Sequential Missile VDM++ Model

Peter Gorm Larsen

2006

1 World

class $World$
instance variables
$sensor : Sensor := \mathsf{new}\ Sensor\,();$
$detector : MissileDetector := \mathsf{new}\ MissileDetector\,();$
$flareControl : FlareController := \mathsf{new}\ FlareController\,();$
$timerRef : Timer := \mathsf{new}\ Timer\,();$
$inputVals : ([Sensor`MissileType \times Sensor`Angle] \times \mathbb{N})^* := [];$

operations
public
$Run : () \xrightarrow{o} (FlareDispenser`MagId \xrightarrow{m} (FlareDispenser`FlareType \times \mathbb{N})^*) \times ((([Sensor`MissileType \times Sensor`Angle] \times \mathbb{N})^*)$
$Run\,() \triangleq$
$(\quad detector.Init(sensor, flareControl)\,;$
$\quad flareControl.Init(detector, timerRef)\,;$
$\quad \mathsf{while}\ \neg\,(sensor.IsFinished\,() \wedge detector.IsFinished\,() \wedge$
$\qquad flareControl.IsFinished\,())$

```
        do (    inputVals := inputVals ↷
                                [mk- (sensor.ReadThreat (), timerRef.GetTime ())];
                if ¬ detector.IsFinished ()
                then detector.Step() ;
                if ¬ flareControl.IsFinished ()
                then flareControl.Step() ;
                timerRef.StepTime() ;
                if ¬ sensor.IsFinished ()
                then sensor.SetThreat()
           ) ;
        return mk- (flareControl.GetResult (), inputVals)
    )
end World
```

Test Suite : vdm.tc
Class : World

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

2 Sensor Class

```
class Sensor
types
    public MissileType = MissileA | MissileB | MissileC | None;
    public Angle = ℕ
    inv num ≜ num ≤ 360
instance variables
    io : SensorIO := new SensorIO ();
    threat : [(MissileType | Consumed) × Angle] := io.readThreat ();

operations
public
    SetThreat : () →o ()
    SetThreat () ≜
      threat := io.readThreat ();
public
    ReadThreat : () →o [MissileType × Angle]
    ReadThreat () ≜
      return threat;
```

public
$IsFinished : () \xrightarrow{o} \mathbb{B}$
$IsFinished\,() \triangleq$
return $threat$ = nil ;
public
$GetThreat : () \xrightarrow{o} [MissileType \times Angle]$
$GetThreat\,() \triangleq$
let $orgthreat = threat$ in
(if $threat \neq$ nil
then $threat :=$ mk- $(\text{CONSUMED}, 0)$;
return $orgthreat$
)
end $Sensor$

Test Suite : vdm.tc
Class : Sensor

Name	**#Calls**	**Coverage**
Sensor‘GetThreat	9	$\surd$
Sensor‘SetThreat	8	$\surd$
Sensor‘IsFinished	45	$\surd$
Sensor‘ReadThreat	22	$\surd$
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 : () →^o SensorIO
        SensorIO () △
          (    let mk- (-, list) =
                        freadval[(Sensor‘MissileType × Sensor‘Angle)+]
                              (
                               "scenario.txt") in
               mvList := list;
               curIndex := 1
          );
public
        readThreat : () →^o [Sensor‘MissileType × Sensor‘Angle]
        readThreat () △
          if curIndex ≤ 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 × Sensor‘Angle] := mk- (NONE, 0);

operations
public
```

```
        Init : Sensor × FlareController →o ()
        Init (newSensor, newFlareController) △
          (    sensorRef := newSensor;
               flareControlRef := newFlareController
          );
public
        Step : () →o ()
        Step () △
          let newthreat = sensorRef.GetThreat () in
          Update(newthreat) ;
        Update : [Sensor`MissileType × Sensor`Angle] →o ()
        Update (newThreat) △
          (    if newThreat = nil ∨ (newThreat ≠ nil ∧ newThreat.#1 ≠
NONE)
               then (    threat := newThreat;
                         flareControlRef.MissileIsHere(newThreat)
                    )
          );
public
        IsFinished : () →o B
        IsFinished () △
          return threat = nil
end MissileDetector
```

Test Suite : vdm.tc
Class : MissileDetector

Name	#Calls	Coverage
MissileDetector`Init	1	√
MissileDetector`Step	9	√
MissileDetector`Update	9	√
MissileDetector`IsFinished	37	√
Total Coverage		**100%**

5 Flare Controller Class

```
class FlareController
instance variables
        dispensers : FlareDispenser`MagId →m FlareDispenser;
        missileDetectorRef : MissileDetector;
        noMoreMissiles : B := false;
```

values

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

operations

public

$Init : MissileDetector \times Timer \xrightarrow{o} ()$
$Init\,(initMissileDetector, initTimerRef) \triangleq$
($missileDetectorRef := initMissileDetector$;
$dispensers := \{mag \mapsto \text{new}\ FlareDispenser\,(mag, initTimerRef) \mid$

$mag \in magids\}$
);

public

$Step : () \xrightarrow{o} ()$
$Step\,() \triangleq$
for all $magid \in magids$
do $dispensers$ (magid) $.Step()$;

public

$IsFinished : () \xrightarrow{o} \mathbb{B}$
$IsFinished\,() \triangleq$
return $noMoreMissiles \wedge$
$\forall\, magid \in magids \cdot dispensers\,(magid).GetCurrentStep\,() =$
0;

public

$GetResult : () \xrightarrow{o} FlareDispenser`MagId \xrightarrow{m} (FlareDispenser`FlareType \times \mathbb{N})^*$
$GetResult\,() \triangleq$
return $\{magid \mapsto dispensers\,(magid).GetResult\,() \mid magid \in magids\}$;

public

$MissileIsHere : [Sensor`MissileType \times Sensor`Angle] \xrightarrow{o} ()$
$MissileIsHere\,(newMissileValue) \triangleq$
(if $newMissileValue = \text{nil}$
then $noMoreMissiles := \text{true}$
elseif $newMissileValue.\#1 \neq \text{NONE}$
then let mk-$(misType, angle) = newMissileValue$,
$magid = Angle2MagId\,(angle)$ in
$dispensers$ (magid) $.NewMissileValue(misType)$
)

functions

$Angle2MagId : Sensor`Angle \rightarrow FlareDispenser`MagId$

$Angle2MagId\,(angle) \triangleq$

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`Step	22	√
FlareController`GetResult	1	√
FlareController`IsFinished	36	√
FlareController`Angle2MagId	7	√
FlareController`MissileIsHere	8	√
Total Coverage		**100%**

6 Flare Dispenser Class

class *FlareDispenser*

instance variables

$magid : MagId;$

$currentMissileValue : Sensor`MissileType := \text{NONE};$

$latestMissileValue : Sensor`MissileType := \text{NONE};$

$outputSequence : (FlareType \times \mathbb{N})^{*} := [];$

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

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

$fresh : \mathbb{B} :=$ 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 $FlareType = \textsc{FlareOneA} \mid \textsc{FlareTwoA} \mid \textsc{FlareOneB} \mid$

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

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

operations

public

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

public

$Step : () \xrightarrow{o} ()$
$Step\,() \triangleq$
(if $interrupt.CheckAwake\,()$
then ($StepAlgorithm()\,;$
if $currentMissileValue \neq \textsc{None}$
then let mk-(-, $delay\text{-}val$) =
$responseDB\,(currentMissileValue)\,(currentStep-$
1) in
$interrupt.Alarm(delay\text{-}val)$
)
);

```
StepAlgorithm : () →o ()
StepAlgorithm () △
  (   if fresh
      then (   fresh := false;
               CheckFreshData()
           ) ;
      if currentMissileValue ≠ NONE
      then StepPlan()
  );
CheckFreshData : () →o ()
CheckFreshData () △
  (   if HigherPriority (latestMissileValue, currentMissileValue)
      then StartPlan(latestMissileValue) ;
      latestMissileValue := NONE;
      numberOfFreshValues := numberOfFreshValues + 1
  );
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
        GetCurrentStep : () ^o→ ℕ
        GetCurrentStep () △
          return currentStep;
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`Step	88	√
FlareDispenser`StepPlan	21	√
FlareDispenser`GetResult	4	√
FlareDispenser`StartPlan	7	√
FlareDispenser`ReleaseAFlare	17	√
FlareDispenser`StepAlgorithm	47	√
FlareDispenser`CheckFreshData	7	√
FlareDispenser`FlareDispenser	4	√
FlareDispenser`GetCurrentStep	33	√
FlareDispenser`HigherPriority	7	√
FlareDispenser`NewMissileValue	7	√
Total Coverage		**100%**

7 Timer Class

```
class Timer
instance variables
        currentTime : ℕ := 0;

values
```

$stepLength : \mathbb{N} = 100$

operations

public

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

$StepTime\,() \triangleq$

$currentTime := currentTime + stepLength;$

public

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

$GetTime\,() \triangleq$

return $currentTime$

end $Timer$

Test Suite : vdm.tc

Class : Timer

Name	**#Calls**	**Coverage**
Timer‘GetTime	121	$\surd$
Timer‘StepTime	22	$\surd$
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

$Alarm : \mathbb{N} \xrightarrow{o} ()$

$Alarm\,(n) \triangleq$

$SetAlarm(n)\ ;$

public

$CheckAwake : () \xrightarrow{o} \mathbb{B}$

$CheckAwake\,() \triangleq$

return $currentAlarm = \mathsf{nil}\ \vee$

$currentAlarm \leq timer.GetTime\,();$

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

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	17	$\surd$
Interrupt‘Inter	7	$\surd$
Interrupt‘GetTime	17	$\surd$
Interrupt‘SetAlarm	17	$\surd$
Interrupt‘Interrupt	4	$\surd$
Interrupt‘CheckAwake	88	$\surd$
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

$fwriteval[@p] : \text{char}^+ \times @p \times filedirective \to \mathbb{B}$
$fwriteval\,(filename, val, fdir) \triangleq$
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