boolean$

public initial *init* $\widehat{=}$
$next := \mathbf{null}$;
$empty := jtrue$

Public Methods

The abstract $insert(\mathbb{Z})$ operation is implemented in terms of calls to two private (internal) synchronized methods $contains(int)$ and $clear()$. This is a choice made by the implementation.

public sync *insert* $\widehat{=}$ **val** $value : int \bullet$
if $size() = 50 \longrightarrow clear()$
$[] \neg size() = 50 \longrightarrow \mathbf{Skip}$
fi ;
if $contains(value) = jtrue \longrightarrow insert(x)$
$[] \neg contains(value) = jtrue \longrightarrow \mathbf{Skip}$
fi

public sync *size* $\widehat{=}$ **res** $ret : int \bullet$
var $size : int \bullet size := 0$;
var $node : List \bullet node :- \mathbf{this}$;
$\mu X \bullet$ **if** $\neg (node.empty = jtrue) \longrightarrow$
$(node := node.next \; ; \; size := size + 1)$
$[] \neg \neg (node.empty = jtrue) \longrightarrow \mathbf{Skip}$
fi ;
$ret := size$

Private Methods

private sync *contains* $\widehat{=}$ **val** *value* : *int*; **res** *ret* : *boolean* •
var *contains* := *jfalse*;
var *node* : *List* • *node* :− **this** ;
μX • **if** $\neg$ (*node.empty* = *jtrue*)$\longrightarrow$
if *node.val* = *value* $\longrightarrow$ *contains* := *jtrue*
$[]\neg$ (*node.val* = *value*) $\longrightarrow$ (*node* := *node.next* ; *X*)
fi ;
$[]\neg\neg$ (*node.empty* = *jtrue*) $\longrightarrow$ **Skip**
fi ;
ret := *contains*

Below we use the **newM** construct to create an object in mission memory. Similarity we have **newI**, **newR** and **newP** to create an object in immortal, per release and private memory, respectively.

private sync *append* $\widehat{=}$ **val** *value* : *int* •
var *node* : *List* • *node* :− **this** ;
μX • **if** $\neg$ (*node.empty* = *jtrue*)$\longrightarrow$
(*node* := *node.next*)
$[]\neg\neg$ (*node.empty* = *jtrue*) $\longrightarrow$ **Skip**
fi ;
node.val := *value*;
if *node.next* = **null** $\longrightarrow$*node.next* :− **newM** *List*
$[]\neg$ *node.next* = **null** $\longrightarrow$*node.next.empty* := *jtrue*
fi ;
node.empty := *jfalse*

private sync *clear* $\widehat{=}$ *empty* := *jtrue*

Logical Methods

Logical method to obtain the elements of the list as a set. May be needed for data refinement.

logical sync *elems* $\widehat{=}$ **res** *col* : $\mathbb{P}$ *int* •
if *node.empty* = *jtrue* $\longrightarrow$ *col* := $\varnothing$
$[]\neg$ *node.empty* = *jtrue* $\longrightarrow$ *col* := *next.elems*() $\cup$ {*val*}
fi

end

8.2 Java Code

```
package jtres;

/* A simple implementation of List. */

public class List {
   private int val;
   private List next;
   private boolean empty;

   /* Invariant: !node.empty => node.next != null. */

   /* However note that it is not an equivalence! */

   public List() {
      next = null;
      empty = true;
   }

   /* Public Methods (Specification) */

   /* The following method must execute within the deadline for Handler1. */

   public synchronized void insert(int value) {
      if (size() == 50) {
         clear();
      }
      if (!contains(value)) {
         append(value);
      }
   }

   /* The following method must execute within the deadline for Handler2. */

   public synchronized int size() {
      int size = 0;
      List node = this;
      while (!node.empty) {
         node = node.next;
         size++;
      }
      return size;
   }

   /* Private Methods (Internal) */

   private synchronized boolean contains(int value) {
      boolean contains = false;
      List node = this;
      while (!node.empty) {
         if (node.val == value) {
            contains = true;
         }
         node = node.next;
      }
```

```
      return contains;
   }

   private synchronized void append(int value) {
      List node = this;
      while (!node.empty) {
         node = node.next;
      }
      node.val = value;
      /* We cannot just overwrite node.next as memory is not reclaimed. */
      if (node.next == null) {
         node.next = new List();
      }
      else {
         /* Bug pointed out by Andy. We need to ignore the former list. */
         node.next.empty = true;
      }
      node.empty = false;
   }

   private synchronized void clear() {
      empty = true;
   }
}
```

9 System

section *System* **parents** *MainSafelet*, *MainMissionSequencer*, *MainMission*, *Handler*1, *Handler*2

Channel Sets

channelset *MissionSequencerSyncChan* ==
$\{\!|$ *start_sequencer*, *done_sequencer* $|\!\}$

channelset *MissionSyncChan* ==
$\{\!|$ *start_mission*, *done_mission*, *end_mission_fw* $|\!\}$

channelset *MissionHideChan* ==
$\{\!|$ *start_mission*, *done_mission* $|\!\}$

channelset *AllHandlersSyncChan* ==
$\{\!|$ *start_handler*, *stop_handler*, *done_handler*, *activate_handlers*, *end_mission_fw* $|\!\}$

channelset *AllHandlersHideChan* ==
$\{\!|$ *start_handler*, *stop_handler*, *done_handler*, *activate_handlers* $|\!\}$

Handlers Process

channelset *HandlerSyncChan* ==
$\{\!|$ *activate_handlers*, *end_mission_fw* $|\!\}$

process $Handlers \mathrel{\widehat=} (Handler1 \llbracket HandlerSyncChan \rrbracket Handler2)$

System Process

channelset *SystemMethChan* ==
$SafeletMethChan \cup MissionSequencerMethChan \cup MissionMethChan \cup HandlerMethChan$

channelset *SystemHideChan* ==
$SystemMethChan \cup \{\!|$ *end_mission_fw* $|\!\}$

process $System \mathrel{\widehat=} ((((((MainSafelet$
$\llbracket MissionSequencerSyncChan \rrbracket MainMissionSequencer) \setminus MissionSequencerSyncChan)$
$\llbracket MissionSyncChan \rrbracket MainMission) \setminus MissionHideChan)$
$\llbracket AllHandlersSyncChan \rrbracket Handlers) \setminus AllHandlersHideChan) \setminus SystemHideChan$