**Description of the**

**USB maker project**

Table of Contents

[1 Overview 3](#_Toc518481413)

[2 Exchanged signals between components 5](#_Toc518481414)

[3 Human Machine Interface (HMI) 6](#_Toc518481415)

[4 Part Tracker 8](#_Toc518481416)

[5 Warehouse 12](#_Toc518481417)

[6 Robotic Arm 13](#_Toc518481418)

[7 Wagon 14](#_Toc518481419)

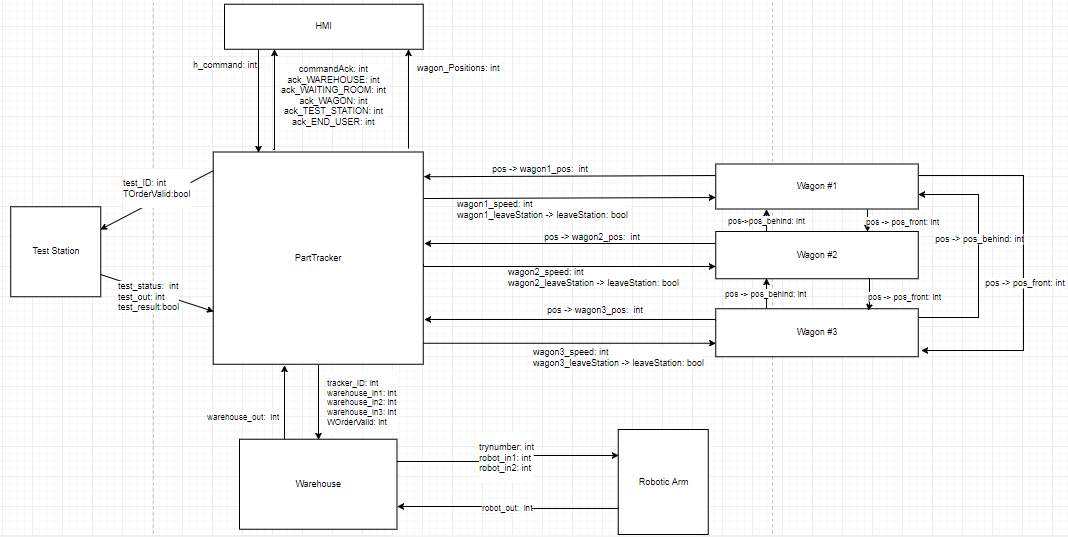
[8 Test Station 15](#_Toc518481420)

[9 Miscellaneous 15](#_Toc518481421)

# Overview

Each component of the prototype (USB maker) was first modelled (in a very abstract way) in VDM by using the Overture tool. The goal of this homogenous co-simulation was to identify the right interaction protocols (signals) among the various components (stations) of the prototype, and not on the model’s accuracy. Therefore, the VDM simulation model includes distinct models for each component of the system (Fig. x):

* H: HMI
* P: Part Tracker;
* W: Warehouse;
* R: Robotic Arm;
* C: Conveyor / Wagon (x3);
* T: Test Station



**system** USBMaker

**instance variables**

/\*h = HMI \*/

**static** h\_command : IntPort := **new** IntPort(0);

**static** commandAck : IntPort := **new** IntPort(0);

**static** wagonPoss : IntPort := **new** IntPort(0);

**static** ack\_WAREHOUSE : IntPort := **new** IntPort(0);

**static** ack\_WAITING\_ROOM : IntPort := **new** IntPort(0);

**static** ack\_WAGON : IntPort := **new** IntPort(0);

**static** ack\_TEST\_STATION : IntPort := **new** IntPort(0);

**static** ack\_END\_USER : IntPort := **new** IntPort(0);

/\*p = part tracker\*/

**static** p\_ID: IntPort := **new** IntPort(0);

/\*w = warehouse\*/

**static** w\_valid: BoolPort := **new** BoolPort(**false**);

**static** w\_in1: IntPort := **new** IntPort(0);

**static** w\_in2: IntPort := **new** IntPort(0);

**static** w\_in3: IntPort := **new** IntPort(0);

**static** w\_out: IntPort := **new** IntPort(0);

/\*r = robot\*/

**static** r\_ID: IntPort := **new** IntPort(0);

**static** r\_in1: IntPort := **new** IntPort(0);

**static** r\_in2: IntPort := **new** IntPort(0);

**static** r\_out: IntPort := **new** IntPort(0);

/\*c = conveyor/wagons\*/

**static** posA: IntPort := **new** IntPort(0);

**static** posB: IntPort := **new** IntPort(0);

**static** posC: IntPort := **new** IntPort(0);

**static** speedA: IntPort := **new** IntPort(GlobalTypes`DEFAULT\_SPEED);

**static** speedB: IntPort := **new** IntPort(GlobalTypes`DEFAULT\_SPEED);

**static** speedC: IntPort := **new** IntPort(GlobalTypes`DEFAULT\_SPEED);

**static** leaveStationA: BoolPort := **new** BoolPort(**false**);

**static** leaveStationB: BoolPort := **new** BoolPort(**false**);

**static** leaveStationC: BoolPort := **new** BoolPort(**false**);

/\*t = test station \*/

**static** t\_ID: IntPort := **new** IntPort(0);

**static** t\_valid: BoolPort := **new** BoolPort(**false**);

**static** t\_out: IntPort := **new** IntPort(0);

**static** t\_status: IntPort := **new** IntPort(0);

**static** testResult: BoolPort := **new** BoolPort(**true**);

/\*perturbations\*/

**static** perturb\_w : IntPort:= **new** IntPort(0);

**static** perturb\_wagA : IntPort:= **new** IntPort(0);

**static** perturb\_wagB : IntPort:= **new** IntPort(0);

**static** perturb\_wagC : IntPort:= **new** IntPort(0);

**static** perturb\_t : IntPort:= **new** IntPort(0);

**static** perturb\_stat : IntPort:= **new** IntPort(0);

**public** **static** hmi: Unit\_HMI:= **new** Unit\_HMI(h\_command, commandAck, wagonPoss, ack\_WAREHOUSE, ack\_WAITING\_ROOM, ack\_WAGON, ack\_TEST\_STATION, ack\_END\_USER,perturb\_stat);

**public** **static** tracker: Unit\_PartTracker := **new** Unit\_PartTracker(h\_command, commandAck, wagonPoss, ack\_WAREHOUSE, ack\_WAITING\_ROOM, ack\_WAGON, ack\_TEST\_STATION, ack\_END\_USER,p\_ID, w\_valid, w\_in1, w\_in2, w\_in3, w\_out, posA, posB, posC, speedA, speedB, speedC, leaveStationA, leaveStationB, leaveStationC, t\_ID, t\_valid, t\_out, t\_status, testResult, perturb\_w, perturb\_wagA, perturb\_wagB, perturb\_wagC, perturb\_t, perturb\_stat);

**public** **static** warehouse: Unit\_Warehouse:= **new** Unit\_Warehouse(p\_ID, w\_valid, w\_in1, w\_in2, w\_in3, w\_out, r\_ID, r\_in1, r\_in2, r\_out, perturb\_w);

**public** **static** robot: Unit\_Robot:= **new** Unit\_Robot(r\_ID, r\_in1, r\_in2, r\_out);

**public** **static** wagonA: Unit\_Wagon:= **new** Unit\_Wagon(**new** IntPort(2), posA, speedA, leaveStationA, posC, posB, perturb\_wagA);

**public** **static** wagonB: Unit\_Wagon:= **new** Unit\_Wagon(**new** IntPort(1), posB, speedB, leaveStationB, posA, posC, perturb\_wagB);

**public** **static** wagonC: Unit\_Wagon:= **new** Unit\_Wagon(**new** IntPort(0), posC, speedC, leaveStationC, posB, posA, perturb\_wagC);

**public** **static** testStation: Unit\_TestStation:= **new** Unit\_TestStation(t\_ID, t\_valid, t\_out, t\_status, testResult, perturb\_t);

# Exchanged signals between components

HP1: HMI to PartTracker, making a command

hmi\_command: IntPort; /\*HMI sending to PartTracker a new order / change / query / cancel order \*/

PH1: PartTracker to HMI, answering a command

commandAck: IntPort; /\*Part Tracker response with item status\*/

PH2-6: PartTracker to HMI, informing which item enters which state

ack\_WAREHOUSE : IntPort; /\* ID of item reaching WAREHOUSE\*/

ack\_WAITING\_ROOM : IntPort; /\* ID of item reaching WAITING\_ROOM\*/

ack\_WAGON : IntPort; /\* ID of item reaching WAGON\*/

ack\_TEST\_STATION : IntPort; /\* ID of item reaching TEST\_STATION\*/

ack\_END\_USER : IntPort; /\* ID of item reaching END\_USER\*/

PH7: PartTracker to HMI, communicating the perturbation status of all units

perturb\_status: IntPort;

PW1: Part Tracker to Warehouse, requesting assembly for each order

tracker\_ID: IntPort; /\*order ID\*/

warehouse\_in1: IntPort; /\*colour <LEFT>\*/

warehouse\_in2: IntPort; /\*colour <MIDDLE>\*/

warehouse\_in3: IntPort; /\*colour <RIGHT>\*/

PW2: Part Tracker to Warehouse, confirming order validity

WOrderValid: BoolPort; /\*order still valid or cancelled\*/

WR1: Warehouse to Roboric Arm, requesting to move a part from location1 to location2

trynumber: IntPort; /\*ID for this request, different from order ID\*/

robot\_in1: IntPort; /\*location 1\*/

robot\_in2: IntPort; /\*location 2\*/

RW1: Robotic Arm to Warehouse, status of the robot

robot\_out: IntPort; /\* FREE = 0 / BUSY = 1 \*/

WP1: Warehouse to Part Tracker, confirming assembly completed

warehouse\_out: IntPort; /\* return order ID \*/

WP2: Warehouse to Part Tracker, communicating perturbation status

perturb\_w: IntPort; /\* perturbation level \*/

CP1: Wagon to Part Tracker, communicates current position (or estimate position)

wagon\_pos: IntPort; /\*wagon positon\*/

CP2: Wagon to Part Tracker, communicating perturbation status

perturb\_wag: IntPort; /\* perturbation level \*/

PC1: Part Tracker to Wagon, setting wagon speed

wagon\_speed: IntPort;

PC2: Part Tracker to Wagon, allowing the wagon to leave station

wagon\_leaveStation: BoolPort; /\*true = permission to leave station\*/

CC1: Wagon to Wagon behind, communicating position in order to avoid collision

wagon\_pos: IntPort;

CC2: Wagon to Wagon in front, requesting to speed up

wagon\_speed: IntPort;

PT1: Part Tracker to Test Station, requesting testing of order ID

test\_ID: IntPort; /\* order ID \*/

TOrderValid: BoolPort; /\*order still valid or cancelled\*/

TP1: Test Station to Part Tracker, status of the robot

test\_status: IntPort; /\* FREE = 0 / BUSY = 1 \*/

TP2: Test Station to Part Tracker, return order ID to indicate that it has been tested

test\_out: IntPort; /\* return order ID \*/

test\_result: BoolPort; /\* quality report: true = item as required \*/

TP3: Test Station to Part Tracker, communicating perturbation status

perturb\_t: IntPort; /\* perturbation level \*/

# Human Machine Interface (HMI)

## Short Description

The HMI unit handles the user interface.

## Communicates with

* Part Tracker

### Internal states

USBlocs : **map** GlobalTypes`OrderID **to** (**<UNDEFINED>** | **<ORDERED>** | **<WAREHOUSE>** | **<WAITING\_ROOM>** | **<WAGON>** | **<TEST\_STATION>** | **<END\_USER>** | **<CANCELLED>**) := {|->};

## Actions

In the VDM model, the HMI unit reads the orders from a \*.csv file that contains indications of when to place an order and (if applicable) when to cancel the order. The purpose of the HMI real implementation of the HMI is to allow the user to place / change / cancel orders at any time through an app.

#### Reading CSV file section

The file orders.csv is read only once. The guarding variable against reading the file multiple times is OrdersHaveBeenRead = **false** , initialized as **false** and set to **true** after reading of the orders.

The file orders.csv contains one order at each line. One order contains 6 values:

GlobalTypes`OrderTime \* /\* time [s.] at which order is placed \*/

GlobalTypes`OptionInt \* /\* requested wagon speed for order \*/

GlobalTypes`ColourInt \* /\* colour of part <LEFT> of the order \*/

GlobalTypes`ColourInt \* /\* colour of part <MIDDLE> of the order \*/

GlobalTypes`ColourInt \* /\* colour of part <RIGHT> of the order \*/

GlobalTypes`CancelTime /\* time [s.] at which order is cancelled \*/

All orders are recorded in the following maps, with respect to their ID (which is the number of the line on which the order is written in orders.csv):

/\* USBitems maps all order info (except cancellation time) \*/

USBitems := USBitems ++ {currid |-> **mk\_**(vals(1),vals(2),vals(3),vals(4),vals(5))};

/\* USBlocs maps item location in the production line \*/

USBlocs := USBlocs ++ {currid |-> **<UNDEFINED>**};

#### Ordering section

The location of all orders is set initially to USBlocs := **<UNDEFINED>**. When the runtime is reaching the order time, USBlocs(idx) is sent to the Part Tracker via message HP1:

/\*order section\*/

**for all** idx **in set** **dom** USBitems **do**

(

**if** idx > 0 **and** OrderLocs(idx) = **<UNDEFINED>** **and**

GlobalTypes`roundTimeToTheSecond(**time**) >= USBordertimes(idx) **and**

(GlobalTypes`roundTimeToTheSecond(**time**) < USBcanceltimes(idx) **or**

USBcanceltimes(idx) < USBordertimes(idx) ) **then**

(

/\*compose message "neworder"\*/

composedCommand := GlobalTypes`composeCommand(1,idx,

USBoptions(idx),USBcolour1(idx),USBcolour2(idx),USBcolour3(idx));

hmi\_command.setValue(composedCommand);

);

);

Similarly, the HMI can request to the Part Tracker to cancel an order, to change an order or to simply enquire about the status of the order (to all of which the Part Tracker responds with message PH1, indicating the status as stored from USBlocs).

The composition rule is as follows:

* **16 bits : order ID**
* **2 bits: command type (00: cancel, 01: newOrder, 02: changeOrder, 03: enquiry)**
* **3 x 3 bits: colour of the left / middle / right part of the item**
* **5 bits: the speed option**

**public** composeCommand: **nat**\***nat**\*OptionInt\*ColourInt\*ColourInt\*ColourInt -> **nat**

composeCommand(command,id,opt,c1,c2,c3) ==

/\*(2^16)\*/ 65536\*id +

/\*(2^14)\*/ 16384\*command +

/\*(2^11)\*/ 2048\*c1 +

/\*(2^ 8)\*/ 256\*c2 +

/\*(2^ 5)\*/ 32\*c3 +

opt;

Examples of composed HMI commands:

New order, ID=1, colour scheme RGB, speed 5: **0000000000000001 0110001000100101**

Cancel order, ID=9, colour scheme RGB, speed 25: **0000000000001001 0010001000111001**

Change, ID=1031, new colours all white, new speed 5: **0000010000000111 1011111111100101**

# Part Tracker

## Short Description

Part Tracker is the unit that handles orders.

## Communicates with

* HMI
* Warehouse
* Wagons
* Test Station

### Internal states

USBlocs : **map** GlobalTypes`OrderID **to** (**<UNDEFINED>** | **<ORDERED>** | **<WAREHOUSE>** | **<WAITING\_ROOM>** | **<WAGON>** | **<TEST\_STATION>** | **<END\_USER>** | **<CANCELLED>**) := {|->};

## Actions

#### Receiving requests from HMI

Upon receiving a command from HMI, the Part Tracker takes appropriate actions. For new order commands, all orders are recorded in the following maps, with respect to their ID:

/\* USBitems maps all order info (except cancellation time) \*/

USBitems := USBitems ++ {currid |-> **mk\_**(vals(1),vals(2),vals(3),vals(4),vals(5))};

/\* USBlocs maps item location in the production line \*/

USBlocs := USBlocs ++ {currid |-> **<ORDERED>**};

/\* USBwagon maps the number of the wagon that the item is on

(or 0 if item is not on any wagon) \*/

USBwagons := USBwagons ++ {currid |-> 0 };

/\* USBordertimes maps the times when the order is received \*/

USBordertimes := USBordertimes ++ {currid |-> vals(1)};

/\* USBoptions maps the rquested speed for the wagon transporting the item \*/

USBoptions := USBoptions ++ {currid |-> vals(2) };

/\* USBcanceltimes maps the times when the order is cancelled by the user \*/

USBcanceltimes := USBcanceltimes ++ {currid |-> vals(6)};

/\* USBordervalidity maps the status of the order

(true = still valid or false = cancelled) \*/

USBordervalidity := USBordervalidity ++ {currid |-> **true**};

The Part Tracker then composes an answer to the HMI, acknowledging the new order.

item\_status.setValue( GlobalTypes`composeAnswer(newOrderCmd,newOrderIdx,

GlobalTypes`StatusToInt(**<ORDERED>**)));

The structure of the answer is in a way similar to the structure of the request:

* **16 bits : order ID**
* **2 bits: command type (00: cancel, 01: newOrder, 02: changeOrder, 03: enquiry)**
* **14 bits: status of the item**

**public** composeAnswer: **nat** \* **nat** \* **nat** -> **nat**

composeAnswer(command,id,status) ==

/\*(2^16)\*/ 65536\*id +

/\*(2^14)\*/ 16384\*command +

status;

For instance, the answer to the HMI command for a new order, ID=1, colour scheme RGB, speed 5 is: **0000000000000001 010000000000001** (status: **<ORDERED>**)

For any other HMI command, the Part Tracker replies with a similarly composed answer.

#### Informing the HMI about activities of the items

For the following internal states, the Part Tracker has a dedicated line to communicate to the HMI that an item has reached that state. The Part Tracker informs the HMI only when an item enters that state, not when the item exits the state, by sending the ID of the order associated with the item on the dedicated line. For instance, the Part Tracker sends the ID =1 of the first order through message ack\_WAREHOUSE, when the order is passed to the warehouse. After the warehouse finishes constructing the item and moves it to the waiting room, the Part Tracker sends the ID=1 through message ack\_WAITING\_ROOM; if in the meantime no other orders were coming in, and nothing is entering the warehouse, the ID on message ack\_WAREHOUSE is still 1 even after the item has long left the warehouse.

ack\_WAREHOUSE | **<WAREHOUSE>**

ack\_WAITING\_ROOM | **<WAITING\_ROOM>**

ack\_WAGON | **<WAGON>**

ack\_TEST\_STATION | **<TEST\_STATION>**

ack\_END\_USER | **<END\_USER>**

The HMI is also informed about the perturbation levels detected by each unit through message perturb\_status, composed in the following way:

**public** composePerturbStatus: **nat** \* **nat** \* **nat** \* **nat** \* **nat** -> **nat**

composePerturbStatus(perturb\_w,perturb\_wagA,perturb\_wagB,perturb\_wagC,perturb\_t) ==

/\*(2^8)\*/ 128\*perturb\_w +

/\*(2^6)\*/ 64\*perturb\_wagA +

/\*(2^4)\*/ 16\*perturb\_wagB +

/\*(2^2)\*/ 4\*perturb\_wagC +

perturb\_t;

/\* 11 11 11 11 11

every two bits = perturbation level for one component...

warehouse - wagonA - wagonB - wagonC - testStation \*/

The perturbation levels are defined as:

**public** PERTURB\_NONE:**nat** = 0;

**public** PERTURB\_LOW :**nat** = 1;

**public** PERTURB\_MED :**nat** = 2;

**public** PERTURB\_HIGH:**nat** = 3;

#### Cancellation section

If the item is not already **<CANCELLED>** or **<UNDEFINED>** or at the **<END\_USER>**, when the HMI requests a cancellation, the validity of the order is changed to **false**. An additional condition is that the order validity of the item is true (to avoid setting the validity to false at every step).

**if** USBlocs(idx) **<>** **<UNDEFINED>** **and**

USBlocs(idx) **<>** **<CANCELLED>** **and**

USBlocs(idx) **<>** **<END\_USER>** **and**

USBordervalidity(idx) = **true** **then**

(

USBordervalidity(idx) := **false**;

);

Cancellation occurs directly for items that are in state **<ORDERED>** or **<WAITING\_ROOM>**.

/\*if still in state ordered => direct cancellation\*/

**if** ( USBlocs(idx) = **<ORDERED>** **or** USBlocs(idx) = **<WAITING\_ROOM>** ) **and**

USBordervalidity(idx) = **false** **then**

(

USBlocs(idx) := **<CANCELLED>**;

);

#### Cancellation during state warehouse

If the cancellation occurs while the item is in state **<WAREHOUSE>**, the signal warehouse order validity is set to **false** and the USBlocs(idx) is updated to state **<CANCELLED>** only when the warehouse finished processing the cancellation (and returns the order ID).

**for all** idx **in set** **dom** USBitems **do**

(

**if** USBlocs(idx) = **<WAREHOUSE>** **then**

(

/\*Confirm order is still valid\*/

WOrderValid.setValue(USBordervalidity(idx));

);

);

#### Warehouse section

When the USBlocs(idx) of the previous warehouse order is not in state **<WAREHOUSE>** any more, the Part Tracker requests the assembly of the next available **<ORDERED>** item.

**for all** idx **in set** **dom** USBitems **do**

( /\*move order to warehouse\*/

i**f** USBlocs(idx) = **<ORDERED>** **and**

(USBlocs(tracker\_ID.getValue()) **<>** **<WAREHOUSE>** ) **then**

(

**if** USBordervalidity(idx) = **true** **then**

(

tracker\_ID.setValue(idx);

WOrderValid.setValue(**true**);

warehouse\_in1.setValue(msgIntPart1);

warehouse\_in2.setValue(msgIntPart2);

warehouse\_in3.setValue(msgIntPart3);

warehouse\_out.setValue(0);

USBlocs(idx) := **<WAREHOUSE>**;

)

);

);

#### Waiting room section

When Warehouse returns the current order ID, the Part Tracker sets the current order USBloc to **<WAITING\_ROOM>** or **<CANCELLED>**, depending on the order validity**.**

/\*Waiting Room section\*/

**let** id = warehouse\_out.getValue() **in**

**if** id **in set** **dom** USBlocs **and** USBlocs(id) = **<WAREHOUSE>** **then**

(

USBlocs(id) := **if** USBordervalidity(id)

**then** **<WAITING\_ROOM>**

**else** **<CANCELLED>**;

);

#### Wagon section

The Part Tracker receives the current location (or an estimate of the location) and speed from each wagon.

Wagons\_pos := [wagonA\_pos.getValue(),wagonB\_pos.getValue(),wagonC\_pos.getValue()];

Wagons\_spds:= [wagonA\_spd.getValue(),wagonB\_spd.getValue(),wagonC\_spd.getValue()];

While WagonK is in loading station, the Part Tracker may make a request that an item is loaded (moved from **<WAITING\_ROOM>** to **<WAGON>**).

**for** widx = 1 **to** 3 **do**

**if** Wagons\_pos(widx) = GlobalTypes`LOADING\_STATION **and**

WagonsLoads(widx) = 0 **then**

**for all** idx **in set** **dom** USBitems **do**

**if** WagonsLoads(widx) = 0 **and**

USBlocs(idx) = **<WAITING\_ROOM>** **then**

(

USBlocs(idx) := **<WAGON>**;

USBwagons(idx) := widx;

WagonsLoads(widx) := idx;

)

When WagonK is in loading station, it waits for a message from the Part Tracker to leave the station. When wagons are not loaded, widx = 0 and the speed of the wagon is set to DEFAULT\_SPEED.

wagonA/B/C\_leaveStationCommand.setValue(**true**);

wagonA/B/C\_speed.setValue( USBoptions(WagonsLoads(widx)) );

When WagonK reaches Test Station, the USB is unloaded automatically if the test station is FREE, therefore the Part Tracker sets the USBlocs to **<TEST\_STATION>**.

**if** Wagons\_pos(widx) = GlobalTypes`TESTING\_STATION **then**

**if** WagonsLoads(widx) **<>** 0 **and**

test\_status.getValue() = GlobalTypes`FREE **then**

(

USBlocs(WagonsLoads(widx)) := **<TEST\_STATION>**;

test\_ID.setValue(WagonsLoads(widx));

TOrderValid.setValue(USBordervalidity(WagonsLoads(widx)));

test\_out.setValue(0);

WagonsLoads(widx) := 0;

);

When WagonK is in testing station, it waits for a message from the Part Tracker to leave the station with DEFAULT\_SPEED .

**if** WagonsLoads(widx) = 0 **then**

(

wagonA/B/C\_leaveStationCommand.setValue(**true**);

wagonA/B/C\_speed.setValue( GlobalTypes`DEFAULT\_SPEED );

);

#### Testing section

While the test station is busy, the test order ID is set to 0. Part Tracker receives the test readings from the Test Station and decides whether to move the USB to **<END\_USER>** or to **<UNDEFINED>** and re-order the item.

#### Cancellation during state test station

If the cancellation occurs while the item is in state **<TEST\_STATION>**, the test order validity is set to **false** and the USBlocs(idx) is updated to state **<CANCELLED>** only when the test station finished processing the cancellation (and returns the order ID).

**for all** idx **in set** **dom** USBitems **do**

**if** USBlocs(idx) = **<TEST\_STATION>** **then**

TOrderValid.setValue(USBordervalidity(idx));

**let** id = test\_out.getValue() **in**

**if** id **in set** **dom** USBlocs **and** USBlocs(id) = **<TEST\_STATION>** **then**

(

**if** USBordervalidity(id) = **true** **then**

USBlocs(id) := **if** test\_result.getValue()

**then** **<END\_USER>**

**else** **<UNDEFINED>**;

**if** USBordervalidity(id) = **false** **then**

USBlocs(id) := **<CANCELLED>**;

);

# Warehouse

## Short Description

Assembles the USB from component parts.

## Communicates with

* Part Tracker
* Robot

### Internal states

STAGE0: **nat** := 0; /\* no pieces moved \*/

STAGE1: **nat** := 1; /\* correct piece <Left> in assembly box \*/

STAGE2: **nat** := 2; /\* correct piece <Middle> in assembly box \*/

STAGE3: **nat** := 3; /\* correct piece <Right> in assembly box \*/

STAGE4: **nat** := 4; /\* assembled item moved into waiting room \*/

## Actions

On receiving an assembly request, the Warehouse sets order ID in STAGE0. While the **<Left>** | **<Middle>** | **<Right>** part is coloured incorrectly, the Warehouse requests the removal of the part to the Robotic Arm, going through stages 1-3. The Robotic Arm returns its state at all times (BUSY/FREE). The Warehouse makes requests only when Robotic Arm is FREE.

When all parts are correct, Warehouse assembles the piece and requests to the Robotic Arm to move the USB to the waiting line.

If the Warehouse receives a cancellation signal, it moves the item to STAGE4 regardless of the current state. After reaching STAGE4, the Warehouse returns to the Part Tracker the order ID so that USBlocs can be updated.

# Robotic Arm

## Short Description

Moves parts or pieces from one location to another.

## Communicates with

* Warehouse

### Internal states

BUSY / FREE

## Actions

On receiving a request, the Robotic Arm executes the command, moving whatever piece / part from location1 to location2. During the action, the Robotic Arm outputs BUSY state and does not accept commands.

# Wagon

## Short Description

Transports the pieces from the waiting room to the test station.

## Communicates with

* Part Tracker
* Wagon in front of it

### Internal states

Location estimation

Speed estimation

## Actions

The Wagon computes an estimate position periodically, having correct (sure) positions at the loading and testing stations. The positions of the loading and testing stations on the circular track are defined in GlobalTypes.

-- min value: MAX\_SPEED + 1; max value: TRACK\_LENGTH-MAX\_SPEED–1

**public** LOADING\_STATION :**nat** = 26;

-- min value: MAX\_SPEED + 1; max value: TRACK\_LENGTH-MAX\_SPEED-1

**public** TESTING\_STATION :**nat** = 70;

**public** TRACK\_LENGTH :**nat** = 100;

**public** DEFAULT\_SPEED :**nat** = 1;

**public** MAX\_SPEED :**nat** = **floor**(TRACK\_LENGTH/4);

The next position of the wagon is computed from the current estimated position, taking into account the current speed of the wagon and the position of the wagon in front of it.

If the wagon in front is too close, the current wagon requests to the wagon in front to speed up until reaching the testing station.

At the loading station, the wagon waits for the ok signal to leave the station, set by Part Tracker. In the current implementation of the Part Tracker, wagons are expected to move out of the loading station (at default speed) even when not loaded (because no pieces are available).

At the testing station, the wagon waits for the ok signal to leave the station, set by Part Tracker. While the Test Station is busy, the first wagon waits in the station and future wagons may queue up behind it (just outside the station).

At any time, the Part Tracker may request any wagon to change speed.

# Test Station

## Short Description

Reads the piece characteristics and reports.

## Communicates with

* Part Tracker

### Internal states

BUSY / FREE

## Actions

On receiving a piece from the wagon, the Test Station reads the characteristics and reports to the Part Tracker. During the testing, the Station outputs BUSY state and does not accept other pieces.

If the Test Station receives a cancellation signal, it interrupts the test count-down regardless of progress and return a random test result.

# Miscellaneous

## Filtering of debug messages

/\*-----------------------------\*\

| 00000 = no debug info

| 00001 = only part tracker

| 00010 = only warehouse

| 00100 = only robot

| 01000 = only wagons

| 10000 = only test station

| 11111 = all debug info

\\*-----------------------------\*/

**public** debugConsoleMsg = 1;

## User Options (for the USB item)

**public** OptionInt = **nat** /\* requested speed of transport in production: value 0 reserved for cancelling the order (requesting speed 0 <=> cancelling order) \*/

**inv** optionint == (optionint <= MAX\_SPEED); /\* speed values \*/

/\* Colour is defined both as int and as string

0 = [0,0,0] = Black;

1 = [0,0,1] = Blue;

2 = [0,1,0] = Green;

3 = [0,1,1] = Cyan;

4 = [1,0,0] = Red;

5 = [1,0,1] = Magenta;

6 = [1,1,0] = Yellow;

7 = [1,1,1] = White;

\*/

**public** ColourInt = **nat**

**inv** colourint == (colourint <= 7);

**public** ColourStr = **<Black>** | **<Blue>** | **<Green>** | **<Cyan>** | **<Red>** | **<Magenta>** | **<Yellow>** | **<White>**;

#### Piece Kind options (for warehouse / robot debug messages)

**public** KindInt = **nat1**

**inv** k == k <= 4; /\*1, 2, 3, 4; consistant with warehouse STAGEs\*/

public KindStr = <Left> | <Middle> | <Right> | <Whole>;

**public** USBpart = OrderID \* KindInt \* ColourInt;

**public** USBitem = OrderID \* OptionInt \* ColourInt \* ColourInt \* ColourInt;

Perturbation levels:

**public** PERTURB\_NONE:**nat** = 0;

**public** PERTURB\_LOW :**nat** = 1;

**public** PERTURB\_MED :**nat** = 2;

**public** PERTURB\_HIGH:**nat** = 3;