The Sequential Missile VDM++ Model

Peter Gorm Larsen and Marcel Verhoef

2007

1 The Overall Class Diagram

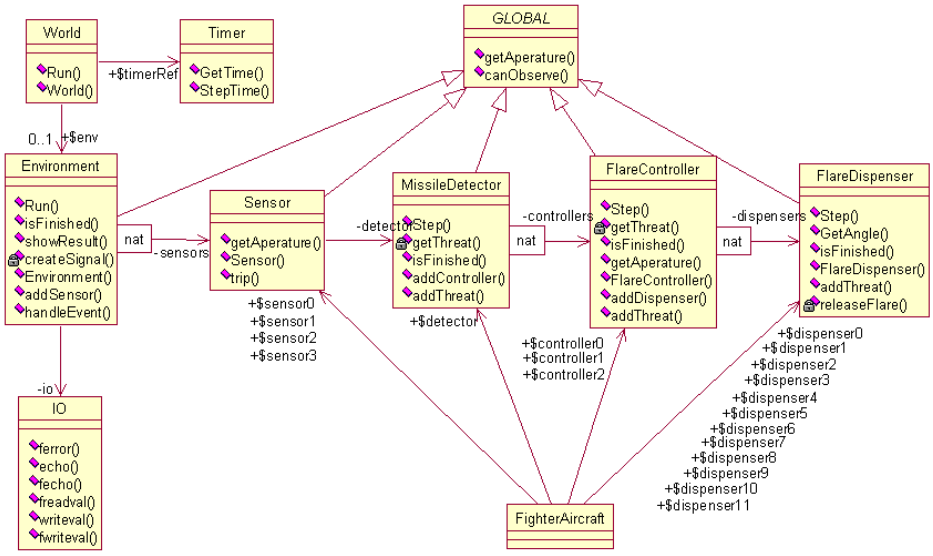

Figure 1: Overview of the classes in the sequential CM model

2 The World Class

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

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)
          );
public
        Run : () ==> ()
        Run () == env.
          Run()
end World
```

Test Suite : vdm.tc
Class : World

Name	#Calls	Coverage
World‘Run	2	$\surd$
World‘World	2	$\surd$
Total Coverage		**100%**

3 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*

4 The Environment Class

class *Environment* is subclass of *GLOBAL*
types
public $inline = EventId \times MissileType \times Angle \times Time;$
public $outline = EventId \times FlareType \times Angle \times Time \times Time$
instance variables

```
        io : IO := new IO ();
        inlines : inline* := [];
        outlines : outline* := [];
        ranges : ℕ m→ (Angle × Angle) := {↦};
        sensors : ℕ m→ Sensor := {↦};
        inv dom ranges = dom sensors
        evid : [EventId] := nil ;
        busy : 𝔹 := true;

operations
public
        Environment : char* o→ Environment
        Environment (fname) ≜
          def mk-(-, input) = io.freadval[inline*] (fname) in
          inlines := input;
public
        addSensor : Sensor o→ ()
        addSensor (psens) ≜
          (   dcl id : ℕ := card dom ranges + 1;
              atomic (   ranges := ranges ⩊ {id ↦ psens.getAperture ()};
                         sensors := sensors ⩊ {id ↦ psens}
              )
          );
public
        Run : () o→ ()
        Run () ≜
          (   while ¬ (isFinished () ∧ CM‘detector.isFinished ())
              do (   evid := createSignal ();
                     CM‘detector.Step() ;
                     World‘timerRef.StepTime()
                 ) ;
              showResult()
          );
private
        createSignal : () o→ [EventId]
        createSignal () ≜
          (   if len inlines > 0
              then (   dcl curtime : Time := World‘timerRef.GetTime (),
                           done : 𝔹 := false;
                       while ¬ 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 eventid
                           )
                      else (   done := true;
                               return nil
                           )
                  )
              else (   busy := false;
                       return nil
                   )
          );
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 B
      isFinished() △
        return inlines = [] ∧ ¬ busy
end Environment
```

Test Suite : vdm.tc
Class : Environment

Name	#Calls	Coverage
Environment‘Run	2	√
Environment‘addSensor	8	√
Environment‘isFinished	404	√

Name	#Calls	Coverage
Environment‘showResult	2	√
Environment‘Environment	2	√
Environment‘handleEvent	42	√
Environment‘createSignal	402	√
Total Coverage		**100%**

5 The Global Class

class $GLOBAL$
values
public
$SENSOR\text{-}APERTURE = 90;$
public
$FLARE\text{-}APERTURE = 120;$
public
$DISPENSER\text{-}APERTURE = 30$
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 $EventId = \mathbb{N};$
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	161	$\surd$
GLOBAL‘getAperture	0	0%
Total Coverage		**96%**

6 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	14	$\surd$
Sensor‘Sensor	8	$\surd$
Sensor‘getAperture	8	$\surd$
Total Coverage		**100%**

7 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 ($ranges := ranges \biguplus \{nid \mapsto pctrl.getAperture\,()\};$
$controllers := controllers \biguplus \{nid \mapsto pctrl\}$
)
);
public
$Step : () \xrightarrow{o} ()$
$Step\,() \triangleq$
(if $threats \neq []$
then 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 $threats > 0;$
for all $id \in$ dom $controllers$
do $controllers$ (id) $.Step()$
);
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 B
        isFinished () ≜
          return ∀ id ∈ dom controllers ·
                      controllers (id).isFinished ()
end MissileDetector
```

Test Suite : vdm.tc
Class : MissileDetector

Name	#Calls	Coverage
MissileDetector‘Step	402	√
MissileDetector‘addThreat	14	√
MissileDetector‘getThreat	14	√
MissileDetector‘isFinished	240	√
MissileDetector‘addController	6	√
Total Coverage		**100%**

8 Flare Controller Class

```
class FlareController is subclass of GLOBAL
instance variables
        private aperture : Angle;
        ranges : N →m (Angle × Angle) := {↦};
        dispensers : N →m FlareDispenser := {↦};
        inv dom ranges = dom dispensers
        threats : (EventId × MissileType × Angle × Time)* := [];
        busy : B := false;
```

```
operations
public
        FlareController : Angle ==o=> FlareController
        FlareController (papp) ==
          aperture := papp;
public
        addDispenser : FlareDispenser ==o=> ()
        addDispenser (pfldisp) ==
          let angle = aperture + pfldisp.GetAngle () in
          (   dcl id : ℕ := card dom ranges + 1;
              atomic (   ranges := ranges ⩊
                                   {id ↦ mk- (angle, DISPENSER-APERTURE)};
                         dispensers := dispensers ⩊ {id ↦ pfldisp}
              )
          );
public
        Step : () ==o=> ()
        Step () ==
          (   if threats ≠ []
              then 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;
              for all id ∈ dom dispensers
              do dispensers     (id )  .Step()
          );
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
```

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

public

$isFinished : () \xrightarrow{o} \mathbb{B}$
$isFinished\,() \triangleq$
return $\forall\, id \in \mathsf{dom}\ dispensers \cdot$
$dispensers\,(id).isFinished\,()$

end *FlareController*

Test Suite : vdm.tc
Class : FlareController

Name	**#Calls**	**Coverage**
FlareController‘Step	1206	√
FlareController‘addThreat	14	√
FlareController‘getThreat	14	√
FlareController‘isFinished	244	√
FlareController‘getAperture	6	√
FlareController‘addDispenser	24	√
FlareController‘FlareController	6	√
Total Coverage		**100%**

9 Flare Dispenser Class

class *FlareDispenser* is subclass of *GLOBAL*

values

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

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

types

public $Plan = PlanStep^*$;
public $PlanStep = FlareType \times Time$

instance variables

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

operations

public

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

public

$Step : () \xrightarrow{o} ()$
$Step\,() \triangleq$
  if len $curplan > 0$
  then (dcl $curtime : Time := World`timerRef.GetTime\,()$,
    $first : PlanStep :=$ hd $curplan$,
    $next : Plan :=$ tl $curplan$;
    let mk- $(fltp, fltime) = first$ in
    (if $fltime \leq curtime$
      then ($releaseFlare(eventid, fltp, fltime, curtime)$;
        $curplan := next$;
        if len $next = 0$
        then ($curprio := 0$;
          $busy :=$ false
)
)
)
) ;

public

$GetAngle : () \xrightarrow{o} \mathbb{N}$
$GetAngle\,() \triangleq$
  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
        releaseFlare : EventId × FlareType × Time × Time →o ()
        releaseFlare (evid, pfltp, pt1, pt2) △ World`env.
          handleEvent(evid, pfltp, aperture, pt1, pt2) ;
public
        isFinished : () →o B
        isFinished () △
          return ¬ busy
end FlareDispenser
```

Test Suite : vdm.tc
Class : FlareDispenser

Name	#Calls	Coverage
FlareDispenser`Step	4824	√
FlareDispenser`GetAngle	24	√
FlareDispenser`addThreat	14	√
FlareDispenser`isFinished	500	√
FlareDispenser`releaseFlare	42	√
FlareDispenser`FlareDispenser	24	√
Total Coverage		**100%**

10 The Timer Class

class $Timer$
instance variables
$currentTime : \mathbb{N} := 0;$

values
$stepLength : \mathbb{N} = 10$
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	1262	$\surd$
Timer'StepTime	402	$\surd$
Total Coverage		**100%**

11 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