The Sequential Missile VDM++ Model

Peter Gorm Larsen and Marcel Verhoef

August 2006

1 The Overall Class Diagram

2 The World Class

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

operations
public
        World : () $\xrightarrow{o}$ 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)
          );
public
        Run : () $\xrightarrow{o}$ ()
        Run () ≜ env.
          Run()
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 FighterAircraft
instance variables
        public static detector : MissileDetector := new MissileDetector ();
        public static sensor0 : Sensor := new Sensor (detector, 0);
        public static sensor1 : Sensor := new Sensor (detector, 90);
        public static sensor2 : Sensor := new Sensor (detector, 180);
        public static sensor3 : Sensor := new Sensor (detector, 270);
        public static controller0 : FlareController := new FlareController (0);
        public static controller1 : FlareController := new FlareController (120);
        public static controller2 : FlareController := new FlareController (240);
        public static dispenser0 : FlareDispenser := new FlareDispenser (0);
        public static dispenser1 : FlareDispenser := new FlareDispenser (30);
        public static dispenser2 : FlareDispenser := new FlareDispenser (60);
        public static dispenser3 : FlareDispenser := new FlareDispenser (90);
        public static dispenser4 : FlareDispenser := new FlareDispenser (0);
        public static dispenser5 : FlareDispenser := new FlareDispenser (30);
        public static dispenser6 : FlareDispenser := new FlareDispenser (60);
        public static dispenser7 : FlareDispenser := new FlareDispenser (90);
        public static dispenser8 : FlareDispenser := new FlareDispenser (0);
        public static dispenser9 : FlareDispenser := new FlareDispenser (30);
        public static dispenser10 : FlareDispenser := new FlareDispenser (60);
        public static dispenser11 : FlareDispenser := new FlareDispenser (90);

end FighterAircraft
```

4 The Environment Class

```
class Environment is subclass of GLOBAL
types
        public inline = EventId × MissileType × Angle × ℕ;
        public outline = EventId × FlareType × Angle × ℕ × ℕ
instance variables
```

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

operations
public
          $Environment :$ 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 $id : \mathbb{N} :=$ card dom $ranges + 1$;
                   atomic (   $ranges := ranges \stackrel{m}{\cup} \{id \mapsto psens.getAperature\,()\}$;
                        $sensors := sensors \stackrel{m}{\cup} \{id \mapsto psens\}$
                   )
             );
public
          $Run : () \xrightarrow{o} ()$
          $Run\,() \triangleq$
             (     while $\neg\,(isFinished\,() \land FighterAircraft`detector.isFinished\,())$
                   do (     $evid := createSignal\,()$;
                            FighterAircraft`detector.$Step()$ ;
                            World`timerRef.$StepTime()$
                       ) ;
                   $showResult()$
             );
private
          $createSignal : () \xrightarrow{o} [EventId]$
          $createSignal\,() \triangleq$
             (     if len $inlines > 0$
                   then (     dcl $curtime : \mathbb{N} := World`timerRef.GetTime\,()$,
                                  $done : \mathbb{B} :=$ false;
                              while $\neg\,done$
```

```
                    do def mk- $(eventid, pmt, pa, pt) =$ hd $inlines$ in
                       if $pt \leq curtime$
                       then (   for all $id \in$ 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
       $handleEvent : EventId \times FlareType \times Angle \times \mathbb{N} \times \mathbb{N} \xrightarrow{o} ()$
       $handleEvent(evid, pfltp, angle, pt1, pt2) \triangleq$
          (   $outlines := outlines \frown [$mk- $(evid, pfltp, angle, pt1, pt2)]$
          );
public
       $showResult : () \xrightarrow{o} ()$
       $showResult() \triangleq$
          def - $= io.writeval[outline^*](outlines)$ in
          skip;
public
       $isFinished : () \xrightarrow{o} \mathbb{B}$
       $isFinished() \triangleq$
          return $inlines = [] \wedge \neg\, busy$
end $Environment$
```

Test Suite : vdm.tc

Class : Environment

Name	**#Calls**	**Coverage**
Environment‘Run	1	√
Environment‘addSensor	4	√
Environment‘isFinished	202	√

Name	**#Calls**	**Coverage**
Environment‘showResult	1	√
Environment‘Environment	1	√
Environment‘handleEvent	21	√
Environment‘createSignal	201	√
Total Coverage		**100%**

5 The Global Class

```
class $GLOBAL$
values
public
        $SENSOR\text{-}APERATURE = 90;$
public
        $FLARE\text{-}APERATURE = 120;$
public
        $DISPENSER\text{-}APERATURE = 30$
types
        public $MissileType$ = MISSILEA | MISSILEB | MISSILEC | NONE;
        public $FlareType$ = FLAREONEA | FLARETWOA | DONOTHINGA |

                             FLAREONEB | FLARETWOB | DONOTHINGB |

                             FLAREONEC | FLARETWOC | 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 $(pangle < pright \lor pangle \geq pleft)$
          else return $(pangle \geq pleft \land pangle < pright)$ ;
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%**

6 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	$\surd$

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

operations
public
        $addController : FlareController \xrightarrow{o} ()$
        $addController\,(pctrl) \triangleq$
          (    dcl $nid : \mathbb{N} :=$ card dom $ranges + 1;$
               atomic (   $ranges := ranges \uplus \{nid \mapsto pctrl.getAperature\,()\};$
                     $controllers := controllers \uplus \{nid \mapsto pctrl\}$
               )
          );
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 $controllers$    $(\mathrm{id}\ )$ $.addThreat(evid, pmt, pa, pt);$
               $busy :=$ len $threats > 0;$
               for all $id \in$ dom $controllers$
               do $controllers$    $(\mathrm{id}\ )$  $.Step()$
          );
public
```

```
        $addThreat : EventId \times MissileType \times Angle \times \mathbb{N} \xrightarrow{o} ()$
        $addThreat\,(evid, pmt, pa, pt) \triangleq$
          (    $threats := threats \curvearrowright [\text{mk-}\,(evid, pmt, pa, pt)];$
               $busy := \text{true}$
          );
private
        $getThreat : () \xrightarrow{o} EventId \times MissileType \times Angle \times \mathbb{N}$
        $getThreat\,() \triangleq$
          (    $\text{dcl } res : EventId \times MissileType \times Angle \times \mathbb{N} := \text{hd } threats;$
               $threats := \text{tl } threats;$
               $\text{return } res$
          );
public
        $isFinished : () \xrightarrow{o} \mathbb{B}$
        $isFinished\,() \triangleq$
          $\text{return } \forall\, id \in \text{dom } controllers \cdot$
                        $controllers\,(id).isFinished\,()$
end $MissileDetector$
```

Test Suite : vdm.tc
Class : MissileDetector

Name	#Calls	Coverage
MissileDetector‘Step	201	√
MissileDetector‘addThreat	7	√
MissileDetector‘getThreat	7	√
MissileDetector‘isFinished	120	√
MissileDetector‘addController	3	√
Total Coverage		**100%**

8 Flare Controller Class

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

```
operations
public
        $FlareController : Angle \xrightarrow{o} FlareController$
        $FlareController\,(papp) \triangleq$
          $aperture := papp;$
public
        $addDispenser : FlareDispenser \xrightarrow{o} ()$
        $addDispenser\,(pfldisp) \triangleq$
          let $angle = aperture + pfldisp.GetAngle\,()$ in
          (    dcl $id : \mathbb{N} :=$ card dom $ranges + 1;$
               atomic  (   $ranges := ranges \uplus \{id \mapsto$ mk- $(angle, DISPENSER\text{-}APERATURE)\};$
                         $dispensers := dispensers \uplus \{id \mapsto pfldisp\}$
                       )
          );
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 $dispensers$     (id )  $.addThreat(evid, pmt, pt)$ ;
               $busy :=$ len $threats > 0;$
               for all $id \in$ dom $dispensers$
               do $dispensers$     (id )  $.Step()$
          );
public
        $getAperature : () \xrightarrow{o} GLOBAL`Angle \times GLOBAL`Angle$
        $getAperature\,() \triangleq$
          return mk- $(aperature, FLARE\text{-}APERATURE);$
public
        $addThreat : EventId \times MissileType \times Angle \times \mathbb{N} \xrightarrow{o} ()$
        $addThreat\,(evid, pmt, pa, pt) \triangleq$
          (    $threats := threats \curvearrowright [$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
        $isFinished : () \xrightarrow{o} \mathbb{B}$
        $isFinished\ () \triangleq$
          return $\forall\ id \in$ dom $dispensers \cdot$
                    $dispensers\ (id).isFinished\ ()$
end $FlareController$
```

Test Suite :	vdm.tc
Class :	FlareController

Name	**#Calls**	**Coverage**
FlareController‘Step	603	√
FlareController‘addThreat	7	√
FlareController‘getThreat	7	√
FlareController‘isFinished	122	√
FlareController‘addDispenser	12	√
FlareController‘getAperature	3	√
FlareController‘FlareController	3	√
Total Coverage		**100%**

9 Flare Dispenser Class

```
class $FlareDispenser$ is subclass of $GLOBAL$
values
        $responseDB : MissileType \xrightarrow{m} Plan = \{$MissileA $\mapsto$ [mk- (FlareOneA, 900),
                                                   mk- (FlareTwoA, 500),
                                                   mk- (DoNothingA, 100),
                                                   mk- (FlareOneA, 500)],
                                                  MissileB $\mapsto$ [mk- (FlareTwoB, 500),
                                                   mk- (FlareTwoB, 700)],
                                                  MissileC $\mapsto$ [mk- (FlareOneC, 400),
                                                   mk- (DoNothingC, 100),
                                                   mk- (FlareTwoC, 400),
                                                   mk- (FlareOneC, 500)]$\}$;
```

```
          $missilePriority : MissileType \xrightarrow{m} \mathbb{N} = \{\textsc{None} \mapsto 0,$
                                                     $\textsc{MissileA} \mapsto 1,$
                                                     $\textsc{MissileB} \mapsto 2,$
                                                     $\textsc{MissileC} \mapsto 3\}$
types
          public $Plan = PlanStep^*;$
          public $PlanStep = FlareType \times \mathbb{N}$
instance variables
          public $curplan : Plan := [];$
          $curprio : \mathbb{N} := 0;$
          $busy : \mathbb{B} :=$ false;
          $angle : \mathbb{N};$
          $eventid : [EventId];$

operations
public
          $FlareDispenser : \mathbb{N} \xrightarrow{o} FlareDispenser$
          $FlareDispenser\,(ang) \triangleq$
            (     $angle := ang$
            );
public
          $Step : () \xrightarrow{o} ()$
          $Step\,() \triangleq$
            if len $curplan > 0$
            then (     dcl $curtime : \mathbb{N} := World`timerRef.GetTime\,(),$
                           $first : PlanStep :=$ 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 $angle$;
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 (    $newplan := newplan \curvearrowright$ [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
        $releaseFlare : EventId \times FlareType \times \mathbb{N} \times \mathbb{N} \xrightarrow{o} ()$
        $releaseFlare\ (evid, pfltp, pt1, pt2) \triangleq$ World`env.
          $handleEvent(evid, pfltp, angle, pt1, pt2)$ ;
public
        $isFinished : () \xrightarrow{o} \mathbb{B}$
        $isFinished\ () \triangleq$
          return $\neg\ busy$
end $FlareDispenser$
```

Test Suite : vdm.tc
Class : FlareDispenser

Name	**#Calls**	**Coverage**
FlareDispenser`Step	2412	√
FlareDispenser`GetAngle	12	√
FlareDispenser`addThreat	7	√
FlareDispenser`isFinished	250	√
FlareDispenser`releaseFlare	21	√
FlareDispenser`FlareDispenser	12	√
Total Coverage		**100%**

10 The Timer Class

```
class $Timer$
instance variables
        $currentTime : \mathbb{N} := 0;$
        $finished : \mathbb{B} :=$ false;

operations
public
        $StepTime : () \xrightarrow{o} ()$
        $StepTime\ () \triangleq$
          $currentTime := currentTime + stepLength;$
public
        $GetTime : () \xrightarrow{o} \mathbb{N}$
        $GetTime\ () \triangleq$
          return $currentTime$
values
        $stepLength : \mathbb{N} = 10$
end $Timer$
```

Test Suite :	vdm.tc
Class :	Timer

Name	**#Calls**	**Coverage**
Timer‘GetTime	631	√
Timer‘StepTime	201	√
Total Coverage		**100%**

11 Standard IO Class

```
class $IO$
types
        public $filedirective =$ START | APPEND
functions
public
        $writeval[@p] : @p \rightarrow \mathbb{B}$
        $writeval\ (val) \triangleq$
          is not yet specified;
public
```

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

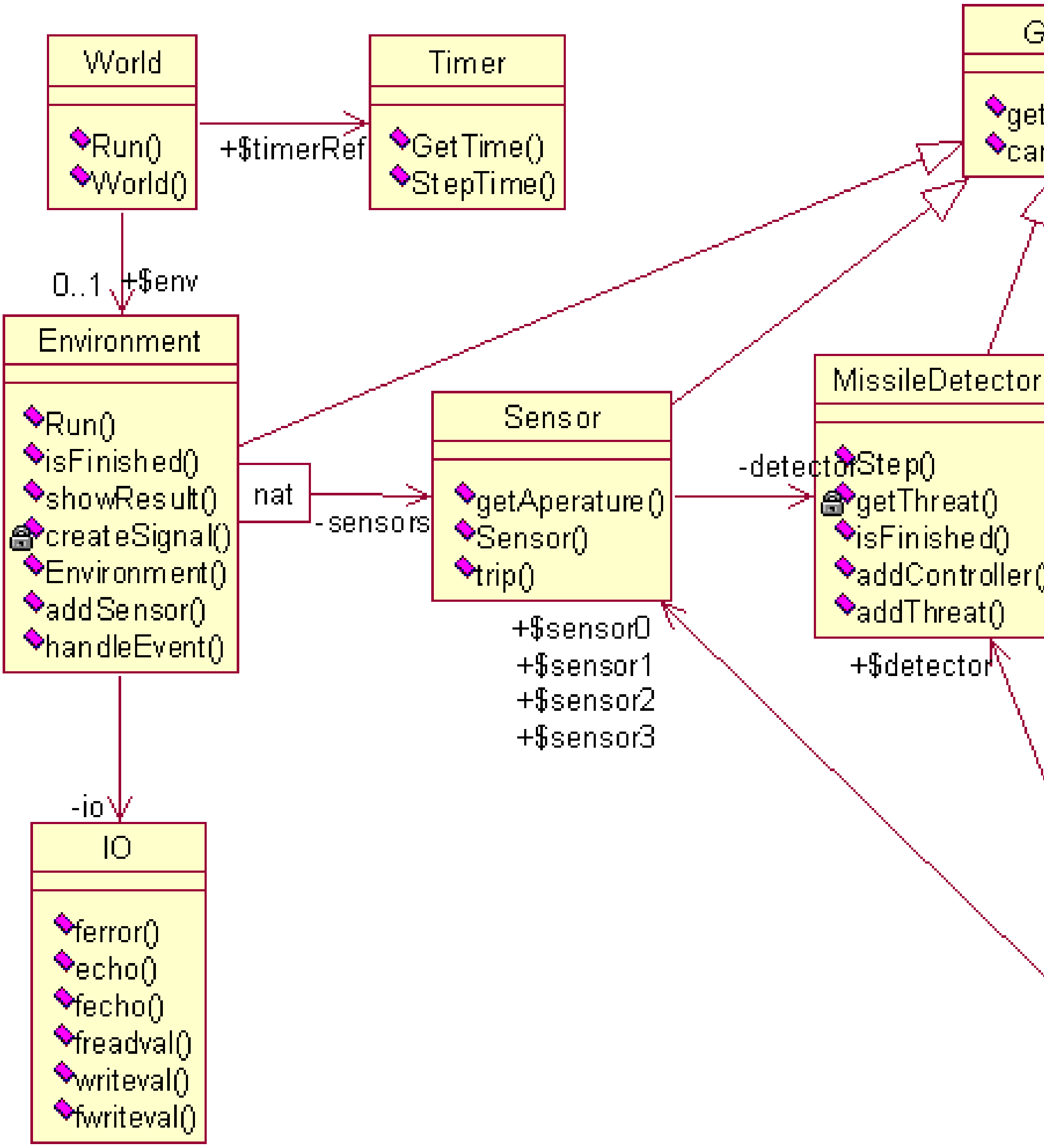

Figure 1: Overview of the classes in the sequential CM model