The Concurrent Missile VDM++ Model

Peter Gorm Larsen and Marcel Verhoef

2006

1 The World Class

class *World*
instance variables
    public static $env : [Environment] := \mathsf{nil}$;
    public static $timerRef : TimeStamp := \mathsf{new}\ TimeStamp\,()$;

operations
public

```
        World : () ->o World
        World () ==
          (   env := new Environment ("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 : () ->o ()
        Run () ==
          (
start
(env) ;
              env.isFinished() ;
              FighterAircraft`detector.isFinished() ;
              env.showResult()
          )
end World
```

Test Suite : vdm.tc
Class : World

Name	#Calls	Coverage
World‘Run	1	$\surd$
World‘World	1	$\surd$
Total Coverage		**100%**

2 The FighterAircraft Class

class $FighterAircraft$
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 $FighterAircraft$

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

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

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

```
                                    do def mk-(applhs, appsize) = ranges(id)
in
                                       if canObserve(pa, applhs, appsize)
                                       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
sync
        mutex(handleEvent);
        per isFinished  ⇒  ¬ busy
thread

        while true
        do (   if busy
               then (   evid := createSignal()
                    ) ;
               World‘timerRef.NotifyAndIncTime()
           )
```

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%**

4 The Global Class

class *GLOBAL*
values
public
$SENSOR\text{-}APERATURE = 90;$
public
$FLARE\text{-}APERATURE = 120;$
public
$DISPENSER\text{-}APERATURE = 30$
types
public $MissileType = \text{MissileA} \mid \text{MissileB} \mid \text{MissileC} \mid \text{None};$
public $FlareType = \text{FlareOneA} \mid \text{FlareTwoA} \mid \text{DoNothingA} \mid$
$\text{FlareOneB} \mid \text{FlareTwoB} \mid \text{DoNothingB} \mid$
$\text{FlareOneC} \mid \text{FlareTwoC} \mid \text{DoNothingC};$
public $Angle = \mathbb{N}$
inv $num \triangleq num < 360;$
public $EventId = \mathbb{N}$
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

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

$getAperature\,() \triangleq$

is subclass responsibility

end $GLOBAL$

Test Suite : vdm.tc

Class : GLOBAL

Name	**#Calls**	**Coverage**
GLOBAL‘canObserve	84	$\surd$
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

$Sensor : MissileDetector \times Angle \xrightarrow{o} Sensor$

$Sensor\,(pmd, psa) \triangleq$

$($ $detector := pmd$;

$aperature := psa$

);

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	$\surd$
Sensor‘Sensor	4	$\surd$
Sensor‘getAperature	4	$\surd$
Total Coverage		**100%**

6 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 $controllers$
$threats : (EventId \times MissileType \times Angle \times \mathbb{N})^* := [];$
$busy : \mathbb{B} :=$ false;

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

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

```
            threats := tl threats;
            return res
        );
public
    isFinished : () ->o ()
    isFinished () triangle=
      for all id in dom controllers
      do controllers (id) .isFinished()
sync
    mutex(addThreat, getThreat);
    per getThreat => len threats > 0;
    per isFinished => not busy
thread

    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 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 × MissileType × Angle × ℕ)* := [];
        busy : 𝔹 := false;

operations
public
        FlareController : Angle ==> FlareController
        FlareController (papp) △
          aperature := papp;
public
        addDispenser : FlareDispenser ==> ()
        addDispenser (pfldisp) △
          let angle = aperature + pfldisp.GetAngle () in
          (   dcl id : ℕ := card dom ranges + 1;
              atomic (   ranges := ranges ⩊ {id ↦ mk- (angle, DISPENSER-APERATURE)};
                         dispensers := dispensers ⩊ {id ↦ pfldisp}
              ) ;

start
(pfldisp)
          );
public
        getAperature : () ==> GLOBAL`Angle × GLOBAL`Angle
        getAperature () △
          return mk- (aperature, FLARE-APERATURE);
public
        addThreat : EventId × MissileType × Angle × ℕ ==> ()
        addThreat (evid, pmt, pa, pt) △
          (   threats := threats ↷ [mk- (evid, pmt, pa, pt)];
              busy := true
          );
private
        getThreat : () ==> EventId × MissileType × Angle × ℕ
        getThreat () △
          (   dcl res : EventId × MissileType × Angle × ℕ := hd threats;
              threats := tl threats;
              return res
          );
public
```

```
        isFinished : () →o ()
        isFinished () △
          for all id ∈ dom dispensers
          do dispensers (id) .isFinished()
sync
        mutex(addThreat, getThreat);
        per getThreat ⇒ len threats > 0;
        per isFinished ⇒ ¬ busy
thread

        (   for all id ∈ dom dispensers
            do
start
(dispensers (id) ) ;
            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 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 $curplan : Plan := [];$

$curprio : \mathbb{N} := 0;$

$busy : \mathbb{B} := \text{false};$

$angle : \mathbb{N};$

$eventid : [EventId];$

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 $newplan : Plan := [],$

$newtime : \mathbb{N} := ptime;$

```
                for mk-(fltp, fltime) in responseDB(pmt)
                do (    newplan := newplan ↷ [mk-(fltp, newtime)];
                        newtime := newtime + fltime
                   );
                def mk-(fltp, fltime) = hd newplan in
                releaseFlare(evid, fltp, fltime, World‘timerRef.GetTime());
                curplan := tl newplan;
                eventid := evid;
                curprio := missilePriority(pmt);
                busy := true
            );
private
    evalQueue : () →^o ()
    evalQueue() △
      (   if len curplan > 0
          then (   dcl curtime : ℕ := 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 × ℕ × ℕ →^o ()
    releaseFlare(evid, pfltp, pt1, pt2) △ World‘env.
      handleEvent(evid, pfltp, angle, 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	4669	√
FlareDispenser‘isFinished	12	√
FlareDispenser‘releaseFlare	21	√
FlareDispenser‘FlareDispenser	12	√
Total Coverage		**100%**

9 The WaitNotify Class

```
class WaitNotify
instance variables
        waitset : ℕ-set := {};

operations
public
        Wait : () ->o ()
        Wait () ≜
          (    AddToWaitSet(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	4692	√
WaitNotify‘Awake	4669	√
WaitNotify‘Notify	0	0%
WaitNotify‘NotifyAll	0	0%
WaitNotify‘AddToWaitSet	4692	√
WaitNotify‘NotifyThread	4572	√
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
$WaitRelative : \mathbb{N} \xrightarrow{o} ()$
$WaitRelative\,(val) \triangleq$
($AddToWakeUpMap(\text{threadid}, currentTime + val)$;
$WaitNotify`Wait()$
);
public
$WaitAbsolute : \mathbb{N} \xrightarrow{o} ()$
$WaitAbsolute\,(val) \triangleq$
($AddToWakeUpMap(\text{threadid}, val)$;
$WaitNotify`Wait()$
);
$AddToWakeUpMap : \mathbb{N} \times \mathbb{N} \xrightarrow{o} ()$
$AddToWakeUpMap\,(tId, val) \triangleq$
$wakeUpMap := wakeUpMap \dagger \{tId \mapsto val\};$
public
$NotifyThread : \mathbb{N} \xrightarrow{o} ()$
$NotifyThread\,(tId) \triangleq$
($wakeUpMap := \{tId\} \lhd wakeUpMap;$
$WaitNotify`NotifyThread(tId)$
);
public
$Notify : () \xrightarrow{o} ()$
$Notify\,() \triangleq$
let $tId \in \text{dom } wakeUpMap$ in
$NotifyThread(tId)$;
public
$NotifyAll : () \xrightarrow{o} ()$
$NotifyAll\,() \triangleq$
($wakeUpMap := \{\mapsto\};$
$WaitNotify`NotifyAll()$
);

public

$NotifyAndIncTime : () \xrightarrow{o} ()$

$NotifyAndIncTime\,() \triangleq$

($currentTime := currentTime + stepLength$;

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	1131	√
TimeStamp‘NotifyAll	0	0%
TimeStamp‘NotifyThread	4572	√
TimeStamp‘WaitAbsolute	0	0%
TimeStamp‘WaitRelative	4693	√
TimeStamp‘AddToWakeUpMap	4692	√
TimeStamp‘NotifyAndIncTime	200	√
Total Coverage		**73%**

11 Standard IO Class

class *IO*

types

public *filedirective* = START | APPEND

functions

public

$writeval[@p] : @p \to \mathbb{B}$

$writeval\,(val) \triangleq$

is not yet specified;

public

$fwriteval[@p] : \mathsf{char}^+ \times @p \times filedirective \to \mathbb{B}$
$fwriteval\,(filename, val, fdir) \triangleq$
is not yet specified;

public

$freadval[@p] : \mathsf{char}^+ \to \mathbb{B} \times [@p]$
$freadval\,(f) \triangleq$
is not yet specified
post let mk-$(b, t) = RESULT$ in
$\neg\, b \;\Rightarrow\; t = \mathsf{nil}$

operations

public

$echo : \mathsf{char}^* \xrightarrow{o} \mathbb{B}$
$echo\,(text) \triangleq$
$fecho(\text{""}, text, \mathsf{nil})$;

public

$fecho : \mathsf{char}^* \times \mathsf{char}^* \times [filedirective] \xrightarrow{o} \mathbb{B}$
$fecho\,(filename, text, fdir) \triangleq$
is not yet specified
pre $filename = \text{""} \;\Leftrightarrow\; fdir = \mathsf{nil}$;

public

$ferror : () \xrightarrow{o} \mathsf{char}^*$
$ferror\,() \triangleq$
is not yet specified

end IO