

TrayAllocatorRealtimeModel.vdmrt

```
-- =====
-- APPENDIX C - Realtime VDM++ Model
-- Amalgamated - (in one file for printing)
-- Deployment scenario 1 - each IC on each CPU
-- =====

-- =====
-- World in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerger - spring 2010
-- =====

class World
  types

  values

  instance variables

    public static env : [SorterEnviroment] := nil;

    loader : [ItemLoader] := nil;

    -- Test files that contains test scenarios
    ---      of items to be feeded on inductions
    testfile1 : seq1 of char := "scenario1.txt";

    testfiles : seq of seq1 of char := [testfile1];

  operations

-- World constructor
public World: () ==> World
World() ==
(
);

-- Prints configuration and result of tray allocation model simulation
public PrintSimulationResult: () ==> ()
PrintSimulationResult() ==
(
  -- Prints configuration of simulation
  IO`print("-----\n");
  IO`print("Simulation completed for sorter configuration\n");
  IO`print("-----\n");
  IO`print("Specified throughput [items/hour]: "
    ^ String`NatToStr(SorterEnviroment`Throughput) ^ "\n");
  IO`print("Sorter speed [mm/sec]: "
    ^ String`NatToStr(SorterEnviroment`Speed) ^ "\n");
  IO`print("Item max size [mm]: "
    ^ String`NatToStr(Item`MaxSize) ^ "\n");
  IO`print("Item min size [mm]: "
    ^ String`NatToStr(Item`MinSize) ^ "\n");
  IO`print("Tray size [mm]: "
    ^ String`NatToStr(Tray`Size) ^ "\n");
  IO`print("Number of trays : "
    ^ String`NatToStr(TrayAllocator`NumOfTrays) ^ "\n");
  IO`print("Number of inductions : "
    ^ String`NatToStr(TrayAllocator`NumOfInductions) ^ "\n");
  IO`print("Induction rate : "
    ^ String`NatToStr(InductionController`InductionRate) ^ "\n");
  IO`print("Induction separation [trays]: "
    ^ String`NatToStr(TrayAllocator`InductionSeperation) ^ "\n");
  IO`print("-----\n");
);
```

TrayAllocatorRealtimeModel.vdmrt

```

-- Prints result of simulation
let r : TrayAllocator`ThroughputResult = SC`allocator.GetThroughput()
in
(
  IO`print("Number of trays with items      : "
    ^ String`NatToStr(r.traysWithItemOnSorter) ^ "\n");
  IO`print("Two tray items on sorter      : "
    ^ String`NatToStr(r.twoTrayItemsOnSorter) ^ "\n");
  IO`print("Number of tray steps          : "
    ^ String`NatToStr(r.traySteps) ^ "\n");
  IO`print("Number of inducted items      : "
    ^ String`NatToStr(r.inductedItems) ^ "\n");
  IO`print("Calculated throughput[items/hour]: "
    ^ String`NatToStr(floor(r.calcThroughput)) ^ "\n");
);

IO`print("-----\n");
if SC`allocator.IsSorterFull() = true
then
  IO`print("      ****      Sorter is full      *****\n")
else
  IO`print("      ****      Sorter is not full    *****\n");
IO`print("-----\n");
);

public Run: () ==> ()
Run() ==
(
  -- Performs model testing for each scenarios specified in test file
  for all test in set {1,...,len testfiles}
  do
  (
    env := new SorterEnviroment();
    loader := new ItemLoader(testfiles(test));
    env.AssignItemLoader(loader);

    start(env); -- Start thread in enviroment

    IO`print("-----\n");
    IO`print("Tray allocation RTD1 model #" ^ String`NatToStr(test)
      ^ ": " ^ testfiles(test) ^ "\n");
    IO`print("-----\n");

    env.isFinished();
    PrintSimulationResult();
  );
);

functions

sync

--thread

traces

end World

-- =====
-- SC - SorterController in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

system SC

```

instance variables

```
-- cpu to deploy induction controller 1
cpuIC1 : CPU := new CPU (<FCFS>,1E6);

-- cpu to deploy induction controller 2
cpuIC2 : CPU := new CPU (<FCFS>,1E6);

-- cpu to deploy induction controller 3
cpuIC3 : CPU := new CPU (<FCFS>,1E6);

-- cpu to deploy induction controller 4
cpuIC4 : CPU := new CPU (<FCFS>,1E6);

-- cpu to deploy tray allocator 4
cpuTA4 : CPU := new CPU (<FCFS>,1E6);

-- bus to connect induction controller 1 to the tray allocator
bus1 : BUS := new BUS (<FCFS>,1E3,{cpuIC1,cpuTA4});

-- bus to connect induction controller 2 to the tray allocator
bus2 : BUS := new BUS (<FCFS>,1E3,{cpuIC2,cpuTA4});

-- bus to connect induction controller 3 to the tray allocator
bus3 : BUS := new BUS (<FCFS>,1E3,{cpuIC3,cpuTA4});

-- bus to connect induction controller 4 to the tray allocator
bus4 : BUS := new BUS (<FCFS>,1E3,{cpuIC4,cpuTA4});

public static ic1 : InductionController := new InductionController(1);
public static ic2 : InductionController := new InductionController(2);
public static ic3 : InductionController := new InductionController(3);
public static ic4 : InductionController := new InductionController(4);
public static inductionGroup : seq of InductionController
                                := [ic1, ic2, ic3, ic4];
public static allocator : TrayAllocator := new TrayAllocator();
```

operations

```
-- SystemController constructor
```

```
public SC: () ==> SC
```

```
SC() ==
```

```
(
```

```
-- set-up ic1
cpuIC1.deploy(ic1);
```

```
-- set-up ic2
cpuIC2.deploy(ic2);
```

```
-- set-up ic3
cpuIC3.deploy(ic3);
```

```
-- set-up ic4
cpuIC4.deploy(ic4);
```

```
-- set-up tray allocator
cpuTA4.deploy(allocator);
```

```
);
```

```
end SC
```

```
-- =====
-- SorterEnvironment in tray allocation for a sortation system
```

-- By José Antonio Esparza and Kim Bjerge - spring 2010

-- =====

class SorterEnviroment

types

values

```

public Speed      : nat1 = 2000;  -- Sorter speed mm/sec
public Throughput : nat  = 10000;  -- Required items/hour
public StepMs      : nat  = 100;
public TUinMS      : nat  = (StepMs * TrayAllocator`TrayStepTimeUnits)
                          / ((Tray`Size * 1000) / Speed);

```

instance variables

```

public inductionGroup : seq of InductionController := [];
itemId : nat := 0;
itemLoader : [ItemLoader] := nil;
busy : bool := true;
msCount : nat := 0;
nextMs : nat := 0;

```

operations

-- SorterEnviroment constructor

```

public SorterEnviroment: () ==> SorterEnviroment
SorterEnviroment() ==
(
);

```

-- Assigning item loader to SorterEnviroment

```

public AssignItemLoader: (ItemLoader) ==> ()
AssignItemLoader(il) ==
(
    itemLoader := il;
);

```

-- Assigning induction group to SorterEnviroment

```

public AssignInductionGroup: seq of InductionController ==> ()
AssignInductionGroup(ig) ==
(
    inductionGroup := ig;
);

```

public isFinished : () ==> ()

isFinished() == **skip**;

-- Create assignments relations between objects

```

CreateAssignments: () ==> ()
CreateAssignments () ==
(
    -- SC`allocator.CreateAllocatorObjs();
    SC`ic1.AssignAllocator(SC`allocator);
    SC`ic2.AssignAllocator(SC`allocator);
    SC`ic3.AssignAllocator(SC`allocator);
    SC`ic4.AssignAllocator(SC`allocator);
    AssignInductionGroup(SC`inductionGroup);
);

```

functions

sync

per isFinished => **not** busy;

```

thread
(
  CreateAssignments();

  -- Start all threads in the system
  start (SC`allocator);
  for all i in set {1,...,TrayAllocator`NumOfInductions}
  do start (inductionGroup(i));

  while busy do
  (
    -- IO`print("< " ^ String`NatToStr(time) ^ ">>>");
    if (time > nextMs)
    then
    (
      nextMs := time + TUinMS;
      for all i in set {1,...,TrayAllocator`NumOfInductions}
      do
      (
        -- Check for item to feed induction at time step
        let size = itemLoader.GetItemAtTimeStep(msCount, i)
        in
        if (size > 0)
        then
        (
          itemId := itemId + 1;
          IO`print("[ " ^ String`NatToStr(msCount)
            ^ ", " ^ String`NatToStr(itemId)
            ^ ", " ^ String`NatToStr(size)
            ^ ", " ^ String`NatToStr(time) ^ "]\n");
          inductionGroup(i).FeedItem(size, itemId);
        );
      );
      msCount := msCount + StepMs;
    );

    -- Check if simulation is finish
    if (time >= itemLoader.GetNumTimeSteps()*(TUinMS/StepMs)) then
    (
      SC`allocator.StopSimulation();
      busy := false;
    );

  );
);

traces

end SorterEnviroment

-- =====
-- ItemLoader in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class ItemLoader
  types

  inline = nat * nat * nat;
  InputTP = int * seq of inline;

```

values**instance variables**

```

io : IO := new IO();

-- Test with mix of 1 and 2 tray items
inlines : seq of inline := [mk_(0,1,100),
                             mk_(0,2,800),
                             mk_(0,3,200),
                             mk_(200,1,200),
                             mk_(200,2,400),
                             mk_(200,3,700),
                             mk_(400,1,800),
                             mk_(400,2,300),
                             mk_(400,3,400),
                             mk_(600,1,600),
                             mk_(600,2,400),
                             mk_(600,3,300),
                             mk_(800,1,900),
                             mk_(800,2,300),
                             mk_(800,3,200),
                             mk_(1000,1,500),
                             mk_(1000,2,300),
                             mk_(1000,3,200)
                           ];

numTimeSteps : nat := 2100;

```

operations

```

-- Loads test scenario from file
public ItemLoader : seq1 of char ==> ItemLoader
ItemLoader(fname) ==
(
  def mk_(-,mk_(timeval,input)) = io.freadval[InputTP](fname)
  in
    (
      numTimeSteps := timeval;
      inlines := input
    );
);

-- Returns number of time steps to simulate
public GetNumTimeSteps : () ==> nat
GetNumTimeSteps() ==
  return numTimeSteps;

-- Returns size of item if found in test scenario
-- Returns zero if no item is found for time step and induction id
public GetItemAtTimeStep : nat * nat1 ==> nat
GetItemAtTimeStep(timeStep, icid) ==
(
  let elm = {e | e in set elems inlines & e.#1 = timeStep
              and e.#2 = icid}
  in
    if elm = {}
    then return 0
    else
      let {mk_(-,-,size)} = elm
      in
        return size;
);

```

functions

```

sync

--thread

traces

end ItemLoader

-- =====
-- Item in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class Item
  types
    public ItemTraySize = nat1
    inv it == it <= MaxTrays;

  values
    public MaxSize : nat = 1500;      -- Item maximum size in mm
    public MinSize : nat = 100;      -- Item minimum size in mm
    public MaxTrays: nat = 2;

  instance variables
    id : nat;                        -- Item ID for induction
    size : nat1;                     -- Item size in mm
    inv SizeLimits(size);

    sizeOfTrays : ItemTraySize;      -- Number of trays item occupies
    trays : set of Tray := {};

  operations

  -- InductionController constructor
  public Item: nat1 * nat ==> Item
  Item(s, i) ==
  (
    size := s;
    sizeOfTrays := size div Tray`Size + 1;
    id := i;
  )
  pre SizeLimits(s);

  -- Return item id
  public GetId: () ==> nat
  GetId() ==
    return id;

  -- Returns the number of trays the item occupies
  public GetSizeOfTrays: () ==> ItemTraySize
  GetSizeOfTrays() ==
    return sizeOfTrays;

  -- Return item size
  public GetSize: () ==> nat
  GetSize() ==
    return size;

  -- Creates association between item and tray
  public AssignItemToTray: Tray ==> ()
  AssignItemToTray(tray) ==
    trays := trays union {tray};

```

TrayAllocatorRealtimeModel.vdmrt

```

-- Release item from sorter ring - Implicit operation
public ReleaseItemFromTrays ()
ext wr trays : set of Tray
post trays = {};

functions

-- Function to check invariant and post condition on limits for size
SizeLimits: nat -> bool
SizeLimits(s) ==
    s >= MinSize and s <= MaxSize;

sync

--thread

traces

end Item

-- =====
-- InductionController in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class InductionController
    types

    values
        public InductionRate : nat = 2;      -- minimum trays between each item

    instance variables
        id : nat1;                          -- Induction ID
        allocator : [TrayAllocator] := nil; -- TrayAllocator
        items : seq of Item := [];           -- set of items ready to be inducted

    operations

    public InductionController: nat ==> InductionController
    InductionController(i) ==
    (
        id := i;
    );

    public AssignAllocator: TrayAllocator ==> ()
    AssignAllocator(a) ==
        allocator := a;

    -- Enviroment feeds a new item on induction
    async
    public FeedItem: nat * nat ==> ()
    FeedItem(icid, size) ==
    (
        --duration (100)
        items := items ^ [new Item(icid, size)];
    );

    -- Returns the next item to be inducted
    GetFirstItem: () ==> Item
    GetFirstItem() ==
        return hd items
    pre len items <> 0;

    -- Removes the first item in sequence and clear priority

```


TrayAllocatorRealtimeModel.vdmrt

```

public InductFirstItem: () ==> ()
InductFirstItem() ==
    items := tl items
pre len items <> 0;

-- Blocked until items to induct
ItemsToInduct: () ==> bool
ItemsToInduct () ==
    return len items <> 0;

-- Thread blocked until removed from Map waitingICs
Wait: () ==> ()
Wait() == skip;

async
WaitInductItem: () ==> ()
WaitInductItem() ==
    -- Request tray allocator to induct item and wait for induction
    let item = GetFirstItem()
    in
    (
        allocator.RequestTray(threadid, id, item);
        Wait();
        InductFirstItem();
    );

InductStep: () ==> ()
InductStep() ==
    if (ItemsToInduct()) then
        WaitInductItem();

functions

sync
    -- Enviroment and TrayAllocator threads
    -- mutex (FeedItem);
    per FeedItem => #active(FeedItem) = 0; -- Equal to mutex
    mutex (FeedItem, InductFirstItem);
    mutex (WaitInductItem);

    -- Permission predicate on Wait operation
    per Wait => threadid not in set dom allocator.icThreadsWaiting;

thread
    -- Time units should be TrayAllocator`TrayStepTimeUnits*InductionRate
    periodic (12000, 0, 0, 0) (InductStep);

traces

end InductionController

-- =====
-- Induction in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class Induction
    types

    values

    instance variables
        priority : nat := 0;      -- priotity of induction
        id : nat1;                -- Induction ID

```

operations

```

public Induction: nat ==> Induction
Induction(i) ==
(
    id := i;
);

-- Returns induction controller UID
public GetId: () ==> nat
GetId() ==
    return id;

-- Returns priority of induction controller
public GetPriority: () ==> nat
GetPriority() ==
    return priority;

-- Returns true if induction is waiting with an item
public IsItemWaiting: () ==> bool
IsItemWaiting() ==
    return priority > 0;

-- Increment priority for number of tray steps waiting
public IncrementPriority: () ==> ()
IncrementPriority() ==
    priority := priority + 1; -- Increment priority wait counter

-- Clear priority when item is inducted
public ClearPriority: () ==> ()
ClearPriority() ==
    priority := 0;

```

functions**sync**

```
--thread
```

traces

```
end Induction
```

```

-- =====
-- Tray in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

```

```
class Tray
```

types

```

    public State = <Empty> | <Full>;
    public UID = nat
    inv u == u <= TrayAllocator`NumOfTrays;      -- Limitation on UID

```

values

```
    public Size      : nat1 = 600                -- Size of any tray mm
```

instance variables

```

    -- It is allowed for a tray to be <Full> with no item associated
    -- in this case an unknown item is detected by the card reader
    state : State := <Empty>;
    item : [Item] := nil;

```

```

-- If an item is associated with a tray the state must be <Full>
inv item <> nil => state = <Full>;

id : UID;                                -- Tray UID

operations

-- Tray constructor
public Tray: UID ==> Tray
Tray(i) ==
(
    id := i;
);

-- Return tray id
public GetId: () ==> nat
GetId() ==
    return id;

-- Returns true if tray is empty
public IsTrayEmpty: () ==> bool
IsTrayEmpty () ==
    return state = <Empty>;

-- Returns true if tray is full
public IsTrayFull: () ==> bool
IsTrayFull () ==
    return state = <Full>;

-- Returns item on tray
public GetItem: () ==> [Item]
GetItem () ==
    return item;

-- Set state of tray
public SetState: State ==> ()
SetState (s) ==
(
    if s = <Empty>
    then -- Remove item if tray is empty
        item := nil;
    state := s;
);

-- Returns state of tray ==> <empty> or <full>
public GetState: () ==> State
GetState () ==
    return state;

-- Puts an item on the tray and creates association between tray and item
public ItemOnTray: Item ==> ()
ItemOnTray (i) ==
(
    atomic -- Only needed if item is assigned before state
    (
        item := i;
        state := <Full>;
    );
    item.AssignItemToTray(self);

    --LogError
    --IO`print("-> Item id " ^ String`NatToStr(item.GetId()) ^
    --          "size " ^ String`NatToStr(item.GetSize()) ^

```

```

TrayAllocatorRealtimeModel.vdmrt

--          "on tray id " ^ String`NatToStr(id) ^ "\n");
)
pre state = <Empty> and item = nil;

functions

sync

--thread

traces

end Tray

-- =====
-- TrayAllocator in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class TrayAllocator

types
  public ThroughputResult::
    traysWithItemOnSorter : nat
    twoTrayItemsOnSorter : nat
    traySteps : nat
    inductedItems : nat
    calcThroughput : real;

values
  public InductionSeperation: nat = 2;
  public NumOfInductions : nat = 4;
  public NumOfTrays : nat = 20;
  public SecInHour : nat = 3600;          -- Number of seconds in an hour
  public TrayStepTimeUnits : nat = 6000 -- Used to simulate tray steps

instance variables
  -- Ensure sufficient number of trays on sorter ring
  -- based on inductions and separation
  inv NumOfTrays > InductionSeperation * NumOfInductions;

  countTraySteps : nat := 0;          -- Used for calculation of throughput
  countItemsInducted : nat := 0;      -- Counts the number of items inducted

  -- Induction group and invariants
  public inductionGroup : seq of Induction := [];
  inv inductionGroup <> [] => len inductionGroup = NumOfInductions;
  -- Induction id and inds of inductionGroup sequence must be the same
  inv forall id in set inds inductionGroup
    & inductionGroup(id).GetId() = id;

  -- Sorter ring and invariants
  public sorterRing : inmap Tray`UID to Tray := {|->};
  inv sorterRing <> {|->} => card dom sorterRing = NumOfTrays;
  -- Tray id and dom of sorterRing map must be the same
  inv forall id in set dom sorterRing & sorterRing(id).GetId() = id;

  -- Tray at card reader and invariants
  public trayAtCardReader : [Tray`UID] := nil;
  -- trayAtCardReader must be a valid tray in sorterRing
  inv trayAtCardReader <> nil => trayAtCardReader in set dom sorterRing;

  -- Allocation "strategy pattern" for one and two tray items
  oneTrayStrategy : [AllocatorOneTray] := nil;

```

```

TrayAllocatorRealtimeModel.vdmrt

twoTrayStrategy : [AllocatorTwoTray] := nil;

-- Map of waiting inductions with an item to be inducted
itemsToInductMap : map nat to (Induction * Item) := {|->};
-- Map of thread ids to IC ids
public icThreadsWaiting : map nat to nat1 := {|->};
inv dom itemsToInductMap = dom icThreadsWaiting;

-- Counting number of trays simulated
trayCount: nat := 0;

-- Flag to stop simulation
busy: bool := true;

operations

-- TrayAllocator constructor
public TrayAllocator: () ==> TrayAllocator
TrayAllocator() ==
(
  CreateAllocatorObjs();
);

public CreateAllocatorObjs: () ==> ()
CreateAllocatorObjs() ==
(
  sorterRing := {num |-> new Tray(num) |
                  num in set {1,...,NumOfTrays}};
  inductionGroup := [new Induction(id) |
                     id in set {1,...,NumOfInductions}];

  -- Creating strategies for allocation of one and two tray items
  oneTrayStrategy := new AllocatorOneTray(self);
  twoTrayStrategy := new AllocatorTwoTray(self);
);

-- Simulate sorter-ring moved one tray step
public CardReader: Tray`UID ==> ()
CardReader(uid) ==
(
  -- Update current tray at card reader
  trayAtCardReader := uid;

  -- Count the number of tray steps
  countTraySteps := countTraySteps + 1;
)
pre uid in set dom sorterRing;

-- Inducting item on sorter if empty trays and no higher induction priority
public InductItem: Induction * Item ==> bool
InductItem(ic, item) ==
(
  decl strategy : AllocatorStrategy;

  -- Determine the strategy to compute the allocation of trays
  let numTrays = item.GetSizeOfTrays()
  in
    cases numTrays:
      1 -> strategy := oneTrayStrategy,
      2 -> strategy := twoTrayStrategy
    end;

  -- Central part of the Tray allocation algorithm
  -- Look for inductions with higher priority

```

```

    if strategy.InductionsWithHigherPriority(ic)
    then
        return false
    else
        let trays = strategy.AllocateTray(ic.GetId())
        in
            if trays = {}
            then
                return false
            else
                (
                    countItemsInducted := countItemsInducted + 1;
                    --LogError
                    --IO`print("*Induction id "
                    --      ^ String`NatToStr(ic.GetId()) ^ "\n");
                    -- Assign item to trays
                    PutItemOnTrays(item, trays);
                    return true;
                )
    )
pre ic in set elems inductionGroup and item.GetSizeOfTrays() <= 2;
-- To be changed if Tray`ItemMaxTrays is increased

-- Assign item on empty trays
private PutItemOnTrays: Item * set of Tray ==> ()
PutItemOnTrays(item, trays) ==
    if trays <> {} then
        let t in set trays
        in
            (
                t.ItemOnTray(item);
                PutItemOnTrays(item, trays \ {t});
            )
pre forall t in set trays & t.IsTrayEmpty();

-- Returns true if sorter is full
public IsSorterFull: () ==> bool
IsSorterFull() ==
    return forall id in set dom sorterRing &
        sorterRing(id).GetState() = <Full>;

-- Returns calculated throughput of soter capacity
-- for current state of sorter ring
public GetThroughput: () ==> ThroughputResult
GetThroughput () ==
    CalculateThroughput(countTraySteps, rng sorterRing, countItemsInducted);

-- Called by InductionController thread requesting to induct item
public RequestTray: nat * nat * Item ==> ()
RequestTray (tid, icid, item) ==
    let ic = inductionGroup(icid)
    in
        AddItem(tid, ic, item)
pre icid in set inds inductionGroup
    and tid not in set dom itemsToInductMap;

-- Add induction waiting with item to induct
AddItem: nat * Induction * Item ==> ()
AddItem (t, ic, item) ==
    atomic (
        itemsToInductMap := itemsToInductMap munion {t |-> mk_(ic, item)};
        icThreadsWaiting := icThreadsWaiting munion {t |-> ic.GetId()}
    )
pre t not in set dom itemsToInductMap;

```

```

-- Release induction waiting with item to induct
ReleaseWaitingIC: nat ==> ()
ReleaseWaitingIC (t) ==
atomic (
    itemsToInductMap := {t} <-: itemsToInductMap;
    icThreadsWaiting := {t} <-: icThreadsWaiting
)
pre t in set dom itemsToInductMap;

-- Returns
CheckItemsToInduct: () ==> ()
CheckItemsToInduct () ==
(
    -- Induct items for all waiting inductions
    for all t in set dom itemsToInductMap
    do
        let mk_(ic, item) = itemsToInductMap(t)
        in
            if InductItem(ic, item) then
                (
                    ic.ClearPriority();
                    ReleaseWaitingIC(t);
                )
            else
                ic.IncrementPriority();
);

public StopSimulation: () ==> ()
StopSimulation () == busy := false;

-- Periodic thread operation that simulates the TrayStep
TrayStep: () ==> ()
TrayStep () ==
(
    if (busy) then
    (
        --IO`print("< " ^ String`NatToStr(time) ^ ">>>");
        trayCount := trayCount + 1;
        --LogError IO`print("< " ^ String`NatToStr(trayCount) ^ ">");

        CardReader(trayCount mod TrayAllocator`NumOfTrays + 1);

        -- Induct items for all waiting inductions
        CheckItemsToInduct();
        --IO`print("<<< " ^ String`NatToStr(time) ^ ">");
    );
);

functions

-- Calculates the current throughput based on items on sorter ring
/*
Calculation as sum of simulation
time steps = number of steps * Tray`TraySize/SorterEnviroment`Speed
throughput calculated as items inducted in simulation time converted to items/hour
calculate the number of items inducted = number of tray with status equal <full>
minus sum of two tray items divided by 2
*/
private CalculateThroughput: nat * set of Tray * nat-> ThroughputResult
CalculateThroughput(steps, trays, items) ==
    let runTime :real = steps * (Tray`Size/SorterEnviroment`Speed),
        traysWithItems = {twi | twi in set trays & twi.IsTrayFull()},

```

```

TrayAllocatorRealtimeModel.vdmrt

traysWith2Items = {tw2i | tw2i in set traysWithItems & tw2i.GetItem() <>
nil
    and tw2i.GetItem().GetSizeOfTrays() = 2},
itemsOnSorter = card traysWithItems,
twoTrayItemsOnSorter = card traysWith2Items / 2,
throughput = itemsOnSorter * SecInHour/runTime
in
    mk_ThroughputResult(itemsOnSorter, twoTrayItemsOnSorter,
        steps, items, throughput)
pre trays <> {};

sync

mutex(RequestTray); -- Only allows one induction at a time to induct items
-- Mutex to ensure synchronization between InductionController and TrayAllocator
mutex(RequestTray, CheckItemsToInduct);

thread
    -- Time units to simulate tray steps jitter 0%
    periodic (6000, 0, 0, 0) (TrayStep);

traces

end TrayAllocator

-- =====
-- Allocator in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- (strategy pattern)
-- =====

class AllocatorStrategy

    instance variables
        protected trayAllocator : [TrayAllocator] := nil;

    operations

        public AllocateTray: nat ==> set of Tray
        AllocateTray (-) ==
            is subclass responsibility;

        public InductionsWithHigherPriority: Induction ==> bool
        InductionsWithHigherPriority(ic) ==
            is subclass responsibility;

    functions

        -- Calculate current tray UID at position in front of induction
        -- based on position of card reader
        protected InductionOffset: Tray`UID * nat -> Tray`UID
        InductionOffset(trayAtCardReader, icid) ==
            ((trayAtCardReader + icid*TrayAllocator`InductionSeperation)
                mod TrayAllocator`NumOfTrays) + 1;

end AllocatorStrategy

-- =====
-- AllocatorOneTray in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- (strategy pattern)
-- =====

class AllocatorOneTray is subclass of AllocatorStrategy

```


operations

```

-- AllocatorOneTray constructor
public AllocatorOneTray: TrayAllocator==> AllocatorOneTray
AllocatorOneTray(ta) ==
(
    trayAllocator := ta;
);

-- Allocates tray if empty at induction offset
public AllocateTray: nat ==> set of Tray
AllocateTray (icid) ==
    def posTray = InductionOffset(trayAllocator.trayAtCardReader, icid)
    in
        if trayAllocator.sorterRing(posTray).IsTrayEmpty()
        then return {trayAllocator.sorterRing(posTray)}
        else return {}
pre icid in set inds trayAllocator.inductionGroup;

-- Returns true if higher priority inductions in induction group
public InductionsWithHigherPriority: Induction ==> bool
InductionsWithHigherPriority(ic) ==
    return exists i in set elems trayAllocator.inductionGroup
    & i.GetId() <> ic.GetId()
    and i.GetPriority() > ic.GetPriority()
pre ic in set elems trayAllocator.inductionGroup;

```

end AllocatorOneTray

```

-- =====
-- AllocatorTwoTray in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- (strategy pattern)
-- =====

```

class AllocatorTwoTray **is subclass of** AllocatorStrategy

operations

```

-- AllocatorTwoTray constructor
public AllocatorTwoTray: TrayAllocator==> AllocatorTwoTray
AllocatorTwoTray(ta) ==
(
    trayAllocator := ta;
);

-- Allocates trays if empty at induction offset and offset + 1
public AllocateTray: nat ==> set of Tray
AllocateTray (icid) ==
    let posTray = InductionOffset(trayAllocator.trayAtCardReader, icid),
    posTrayNext = if (posTray - 1) = 0 then
        TrayAllocator.NumOfTrays else posTray - 1
    in
        if trayAllocator.sorterRing(posTray).IsTrayEmpty() and
        trayAllocator.sorterRing(posTrayNext).IsTrayEmpty()
        then return {trayAllocator.sorterRing(posTray),
                    trayAllocator.sorterRing(posTrayNext)}
        else return {}
pre icid in set inds trayAllocator.inductionGroup;

-- Returns true if higher priority inductions in induction group
public InductionsWithHigherPriority: Induction ==> bool

```

```

TrayAllocatorRealtimeModel.vdmrt

InductionsWithHigherPriority(ic) ==
    return exists i in set elems trayAllocator.inductionGroup
        & i.GetId() <> ic.GetId()
        and i.GetPriority() > ic.GetPriority()
pre ic in set elems trayAllocator.inductionGroup;

end AllocatorTwoTray

-- =====
-- String helper class for converting numbers
-- By José Antonio Esparza and Kim Bjerge - spring 2010
-- =====

class String
    types

    values

    instance variables
        static numeric : seq1 of char := "0123456789";

    operations

    static public NatToStr: nat ==> seq1 of char
    NatToStr (val) ==
    (
        decl string : seq1 of char := " ";
        decl x1 : nat := val;
        decl x2 : nat;

        if val = 0 then string := "0";

        while x1 > 0 do
        (
            x2 := (x1 mod 10) + 1;
            string := [numeric(x2)] ^ string;
            x1 := x1 div 10;
        );

        return string;
    );

    functions

end String

```