Oil rig access management system
TI-VDM2 project

Kim Christensen

Delivery date: 19/03-2009
Supervisor: Peter Gorm Larsen

May 27, 2009

Contents

Index

AccessController, 7, **7**, *7*, *16, 17*, *21*
AccessControllerTest, 27
addArea, **19**
addBoat, **11**
addEquipment, **11**
addEvent, **7**
addOutline, **17**
addPerson, **19**
Aid, **6**, *6*, *8–11*, *19–21*, **22**, *22*
AreaController, 9, **10**, *10*, *21*
AreaControllerTest, 25
areas, 8, 9, 13
AttemptRemoveLifeboat, **8**

Bid, **6**, *6*, *8, 9*, *11*, *13–15*, *18, 19*, *21*, **22**, *22, 23*
boatMaintenance, **14**
BUS, *21*

canRemoveBoat, **11**
canRemoveEquipment, **11**
changeArea, **22**
changeBoat, **12**, **23**
CPU, *21*

disable, **20**
doAddStaying, **13**
doAddTransit, **13**

enable, **20**
enterPerson, **12**
Environment, *6*, 16, **16**, *16*
event, **6**, *7*, *16*
eventType, **6**, *6*

findBoat, **13, 14**
FireEquipment, *9*, *11*, *21*

getArea, **22**
getAreas, **20**
getBoat, **22**
getCap, **15**
getCrew, **20**
getId, **11**, **19**, **23**
getInterCrew, **20**
getRemainingCap, **19**
GLOBAL, 6

init, **18**
Intend, **6**, *6*, *8*, *12*
IO, *16*
isDedicated, **20**
isFinished, **7**, 9, **10**, 15, **18**, 18
isRemovable, **8**
isRoomForStay, **15**
isRoomForTransit, **15**

leave, **12**
Lifeboat, *9*, *11*, 18, **19**, *19*, *21*, *23*
Lifeboats, 8, 9, 11–13, 15, 26
LifeboatTest, 23
Linline, **6**, *6*, *16*

min, **15**
MoveLine, **6**, *6*
movePerson, **8**

ORAMS, 21, **22**, *22*
outline, **6**, *16, 17*

Person, *9*, *21*, 22, **22**, *22, 24*
persons, 8, 9, 13
PersonTest, 24
Pid, **6**, *6*, *8–10*, *12–14*, *19–21*, **22**, *22, 23*
Pinline, **6**, *6, 16*

redistribute, **9**
removeArea, **19**
removeBoat, **11**
RemoveBoatLine, **6**, *6*

removeEquipment, **11**
removeLifeboat, **8**
removePerson, **19**
Run, **6**
RunTest, **23, 24**, **26, 27**

SafetyEquipment, 23
sendEvents, **17**, 18
setController, **17**
SetUp, **23–25**, **27**
showResult, **17**
step, **7**, 9, **10**, 15
String, **6**, *6*, *16*

TearDown, **24, 25**, **27**
Time, *6*, **7**, *10*, *17*
tPerson, **10**, *10*, *13*
TransError, **6**, *6*

updateCap, **15**

World, 6, **6**, *6*, *27*

```
class
$World$ is subclass of $GLOBAL$
instance variables
        public static $env : [Environment] := $ nil ;

operations
public
        $World : String \xrightarrow{o} World$
        $World\ (scenario) \triangleq$
          (   $env := $ new $Environment\ (scenario);$
              $env.setController(ORAMS`ctl)$
          );
public
        $Run : () \xrightarrow{o} ()$
        $Run\ () \triangleq$
          (   start(env) ;
              start(ORAMS`ctl) ;
              start(ORAMS`a1) ;
              start(ORAMS`a2) ;
              start(ORAMS`a3) ;
              $env.isFinished()$ ;
              ORAMS`ctl.$isFinished()$ ;
              ORAMS`a1.$isFinished()$ ;
              ORAMS`a2.$isFinished()$ ;
              ORAMS`a3.$isFinished()$ ;
              $env.showResult()$
          )
end
$World$
class
$GLOBAL$
types
        public $Bid = $ token;
        public $Aid = $ token;
        public $Pid = $ token;
        public $TransError = Time \times String \times Aid \times Pid \times \mathbb{N} \times$ char;
        public $MoveLine = Time \times String \times Aid \times Pid \times Intend \times \mathbb{B} \times$ char;
        public $RemoveBoatLine = Time \times String \times Bid \times \mathbb{B} \times$ char;
        public $outline = TransError \mid MoveLine \mid RemoveBoatLine;$
        public $eventType = \textsc{removeBoat} \mid \textsc{movePerson};$
        public $Linline = eventType \times Bid \times [\mathbb{N}] \times [\mathbb{N}] \times \mathbb{N};$
        public $Pinline = eventType \times Pid \times Aid \times Intend \times \mathbb{N};$
        public $event = Linline \mid Pinline;$
        public $Intend = \textsc{Stay} \mid \textsc{Transit};$
        public $String = $ char$^*$;
        public  $Time = \mathbb{N}$
end
$GLOBAL$
class
$AccessController$ is subclass of $GLOBAL$
instance variables
        $eventQueue : event^* := []$;
        $busy : \mathbb{B} :=$ true;

operations
public
        $AccessController : () \xrightarrow{o} AccessController$
        $AccessController\,() \triangleq$
          return ;
public
        $addEvent : event \xrightarrow{o} ()$
        $addEvent\,(evt) \triangleq$
          $eventQueue := eventQueue \curvearrowright [evt]$;
public
        $isFinished : () \xrightarrow{o} ()$
        $isFinished\,() \triangleq$
          skip;
public
        $step : () \xrightarrow{o} ()$
        $step\,() \triangleq$ duration $(10)$
          (    if len $eventQueue > 0$
               then while len $eventQueue > 0$
                    do (    def mk- $(type, id1, id2, intend, \text{-}) =$ hd $eventQueue$ in
                            (     cases  $type$:
                                     $\textsc{removeBoat} \rightarrow AttemptRemoveLifeboat(id1)$ ,
                                     $\textsc{movePerson} \rightarrow movePerson(id1, id2, intend)$
                                  end;
                                  $eventQueue :=$ tl $eventQueue$
                            )
                       )
          );
public
```

```
        $movePerson : Pid \times Aid \times Intend \xrightarrow{o} ()$
        $movePerson\,(person, area, intend) \triangleq$
          (   let $success = ORAMS`areas\,(area).enterPerson\,(person, intend)$ in
              World`env.$addOutline$(mk_ (time,
                          "$Person\ \ move$ ",
                          $person$,
                          $area$,
                          $intend$,
                          $success$,
                          '$\backslash n$'))
          )
        pre  $person \in$ dom $ORAMS`persons$ ;
public
        $AttemptRemoveLifeboat : Bid \xrightarrow{o} ()$
        $AttemptRemoveLifeboat\,(boat) \triangleq$
          (   if $isRemovable\,(boat)$
              then (    $removeLifeboat(boat)$ ;
                        World`env.$addOutline$(mk_ (time,
                                   "$Lifeboat\ \ disable$ ",
                                   $boat$,
                                   true,
                                   '$\backslash n$'))
                   )
              else World`env.$addOutline$(mk_ (time,
                               "$Lifeboat\ \ disable$ ",
                               $boat$,
                               false,
                               '$\backslash n$'))
          );
public
        $removeLifeboat : Bid \xrightarrow{o} ()$
        $removeLifeboat\,(boat) \triangleq$
          (   $ORAMS`Lifeboats$     (boat )  .$disable()$ ;
              $redistribute(boat)$ ;
              for all $a \in$  $ORAMS`Lifeboats\,(boat).getAreas\,()$
              do $ORAMS`areas$     (a )  .$removeBoat(boat)$
          )
        pre  $isRemovable\,(boat) \wedge boat \in$ dom $ORAMS`Lifeboats$ ;
public
        $isRemovable : Bid \xrightarrow{o} \mathbb{B}$
        $isRemovable\,(boat) \triangleq$
          (   let $crew = ORAMS`Lifeboats\,(boat).getCrew\,()$,
                  $areas = ORAMS`Lifeboats\,(boat).getAreas\,()$ in
              (   if $\exists\, p \in crew \cdot ORAMS`persons\,(p).getArea\,() \notin areas$
                  then return false
```

```
                else (   for all $a \in areas$
                         do if $\neg$ $ORAMS`areas(a).canRemoveBoat(boat)$
                            then return false
                       ) ;
                return true
            )
        )
        pre $boat \in$ dom $ORAMS`Lifeboats$ ;
private
        $redistribute : Bid \xrightarrow{o} ()$
        $redistribute(boat) \triangleq$
          (   let $crew = (ORAMS`Lifeboats(boat)).getCrew()$ in
              for all $p \in crew$
              do $ORAMS`areas(ORAMS`persons(p).getArea()).changeBoat(p)$
          )
        pre $boat \in$ dom $ORAMS`Lifeboats$
sync
        mutex($step$);
        mutex($step, addEvent$);
        per $isFinished \Rightarrow$ len $eventQueue = 0$
thread
        periodic (250,1,3,0)(step)
        end $AccessController$
```

Test Suite : vdm.tc
Class : AccessController

Name	**#Calls**	**Coverage**
AccessController‘step	81	0%
AccessController‘addEvent	24	√
AccessController‘isFinished	2	√
AccessController‘movePerson	23	73%
AccessController‘isRemovable	5	82%
AccessController‘redistribute	2	76%
AccessController‘removeLifeboat	2	69%
AccessController‘AccessController	2	√
AccessController‘AttemptRemoveLifeboat	3	√
Total Coverage		**81%**

```
class
$AreaController$ is subclass of $GLOBAL$
instance variables
        $id : Aid$;
        $boats : Bid \xrightarrow{m} Lifeboat := \{\mapsto\}$;
        $equip : FireEquipment$-set $:= \{\}$;
        $pStay : Pid \xrightarrow{m} Person := \{\mapsto\}$;
        $pTransit : tPerson$-set $:= \{\}$;
        $stayCap : \mathbb{N} := 0$;
        $transCap : \mathbb{N} := 0$;
        $busy : \mathbb{B} :=$ true;
        inv card dom $pStay \leq stayCap$
        inv card $pTransit \leq transCap$

types
        $tPerson :: person : Pid$
                    $outTime : \mathbb{N}$
                    $late : \mathbb{B}$
operations
public
        $AreaController : Aid \xrightarrow{o} AreaController$
        $AreaController\ (aid) \triangleq$
          (    $id := aid$;
               $transCap := 2$
          );
public
        $step : () \xrightarrow{o} ()$
        $step\ () \triangleq$ duration (1)
          (    dcl $curtime : Time :=$ time;
               $boatMaintenance()$ ;
               for all $p \in pTransit$
               do if $p.outTime \leq curtime \land \neg\, p.late$
                  then (    dcl $ap : tPerson := \mu\ (p, late \mapsto$ true);
                            $pTransit := pTransit \setminus \{p\}$;
                            $pTransit := pTransit \cup \{ap\}$;
                            World‘env.$addOutline($mk_
                                      (
                                       $curtime$,
                                       "Error person stayed too long ",
                                       $id$,
                                       $p.person$,
                                       $p.outTime$,
                                       '\n'))
                       )
          );
public
        $isFinished : () \xrightarrow{o} ()$
        $isFinished\ () \triangleq$
          $busy :=$ false;
public
```

```
        $getId : () \xrightarrow{o} Aid$
        $getId\ () \triangleq$
          return $id$;
public
        $addEquipment : FireEquipment \xrightarrow{o} ()$
        $addEquipment\ (fe) \triangleq$
          $equip := \{fe\} \cup equip$;
public
        $removeEquipment : FireEquipment \xrightarrow{o} ()$
        $removeEquipment\ (fe) \triangleq$
          $equip := equip \setminus \{fe\}$
        pre  $stayCap -$ card dom $pStay > 0$ ;
public
        $canRemoveEquipment : () \xrightarrow{o} \mathbb{B}$
        $canRemoveEquipment\ () \triangleq$
          (    $updateCap()$ ;
               if card $equip > stayCap$
               then return true
               else return $(stayCap - 1 -$ card dom $pStay) \geq 0$
          );
public
        $addBoat : Lifeboat \xrightarrow{o} ()$
        $addBoat\ (boat) \triangleq$
          (    $boats := boats \uplus_m \{boat.getId\ () \mapsto boat\}$;
               $boat.addArea(id)$
          )
        pre  $boat.getId\ () \notin$ dom $boats$ ;
public
        $canRemoveBoat : Bid \xrightarrow{o} \mathbb{B}$
        $canRemoveBoat\ (boat) \triangleq$
          (    return card $(ORAMS`Lifeboats\ (boat).getCrew\ () \cap$ dom $pStay) \leq$
                      $getCap\ ($dom $(\{boat\} \lessdot boats))$
          );
public
        $removeBoat : Bid \xrightarrow{o} ()$
        $removeBoat\ (bid) \triangleq$
          (    $boats\ \ \ (bid\ )\ \ .disable()$ ;
               let $ps = boats\ (bid).getInterCrew\ ($dom $pStay)$ in
               for all $p \in\ ps$
               do $changeBoat(p)$ ;
               $boats\ \ \ (bid\ )\ \ .removeArea(id)$ ;
               $boats\ \ \ (bid\ )\ \ .enable()$ ;
               $boats := \{bid\} \lessdot boats$
          )
        pre  $id \in$ dom $boats \wedge canRemoveBoat\ (bid)$
        post $id \notin \text{dom } boats$ ;
public
        $leave : Pid \xrightarrow{o} ()$
        $leave\,(person) \triangleq$
          if $(\exists\, p \in \text{dom } pStay \cdot p = person)$
          then (   $pStay := \{person\} \lhd pStay$
               )
          elseif $(\exists\, p \in pTransit \cdot p.person = person)$
          then (   let  $pT \in pTransit$ be st  $pT.person = person$ in
                   $pTransit := pTransit \setminus \{pT\}$
               )
        pre  $person \in \text{dom } pStay \lor \exists\, p \in pTransit \cdot p.person = person$ ;
public
        $changeBoat : Pid \xrightarrow{o} ()$
        $changeBoat\,(p) \triangleq$
          (   let $b = pStay\,(p).getBoat\,()$ in
              (   let $newBoat = findBoat\,(b)$ in
                  (   $ORAMS`Lifeboats$    (b )  $.removePerson(p)$ ;
                      $boats$    (newBoat )  $.addPerson(p)$ ;
                      $pStay$    (p )  $.changeBoat(newBoat)$
                  )
              )
          );
public
        $enterPerson : Pid \times Intend \xrightarrow{o} \mathbb{B}$
        $enterPerson\,(person, intend) \triangleq$
          (   $updateCap()$ ;
               cases  $intend$:
                 STAY $\rightarrow$ return $doAddStaying\,(person)$,
                 TRANSIT $\rightarrow$ return $doAddTransit\,(person)$
              end
          )
        pre  $person \notin \text{dom } pStay \lor$
             $\exists\, p \in pTransit \cdot p.person = person$ ;
private
```

```
        $doAddStaying : Pid \xrightarrow{o} \mathbb{B}$
        $doAddStaying\,(person) \triangleq$
          if $isRoomForStay\,()$
          then (    let $boat = findBoat\,()$ in
                    (    $boats$    (boat )  $.addPerson(person)$ ;
                         $pStay := \{person \mapsto ORAMS`persons\,(person)\} \uplus pStay$;
                         if $pStay\,(person).getArea\,() \neq$ nil
                         then $ORAMS`areas$      $(ORAMS`persons\,(person).getArea$
( )   )  $.leave(person)$ ;
                         if $pStay\,(person).getBoat\,() \neq$ nil
                         then $ORAMS`Lifeboats$    $(ORAMS`persons\,(person).getBoat$
( )   )  $.removePerson(person)$ ;
                         $pStay$     (person )  $.changeArea((ORAMS`areas^{-1})$ (self)) ;
                         $pStay$     (person )  $.changeBoat(boat)$ ;
                         return true
                    )
               )
          else return false ;
private
        $doAddTransit : Pid \xrightarrow{o} \mathbb{B}$
        $doAddTransit\,(person) \triangleq$
          if $isRoomForTransit\,()$
          then (    dcl $p : tPerson :=$ mk_$tPerson\,(person,$
                                             time $+ 250,$
                                             false);
                    $pTransit := \{p\} \cup pTransit$;
                    if $ORAMS`persons\,(person).getArea\,() \neq$ nil
                    then $ORAMS`areas$     $(ORAMS`persons\,(person).getArea$     ( )
)  $.leave(person)$ ;
                    $ORAMS`persons$    (person )  $.changeArea((ORAMS`areas^{-1})$ (self));
                    return true
               )
          else return false;
private
        $findBoat : Bid \xrightarrow{o} Bid$
        $findBoat\,(b) \triangleq$
          (    let $bids = \{b\} \lessdot boats$ in
               (    let $bs = \{boat \mid boat \in$ dom $bids \cdot$
                                   $bids\,(boat).getRemainingCap\,() > 0\}$ in
                    (    if $\exists\, boat \in bs \cdot boats\,(boat).isDedicated\,()$
                         then (     let   $boat \in bs$ be st   $boats\,(boat).isDedicated\,()$ in
                                   return $boat$
                              )
```

```
                  else (   let $boat \in bs$ in
                           return $boat$
                        )
                 )
             )
         );
private
    $findBoat : () \xrightarrow{o} Bid$
    $findBoat\,() \triangleq$
      (   let $bs = \{b \mid b \in$ dom $boats \cdot boats\,(b).getRemainingCap\,() > 0\}$ in
          (   if $\exists\, b \in bs \cdot boats\,(b).isDedicated\,()$
              then (   let  $b \in bs$ be st  $boats\,(b).isDedicated\,()$ in
                       return $b$
                    )
              else (   let $b \in bs$ in
                       return $b$
                    )
          )
      );
private
    $boatMaintenance : () \xrightarrow{o} ()$
    $boatMaintenance\,() \triangleq$
      (   dcl $remCap : \mathbb{N} := 0;$
          let $bs = \{b \mid b \in$ rng $boats \cdot b.isDedicated\,()\}$ in
          (   for all $b \in\ bs$
              do $remCap := remCap + b.getRemainingCap\,();$
              if $remCap > 0$
              then (   let $sharedBoats =$ rng $boats \setminus bs$ in
                       (   dcl $crew : Pid$-set $:= \bigcup\{cs.getInterCrew\,($dom $pStay) \mid$
$cs \in sharedBoats\};$
                           while $remCap > 0 \wedge crew \neq \{\}$
                           do (   let $p \in crew$ in
                                  (   $changeBoat(p)$ ;
                                      $crew := crew \setminus \{p\}$
                                  ) ;
                                  $remCap := remCap - 1$
                              )
                       )
                    )
          )
      );
private
```

```
        $isRoomForStay : () \xrightarrow{o} \mathbb{B}$
        $isRoomForStay\ () \triangleq$
          (   $updateCap()$ ;
              return $stayCap -$ card dom $pStay > 0$
          );
private
        $isRoomForTransit : () \xrightarrow{o} \mathbb{B}$
        $isRoomForTransit\ () \triangleq$
          return $transCap -$ card $pTransit > 0$;
private
        $updateCap : () \xrightarrow{o} ()$
        $updateCap\ () \triangleq$
          (   dcl $bCap : \mathbb{N} := 0$;
              for all $b \in$ dom $boats$
              do $bCap := bCap + ORAMS`Lifeboats\ (b).getRemainingCap\ ()$;
              $stayCap := min\ (bCap +$ card dom $pStay,$ card $equip)$
          );
private
        $getCap : Bid$-set $\xrightarrow{o} \mathbb{N}$
        $getCap\ (bs) \triangleq$
          (   dcl $bCap : \mathbb{N} := 0$;
              for all $b \in\ bs$
              do $bCap := bCap + boats\ (b).getRemainingCap\ ()$;
              return $bCap$
          )
functions
private
        $min : \mathbb{N} \times \mathbb{N} \tilde{\rightarrow} \mathbb{N}$
        $min\ (n1, n2) \triangleq$
          if $n1 < n2$
          then $n1$
          else $n2$
sync
        per $isFinished \Rightarrow$ #active $(step) = 0$;
        mutex$(leave, step)$;
        mutex$(enterPerson, step)$ thread
        periodic (250,1,3,0)(step)
        end $AreaController$
```

Test Suite : vdm.tc
Class : AreaController

Name	#Calls	Coverage
AreaController‘min	58	√
AreaController‘step	240	0%
AreaController‘getId	14	√

Name	**#Calls**	**Coverage**
AreaController‘leave	15	75%
AreaController‘getCap	11	$\surd$
AreaController‘addBoat	18	66%
AreaController‘updateCap	58	$\surd$
AreaController‘changeBoat	5	$\surd$
AreaController‘isFinished	6	$\surd$
AreaController‘removeBoat	7	61%
AreaController‘enterPerson	28	56%
AreaController‘findBoat	22	$\surd$
AreaController‘addEquipment	34	$\surd$
AreaController‘doAddStaying	25	$\surd$
AreaController‘doAddTransit	3	95%
AreaController‘canRemoveBoat	11	$\surd$
AreaController‘isRoomForStay	25	$\surd$
AreaController‘AreaController	6	$\surd$
AreaController‘boatMaintenance	240	$\surd$
AreaController‘removeEquipment	4	41%
AreaController‘findBoat	5	$\surd$
AreaController‘isRoomForTransit	3	$\surd$
AreaController‘canRemoveEquipment	5	$\surd$
Total Coverage		**89%**

```
class
$Environment$ is subclass of $GLOBAL$
instance variables
        $ctl : [AccessController] :=$ nil ;
        $Linlines : Linline^* := []$;
        $Pinlines : Pinline^* := []$;
        $events : event^* := []$;
        $busy : \mathbb{B} :=$ true;
        $running : \mathbb{B} :=$ true;
        $outlines : outline^* := []$;
        $io : IO :=$ new $IO\,()$;

operations
public
        $Environment : String \xrightarrow{o} Environment$
        $Environment\,(sfname) \triangleq$
          (   def mk- (-, $input1) = io.freadval[event^*]\,(sfname)$ in
                $events := input1$;
                $init()$
          );
public
```

```
        $setController : AccessController \xrightarrow{o} ()$
        $setController\,(actl) \triangleq$
          $ctl := actl;$
public
        $sendEvents : () \xrightarrow{o} ()$
        $sendEvents\,() \triangleq$
          (    if len $events > 0$
               then (    dcl $curtime : Time :=$ time,
                             $done : \mathbb{B} :=$ false;
                         while $\neg\, done$
                         do def mk- $(type, id1, id2, intend, pt) =$ hd $events$ in
                            if $pt \leq curtime$
                            then (    $ctl.addEvent($mk_$(type, id1, id2, intend, pt))$ ;
                                      $events :=$ tl $events;$
                                      $done :=$ len $events = 0$
                                 )
                            else $done :=$ true
                    )
               else (    $running :=$ false
                    )
          );
public
        $addOutline : outline \xrightarrow{o} ()$
        $addOutline\,(line) \triangleq$
          (    $outlines := outlines \curvearrowright [line]$
          );
public
        $showResult : () \xrightarrow{o} ()$
        $showResult\,() \triangleq$
          def - $= io.writeval[outline^*]\,(outlines)$ in
          skip;
private
```

```
    $init : () \xrightarrow{o} ()$
    $init\ () \triangleq$
      (   ORAMS`a1.$addBoat(ORAMS`l1)$ ;
          ORAMS`a1.$addEquipment(ORAMS`f1)$ ;
          ORAMS`a1.$addEquipment(ORAMS`f2)$ ;
          ORAMS`a1.$addEquipment(ORAMS`f3)$ ;
          ORAMS`a2.$addBoat(ORAMS`l2)$ ;
          ORAMS`a2.$addBoat(ORAMS`l3)$ ;
          ORAMS`a2.$addEquipment(ORAMS`f4)$ ;
          ORAMS`a2.$addEquipment(ORAMS`f5)$ ;
          ORAMS`a2.$addEquipment(ORAMS`f6)$ ;
          ORAMS`a3.$addBoat(ORAMS`l3)$ ;
          ORAMS`a3.$addBoat(ORAMS`l4)$ ;
          ORAMS`a3.$addEquipment(ORAMS`f7)$ ;
          ORAMS`a3.$addEquipment(ORAMS`f8)$ ;
          ORAMS`a3.$addEquipment(ORAMS`f9)$ ;
          ORAMS`a3.$addEquipment(ORAMS`f10)$
      );
public
    $isFinished : () \xrightarrow{o} ()$
    $isFinished\ () \triangleq$
      skip
sync
    mutex($sendEvents$);
    per $isFinished \Rightarrow \neg running$
thread
    periodic (1000,1,3,0)(sendEvents)
    end $Environment$
```

Test Suite : vdm.tc
Class : Environment

Name	**#Calls**	**Coverage**
Environment`init	3	$\surd$
Environment`addOutline	27	$\surd$
Environment`isFinished	2	$\surd$
Environment`sendEvents	21	$\surd$
Environment`showResult	2	$\surd$
Environment`Environment	3	$\surd$
Environment`setController	3	$\surd$
Total Coverage		**100%**

```
class
$Lifeboat$ is subclass of $SafetyEquipment$
instance variables
    $id : Bid;$
```

```
        $acceptingCrew : \mathbb{B};$
        $crew : Pid\text{-set} := \{\};$
        $areas : Aid\text{-set} := \{\};$
        inv $\text{card } crew \leq capacity$

operations
public
        $Lifeboat : \mathbb{N} \times Bid \xrightarrow{o} Lifeboat$
        $Lifeboat\,(cap, bid) \triangleq$
          (    $acceptingCrew := \text{true};$
               $capacity := cap;$
               $id := bid$
          );
public
        $getId : () \xrightarrow{o} Bid$
        $getId\,() \triangleq$
          return $id;$
public
        $getRemainingCap : () \xrightarrow{o} \mathbb{N}$
        $getRemainingCap\,() \triangleq$
          if $acceptingCrew$
          then return $capacity - \text{card } crew$
          else return $0$ ;
public
        $addPerson : Pid \xrightarrow{o} ()$
        $addPerson\,(p) \triangleq$
          $crew := crew \cup \{p\}$
        pre  $capacity - \text{card } crew > 0 \land acceptingCrew$
        post $p \in crew$ ;
public
        $removePerson : Pid \xrightarrow{o} ()$
        $removePerson\,(p) \triangleq$
          $crew := crew \setminus \{p\}$
        pre  $p \in crew$
        post $p \notin crew$ ;
public
        $addArea : Aid \xrightarrow{o} ()$
        $addArea\,(a) \triangleq$
          $areas := areas \cup \{a\};$
public
        $removeArea : Aid \xrightarrow{o} ()$
        $removeArea\,(a) \triangleq$
          $areas := areas \setminus \{a\};$
public
```

$isDedicated : () \xrightarrow{o} \mathbb{B}$
$isDedicated\ () \triangleq$
  return card $areas = 1;$
public
$getAreas : () \xrightarrow{o} Aid$-set
$getAreas\ () \triangleq$
  return $areas;$
public
$disable : () \xrightarrow{o} ()$
$disable\ () \triangleq$
  $acceptingCrew :=$ false;
public
$enable : () \xrightarrow{o} ()$
$enable\ () \triangleq$
  $acceptingCrew :=$ true;
public
$getInterCrew : Pid$-set $\xrightarrow{o} Pid$-set
$getInterCrew\ (ps) \triangleq$
  return $crew \cap ps;$
public
$getCrew : () \xrightarrow{o} Pid$-set
$getCrew\ () \triangleq$
  return $crew$
sync
  mutex$(addPerson);$
  mutex$(addPerson, removePerson, getRemainingCap, getCrew, getInterCrew);$
  mutex$(addArea, removeArea, getAreas, isDedicated);$
  mutex$(disable, enable, getRemainingCap)$ end $Lifeboat$

Test Suite : vdm.tc
Class : Lifeboat

Name	**#Calls**	**Coverage**
Lifeboat‘getId	44	√
Lifeboat‘enable	8	√
Lifeboat‘addArea	20	√
Lifeboat‘disable	10	√
Lifeboat‘getCrew	20	√
Lifeboat‘Lifeboat	9	√
Lifeboat‘getAreas	9	√
Lifeboat‘addPerson	29	31%
Lifeboat‘removeArea	8	√
Lifeboat‘isDedicated	462	√
Lifeboat‘getInterCrew	94	√
Lifeboat‘removePerson	18	45%

Name	**#Calls**	**Coverage**
Lifeboat‘getRemainingCap	376	$\surd$
Total Coverage		**76%**

system
ORAMS
instance variables

$cpu1 : CPU :=$ new $CPU\,(\mathrm{FCFS}, 1000000)$;
$cpu2 : CPU :=$ new $CPU\,(\mathrm{FCFS}, 1000000)$;
$bus0 : BUS :=$ new $BUS\,(\mathrm{FCFS}, 1000000, \{cpu1, cpu2\})$;
public static $a1 : AreaController :=$ new $AreaController$ (mk_token ($\mathrm{A1}$));
public static $a2 : AreaController :=$ new $AreaController$ (mk_token ($\mathrm{A2}$));
public static $a3 : AreaController :=$ new $AreaController$ (mk_token ($\mathrm{A3}$));
public static $ctl : AccessController :=$ new $AccessController\,()$;
public static $l1 : Lifeboat :=$ new $Lifeboat$ (2, mk_token ($\mathrm{B1}$));
public static $l2 : Lifeboat :=$ new $Lifeboat$ (2, mk_token ($\mathrm{B2}$));
public static $l3 : Lifeboat :=$ new $Lifeboat$ (2, mk_token ($\mathrm{B3}$));
public static $l4 : Lifeboat :=$ new $Lifeboat$ (2, mk_token ($\mathrm{B4}$));
public static $p1 : Person :=$ new $Person$ (mk_token ($\mathrm{P1}$));
public static $p2 : Person :=$ new $Person$ (mk_token ($\mathrm{P2}$));
public static $p3 : Person :=$ new $Person$ (mk_token ($\mathrm{P3}$));
public static $p4 : Person :=$ new $Person$ (mk_token ($\mathrm{P4}$));
public static $p5 : Person :=$ new $Person$ (mk_token ($\mathrm{P5}$));
public static $f1 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f2 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f3 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f4 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f5 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f6 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f7 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f8 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f9 : FireEquipment :=$ new $FireEquipment\,()$;
public static $f10 : FireEquipment :=$ new $FireEquipment\,()$;
public static $persons : Pid \xleftrightarrow{m} Person := \{p1.getId\,() \mapsto p1,$
$p2.getId\,() \mapsto p2,$
$p3.getId\,() \mapsto p3,$
$p4.getId\,() \mapsto p4,$
$p5.getId\,() \mapsto p5\}$;
public static $Lifeboats : Bid \xleftrightarrow{m} Lifeboat := \{l1.getId\,() \mapsto l1,$
$l2.getId\,() \mapsto l2,$
$l3.getId\,() \mapsto l3,$
$l4.getId\,() \mapsto l4\}$;
public static $areas : Aid \xleftrightarrow{m} AreaController := \{a1.getId\,() \mapsto a1,$
$a2.getId\,() \mapsto a2,$
$a3.getId\,() \mapsto a3\}$;

```
types
          public  $Bid$ = token;
          public  $Aid$ = token;
          public  $Pid$ = token
operations
public
          $ORAMS : () \xrightarrow{o} ORAMS$
          $ORAMS\,() \triangleq$
             (    $cpu1.deploy(ctl)$ ;
                  $cpu1.deploy(a1)$ ;
                  $cpu1.deploy(a2)$ ;
                  $cpu1.deploy(a3)$ ;
                  $cpu2.deploy(l1)$ ;
                  $cpu2.deploy(l2)$ ;
                  $cpu2.deploy(l3)$ ;
                  $cpu2.deploy(l4)$
             )
end
$ORAMS$
class
$Person$ is subclass of $GLOBAL$
instance variables
          $area : [Aid] :=$ nil ;
          $boat : [Bid] :=$ nil ;
          $id : Pid$;

operations
public
          $Person : Pid \xrightarrow{o} Person$
          $Person\,(pid) \triangleq$
             $id := pid$;
public
          $getArea : () \xrightarrow{o} [Aid]$
          $getArea\,() \triangleq$
             return $area$;
public
          $getBoat : () \xrightarrow{o} [Bid]$
          $getBoat\,() \triangleq$
             return $boat$;
public
          $changeArea : Aid \xrightarrow{o} ()$
          $changeArea\,(a) \triangleq$
             $area := a$
          pre  $a \in$ dom $ORAMS`areas$ ;
```

public
    $changeBoat : Bid \xrightarrow{o} ()$
    $changeBoat\,(b) \triangleq$
        $boat := b$
    pre $b \in$ dom $ORAMS`Lifeboats$;
public
    $getId : () \xrightarrow{o} Pid$
    $getId\,() \triangleq$
        return id
end
Person

Test Suite : vdm.tc
Class : AreaController

Name	#Calls	Coverage
Total Coverage		1%

class
SafetyEquipment is subclass of *GLOBAL*
instance variables
    protected $capacity : \mathbb{N}$;

end
SafetyEquipment
class
LifeboatTest is subclass of *TestCase*
operations
protected
    $SetUp : () \xrightarrow{o} ()$
    $SetUp\,() \triangleq$
        skip;
protected
    $RunTest : () \xrightarrow{o} ()$
    $RunTest\,() \triangleq$
        (dcl $l1 : Lifeboat :=$ new $Lifeboat\,(2,$ mk_token $(\text{TESTBOAT}))$;

```
            AssertTrue(l1.getId () = mk_token (TESTBOAT)) ;
            AssertTrue(l1.getRemainingCap () = 2) ;
            l1.disable() ;
            AssertFalse(l1.getRemainingCap () = 2) ;
            l1.enable() ;
            l1.addArea(ORAMS‘a1.getId ()) ;
            AssertTrue(l1.isDedicated ()) ;
            l1.addArea(ORAMS‘a2.getId ()) ;
            AssertFalse(l1.isDedicated ()) ;
            AssertTrue(l1.getAreas () = {ORAMS‘a1.getId (), ORAMS‘a2.getId ()});
            l1.removeArea(ORAMS‘a1.getId ()) ;
            AssertTrue(l1.isDedicated ()) ;
            l1.addPerson(ORAMS‘p1.getId ()) ;
            l1.addPerson(ORAMS‘p2.getId ()) ;
            AssertTrue(l1.getCrew () = {ORAMS‘p1.getId (), ORAMS‘p2.getId ()});
            AssertTrue(l1.getInterCrew ({ORAMS‘p1.getId ()}) = {ORAMS‘p1.getId ()});
            AssertFalse(l1.getInterCrew ({ORAMS‘p3.getId ()}) = {ORAMS‘p1.getId ()});
            AssertTrue(l1.getRemainingCap () = 0) ;
            l1.removePerson(ORAMS‘p1.getId ()) ;
            AssertTrue(l1.getRemainingCap () = 1) ;
            AssertFalse(l1.getInterCrew ({ORAMS‘p1.getId ()}) = {ORAMS‘p1.getId ()})
        );
protected
    TearDown : () $\xrightarrow{o}$ ()
    TearDown () $\triangleq$
        skip
end
LifeboatTest
class
PersonTest is subclass of TestCase
operations
protected
    SetUp : () $\xrightarrow{o}$ ()
    SetUp () $\triangleq$
        skip;
protected
    RunTest : () $\xrightarrow{o}$ ()
    RunTest () $\triangleq$
        (   dcl p1 : Person := new Person (mk_token (TESTPERSON));
            $AssertTrue(p1.getId\,() = \mathsf{mk\_token}\,(\textsc{testperson}))\;;$
            $AssertTrue(p1.getArea\,() = \mathsf{nil}\;)\;;$
            $AssertTrue(p1.getBoat\,() = \mathsf{nil}\;)\;;$
            $p1.changeArea(ORAMS`a1.getId\,())\;;$
            $p1.changeBoat(ORAMS`l1.getId\,())\;;$
            $AssertTrue(p1.getArea\,() = ORAMS`a1.getId\,())\;;$
            $AssertTrue(p1.getBoat\,() = ORAMS`l1.getId\,())$
        );
protected
      $TearDown : () \xrightarrow{o} ()$
      $TearDown\,() \triangleq$
        skip
end
$PersonTest$
class
$AreaControllerTest$ is subclass of $TestCase$
operations
protected
      $SetUp : () \xrightarrow{o} ()$
      $SetUp\,() \triangleq$
        (   ORAMS`a1.$addEquipment(ORAMS`f1)\;;$
            ORAMS`a1.$addEquipment(ORAMS`f2)\;;$
            ORAMS`a1.$addEquipment(ORAMS`f3)$
        );
protected
```

```
RunTest : () $\xrightarrow{o}$ ()
RunTest () $\triangleq$
  (   AssertTrue(ORAMS`a1.getId () = mk_token (A1)) ;
      AssertTrue(ORAMS`p1.getArea () = nil ) ;
      AssertFalse(ORAMS`a1.enterPerson (mk_token (P1), STAY)) ;
      AssertTrue(ORAMS`p1.getArea () = nil ) ;
      ORAMS`a1.addBoat(ORAMS`l1) ;
      AssertTrue(ORAMS`a1.enterPerson (mk_token (P1), STAY)) ;
      AssertTrue(ORAMS`p1.getArea () = mk_token (A1)) ;
      AssertTrue(ORAMS`p1.getBoat () = ORAMS`l1.getId ()) ;
      AssertFalse(ORAMS`a1.canRemoveBoat (ORAMS`l1.getId ())) ;
      AssertTrue(ORAMS`a1.enterPerson (mk_token (P2), TRANSIT)) ;
      AssertTrue(ORAMS`p2.getArea () = mk_token (A1)) ;
      ORAMS`a2.addEquipment(ORAMS`f4) ;
      ORAMS`a2.addBoat(ORAMS`l3) ;
      AssertTrue(ORAMS`a2.enterPerson (mk_token (P2), STAY)) ;
      AssertFalse(ORAMS`p2.getArea () = mk_token (A1)) ;
      ORAMS`a1.addBoat(ORAMS`l3) ;
      ORAMS`a1.changeBoat(mk_token (P1)) ;
      AssertTrue(ORAMS`p1.getBoat () = ORAMS`l3.getId ()) ;
      AssertTrue(ORAMS`Lifeboats (ORAMS`l1.getId ()).getCrew ()∩{mk_token (P1)} =
{}) ;
      AssertTrue(ORAMS`a1.canRemoveBoat (ORAMS`l1.getId ())) ;
      ORAMS`a1.removeBoat(ORAMS`l1.getId ()) ;
      AssertTrue(ORAMS`l1.getAreas () ∩ {mk_token (A1)} = {}) ;
      AssertTrue(ORAMS`a1.canRemoveEquipment ()) ;
      ORAMS`a1.removeEquipment(ORAMS`f3) ;
      AssertTrue(ORAMS`a1.canRemoveEquipment ()) ;
      ORAMS`a1.removeEquipment(ORAMS`f2) ;
      AssertFalse(ORAMS`a1.canRemoveEquipment ()) ;
      ORAMS`a2.leave(mk_token (P2)) ;
      AssertTrue(ORAMS`a2.enterPerson (mk_token (P1), TRANSIT)) ;
      ORAMS`a2.leave(mk_token (P1)) ;
      AssertTrue(ORAMS`a2.canRemoveBoat (ORAMS`l3.getId ())) ;
      ORAMS`a2.removeBoat(ORAMS`l3.getId ()) ;
      AssertTrue(ORAMS`a1.canRemoveBoat (ORAMS`l3.getId ())) ;
      ORAMS`a1.removeBoat(ORAMS`l3.getId ()) ;
      AssertTrue(ORAMS`a1.canRemoveEquipment ()) ;
      ORAMS`a1.removeEquipment(ORAMS`f1) ;
      AssertTrue(ORAMS`a2.canRemoveEquipment ()) ;
      ORAMS`a1.removeEquipment(ORAMS`f4) ;
      AssertTrue(ORAMS`p1.getBoat () = mk_token (B3)) ;
      AssertTrue(ORAMS`p2.getBoat () = mk_token (B3)) ;
      ORAMS`l3.removePerson(mk_token (P1)) ;
      ORAMS`l3.removePerson(mk_token (P2))
  );
```

```
protected
        $TearDown : () \xrightarrow{o} ()$
        $TearDown\ () \triangleq$
          (    skip
          )
end
$AreaControllerTest$
class
$AccessControllerTest$ is subclass of $TestCase$
instance variables
        public static $w : [World] :=$ nil ;

operations
protected
        $SetUp : () \xrightarrow{o} ()$
        $SetUp\ () \triangleq$
          (    $w :=$ new $World$ ("$ballancing.txt$")
          );
protected
        $RunTest : () \xrightarrow{o} ()$
        $RunTest\ () \triangleq$
          (    ORAMS‘ctl.$movePerson$(mk_token (P1), mk_token (A1), STAY) ;
               $AssertTrue$($ORAMS$‘$p1.getArea$ () = mk_token (A1)) ;
               $AssertTrue$($ORAMS$‘$p1.getBoat$ () = $ORAMS$‘$l1.getId$ ()) ;
               $AssertFalse$($ORAMS$‘$ctl.isRemovable$ ($ORAMS$‘$l1.getId$ ())) ;
               ORAMS‘ctl.$movePerson$(mk_token (P1), mk_token (A2), STAY) ;
               $AssertTrue$($ORAMS$‘$p1.getArea$ () = mk_token (A2)) ;
               $AssertTrue$($ORAMS$‘$p1.getBoat$ () = $ORAMS$‘$l2.getId$ ()) ;
               $AssertTrue$($ORAMS$‘$ctl.isRemovable$ ($ORAMS$‘$l3.getId$ ())) ;
               ORAMS‘ctl.$removeLifeboat$($ORAMS$‘$l3.getId$ ()) ;
               $AssertFalse$($ORAMS$‘$l3.isDedicated$ ())
          );
protected
        $TearDown : () \xrightarrow{o} ()$
        $TearDown\ () \triangleq$
          skip
end
$AccessControllerTest$
```