

TrayAllocatorConcurrentModel.vdmpp

```
-- =====
-- APPENDIX B - Complete VDM++ Concurrent Model
-- Amalgamated - (in one file for printing)
-- =====

-- =====
-- World in tray allocation for a sortation system
-- By José Antonio Esparza and Kim Bjerge - 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"; -- One tray items
    testfile2 : seq1 of char := "scenario2.txt"; -- Two tray items
    testfile3 : seq1 of char := "scenario3.txt"; -- Mix of items

    -- Change here scenario file to be used
    testfiles : seq of seq1 of char := [testfile3];

    public static timerRef : TimeStamp := new TimeStamp();

  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");
);
```

```

TrayAllocatorConcurrentModel.vdmpp

IO`print("Induction separation      [trays]: " ^
        String`NatToStr(TrayAllocator`InductionSeperation) ^ "\n");
IO`print("-----\n");

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

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

functions

sync

--thread

traces

end World

-- =====
-- ItemLoader in tray allocation for a sortation system

```



```

    in
      if elm = {}
      then
        return 0
      else
        let {si} = elm
        in
          return si;
    );

  functions

  sync

  traces

end ItemLoader

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

\begin{vdm_al}
class TimeStamp

\end{vdm_al}
Class used for concurrent VDM++ models. All threads should call the following
operations:
- WaitRelative(t): makes the thread periodic with t = the period
- NotifyAll(): notified all threads sleeping in the wakeUpMap
- Awake(): puts the thread to sleep - will wakeup when t time units has passed
\begin{vdm_al}

values

\end{vdm_al}
The step length with which the currentTime is incremented
One time step in the trayallocation model is equal to one tray step
\begin{vdm_al}

public stepLength : nat = 10;

instance variables

currentTime : nat := 0;
wakeUpMap : map nat to nat := {|->};

operations

\end{vdm_al}
WaitRelative: sleeps the calling thread for 'val' time units - relative to the
currentTime
\begin{vdm_al}

public WaitRelative : nat ==> ()
WaitRelative(val) ==
  AddToWakeUpMap(threadid, currentTime + val);

\end{vdm_al}
WaitAbsolute: sleeps the calling thread undtil a specific time
\begin{vdm_al}

public WaitAbsolute : nat ==> ()

```

```

WaitAbsolute(val) ==
  AddToWakeUpMap(threadid, val);

\end{vdm_al}
AddToWakeUpMap: Utility operation - adding the thread to the wakeUpMap
\begin{vdm_al}

AddToWakeUpMap : nat * nat ==> ()
AddToWakeUpMap(tId, val) ==
  wakeUpMap := wakeUpMap ++ { tId |-> val };

\end{vdm_al}
NotifyThread: notified a specific thread - removing it from the wakeUpMap
\begin{vdm_al}

public NotifyThread : nat ==> ()
NotifyThread(tId) ==
  wakeUpMap := {tId} <-: wakeUpMap;

\end{vdm_al}
NotifyAll: notifies all threads - waking up threads which wakeUpTime is up
\begin{vdm_al}

public NotifyAll : () ==> ()
NotifyAll() ==
  let threadSet : set of nat = {th | th in set dom wakeUpMap & wakeUpMap(th) <=
    currentTime }
  in
    for all t in set threadSet
    do
      NotifyThread(t);

\end{vdm_al}
NotifyAndIncTime: Must only be used by ONE thread - usually the Environment thread.
Increments the currentTime with stepLength time units, and notifies all threads.
\begin{vdm_al}

public NotifyAndIncTime : () ==> ()
NotifyAndIncTime() ==
  (currentTime := currentTime + stepLength;
   NotifyAll();
  );

\end{vdm_al}
GetTime: Returns the currentTime.
\begin{vdm_al}

public GetTime : () ==> nat
GetTime() ==
  return currentTime;

\end{vdm_al}
Awake: Used to sleep threads - will not wake up until threadid is removed from
wakeUpMap
\begin{vdm_al}

public Awake: () ==> ()
Awake() == skip;

sync
  per Awake => threadid not in set dom wakeUpMap;

  mutex(NotifyAll);
  mutex(AddToWakeUpMap);

```

```

    mutex(AddToWakeUpMap, NotifyAll);

end TimeStamp
\end{vdm_al}

\begin{rtinfo}[TimeStamp`NotifyAndIncTime]
{vdm.tc}[TimeStamp]
\end{rtinfo}

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

class SorterEnviroment
  types

  values
    public Speed      : nat1 = 2000;  -- Sorter speed mm/sec
    public Throughput : nat  = 10000;  -- Required items/hour

  instance variables

    public static sc : SC := new SC();
    public static trayStep : TrayStep := new TrayStep(Speed);

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

  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;
    );

    -- Used by traces in TestSernarios
    public FeedItemOnInduction: nat * Item ==> ()
    FeedItemOnInduction(ic, item) ==
      inductionGroup(ic).FeedItem(item);

    public isFinished : () ==> ()
    isFinished() == skip;

  functions

  sync

```

```

per isFinished => not busy;

thread
(
    sc.CreateAssignments(World`env);

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

    while busy do
    (
        for all i in set {1,...,TrayAllocator`NumOfInductions}
        do
        (
            -- Check for item to feed induction at time step
            let size = itemLoader.GetItemAtTimeStep(World`timerRef.GetTime(), i)
            in
                if (size > 0)
                then
                (
                    itemId := itemId + 1;
                    inductionGroup(i).FeedItem(new Item(size, itemId));
                );
        );

        --IO`print(".");
        World`timerRef.WaitRelative(0);
        World`timerRef.NotifyAndIncTime();
        World`timerRef.Awake();

        -- Check if simulation is finish
        if (World`timerRef.GetTime() >= itemLoader.GetNumTimeSteps()) then
            busy := false;
    );
);

traces

end SorterEnviroment

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

class TrayStep

instance variables

    public static stepTime : int := -1;          -- Time it takes to move a tray in ms
    public static counts : nat := 0;             -- Counting number of tray steps
    waitTrayStepMap : map nat to nat := {|->}; -- Map waiting threads to tray step
count

operations

```

TrayAllocatorConcurrentModel.vdmpp

```

-- TrayStep constructor
public TrayStep : nat1 ==> TrayStep
TrayStep(speed) ==
(
    stepTime := (Tray`Size*1000)/speed;    -- [mm]*1000 / [mm/sec]
);

-- Thread waiting until tray steps passed
public WaitSteps : nat ==> ()
WaitSteps(steps) ==
(
    AddToMap(threadid, counts + steps);
    Wait();
);

-- Add threadid to map
AddToMap : nat * nat ==> ()
AddToMap(tId, step) ==
    waitTrayStepMap := waitTrayStepMap ++ { tId |-> step };

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

-- Threadid remove from map of waiting threads
RemoveFromMap : nat ==> ()
RemoveFromMap(tId) ==
    waitTrayStepMap := {tId} <-: waitTrayStepMap;

-- Notifies waiting threads if TraySteps passed
NotifyWaitingThreads : () ==> ()
NotifyWaitingThreads() ==
let threadSet : set of nat = {th | th in set dom waitTrayStepMap &
                                waitTrayStepMap(th) <= counts }
in
    for all t in set threadSet
    do
        RemoveFromMap(t);

public static GetCounts : () ==> nat
GetCounts () == return counts;

IncCounts : () ==> ()
IncCounts () == counts := counts + 1;

sync
per Wait => threadid not in set dom waitTrayStepMap;

-- Block reading counter while incrementing
-- mutex(GetCounts, IncCounts);
-- Causes error when added

thread

while true do
(
    IncCounts();
    NotifyWaitingThreads();

    IO`print("< " ^ String`NatToStr(counts) ^ ">");
    World`timerRef.WaitRelative(stepTime);
    World`timerRef.NotifyAll();
    World`timerRef.Awake();

```



```

    )

end TrayStep

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

class SC

    types

    values

    instance variables
        public static ic1 : InductionController := new InductionController(1);
        public static ic2 : InductionController := new InductionController(2);
        public static ic3 : InductionController := new InductionController(3);
        public static inductionGroup : seq of InductionController :=
                                                    [ic1, ic2, ic3];

        public static allocator : TrayAllocator :=
                                                    new TrayAllocator(inductionGroup);

    operations

    -- SystemController constructor
    public SC: () ==> SC
    SC() ==
    (
    );

    -- Create assignments relations between objects
    public CreateAssignments: SorterEnvironment ==> ()
    CreateAssignments (env) ==
    (
        ic1.AssignAllocator(allocator);
        ic2.AssignAllocator(allocator);
        ic3.AssignAllocator(allocator);
        env.AssignInductionGroup(inductionGroup);
    );

    functions

    sync

    --thread

    traces

end SC

-- =====
-- 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;          -- trays between each item
        public InductionRateInMs : nat = (InductionRate * Tray`Size * 1000)
                                           / SorterEnvironment`Speed;

```

```
-- [mm] / [mm/msec]
```

instance variables

```
priority : nat := 0;           -- priority of induction
id : nat1;                     -- Induction ID
allocator : TrayAllocator;     -- TrayAllocator
items : seq of Item := [];     -- items ready to be inducted
stepCount : nat := 0;         -- Steps between inducting items
selfIC : InductionController;

-- If induction is waiting there must be items in sequence
inv priority > 0 => len items > 0;
```

operations

```
-- InductionController constructors
```

```
public InductionController: TrayAllocator * nat ==> InductionController
InductionController(a, n) ==
```

```
(
  allocator := a;
  id := n;
  selfIC := self;
);
```

```
public InductionController: nat ==> InductionController
InductionController(n) ==
```

```
(
  id := n;
  selfIC := self;
);
```

```
public AssignAllocator: TrayAllocator ==> ()
```

```
AssignAllocator(a) ==
  allocator := a;
```

```
-- 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;
```

```
-- Get size of waiting item in number of trays
```

```
public GetSizeOfWaitingItem: () ==> nat
GetSizeOfWaitingItem() ==
  if not IsItemWaiting()
  then
    return 0 -- No waiting items
  else
    let item = hd items
    in item.GetSizeOfTrays();
```

```
-- Environment feeds a new item on induction
```

```
public FeedItem: Item ==> ()
FeedItem(i) ==
  items := items ^ [i];
```

```

-- 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
public InductFirstItem: () ==> ()
InductFirstItem() ==
    atomic -- Due to invariant
    (
        items := tl items;
        priority := 0
    )
pre len items <> 0;

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

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

functions

sync
    -- Enviroment and TrayAllocator threads
    mutex (FeedItem, InductFirstItem);

    -- TrayAllocator and InductionController threads
    mutex (InductFirstItem);
    mutex (IncrementPriority);
    mutex (GetFirstItem, IncrementPriority, InductFirstItem);

    -- Block thread if no items to induct
    per ItemsToInduct => len items > 0;

thread
(
    while (ItemsToInduct()) do
    (
        -- Request tray allocator to induct item and wait for induction
        let item = GetFirstItem()
        in
            allocator.RequestTray(threadid, selfIC, item);
            allocator.Wait();

            -- Wait time equal InductionRate in tray steps
            -- to keep separation between items
            SorterEnviroment`trayStep.WaitSteps(InductionRate);
    );
);
traces

end InductionController

-- =====
-- 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;           -- Limitation on trays an items

  values
    public MaxSize : nat = 1500;       -- Item maximum size in mm
    public MinSize : nat = 100;       -- Item minimum size in mm
    public MaxTrays: nat = 2;         -- Number of trays an item occupies

  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};

  -- 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

-- =====
-- 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 ==> ()

```

TrayAllocatorConcurrentModel.vdmpp

```

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);

    IO`print("-> Item id " ^ String`NatToStr(item.GetId()) ^
        "size " ^ String`NatToStr(item.GetSize()) ^
        "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 = 3;
        public NumOfTrays : nat = 20;
        public SecInHour : nat = 3600 -- Number of seconds in an hour

    instance variables
        -- Ensure sufficient number of trays on sorter ring

```

TrayAllocatorConcurrentModel.vdmpp

```

-- 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 InductionController := [];
inv 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 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;
twoTrayStrategy : AllocatorTwoTray;

-- Map of waiting inductions with an item to be inducted
itemsToInductMap : map nat to (InductionController * Item) := {|->};

operations

-- TrayAllocator constructor
public TrayAllocator: seq of InductionController ==> TrayAllocator
TrayAllocator(ig) ==
(
    sorterRing := {num |-> new Tray(num) | num in set {1,...,NumOfTrays}};
    inductionGroup := ig;

    -- 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 * Tray`State ==> ()
CardReader(uid, state) ==
(
    -- Update current tray at card reader
    trayAtCardReader := uid;

    -- Simulate change of Tray state !!!
    -- sorterRing(trayAtCardReader).SetState(state);

    -- 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: InductionController * Item ==> bool
InductItem(ic, item) ==
(

```

```

dcl 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
-- before allocation of empty trays
if strategy.InductionsWithHigherPriority(ic)
then
    return false
else
    let trays = strategy.AllocateTray(ic.GetId())
    in
        if trays = {}
        then
            return false
        else
            (
                countItemsInducted := countItemsInducted + 1;
                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 * InductionController * Item ==> ()
RequestTray (t, ic, item) ==
    AddItem(t, ic, item)
pre t not in set dom itemsToInductMap;

-- Thread blocked until removed from Map waitingICs

```



```

public Wait: () ==> ()
Wait() == skip;

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

-- Release induction waiting with item to induct
ReleaseWaitingIC: nat ==> ()
ReleaseWaitingIC (t) ==
    itemsToInductMap := {t} <-: itemsToInductMap
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.InductFirstItem();
                    ReleaseWaitingIC(t);
                )
            else
                ic.IncrementPriority();
);

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()},
    traysWith2Items = {tw2i | tw2i in set traysWithItems &
        tw2i.GetItem() <> nil and tw2i.GetItem().GetSzieOfTrays() = 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

```

TrayAllocatorConcurrentModel.vdmpp

```

mutex(RequestTray, CheckItemsToInduct);
-- Permission predicate on Wait operation
per Wait ==> threadid not in set dom itemsToInductMap;

thread
  while (true) do
  (
    -- Wait time equal to one tray step
    SorterEnviroment`trayStep.WaitSteps(1);

    CardReader(TrayStep`GetCounts() mod TrayAllocator`NumOfTrays + 1, <Empty>);

    -- Induct items for all waiting inductions
    CheckItemsToInduct();
  );

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: InductionController ==> 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: InductionController ==> 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)}
                           -- Return the set of 2 empty trays
                else return {}
pre icid in set inds trayAllocator.inductionGroup;

-- Returns true if higher priority inductions in induction group
public InductionsWithHigherPriority: InductionController ==> bool
InductionsWithHigherPriority(ic) ==
    return exists i in set elems trayAllocator.inductionGroup

```

```

TrayAllocatorConcurrentModel.vdmpp

        & 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) ==
    (
        dcl string : seq1 of char := " ";
        dcl x1 : nat := val;
        dcl 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

```