The Concurrent Missile VDM++ Model

Peter Gorm Larsen and Marcel Verhoef

2006

1   The World Class

class $World$
instance variables
  public static $env : [Environment] :=$ nil ;
  public static $timerRef : TimeStamp :=$ new $TimeStamp$ ();

operations
public

$$World : () \xrightarrow{o} World$$
$$World\,() \triangleq$$

($env := $ new $Environment\,(\texttt{"scenario.txt"})$;

$env.addSensor(FighterAircraft`sensor0)$;

$env.addSensor(FighterAircraft`sensor1)$;

$env.addSensor(FighterAircraft`sensor2)$;

$env.addSensor(FighterAircraft`sensor3)$;

FighterAircraft`controller0.$addDispenser(FighterAircraft`dispenser0)$;

FighterAircraft`controller0.$addDispenser(FighterAircraft`dispenser1)$;

FighterAircraft`controller0.$addDispenser(FighterAircraft`dispenser2)$;

FighterAircraft`controller0.$addDispenser(FighterAircraft`dispenser3)$;

FighterAircraft`detector.$addController(FighterAircraft`controller0)$;

FighterAircraft`controller1.$addDispenser(FighterAircraft`dispenser4)$;

FighterAircraft`controller1.$addDispenser(FighterAircraft`dispenser5)$;

FighterAircraft`controller1.$addDispenser(FighterAircraft`dispenser6)$;

FighterAircraft`controller1.$addDispenser(FighterAircraft`dispenser7)$;

FighterAircraft`detector.$addController(FighterAircraft`controller1)$;

FighterAircraft`controller2.$addDispenser(FighterAircraft`dispenser8)$;

FighterAircraft`controller2.$addDispenser(FighterAircraft`dispenser9)$;

FighterAircraft`controller2.$addDispenser(FighterAircraft`dispenser10)$;

FighterAircraft`controller2.$addDispenser(FighterAircraft`dispenser11)$;

FighterAircraft`detector.$addController(FighterAircraft`controller2)$;

start

(FighterAircraft`detector)

);

public

$$Run : () \xrightarrow{o} ()$$
$$Run\,() \triangleq$$

(

start

(env) ;

$env.isFinished()$;

FighterAircraft`detector.$isFinished()$;

$env.showResult()$

)

end $World$

Test Suite :   vdm.tc

Class :   World

2

Name	#Calls	Coverage
World'Run	1	√
World'World	1	√
Total Coverage		**100%**

2 The FighterAircraft Class

class *FighterAircraft*
instance variables

      public static $detector : MissileDetector :=$ new $MissileDetector\,();$
      public static $sensor0 : Sensor :=$ new $Sensor\,(detector, 0);$
      public static $sensor1 : Sensor :=$ new $Sensor\,(detector, 90);$
      public static $sensor2 : Sensor :=$ new $Sensor\,(detector, 180);$
      public static $sensor3 : Sensor :=$ new $Sensor\,(detector, 270);$
      public static $controller0 : FlareController :=$ new $FlareController\,(0);$
      public static $controller1 : FlareController :=$ new $FlareController\,(120);$
      public static $controller2 : FlareController :=$ new $FlareController\,(240);$
      public static $dispenser0 : FlareDispenser :=$ new $FlareDispenser\,(0);$
      public static $dispenser1 : FlareDispenser :=$ new $FlareDispenser\,(30);$
      public static $dispenser2 : FlareDispenser :=$ new $FlareDispenser\,(60);$
      public static $dispenser3 : FlareDispenser :=$ new $FlareDispenser\,(90);$
      public static $dispenser4 : FlareDispenser :=$ new $FlareDispenser\,(0);$
      public static $dispenser5 : FlareDispenser :=$ new $FlareDispenser\,(30);$
      public static $dispenser6 : FlareDispenser :=$ new $FlareDispenser\,(60);$
      public static $dispenser7 : FlareDispenser :=$ new $FlareDispenser\,(90);$
      public static $dispenser8 : FlareDispenser :=$ new $FlareDispenser\,(0);$
      public static $dispenser9 : FlareDispenser :=$ new $FlareDispenser\,(30);$
      public static $dispenser10 : FlareDispenser :=$ new $FlareDispenser\,(60);$
      public static $dispenser11 : FlareDispenser :=$ new $FlareDispenser\,(90);$

end *FighterAircraft*

3 The Environment Class

class *Environment* is subclass of *GLOBAL*
types

      public $inline = EventId \times MissileType \times Angle \times \mathbb{N};$
      public $outline = EventId \times FlareType \times Angle \times \mathbb{N} \times \mathbb{N}$
instance variables

$io : IO := $ new $IO\,();$
$inlines : inline^* := [\,];$
$busy : \mathbb{B} := $ true;
$outlines : outline^* := [\,];$
$ranges : \mathbb{N} \xrightarrow{m} (Angle \times Angle) := \{\mapsto\};$
$sensors : \mathbb{N} \xrightarrow{m} Sensor := \{\mapsto\};$
inv dom $ranges = $ dom $sensors$
$evid : [EventId] := $ nil ;

operations
public
$\qquad Environment : $ char$^* \xrightarrow{o} Environment$
$\qquad Environment\,(fname) \triangleq$
$\qquad\quad$ def mk- $(\text{-}, input) = io.freadval[inline^*]\,(fname)$ in
$\qquad\quad inlines := input;$
public
$\qquad addSensor : Sensor \xrightarrow{o} ()$
$\qquad addSensor\,(psens) \triangleq$
$\qquad\quad ($ dcl $id : \mathbb{N} := $ card dom $ranges + 1;$
$\qquad\qquad$ atomic $(\quad ranges := ranges \uplus \{id \mapsto psens.getAperature\,()\};$
$\qquad\qquad\qquad sensors := sensors \uplus \{id \mapsto psens\}$
$\qquad\qquad)$
$\qquad\quad);$
private
$\qquad createSignal : () \xrightarrow{o} [EventId]$
$\qquad createSignal\,() \triangleq$
$\qquad\quad ($ if len $inlines > 0$
$\qquad\qquad$ then $($ dcl $curtime : \mathbb{N} := World`timerRef.GetTime\,(),$
$\qquad\qquad\qquad done : \mathbb{B} := $ false;
$\qquad\qquad\qquad$ while $\neg\, done$
$\qquad\qquad\qquad$ do def mk- $(eventid, pmt, pa, pt) = $ hd $inlines$ in
$\qquad\qquad\qquad\quad$ if $pt \leq curtime$
$\qquad\qquad\qquad\quad$ then $(\quad$ for all $id \in $ dom $ranges$

4

```
                                      do def mk-(papplhs, pappsize) = ranges(id)
in
                                          if canObserve(pa, papplhs, pappsize)
                                          then sensors  (id) .trip(eventid, pmt, pa);
                                      inlines := tl inlines;
                                      done := len inlines = 0;
                                      return eventid
                              )
                      else (    done := true;
                                return nil
                              )
              )
          else (    busy := false;
                    return nil
              )
      );
public
      handleEvent : EventId × FlareType × Angle × ℕ × ℕ →ᵒ ()
      handleEvent (evid, pfltp, angle, pt1, pt2) ≜
      (    outlines := outlines ⌢ [mk-(evid, pfltp, angle, pt1, pt2)]
      );
public
      showResult : () →ᵒ ()
      showResult () ≜
        def - = io.writeval[outline*](outlines) in
        skip;
public
      isFinished : () →ᵒ ()
      isFinished () ≜
        skip;
      ResponseStarted : () →ᵒ ()
      ResponseStarted () ≜
        evid := nil
sync
      mutex(handleEvent);
      per isFinished ⇒ ¬ busy;
      per ResponseStarted ⇒ evid ∈ {out.#1 | out ∈ elems outlines}
thread

      while true
```

```
do (    if busy
        then (    evid := createSignal ()
              ) ;
        World'timerRef.NotifyAndIncTime()
    )
end Environment
```

Test Suite :   vdm.tc
Class :   FlareController

Name	#Calls	Coverage
Total Coverage		**1%**

4   The Global Class

```
class GLOBAL
values
public
        SENSOR-APERATURE = 90;
public
        FLARE-APERATURE = 120;
public
        DISPENSER-APERATURE = 30
types
        public MissileType = MISSILEA | MISSILEB | MISSILEC | NONE;
        public FlareType = FLAREONEA | FLARETWOA | DONOTHINGA |

                           FLAREONEB | FLARETWOB | DONOTHINGB |

                           FLAREONEC | FLARETWOC | DONOTHINGC;
        public Angle = ℕ
        inv num △ num < 360;
        public EventId = ℕ
operations
public
        canObserve : Angle × Angle × Angle →ᵒ 𝔹
        canObserve (pangle, pleft, psize) △
          def pright = (pleft + psize) mod 360 in
          if pright < pleft
          then return (pangle < pright ∨ pangle ≥ pleft)
          else return (pangle ≥ pleft ∧ pangle < pright) ;
```

6

public

$$getAperature : () \xrightarrow{o} Angle \times Angle$$
$$getAperature\,() \triangleq$$

   is subclass responsibility

end $GLOBAL$

   Test Suite :   vdm.tc

   Class :   GLOBAL

Name	#Calls	Coverage
GLOBAL'canObserve	77	$\sqrt{}$
GLOBAL'getAperature	0	0%
Total Coverage		**96%**

5   Sensor Class

class $Sensor$ is subclass of $GLOBAL$

instance variables

   private $detector : MissileDetector$;

   private $aperature : Angle$;

operations

public

$$Sensor : MissileDetector \times Angle \xrightarrow{o} Sensor$$
$$Sensor\,(pmd, psa) \triangleq$$
$$(\quad detector := pmd;$$
$$\qquad aperature := psa$$
$$)\,;$$

public

$$getAperature : () \xrightarrow{o} GLOBAL'Angle \times GLOBAL'Angle$$
$$getAperature\,() \triangleq$$

   return mk-$(aperature, SENSOR\text{-}APERATURE)$;

public

$$trip : EventId \times MissileType \times Angle \xrightarrow{o} ()$$
$$trip\,(evid, pmt, pa) \triangleq detector\,.$$

   $addThreat(evid, pmt, pa, World'timerRef\,.GetTime\,())$

   pre $canObserve\,(pa, aperature, SENSOR\text{-}APERATURE)$

end $Sensor$

   Test Suite :   vdm.tc

   Class :   Sensor

Name	#Calls	Coverage
Sensor'trip	7	61%
Sensor'Sensor	4	$\checkmark$
Sensor'getAperature	4	$\checkmark$
Total Coverage		**77%**

6   Missile Detector Class

class $MissileDetector$ is subclass of $GLOBAL$
instance variables
$\qquad ranges : \mathbb{N} \xrightarrow{m} (Angle \times Angle) := \{\mapsto\};$
$\qquad controllers : \mathbb{N} \xrightarrow{m} FlareController := \{\mapsto\};$
$\qquad$ inv dom $ranges =$ dom $controllers$
$\qquad threats : (EventId \times MissileType \times Angle \times \mathbb{N})^{*} := [];$
$\qquad busy : \mathbb{B} := $ false;

operations
public
$\qquad addController : FlareController \xrightarrow{o} ()$
$\qquad addController\,(pctrl) \triangleq$
$\qquad\quad (\quad$ dcl $nid : \mathbb{N} := $ card dom $ranges + 1;$
$\qquad\qquad$ atomic $(\quad ranges := ranges \uplus \{nid \mapsto pctrl.getAperature\,()\};$
$\qquad\qquad\qquad controllers := controllers \uplus \{nid \mapsto pctrl\}$
$\qquad\qquad) ;$

start
(pctrl)
$\qquad\qquad);$
public
$\qquad addThreat : EventId \times MissileType \times Angle \times \mathbb{N} \xrightarrow{o} ()$
$\qquad addThreat\,(evid, pmt, pa, pt) \triangleq$
$\qquad\quad (\quad threats := threats \frown [\text{mk-}\,(evid, pmt, pa, pt)];$
$\qquad\qquad busy := $ true
$\qquad\qquad);$
private
$\qquad getThreat : () \xrightarrow{o} EventId \times MissileType \times Angle \times \mathbb{N}$
$\qquad getThreat\,() \triangleq$
$\qquad\quad (\quad$ dcl $res : EventId \times MissileType \times Angle \times \mathbb{N} := $ hd $threats;$

8

```
                threats := tl threats;
                return res
            );
public
        isFinished : () --o--> ()
        isFinished () /\
            for all id ∈ dom controllers
            do controllers    (id ) .isFinished()
sync
        mutex(addThreat, getThreat);
        per getThreat  ⇒  len threats > 0;
        per isFinished  ⇒  ¬ busy
thread

        while true
        do (    def mk- (evid, pmt, pa, pt) = getThreat () in
                for all id ∈ dom ranges
                do def mk- (papplhs, pappsize) = ranges (id) in
                    if canObserve (pa, papplhs, pappsize)
                    then controllers    (id ) .addThreat(evid, pmt, pa, pt) ;
                busy := len threats > 0
            )
end MissileDetector
```

Test Suite :   vdm.tc
Class :   MissileDetector

Name	#Calls	Coverage
MissileDetector'addThreat	7	√
MissileDetector'getThreat	7	√
MissileDetector'isFinished	1	√
MissileDetector'addController	3	√
Total Coverage		**100%**

7 Flare Controller Class

```
class FlareController is subclass of GLOBAL
instance variables
        private aperature : Angle;
        ranges : ℕ --m--> (Angle × Angle) := {↦};
        dispensers : ℕ --m--> FlareDispenser := {↦};
```

inv dom $ranges$ = dom $dispensers$
$threats : (EventId \times MissileType \times Angle \times \mathbb{N})^* := [];$
$busy : \mathbb{B} := \mathsf{false};$

operations
public

$FlareController : Angle \xrightarrow{o} FlareController$
$FlareController\,(papp) \triangleq$
  $aperature := papp;$

public

$addDispenser : FlareDispenser \xrightarrow{o} ()$
$addDispenser\,(pfldisp) \triangleq$
  let $angle = aperature + pfldisp.GetAngle\,()$ in
  (dcl $id : \mathbb{N} := \mathsf{card\ dom}\ ranges + 1;$
       atomic ($ranges := ranges \uplus \{id \mapsto \mathsf{mk\text{-}}\,(angle, DISPENSER\text{-}APERATURE)\};$
            $dispensers := dispensers \uplus \{id \mapsto pfldisp\}$
)

start
(pfldisp)
       );
public

$getAperature : () \xrightarrow{o} GLOBAL`Angle \times GLOBAL`Angle$
$getAperature\,() \triangleq$
  return $\mathsf{mk\text{-}}\,(aperature, FLARE\text{-}APERATURE);$
public

$addThreat : EventId \times MissileType \times Angle \times \mathbb{N} \xrightarrow{o} ()$
$addThreat\,(evid, pmt, pa, pt) \triangleq$
  ($threats := threats \curvearrowright [\mathsf{mk\text{-}}\,(evid, pmt, pa, pt)];$
       $busy := \mathsf{true}$
  );
private

$getThreat : () \xrightarrow{o} EventId \times MissileType \times Angle \times \mathbb{N}$
$getThreat\,() \triangleq$
  (dcl $res : EventId \times MissileType \times Angle \times \mathbb{N} := \mathsf{hd}\ threats;$
       $threats := \mathsf{tl}\ threats;$
       return res
  );
public

10

$$isFinished : () \xrightarrow{o} ()$$
$$isFinished\,() \triangleq$$
    for all $id \in$ dom $dispensers$
    do $dispensers$ (id) $.isFinished()$

sync

    mutex($addThreat, getThreat$);
    per $getThreat \Rightarrow$ len $threats > 0$;
    per $isFinished \Rightarrow \neg\,busy$

thread

    (for all $id \in$ dom $dispensers$
      do

start
($dispensers$ (id)) ;
      while true
      do (def mk- $(evid, pmt, pa, pt) = getThreat\,()$ in
          for all $id \in$ dom $ranges$
          do def mk- $(papplhs, pappsize) = ranges\,(id)$ in
            if $canObserve\,(pa, papplhs, pappsize)$
              then $dispensers$ (id) $.addThreat(evid, pmt, pt)$;
          $busy :=$ len $threats > 0$
)
)
end $FlareController$

Test Suite :   vdm.tc
Class :   FlareController

Name	#Calls	Coverage
FlareController'addThreat	7	√
FlareController'getThreat	7	√
FlareController'isFinished	3	√
FlareController'addDispenser	12	√
FlareController'getAperature	3	√
FlareController'FlareController	3	√
Total Coverage		**100%**

8   Flare Dispenser Class

class $FlareDispenser$ is subclass of $GLOBAL$
values

$$responseDB : MissileType \xrightarrow{m} Plan = \{\text{MISSILEA} \mapsto [\text{mk-}(\text{FLAREONEA}, 900),$$
$$\text{mk-}(\text{FLARETWOA}, 500),$$
$$\text{mk-}(\text{DONOTHINGA}, 100),$$
$$\text{mk-}(\text{FLAREONEA}, 500)],$$
$$\text{MISSILEB} \mapsto [\text{mk-}(\text{FLARETWOB}, 500),$$
$$\text{mk-}(\text{FLARETWOB}, 700)],$$
$$\text{MISSILEC} \mapsto [\text{mk-}(\text{FLAREONEC}, 400),$$
$$\text{mk-}(\text{DONOTHINGC}, 100),$$
$$\text{mk-}(\text{FLARETWOC}, 400),$$
$$\text{mk-}(\text{FLAREONEC}, 500)]\};$$

$$missilePriority : MissileType \xrightarrow{m} \mathbb{N} = \{\text{NONE} \mapsto 0,$$
$$\text{MISSILEA} \mapsto 1,$$
$$\text{MISSILEB} \mapsto 2,$$
$$\text{MISSILEC} \mapsto 3\}$$

types

  public $Plan = PlanStep^*$;
  public $PlanStep = FlareType \times \mathbb{N}$

instance variables

  public $curplan : Plan := []$;
  $curprio : \mathbb{N} := 0$;
  $busy : \mathbb{B} := \mathsf{false}$;
  $angle : \mathbb{N}$;
  $eventid : [EventId]$;

operations
public

  $FlareDispenser : \mathbb{N} \xrightarrow{o} FlareDispenser$
  $FlareDispenser\,(ang) \triangleq$
   ($angle := ang$
   );

public

  $GetAngle : () \xrightarrow{o} \mathbb{N}$
  $GetAngle\,() \triangleq$
   return $angle$;

public

  $addThreat : EventId \times MissileType \times \mathbb{N} \xrightarrow{o} ()$
  $addThreat\,(evid, pmt, ptime) \triangleq$
   if $missilePriority\,(pmt) > curprio$
   then (dcl $newplan : Plan := []$,
       $newtime : \mathbb{N} := ptime$;

$$\text{for mk-}(\mathit{fltp}, \mathit{fltime}) \text{ in } \mathit{responseDB}\,(\mathit{pmt})$$

$$\text{do } (\quad \mathit{newplan} := \mathit{newplan} \curvearrowright [\text{mk-}(\mathit{fltp}, \mathit{newtime})];$$

$$\mathit{newtime} := \mathit{newtime} + \mathit{fltime}$$

$$)\;;$$

$$\text{def mk-}(\mathit{fltp}, \mathit{fltime}) = \text{hd } \mathit{newplan} \text{ in}$$

$$\mathit{releaseFlare}(\mathit{evid}, \mathit{fltp}, \mathit{fltime}, \mathit{World`timerRef}.\mathit{GetTime}\,());$$

$$\mathit{curplan} := \text{tl } \mathit{newplan};$$

$$\mathit{eventid} := \mathit{evid};$$

$$\mathit{curprio} := \mathit{missilePriority}\,(\mathit{pmt});$$

$$\mathit{busy} := \text{true}$$

$$)\;;$$

private

$$\mathit{evalQueue} : () \xrightarrow{o} ()$$

$$\mathit{evalQueue}\,() \triangleq$$

$$(\quad \text{if len } \mathit{curplan} > 0$$

$$\text{then } (\quad \text{dcl } \mathit{curtime} : \mathbb{N} := \mathit{World`timerRef}.\mathit{GetTime}\,(),$$

$$\mathit{done} : \mathbb{B} := \text{false};$$

$$\text{while } \neg\, \mathit{done}$$

$$\text{do } (\quad \text{dcl } \mathit{first} : \mathit{PlanStep} := \text{hd } \mathit{curplan},$$

$$\mathit{next} : \mathit{Plan} := \text{tl } \mathit{curplan};$$

$$\text{let mk-}(\mathit{fltp}, \mathit{fltime}) = \mathit{first} \text{ in}$$

$$(\quad \text{if } \mathit{fltime} \leq \mathit{curtime}$$

$$\text{then } (\quad \mathit{releaseFlare}(\mathit{eventid}, \mathit{fltp}, \mathit{fltime}, \mathit{curtime});$$

$$\mathit{curplan} := \mathit{next};$$

$$\text{if len } \mathit{next} = 0$$

$$\text{then } (\quad \mathit{curprio} := 0;$$

$$\mathit{done} := \text{true};$$

$$\mathit{busy} := \text{false}$$

$$)$$

$$)$$

$$\text{else } \mathit{done} := \text{true}$$

$$)$$

$$)$$

$$)$$

$$);$$

private

$$\mathit{releaseFlare} : \mathit{EventId} \times \mathit{FlareType} \times \mathbb{N} \times \mathbb{N} \xrightarrow{o} ()$$

$$\mathit{releaseFlare}\,(\mathit{evid}, \mathit{pfltp}, \mathit{pt1}, \mathit{pt2}) \triangleq \mathit{World`env}.$$

$$\mathit{handleEvent}(\mathit{evid}, \mathit{pfltp}, \mathit{angle}, \mathit{pt1}, \mathit{pt2})\;;$$

public

$$isFinished : () \xrightarrow{o} ()$$
$$isFinished () \triangleq$$
   skip
sync
   $mutex(addThreat, evalQueue);$
   per $isFinished \Rightarrow \neg busy$
thread

   while true
   do (World'timerRef.$WaitRelative(TimeStamp'stepLength)$;
        $evalQueue()$
)
end $FlareDispenser$

Test Suite :   vdm.tc
Class :   FlareDispenser

Name	#Calls	Coverage
FlareDispenser'GetAngle	12	√
FlareDispenser'addThreat	7	√
FlareDispenser'evalQueue	4753	√
FlareDispenser'isFinished	12	√
FlareDispenser'releaseFlare	21	√
FlareDispenser'FlareDispenser	12	√
Total Coverage		**100%**

9   The WaitNotify Class

class $WaitNotify$
instance variables
   $waitset : \mathbb{N}\text{-set} := \{\};$

operations
public
   $Wait : () \xrightarrow{o} ()$
   $Wait () \triangleq$
      ($AddToWaitSet(\text{threadid})$;
         $Awake()$
      );
public

14

$Notify : () \xrightarrow{o} ()$
$Notify\,() \triangleq$
   let $p \in waitset$ in
   $waitset := waitset \setminus \{p\}$;

public

$NotifyThread : \mathbb{N} \xrightarrow{o} ()$
$NotifyThread\,(tId) \triangleq$
   $waitset := waitset \setminus \{tId\}$;

public

$NotifyAll : () \xrightarrow{o} ()$
$NotifyAll\,() \triangleq$
   $waitset := \{\}$;

private

$AddToWaitSet : \mathbb{N} \xrightarrow{o} ()$
$AddToWaitSet\,(n) \triangleq$
   $waitset := waitset \cup \{n\}$;

private

$Awake : () \xrightarrow{o} ()$
$Awake\,() \triangleq$
   skip

sync

   per $Awake \Rightarrow$ threadid $\notin waitset$;
   mutex($AddToWaitSet$)
end $WaitNotify$

Test Suite :   vdm.tc
Class :   WaitNotify

Name	#Calls	Coverage
WaitNotify'Wait	4776	$\checkmark$
WaitNotify'Awake	4754	$\checkmark$
WaitNotify'Notify	0	0%
WaitNotify'NotifyAll	0	0%
WaitNotify'AddToWaitSet	4776	$\checkmark$
WaitNotify'NotifyThread	4650	$\checkmark$
Total Coverage		**60%**

10   TimeStamp Class

class $TimeStamp$ is subclass of $WaitNotify$

values
public
$$stepLength : \mathbb{N} = 10$$
instance variables
$$currentTime : \mathbb{N} := 0;$$
$$wakeUpMap : \mathbb{N} \xrightarrow{m} \mathbb{N} := \{\mapsto\};$$

operations
public
$$WaitRelative : \mathbb{N} \xrightarrow{o} ()$$
$$WaitRelative\,(val) \triangleq$$
$$(\quad AddToWakeUpMap(\mathsf{threadid}, currentTime + val)\,;$$
$$\qquad WaitNotify`Wait()$$
$$);$$
public
$$WaitAbsolute : \mathbb{N} \xrightarrow{o} ()$$
$$WaitAbsolute\,(val) \triangleq$$
$$(\quad AddToWakeUpMap(\mathsf{threadid}, val)\,;$$
$$\qquad WaitNotify`Wait()$$
$$);$$
$$AddToWakeUpMap : \mathbb{N} \times \mathbb{N} \xrightarrow{o} ()$$
$$AddToWakeUpMap\,(tId, val) \triangleq$$
$$wakeUpMap := wakeUpMap \dagger \{tId \mapsto val\};$$
public
$$NotifyThread : \mathbb{N} \xrightarrow{o} ()$$
$$NotifyThread\,(tId) \triangleq$$
$$(\quad wakeUpMap := \{tId\} \lhd wakeUpMap;$$
$$\qquad WaitNotify`NotifyThread(tId)$$
$$);$$
public
$$Notify : () \xrightarrow{o} ()$$
$$Notify\,() \triangleq$$
$$\mathsf{let}\ tId \in \mathsf{dom}\ wakeUpMap\ \mathsf{in}$$
$$NotifyThread(tId)\,;$$
public
$$NotifyAll : () \xrightarrow{o} ()$$
$$NotifyAll\,() \triangleq$$
$$(\quad wakeUpMap := \{\mapsto\};$$
$$\qquad WaitNotify`NotifyAll()$$
$$);$$

public

$$NotifyAndIncTime : () \xrightarrow{o} ()$$
$$NotifyAndIncTime () \triangleq$$
$$(\quad currentTime := currentTime + stepLength;$$

      for all $t \in$ dom $(wakeUpMap \rhd \{currentTime\})$
      do $NotifyThread(t)$
$$);$$

public

$$GetTime : () \xrightarrow{o} \mathbb{N}$$
$$GetTime () \triangleq$$

    return $currentTime$

sync

    mutex($AddToWakeUpMap, Notify, NotifyThread, NotifyAll$)

end $TimeStamp$

Test Suite :   vdm.tc
Class :   TimeStamp

Name	#Calls	Coverage
TimeStamp'Notify	0	0%
TimeStamp'GetTime	1134	√
TimeStamp'NotifyAll	0	0%
TimeStamp'NotifyThread	4650	√
TimeStamp'WaitAbsolute	0	0%
TimeStamp'WaitRelative	4777	√
TimeStamp'AddToWakeUpMap	4776	√
TimeStamp'NotifyAndIncTime	200	√
Total Coverage		**73%**

11 Standard IO Class

class IO
types

    public $filedirective = \text{START} \mid \text{APPEND}$

functions
public

$$writeval[@p] : @p \rightarrow \mathbb{B}$$
$$writeval (val) \triangleq$$

    is not yet specified;

public

$fwriteval[@p] : \mathsf{char}^+ \times @p \times \mathit{filedirective} \rightarrow \mathbb{B}$

$fwriteval\,(\mathit{filename}, \mathit{val}, \mathit{fdir}) \triangleq$

   is not yet specified;

public

$freadval[@p] : \mathsf{char}^+ \rightarrow \mathbb{B} \times [@p]$

$freadval\,(f) \triangleq$

   is not yet specified

post let mk-$(b, t) = RESULT$ in

   $\neg\, b \;\Rightarrow\; t = \mathsf{nil}$

operations

public

$echo : \mathsf{char}^* \xrightarrow{o} \mathbb{B}$

$echo\,(\mathit{text}) \triangleq$

   $fecho(\texttt{""}, \mathit{text}, \mathsf{nil}\,)$;

public

$fecho : \mathsf{char}^* \times \mathsf{char}^* \times [\mathit{filedirective}] \xrightarrow{o} \mathbb{B}$

$fecho\,(\mathit{filename}, \mathit{text}, \mathit{fdir}) \triangleq$

   is not yet specified

pre $\mathit{filename} = \texttt{""} \;\Leftrightarrow\; \mathit{fdir} = \mathsf{nil}$;

public

$ferror : () \xrightarrow{o} \mathsf{char}^*$

$ferror\,() \triangleq$

   is not yet specified

   end IO