# The Concurrent Missile VDM++ Model

Peter Gorm Larsen and Marcel Verhoef

2007

## 1 The Overall Class Diagram

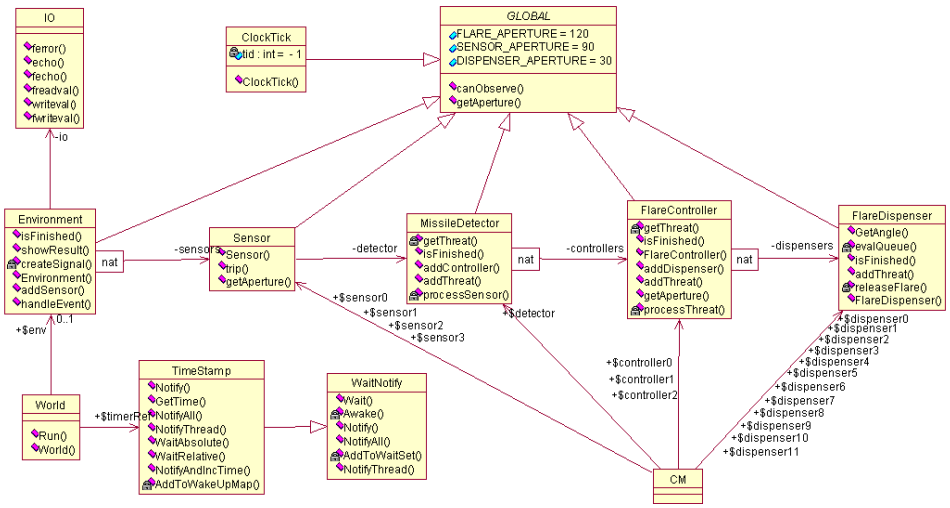


Figure 1: Overview of the classes in the concurrent CM model

# 2 The World Class

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

operations
public
        World : () ==> World
        World () ==
          (     env := new Environment ("scenario.txt");
                env.addSensor(CM`sensor0) ;
                env.addSensor(CM`sensor1) ;
                env.addSensor(CM`sensor2) ;
                env.addSensor(CM`sensor3) ;
                CM`controller0.addDispenser(CM`dispenser0) ;
                CM`controller0.addDispenser(CM`dispenser1) ;
                CM`controller0.addDispenser(CM`dispenser2) ;
                CM`controller0.addDispenser(CM`dispenser3) ;
                CM`detector.addController(CM`controller0) ;
                CM`controller1.addDispenser(CM`dispenser4) ;
                CM`controller1.addDispenser(CM`dispenser5) ;
                CM`controller1.addDispenser(CM`dispenser6) ;
                CM`controller1.addDispenser(CM`dispenser7) ;
                CM`detector.addController(CM`controller1) ;
                CM`controller2.addDispenser(CM`dispenser8) ;
                CM`controller2.addDispenser(CM`dispenser9) ;
                CM`controller2.addDispenser(CM`dispenser10) ;
                CM`controller2.addDispenser(CM`dispenser11) ;
                CM`detector.addController(CM`controller2) ;

start
(CM`detector)
          );
public
        Run : () ==> ()
        Run () ==
          (
start
```

(env) ;
*env.isFinished*() ;
CM‘detector.*isFinished*() ;
*env.showResult*()
)
end *World*

**Test Suite :** vdm.tc
**Class :** World

| Name | #Calls | Coverage |
|---|---|---|
| World‘Run | 1 | √ |
| World‘World | 1 | √ |
| **Total Coverage** | | **100%** |

# 3 The FighterAircraft Class

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

end *CM*

# 4 The Environment Class

class $Environment$ is subclass of $GLOBAL$
types
public $InputTP = (Time \times inline^*)$;
public $inline = EventId \times MissileType \times Angle \times Time$;
public $outline = EventId \times FlareType \times Angle \times Time \times Time$
instance variables
$io : IO := \text{new } IO\,()$;
$inlines : inline^* := []$;
$outlines : outline^* := []$;
$ranges : \mathbb{N} \xrightarrow{m} (Angle \times Angle) := \{\mapsto\}$;
$sensors : \mathbb{N} \xrightarrow{m} Sensor := \{\mapsto\}$;
inv dom $ranges$ = dom $sensors$
$busy : \mathbb{B} := \text{true}$;
$simtime : Time$;

operations
public
$Environment : \text{char}^* \xrightarrow{o} Environment$
$Environment\,(fname) \triangleq$
def mk-(-, mk-$(timeval, input)) = io.freadval[InputTP]\,(fname)$ in
( $inlines := input$;
$simtime := timeval$
);
public
$addSensor : Sensor \xrightarrow{o} ()$
$addSensor\,(psens) \triangleq$
( dcl $id : \mathbb{N} := \text{card dom } ranges + 1$;
atomic ( $ranges := ranges \biguplus \{id \mapsto psens.getAperture\,()\}$;
$sensors := sensors \biguplus \{id \mapsto psens\}$
)
);
private
$createSignal : () \xrightarrow{o} ()$
$createSignal\,() \triangleq$
( if len $inlines > 0$
then ( dcl $curtime : Time := World`timerRef.GetTime\,()$,
$done : \mathbb{B} := \text{false}$;
while $\neg\, done$

```
                    do def mk-(eventid, pmt, pa, pt) = hd inlines in
                       if pt ≤ curtime
                       then (   for all id ∈ dom ranges
                                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
                            )
                       else (   done := true;
                                return
                            )
                 )
              else (   busy := false;
                       return
                 )
        );
public
      handleEvent : EventId × FlareType × Angle × Time × Time →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
sync
      mutex(handleEvent);
      per isFinished ⇒ ¬ busy
thread

      (
start
```

```
(new ClockTick     (threadid )   ) ;
            while World‘timerRef.GetTime () < simtime
            do (    if busy
                    then createSignal() ;
                    World‘timerRef.NotifyAndIncTime() ;
                    World‘timerRef.WaitRelative(0)
               ) ;
            busy := false
        )
end Environment
```

**Test Suite :** vdm.tc
**Class :** Environment

| Name | #Calls | Coverage |
|---|---|---|
| Environment‘addSensor | 4 | √ |
| Environment‘isFinished | 1 | √ |
| Environment‘showResult | 1 | √ |
| Environment‘Environment | 1 | √ |
| Environment‘handleEvent | 21 | √ |
| Environment‘createSignal | 82 | √ |
| **Total Coverage** | | **100%** |

# 5 The Global Class

```
class GLOBAL
values
public
        SENSOR-APERTURE = 90;
public
        FLARE-APERTURE = 120;
public
        DISPENSER-APERTURE = 30
types
        public MissileType = MissileA | MissileB | MissileC | None;
        public FlareType = FlareOneA | FlareTwoA | DoNothingA |
                           FlareOneB | FlareTwoB | DoNothingB |
                           FlareOneC | FlareTwoC | DoNothingC;
        public Angle = ℕ
```

inv $num \triangleq num < 360$;
public $EventId = \mathbb{N}$;
public $Time = \mathbb{N}$
operations
public
$canObserve : Angle \times Angle \times Angle \xrightarrow{o} \mathbb{B}$
$canObserve\,(pangle, pleft, psize) \triangleq$
def $pright = (pleft + psize)$ mod 360 in
if $pright < pleft$
then return $(pangle < pright \vee pangle \geq pleft)$
else return $(pangle \geq pleft \wedge pangle < pright)$ ;
public
$getAperture : () \xrightarrow{o} Angle \times Angle$
$getAperture\,() \triangleq$
is subclass responsibility
end $GLOBAL$

**Test Suite :** vdm.tc
**Class :** GLOBAL

| **Name** | **#Calls** | **Coverage** |
|---|---|---|
| GLOBAL‘canObserve | 84 | $\surd$ |
| GLOBAL‘getAperture | 0 | 0% |
| **Total Coverage** | | **96%** |

# 6 The Sensor Class

class $Sensor$ is subclass of $GLOBAL$
instance variables
private $detector : MissileDetector$;
private $aperture : Angle$;

operations
public
$Sensor : MissileDetector \times Angle \xrightarrow{o} Sensor$
$Sensor\,(pmd, psa) \triangleq$
( $detector := pmd$;
$aperture := psa$
);
public

$getAperture : () \xrightarrow{o} GLOBAL`Angle \times GLOBAL`Angle$
$getAperture\,() \triangleq$
return mk- $(aperture, SENSOR\text{-}APERTURE)$;
public
$trip : EventId \times MissileType \times Angle \xrightarrow{o} ()$
$trip\,(evid, pmt, pa) \triangleq detector.$
$addThreat(evid, pmt, pa, World`timerRef.GetTime\,())$
pre $canObserve\,(pa, aperture, SENSOR\text{-}APERTURE)$
end *Sensor*

**Test Suite :** vdm.tc
**Class :** Sensor

| **Name** | **#Calls** | **Coverage** |
|---|---|---|
| Sensor`trip | 7 | $\surd$ |
| Sensor`Sensor | 4 | $\surd$ |
| Sensor`getAperture | 4 | $\surd$ |
| **Total Coverage** | | **100%** |

# 7 The Missile Detector Class

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

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

start
(pctrl)
);

```
public
        addThreat : EventId × MissileType × Angle × Time →o ()
        addThreat (evid, pmt, pa, pt) △
          (     threats := threats ↷ [mk- (evid, pmt, pa, pt)];
                busy := true
          );
private
        getThreat : () →o EventId × MissileType × Angle × Time
        getThreat () △
          (     dcl res : EventId × MissileType × Angle × Time := hd threats;
                threats := tl threats;
                return res
          );
public
        isFinished : () →o ()
        isFinished () △
          for all id ∈ dom controllers
          do controllers (id) .isFinished() ;
        processSensor : () →o ()
        processSensor () △
          (     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
          )
sync
        mutex(addThreat, getThreat);
        per getThreat ⇒ len threats > 0;
        per isFinished ⇒ ¬ busy
thread

        while true
        do processSensor()
end MissileDetector
```

**Test Suite :** vdm.tc
**Class :** MissileDetector

| Name | #Calls | Coverage |
|---|---|---|
| MissileDetector‘addThreat | 7 | √ |

| Name | #Calls | Coverage |
|---|---|---|
| MissileDetector‘getThreat | 7 | $\surd$ |
| MissileDetector‘isFinished | 1 | $\surd$ |
| MissileDetector‘addController | 3 | $\surd$ |
| MissileDetector‘processSensor | 8 | $\surd$ |
| **Total Coverage** | | **100%** |

# 8 The Flare Controller Class

class $FlareController$ is subclass of $GLOBAL$
instance variables

$\quad$ private $aperture : Angle$;
$\quad$ $ranges : \mathbb{N} \xrightarrow{m} (Angle \times Angle) := \{\mapsto\}$;
$\quad$ $dispensers : \mathbb{N} \xrightarrow{m} FlareDispenser := \{\mapsto\}$;
$\quad$ inv dom $ranges$ = dom $dispensers$
$\quad$ $threats : (EventId \times MissileType \times Angle \times Time)^* := []$;
$\quad$ $busy : \mathbb{B} :=$ false;

operations
public

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

public

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

start
(pfldisp)
$\quad\quad$ );
public

```
        getAperture : () ^o→ GLOBAL`Angle × GLOBAL`Angle
        getAperture () ≜
          return mk-(aperture, FLARE-APERTURE);
public
        addThreat : EventId × MissileType × Angle × Time ^o→ ()
        addThreat (evid, pmt, pa, pt) ≜
          (   threats := threats ↷ [mk-(evid, pmt, pa, pt)];
              busy := true
          );
private
        getThreat : () ^o→ EventId × MissileType × Angle × Time
        getThreat () ≜
          (   dcl res : EventId × MissileType × Angle × Time := hd threats;
              threats := tl threats;
              return res
          );
public
        isFinished : () ^o→ ()
        isFinished () ≜
          for all id ∈ dom dispensers
          do dispensers (id) .isFinished() ;
        processThreat : () ^o→ ()
        processThreat () ≜
          (   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 dispensers (id) .addThreat(evid, pmt, pt) ;
              busy := len threats > 0
          )
sync
        mutex(addThreat, getThreat);
        per getThreat ⇒ len threats > 0;
        per isFinished ⇒ len threats = 0
thread

        (   for all id ∈ dom dispensers
            do
start
(dispensers (id) ) ;
            while true
```

```
        do processThreat()
    )
end FlareController
```

**Test Suite :** vdm.tc
**Class :** FlareController

| Name | #Calls | Coverage |
|---|---|---|
| FlareController`addThreat | 7 | √ |
| FlareController`getThreat | 7 | √ |
| FlareController`isFinished | 3 | √ |
| FlareController`getAperture | 3 | √ |
| FlareController`addDispenser | 12 | √ |
| FlareController`processThreat | 10 | √ |
| FlareController`FlareController | 3 | √ |
| **Total Coverage** | | **100%** |

# 9 The Flare Dispenser Class

class *FlareDispenser* is subclass of *GLOBAL*
values

$$\begin{aligned} \mathit{responseDB} : \mathit{MissileType} \xrightarrow{m} \mathit{Plan} = \{&\textsc{MissileA} \mapsto [\mathsf{mk\text{-}}(\textsc{FlareOneA}, 900),\\ &\mathsf{mk\text{-}}(\textsc{FlareTwoA}, 500),\\ &\mathsf{mk\text{-}}(\textsc{DoNothingA}, 100),\\ &\mathsf{mk\text{-}}(\textsc{FlareOneA}, 500)],\\ &\textsc{MissileB} \mapsto [\mathsf{mk\text{-}}(\textsc{FlareTwoB}, 500),\\ &\mathsf{mk\text{-}}(\textsc{FlareTwoB}, 700)],\\ &\textsc{MissileC} \mapsto [\mathsf{mk\text{-}}(\textsc{FlareOneC}, 400),\\ &\mathsf{mk\text{-}}(\textsc{DoNothingC}, 100),\\ &\mathsf{mk\text{-}}(\textsc{FlareTwoC}, 400),\\ &\mathsf{mk\text{-}}(\textsc{FlareOneC}, 500)]\}; \end{aligned}$$

$$\begin{aligned} \mathit{missilePriority} : \mathit{MissileType} \xrightarrow{m} \mathbb{N} = \{&\textsc{None} \mapsto 0,\\ &\textsc{MissileA} \mapsto 1,\\ &\textsc{MissileB} \mapsto 2,\\ &\textsc{MissileC} \mapsto 3\} \end{aligned}$$

types

$$\mathsf{public}\ \mathit{Plan} = \mathit{PlanStep}^*;$$

$$\mathsf{public}\ \mathit{PlanStep} = \mathit{FlareType} \times \mathit{Time}$$

instance variables

$$\mathsf{public}\ \mathit{curplan} : \mathit{Plan} := [];$$

```
        curprio : ℕ := 0;
        busy : 𝔹 := false;
        aperture : Angle;
        eventid : [EventId];

operations
public
        FlareDispenser : Angle →o FlareDispenser
        FlareDispenser (ang) △
          aperture := ang;
public
        GetAngle : () →o ℕ
        GetAngle () △
          return aperture;
public
        addThreat : EventId × MissileType × Time →o ()
        addThreat (evid, pmt, ptime) △
          if missilePriority (pmt) > curprio
          then (    dcl newplan : Plan := [],
                        newtime : Time := ptime;
                    for mk- (fltp, fltime) in responseDB (pmt)
                    do (    newplan := newplan ↷ [mk- (fltp, newtime)];
                            newtime := newtime + fltime
                       ) ;
                    def mk- (fltp, fltime) = hd newplan;
                        t = World`timerRef.GetTime () in
                    releaseFlare(evid, fltp, fltime, t) ;
                    curplan := tl newplan;
                    eventid := evid;
                    curprio := missilePriority (pmt);
                    busy := true
                )
        pre  pmt ∈ dom missilePriority ∧
             pmt ∈ dom responseDB ;
private
        evalQueue : () →o ()
        evalQueue () △
          (    if len curplan > 0
               then (    dcl curtime : Time := World`timerRef.GetTime (),
                             done : 𝔹 := false;
                         while ¬ done
```

```
                    do (    dcl first : PlanStep := hd curplan,
                                next : Plan := tl curplan;
                            let mk- (fltp, fltime) = first in
                            if fltime ≤ curtime
                            then (    releaseFlare(eventid, fltp, fltime, curtime);
                                      curplan := next;
                                      if len next = 0
                                      then (    curprio := 0;
                                                done := true;
                                                busy := false
                                            )
                                 )
                            else done := true
                        )
                )
        );
private
        releaseFlare : EventId × FlareType × Time × Time →o ()
        releaseFlare (evid, pfltp, pt1, pt2) △ World‘env.
          handleEvent(evid, pfltp, aperture, pt1, pt2) ;
public
        isFinished : () →o ()
        isFinished () △
          skip
sync
        mutex(addThreat, evalQueue);
        per isFinished  ⇒  ¬ 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 | 4776 | √ |

| Name | #Calls | Coverage |
|---|---|---|
| FlareDispenser`isFinished | 12 | √ |
| FlareDispenser`releaseFlare | 21 | √ |
| FlareDispenser`FlareDispenser | 12 | √ |
| **Total Coverage** | | **100%** |

# 10 The WaitNotify Class

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

operations
public
$Wait : () \xrightarrow{o} ()$
$Wait\,() \triangleq$
( $AddToWaitSet(\text{threadid})$ ;
$Awake()$
);
public
$Notify : () \xrightarrow{o} ()$
$Notify\,() \triangleq$
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 | 5200 | $\surd$ |
| WaitNotify‘Awake | 5174 | $\surd$ |
| WaitNotify‘Notify | 0 | 0% |
| WaitNotify‘NotifyAll | 0 | 0% |
| WaitNotify‘AddToWaitSet | 5200 | $\surd$ |
| WaitNotify‘NotifyThread | 5187 | $\surd$ |
| **Total Coverage** | | **60%** |

# 11 The 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
$WaitRelative : \mathbb{N} \xrightarrow{o} ()$
$WaitRelative\,(val) \triangleq$
( $AddToWakeUpMap$(threadid, $currentTime + val$) ;
$WaitNotify‘Wait()$
);
public

```
        WaitAbsolute : ℕ →^o ()
        WaitAbsolute (val) ≙
          (   AddToWakeUpMap(threadid, val) ;
              WaitNotify`Wait()
          );
        AddToWakeUpMap : ℕ × ℕ →^o ()
        AddToWakeUpMap (tId, val) ≙
          wakeUpMap := wakeUpMap † {tId ↦ val};
public
        NotifyThread : ℕ →^o ()
        NotifyThread (tId) ≙
          (   wakeUpMap := {tId} ⩤ wakeUpMap;
              WaitNotify`NotifyThread(tId)
          );
public
        Notify : () →^o ()
        Notify () ≙
          let tId ∈ dom wakeUpMap in
          NotifyThread(tId) ;
public
        NotifyAll : () →^o ()
        NotifyAll () ≙
          (   wakeUpMap := {↦};
              WaitNotify`NotifyAll()
          );
public
        NotifyAndIncTime : () →^o ()
        NotifyAndIncTime () ≙
          (   currentTime := currentTime + stepLength;
              for all t ∈ dom (wakeUpMap ▷ {1, ..., currentTime})
              do NotifyThread(t)
          );
public
        GetTime : () →^o ℕ
        GetTime () ≙
          return currentTime
sync
        mutex(AddToWakeUpMap);
        mutex(AddToWakeUpMap, Notify, NotifyThread, NotifyAll)
end TimeStamp
```

**Test Suite :** vdm.tc
**Class :** TimeStamp

| Name | #Calls | Coverage |
|---|---|---|
| TimeStamp‘Notify | 0 | 0% |
| TimeStamp‘GetTime | 1325 | √ |
| TimeStamp‘NotifyAll | 0 | 0% |
| TimeStamp‘NotifyThread | 5187 | √ |
| TimeStamp‘WaitAbsolute | 0 | 0% |
| TimeStamp‘WaitRelative | 5200 | √ |
| TimeStamp‘AddToWakeUpMap | 5200 | √ |
| TimeStamp‘NotifyAndIncTime | 200 | √ |
| **Total Coverage** | | **74%** |

# 12 The ClockTick Class

class $ClockTick$ is subclass of $GLOBAL$
instance variables
$tid : \mathbb{Z} := -1;$

operations
public
$ClockTick : \mathbb{N} \xrightarrow{o} ClockTick$
$ClockTick\,(t) \triangleq$
$tid := t$
thread

while true
do ( World‘timerRef.$NotifyThread(tid)$ ;
World‘timerRef.$WaitRelative(1)$
)
end $ClockTick$

**Test Suite :** vdm.tc
**Class :** ClockTick

| Name | #Calls | Coverage |
|---|---|---|
| ClockTick‘ClockTick | 1 | √ |
| **Total Coverage** | | **100%** |

# 13 The Standard IO Class

class $IO$
types
public $filedirective = \text{START} \mid \text{APPEND}$
functions
public
$writeval[@p] : @p \to \mathbb{B}$
$writeval\,(val) \triangleq$
is not yet specified;
public
$fwriteval[@p] : \text{char}^+ \times @p \times filedirective \to \mathbb{B}$
$fwriteval\,(filename, val, fdir) \triangleq$
is not yet specified;
public
$freadval[@p] : \text{char}^+ \to \mathbb{B} \times [@p]$
$freadval\,(f) \triangleq$
is not yet specified
post let mk-$(b, t) = RESULT$ in
$\neg\, b \;\Rightarrow\; t = \text{nil}$
operations
public
$echo : \text{char}^* \xrightarrow{o} \mathbb{B}$
$echo\,(text) \triangleq$
$fecho(\text{""}, text, \text{nil})$ ;
public
$fecho : \text{char}^* \times \text{char}^* \times [filedirective] \xrightarrow{o} \mathbb{B}$
$fecho\,(filename, text, fdir) \triangleq$
is not yet specified
pre $filename = \text{""} \;\Leftrightarrow\; fdir = \text{nil}$ ;
public
$ferror : () \xrightarrow{o} \text{char}^*$
$ferror\,() \triangleq$
is not yet specified
end $IO$

# Index