# Concurrent Missile VDM++ Model

Peter Gorm Larsen

2006

## 1 World

```
class World
instance variables
        sensor : Sensor := new Sensor ();
        detector : MissileDetector := new MissileDetector ();
        flareControl : FlareController := new FlareController ();
        timerRef : Timer := new Timer ();

operations
public
        Run : () ->o FlareDispenser`MagId ->m (FlareDispenser`FlareType × ℕ)*
        Run () ≜
          (     sensor.Init(timerRef) ;
                detector.Init(sensor, flareControl) ;
                flareControl.Init(detector, timerRef) ;

start
(sensor) ;

start
(detector) ;

start
(flareControl) ;
                sensor.IsFinished() ;
                detector.IsFinished() ;
                return flareControl.IsFinished ()
          )
end World
```

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

| Name | #Calls | Coverage |
|---|---|---|
| World`Run | 1 | $\surd$ |
| **Total Coverage** | | **100%** |

# 2 Sensor Class

class *Sensor*
types
public *MissileType* = MISSILEA | MISSILEB | MISSILEC | NONE;
public *Angle* = $\mathbb{N}$
inv *num* $\triangleq$ *num* $\leq 360$
instance variables
*io* : *SensorIO* := new *SensorIO* ();
*threat* : [(*MissileType* | CONSUMED) $\times$ *Angle*] := *io.readThreat* ();
*timerRef* : *Timer*;

thread

```
(   while threat ≠ nil
    do (    ThreatConsumed() ;
            SetThreat() ;
            timerRef.StepTime()
       ) ;
    timerRef.Finished()
)
```

operations
*ThreatConsumed* : () $\xrightarrow{o}$ ()
*ThreatConsumed* () $\triangleq$
skip
sync
per *ThreatConsumed* $\Rightarrow$ *threat* $\neq$ nil $\wedge$ *threat*.#1 = CONSUMED
operations
public
*Init* : *Timer* $\xrightarrow{o}$ ()
*Init* (*newTimer*) $\triangleq$
*timerRef* := *newTimer*;

$SetThreat : () \xrightarrow{o} ()$
$SetThreat\,() \triangleq$
$threat := io.readThreat\,()$

sync

mutex($SetThreat$, $GetThreat$)

operations

public

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

sync

per $IsFinished \Rightarrow threat =$ nil

operations

public

$GetThreat : () \xrightarrow{o} [MissileType \times Angle]$
$GetThreat\,() \triangleq$
let $orgThreat = threat$ in
( if $threat \neq$ nil
then $threat :=$ mk- (CONSUMED, 0);
return $orgThreat$
)

sync

per $GetThreat \Rightarrow threat \neq$ nil $\Rightarrow threat.\#1 \neq$ CONSUMED

end $Sensor$

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

| Name | #Calls | Coverage |
|---|---|---|
| Sensor‘Init | 1 | √ |
| Sensor‘GetThreat | 9 | √ |
| Sensor‘SetThreat | 8 | √ |
| Sensor‘IsFinished | 1 | √ |
| Sensor‘ThreatConsumed | 8 | √ |
| **Total Coverage** | | **100%** |

# 3 Sensor IO Class

class $SensorIO$ is subclass of $IO$

instance variables

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

$mvList : (Sensor`MissileType \times Sensor`Angle)^* := [];$

operations
public
$SensorIO : () \xrightarrow{o} SensorIO$
$SensorIO\,() \triangleq$
( let mk-(-, $list$) =
$freadval[(Sensor`MissileType \times Sensor`Angle)^+]$
( "$scenario.txt$") in
$mvList := list;$
$curIndex := 1;$
return self
);
public
$readThreat : () \xrightarrow{o} [Sensor`MissileType \times Sensor`Angle]$
$readThreat\,() \triangleq$
if $curIndex \leq$ len $mvList$
then ( $curIndex := curIndex + 1;$
return $mvList\,(curIndex - 1)$
)
else return nil
end *SensorIO*

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

| Name | #Calls | Coverage |
|---|---|---|
| SensorIO`SensorIO | 1 | √ |
| SensorIO`readThreat | 9 | √ |
| **Total Coverage** | | **100%** |

# 4 Missile Detector Class

class *MissileDetector*
instance variables
$sensorRef : Sensor;$
$flareControlRef : FlareController;$
$threat : [Sensor`MissileType \times Sensor`Angle] :=$ mk-(NONE, 0);

thread

```
        while threat ≠ nil
        do let newThreat = sensorRef.GetThreat () in
           Update(newThreat)
operations
public
        IsFinished : () →o ()
        IsFinished () △
          skip
sync
        per IsFinished  ⇒  threat = nil
operations
        Update : [Sensor`MissileType × Sensor`Angle] →o ()
        Update (newThreat) △
          (    if newThreat = nil  ∨ (newThreat ≠ nil  ∧ newThreat.#1 ≠
NONE)
               then (    threat := newThreat;
                         flareControlRef.MissileIsHere(threat)
                    )
          );
public
        Init : Sensor × FlareController →o ()
        Init (newSensor, newFlareController) △
          (    sensorRef := newSensor;
               flareControlRef := newFlareController
          )
end MissileDetector
```

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

| Name | #Calls | Coverage |
|---|---|---|
| MissileDetector`Init | 1 | √ |
| MissileDetector`Update | 9 | √ |
| MissileDetector`IsFinished | 1 | √ |
| **Total Coverage** | | **100%** |

# 5 Flare Controller Class

```
class FlareController
instance variables
        dispensers : FlareDispenser`MagId →m FlareDispenser;
```

$missileDetectorRef : MissileDetector;$
$noMoreMissiles : \mathbb{B} := \mathsf{false};$

values
$mag1 : FlareDispenser`MagId = \mathsf{mk\text{-}token}("Magazine\ 1");$
$mag2 : FlareDispenser`MagId = \mathsf{mk\text{-}token}("Magazine\ 2");$
$mag3 : FlareDispenser`MagId = \mathsf{mk\text{-}token}("Magazine\ 3");$
$mag4 : FlareDispenser`MagId = \mathsf{mk\text{-}token}("Magazine\ 4");$
$magids : FlareDispenser`MagId\text{-}\mathsf{set} = \{mag1, mag2, mag3, mag4\}$
thread

for all $magid \in magids$
do
start
$(dispensers\ (\text{magid})\ )$
operations
public
$Init : MissileDetector \times Timer \xrightarrow{o} ()$
$Init\,(initMissileDetector, initTimerRef) \triangleq$
$(\ missileDetectorRef := initMissileDetector;$
$dispensers := \{mag \mapsto \mathsf{new}\ FlareDispenser\,(mag, initTimerRef) \mid$

$mag \in magids\}$
);
public
$IsFinished : () \xrightarrow{o} FlareDispenser`MagId \xrightarrow{m} (FlareDispenser`FlareType \times \mathbb{N})^*$
$IsFinished\,() \triangleq$
$(\ \mathsf{for\ all}\ magid \in magids$
$\mathsf{do}\ dispensers\ (\text{magid})\ .IsFinished()\,;$
$\mathsf{return}\ \{magid \mapsto dispensers\,(magid).GetResult\,() \mid magid \in magids\}$
)
sync
per $IsFinished \Rightarrow noMoreMissiles$
operations
public
$MissileIsHere : [Sensor`MissileType \times Sensor`Angle] \xrightarrow{o} ()$
$MissileIsHere\,(newMissileValue) \triangleq$
$(\ \mathsf{if}\ newMissileValue = \mathsf{nil}$
$\mathsf{then}\ noMoreMissiles := \mathsf{true}$

```
            elseif newMissileValue.#1 ≠ NONE
            then let mk-(misType, angle) = newMissileValue,
                     magid = Angle2MagId (angle) in
                 dispensers     (magid )   .NewMissileValue(misType)
        )
functions
    Angle2MagId : Sensor`Angle → FlareDispenser`MagId
    Angle2MagId (angle) ≜
      if angle < 90
      then mag1
      elseif angle < 180
      then mag2
      elseif angle < 270
      then mag3
      else mag4
end FlareController
```

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

| Name | #Calls | Coverage |
|---|---|---|
| FlareController`Init | 1 | √ |
| FlareController`IsFinished | 1 | √ |
| FlareController`Angle2MagId | 7 | √ |
| FlareController`MissileIsHere | 8 | √ |
| **Total Coverage** | | **100%** |

# 6 Flare Dispenser Class

```
class FlareDispenser
instance variables
    magid : MagId;
    currentMissileValue : Sensor`MissileType := NONE;
    latestMissileValue : Sensor`MissileType := NONE;
    outputSequence : (FlareType × ℕ)* := [];
    currentStep : ℕ := 0;
    fresh : 𝔹 := false;
    interrupt : Interrupt;

values
```

$responseDB : Sensor`MissileType \xrightarrow{m} Plan = \{\textsc{MissileA} \mapsto [\text{mk-}(\textsc{FlareOneA}, 900), \text{mk-}$
$\text{mk-}(\textsc{DoNothingA}, 100), \text{mk-}(\textsc{FlareOne}$
$\textsc{MissileB} \mapsto [\text{mk-}(\textsc{FlareTwoB}, 500), \text{mk-}$
$\textsc{MissileC} \mapsto [\text{mk-}(\textsc{FlareOneC}, 400), \text{mk-}$
$\text{mk-}(\textsc{FlareTwoC}, 400), \text{mk-}(\textsc{FlareOneC}$

$missilePriority : Sensor`MissileType \xrightarrow{m} \mathbb{N} = \{\textsc{MissileA} \mapsto 1,$
$\textsc{MissileB} \mapsto 2,$
$\textsc{MissileC} \mapsto 3,$
$\textsc{None} \mapsto 0\}$

types

public $MagId = \text{token};$
$Plan = PlanStep^*;$
public $PlanStep = FlareType \times \mathbb{N};$
public $FlareType = \textsc{FlareOneA} \mid \textsc{FlareTwoA} \mid \textsc{FlareOneB} \mid$

$\textsc{FlareTwoB} \mid \textsc{FlareOneC} \mid \textsc{FlareTwoC} \mid$

$\textsc{DoNothingA} \mid \textsc{DoNothingB} \mid \textsc{DoNothingC}$

thread

while true
do ( $StepAlgorithm()$ ;
if $currentMissileValue \neq \textsc{None}$
then let mk-$(\text{-}, delay\text{-}val) =$
$responseDB\,(currentMissileValue)\,(currentStep-$
$1)$ in
$interrupt.Alarm(delay\text{-}val)$
)

operations

public

$FlareDispenser : MagId \times Timer \xrightarrow{o} FlareDispenser$
$FlareDispenser\,(mid, t) \triangleq$
( $magid := mid;$
$interrupt :=$ new $Interrupt\,(t)$
);

```
        StepAlgorithm : () ^o→ ()
        StepAlgorithm () ≜
          (   if fresh
              then (   fresh := false;
                       CheckFreshData()
                   ) ;
              if currentMissileValue ≠ NONE
              then StepPlan()
          )
sync
        per StepAlgorithm  ⇒  fresh = true ∨ currentMissileValue ≠
NONE
operations
        CheckFreshData : () ^o→ ()
        CheckFreshData () ≜
          (   if HigherPriority (latestMissileValue, currentMissileValue)
              then StartPlan(latestMissileValue) ;
              latestMissileValue := NONE
          );
        HigherPriority : Sensor`MissileType × Sensor`MissileType ^o→ 𝔹
        HigherPriority (latest, current) ≜
          return missilePriority (latest) > missilePriority (current);
        StartPlan : Sensor`MissileType ^o→ ()
        StartPlan (newMissileValue) ≜
          (   currentMissileValue := newMissileValue;
              currentStep := 1
          );
        ReleaseAFlare : FlareType ^o→ ()
        ReleaseAFlare (ft) ≜
          outputSequence := outputSequence↷[mk- (ft, interrupt.GetTime ())];
        StepPlan : () ^o→ ()
        StepPlan () ≜
          if currentStep ≤ len responseDB (currentMissileValue)
          then (   let mk- (flare, -) = responseDB (currentMissileValue) (currentStep) in
                   ReleaseAFlare(flare) ;
                   currentStep := currentStep + 1
               )
          else (   currentMissileValue := NONE;
                   currentStep := 0
               ) ;
```

```
public
        GetResult : () ->o (FlareType × ℕ)*
        GetResult () △
          return outputSequence;
public
        IsFinished : () ->o ()
        IsFinished () △
          skip
sync
        per IsFinished  ⇒  currentStep = 0
operations
public
        NewMissileValue : Sensor`MissileType ->o ()
        NewMissileValue (misType) △
          (   interrupt.Inter() ;
              latestMissileValue := misType;
              fresh := true
          )
end FlareDispenser
```

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

| Name | #Calls | Coverage |
|---|---|---|
| FlareDispenser`StepPlan | 30 | √ |
| FlareDispenser`GetResult | 4 | √ |
| FlareDispenser`StartPlan | 7 | √ |
| FlareDispenser`IsFinished | 4 | √ |
| FlareDispenser`ReleaseAFlare | 24 | √ |
| FlareDispenser`StepAlgorithm | 30 | √ |
| FlareDispenser`CheckFreshData | 7 | √ |
| FlareDispenser`FlareDispenser | 4 | √ |
| FlareDispenser`HigherPriority | 7 | √ |
| FlareDispenser`NewMissileValue | 7 | √ |
| **Total Coverage** | | **100%** |

# 7 Timer Class

```
class Timer
instance variables
```

$currentTime : \mathbb{N} := 0;$
$finished : \mathbb{B} := \mathsf{false};$

operations
public
$Finished : () \xrightarrow{o} ()$
$Finished\ () \triangleq$
$finished := \mathsf{true}$
sync
mutex($StepTime$, $GetTime$)
operations
public
$StepTime : () \xrightarrow{o} ()$
$StepTime\ () \triangleq$
$currentTime := currentTime + stepLength;$
public
$GetTime : () \xrightarrow{o} \mathbb{N}$
$GetTime\ () \triangleq$
return $currentTime$
values
$stepLength : \mathbb{N} = 100$
end $Timer$

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

| Name | #Calls | Coverage |
|---|---|---|
| Timer‘GetTime | 48 | √ |
| Timer‘Finished | 1 | √ |
| Timer‘StepTime | 8 | √ |
| **Total Coverage** | | **100%** |

# 8 Interrupt Class

class $Interrupt$
instance variables
$timer : Timer;$
$currentAlarm : [\mathbb{N}] := \mathsf{nil}\ ;$

operations
public

$Interrupt : Timer \xrightarrow{o} Interrupt$
$Interrupt\,(t) \triangleq$
$timer := t;$

public

$Alarm : \mathbb{N} \xrightarrow{o} ()$
$Alarm\,(n) \triangleq$
$SetAlarm(n)\,;$
$SetAlarm : \mathbb{N} \xrightarrow{o} ()$
$SetAlarm\,(n) \triangleq$
$currentAlarm := timer.GetTime\,() + n;$

public

$Inter : () \xrightarrow{o} ()$
$Inter\,() \triangleq$
$currentAlarm :=$ nil ;

public

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

end *Interrupt*

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

| **Name** | **#Calls** | **Coverage** |
|---|---|---|
| Interrupt‘Alarm | 24 | √ |
| Interrupt‘Inter | 7 | √ |
| Interrupt‘GetTime | 24 | √ |
| Interrupt‘SetAlarm | 24 | √ |
| Interrupt‘Interrupt | 4 | √ |
| **Total Coverage** | | **100%** |

# 9 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 $IO$