Assignment 2

The object of the assignment 2 was to create an agent based system to control Fastory simulator. The DACS methodology was used to define the system.

Specification of the problem

System's control interface is provided by the simulator and is composed of events and commands to it. Events are composed of conveyors' state changes and robots' pen changes, drawing ends, paper loaded/unloaded, pallet loaded/unloaded and ink's state events. Commands are conveyors' movements and robots' pen changes, paper/pallet loading/unloading, drawing commands. Also the states of the conveyors and robots' pen color can be queried.

Mechanical components can be divided to resources, materials and auxiliary components. We determined as resources robots and conveyors. Every workstation has a robot and workstations 1 and 7 have 3 conveyors as the others have 5. As materials we determined frames, screens, keyboards (although they really are only drawings) and paper. Auxiliary equipments are pallets and pens.

Next step was to determine operation conditions. As inputs we qualified colors of frame, screen and keyboard set up by order. Output of the process is a drawing of a mobile phone. As disturbance we saw bugs of the simulator. In simulator we did not recognize any changes that might affect production.

Production goal of the system is to produce a drawing of a mobile phone and deliver it as is determined in an order. As requirements we saw that product should be done as quickly as possible, which in practice means that the drawing should be finished in one cycle of line. This means that a pallet should, after loading, go to get paper and in one cycle of line to finish a product. Then return to deliver the paper and then proceed to get unloaded out of the production line. This means total of 2 cycles in the line for a pallet. Also we as a requirement to minimize pen changes of robots but saw that it should not cause extra cycles in production line for pallets.

Analysis of control decisions

Analysis of control decisions consists of identification of effectoric decisions and decision dependencies. We started with effectoric decisions and at this point we were able to identify following decisions:

Attribute	**Description**
ID	D1
Title	Choose an exit at Z1
Parameters	WS's occupation state
Control Interface	WS *
Trigger	P arrives at Z1
Decision Space	{exit to Z2, exit to Z4}
Local Decision Rule	Choose exit to Z2 if workstation not occupied.

Attribute	**Description**
ID	D2
Title	Choose an exit at Z2
Parameters	state of Z3
Control Interface	WS *
Trigger	P arrives to Z2
Decision Space	{exit to Z3}
Local Decision Rule	Exit to Z3 when Z3 free

Attribute	**Description**
ID	D3
Title	Choose an exit at Z3
Parameters	State o Z5
Control Interface	WS *
Trigger	P wants to leave Z3
Decision Space	{exit to Z5}
Local Decision Rule	Exit to Z5 when Z5 free

Attribute	**Description**

ID	D4
Title	Choose an exit at Z4
Parameters	state of Z5
Control Interface	WS *
Trigger	P arrives to Z4
Decision Space	{exit to Z5}
Local Decision Rule	Exit to Z5 when Z5 free

Attribute	**Description**
ID	D5
Title	Change WS's color
Parameters	Pallet P's desire for color
Control Interface	WS *
Trigger	pallet P asks WS to change color
Decision Space	{green, red, blue}
Local Decision Rule	Change color to desired one

Attribute	**Description**
ID	D6
Title	Work pallet at WS
Parameters	pallet P's desire of part
Control Interface	WS *
Trigger	Pallet arrives to Z3
Decision Space	{not, frame, screen, keyboard}
Local Decision Rule	Work desired part if pallet has an order on WS

Attribute	**Description**
ID	D7
Title	Load pallet
Parameters	
Control Interface	WS 7
Trigger	A new order
Decision Space	{Load}

Local Decision Rule	Load when Z3 free

Attribute	**Description**
ID	D8
Title	unload pallet
Parameters	pallet P's order state
Control Interface	WS 7
Trigger	Pallet arrives to Z2 of WS7
Decision Space	{yes, no}
Local Decision Rule	Unload if P has no order

Attribute	**Description**
ID	D9
Title	Load paper on pallet
Parameters	pallet P's paper situation
Control Interface	WS 1
Trigger	P arrives to Z2 of WS1
Decision Space	{yes, no}
Local Decision Rule	Load if pallet has no paper

Attribute	**Description**
ID	D10
Title	Unload paper from pallet
Parameters	pallet P's paper situation
Control Interface	WS 1
Trigger	P arrives to Z2 of WS1
Decision Space	{yes, no}
Local Decision Rule	Unload if pallet has finished order

Attribute	**Description**
ID	D11
Title	Pallet's next WS
Parameters	pallet P's desire for part, WSs' states

Control Interface	pallet P
Trigger	Pallet has finished its last desire
Decision Space	{WS 1-12}
Local Decision Rule	Go to WS that satisfies pallet's desire best

After identification of decisions, the next step was to identify decision dependencies. We found the following dependencies:

Attribute	**Description**
ID	DP1
Title	Route pallet through WS
Decision Tasks	D1-D4
Constraints	Either 1-4-5 or 1-2-3-5. First only if there is another pallet getting worked at the Z3
Pregerences	

Attribute	**Description**
ID	DP2
Title	Change color of WS
Decision Tasks	D5, change is asked by pallet
Constraints	Change to color that is defined by order of pallet
Pregerences	

Attribute	**Description**
ID	DP3
Title	What part to be made
Decision Tasks	D6
Constraints	Make part that is defined by order of the pallet not an other
Pregerences	

Attribute	**Description**

ID	DP4
Title	Load pallet
Decision Tasks	D7, happens when new order is made in UI
Constraints	
Pregerences	

Attribute	**Description**
ID	DP5
Title	Unload pallet
Decision Tasks	D8, pallet knows if it has an order or not
Constraints	
Pregerences	

Attribute	**Description**
ID	DP6
Title	Load paper
Decision Tasks	D9, pallet knows if it needs a paper
Constraints	
Pregerences	

Attribute	**Description**
ID	DP7
Title	Unload paper
Decision Tasks	D10, pallet knows it has completed drawing
Constraints	
Pregerences	

Attribute	**Description**
ID	DP8
Title	Choose WS
Decision Tasks	D11, pallet use info of itself and info of WS configurations
Constraints	If has not a paper go to WS1, if completed

	product go to WS1, if no order go to WS7
Pregerences	Go to nearest WS but avoid changing color of WS

Identification of agents

From the above identifications we decided that there is need for two kinds of agents. First is workstation agent which controls the interface of simulator. Natural way to distribute the decision tasks were to assign one workstation agent for each workstation. Now there is one agent controlling routing and working of a workstation. Second kind of agent that we identified was pallet agent that's work is to identify desired destination workstations for pallets. Natural way to distribute the decision tasks were to assign one pallet agent for one pallet. In total the system was seen to compose of 12 workstation agents and equal amount of pallet agents as there are orders to products.

Selection of interaction protocols

As we had proceeded at this face, we noticed that the workstation agents could be purely reactive. All they needed to work were the sensor information and knowledge if it had order, from who it was and what part was to be made. Therefore the real need for interaction protocol was for the pallet agent that it could determine the next destination and make a reservation for the workstation. And this can be restricted to drawing workstations because there is no need for reservation of loading or paper loading workstations, because there are only one of both, which means that there is nothing to negotiate.

After a thought, we came to a conclusion that the negotiation protocol should be based on conflict resolution. Pallet agent should evaluate the nearest workstation and if a threshold is not passed then the robot should start evaluate the second nearest and so on. If threshold is not passed with any workstation agent, the pallet agent should lower its expectations and start a new round of evaluations. This protocol enables the pallet agent to choose the nearest workstation suitable, which serves the desire to finish product in one cycle of line. Also we came to a conclusion that the number of workstation to negotiate with should limited according to "hardness" of the order. If all parts were to

be same color, then the order was easiest and should be made in one workstation to minimize pen changes and there ought not to be any limitation which station it was. Hardest order was in which the colors successively were all different. This, we thought, should limit the possible next workstations to closest ones to ensure that the product is drawn in one cycle.

Description of programs

We implemented 3 programs. One for user interface, one for pallet agents, which also has pallet manager that's mission is to handle creation of new pallet agents. Third program was for the workstation agents. Next let's describe all the programs.

The implementation was started by creating a user interface (UserInt.js). Interface should allow making of order and inform if the order was valid. The interface is presented in figure 1.

Figure 1. *UI*

User interface expects as color value either "blue", "red" or "green"; as frame value "1", "2" or "3"; as screen value "4", "5" or "6" and as keyboard value "7", "8" or "9". If the

order is valid, then “Valid Order!” message appears. If values are invalid then message will inform of error and what part of the order was invalid.

Next program was pallet agent and manager (palletManager.js). Pallet manager is handling new orders coming from user interface. It also handles creation of new pallet agents in an array. It expects information of the created pallet’s ID from workstation 7 agent and then creates a pallet agent with right ID. Although, the manager checks first that an agent with the ID has not already been created. If is, then it only updates the old agent. Manager also handles destination and paper status requests from workstation agents 1 and 7. Pallet agent is an object that is presented in figure 2.

```
var Pallet = function(id,fr,fr_c,scr,scr_c,kb,kb_c, pt){
    this._id = id;
    this.hardness = 0;
    this.hasPaper = false;
    this.destination = 1;
    this._frame = fr;
    this._f_colour = fr_c;
    this._screen = scr;
    this._s_colour = scr_c;
    this._keyboard = kb;
    this._k_colour = kb_c;
    this._port = pt;
}
```

Figure 2. *Pallet object*

In figure 2, everything should be self explaining except hardness. Hardness is in this case factor that presents the hardness of manufacture. If all colors are same, then hardness factor is 1, if screen’s colour is same with either of them but not with both then factor is 2 and if there are no same successive colours then the factor is 3. The hardness factor affects the number of workstations the pallet agent is free to negotiate with for frame and screen. For keyboard negotiation it has no effect because the keyboard is enough to get drawn in any of the remaining workstations before workstation 1. Pallet agent server is supposed to handle the following requests made with post method: “PaperLoaded”, “PaperUnloaded”, “Framed”, “Screened” and “Keyboarded”. PaperLoaded request changes pallet’s hasPaper value to true and triggers negotiation with workstations for drawing a frame. PaperUnloaded request changes hasPaper value back to false and sets destination to 7 (WS). Framed request informs the pallet that frame has been made and triggers negotiations for drawing a screen. Screened request

informs the pallet agent that screen has been drawn and triggers negotiations for drawing a keyboard. Keyboarded request informs pallet agent that keyboard has been drawn and sets destination to (WS) 1 to deliver the drawn paper.

Third program is for workstation agents (WS.js). Workstation object is in figure 3.

```
var WS = function(number){
    this._number = number; //Number of the ws
    this.Busy = false; // Flag for working ws
    this.Order = false; // Flag for reserved ws
    this.ordererID = 0; // Pallet ID of the one that has
    this.ordererPort = 0; // Pallet agent's port nubmer
    this.orderedPart = 0; // Desired part to manifacture
    this.Colour = 'RED'; // Colour of ws
    this.Z1 = -1; // Pallet ID currently at Z1
    this.Z2 = -1; // Pallet ID currently at Z1
    this.Z3 = -1; // Pallet ID currently at Z1
    this.Z4 = -1; // Pallet ID currently at Z1
    this.Z5 = -1; // Pallet ID currently at Z1
}
```

Figure 3. *WS object*

Figure 3 should be self explaining. For WS agents there are three different kinds of servers. One kind for WS 1, other for WS 7 and third kind for rest. Server for WS 1 is supposed to handle following requests of post method: Z1_Change, Z2_Change, Z3_Change, Z5_Change, PaperLoaded, PaperUnloaded, ForceMove. Z1_Change request makes the WS agent to refresh its Z1 value and if change is for arriving pallet then this also triggers a request to simulator to move pallet forward to Z2. Z2_Change works as Z1 except it naturally moves pallet to Z3. Z3_Chance triggers the agent to request destination and paper information of the arrived pallet and if it is need of paper or removal of paper, the agent commands simulator to do desired. Z5_Change refreshes the Z5 value. PaperLoaded abd PaperUnloaded triggers the agent to inform pallet agent that its paper status has been changed. ForceMove triggers the agent to command pallet movement from Z3 station and is used after pallet has got paper (un)loaded and has desire to leave workstation.

WS 7 agent has the same requests to handle as WS 1 except that PaperLoaded and PaperUnloaded are replaced with PalletLoaded, PalletUnloaded and LoadNew. PalletLoaded request triggers the agent to send the pallet ID of created pallet to pallet

manager. PalletUnloaded has no role, because subscription to the event does not work. LoadNew is request from pallet manager to trigger loading of new pallet to line.

Servers for other workstations handle Z*_Changes and furthermore requests for DrawEndExecution, ChangeColour, PenChanged, Colour, hasOrder, Order and ForceMove. DrawEndExecution is request from simulator to mark end of drawing and triggers Framed, Screened or Keyboarded request to pallet agent depending on drawn part. ChangeColour is request from pallet agent and triggers request to change pen to simulator. PenChanged is request from simulator and triggers the agent to refresh its colour value. Colour is request from pallet agent and WS agent response the color value of the robot. HasOrder is a request from pallet agent and triggers a response whether workstation already has an order. Order is a request from pallet agent to reserve the workstation and triggers the WS agent to refresh the orderer* values to values of the pallet agent given in the order.

One day in the life of a order and interactions between agents

Let's take a look at a life of an order from ordering and to finally unloading of the pallet that delivered the order. At the same time we check out interactions by using sequence diagrams. The story begins from UI, where user writes a valid order and this is seen at the simulator as a loaded pallet. The interaction is presented in figure 4.

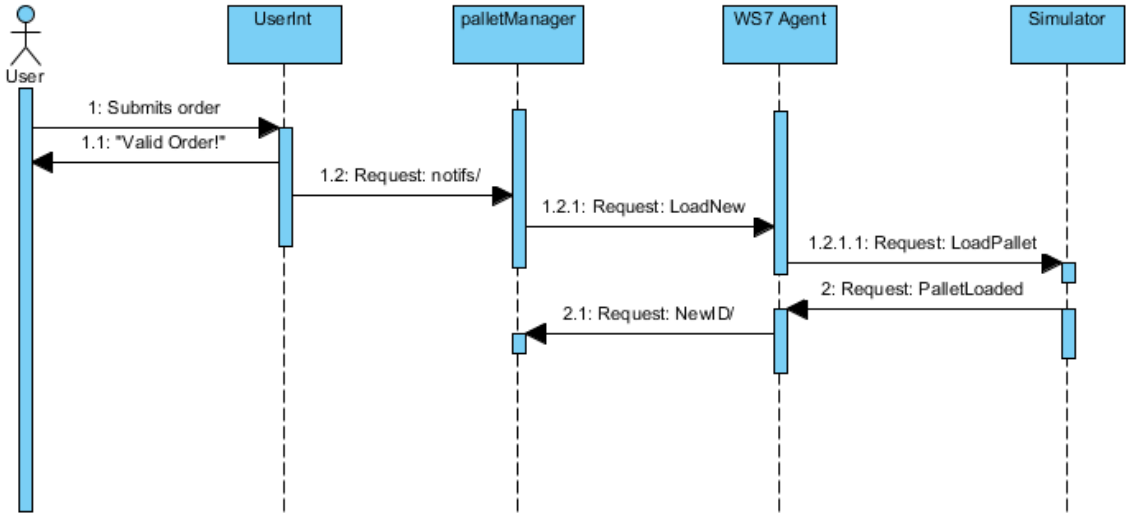

Figure 4. *Pallet load*

At the end of this interaction the pallet agent is created to handle the pallet. The pallet agents destination is set at 1, which it heads. The normal movement interactions are not presented here because they only consists of request Z*_Changed to WS agent, which in turn requests TransZone** from simulator. When pallet arrives to WS 1 the following interaction happens:

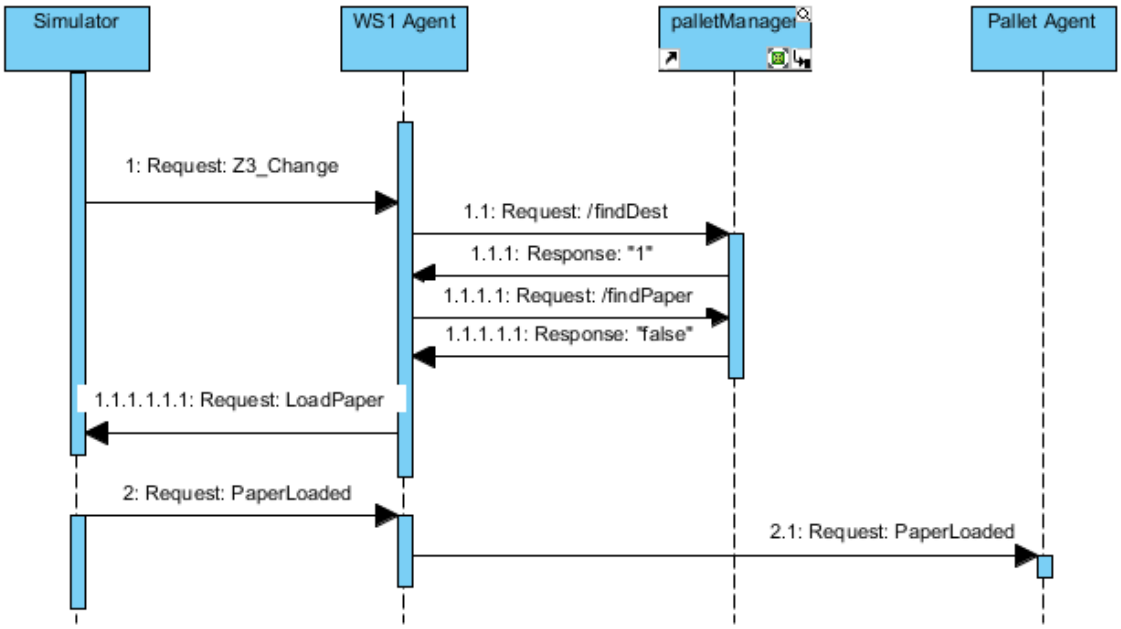

Figure 5. *Paper load*

At this point pallet agent change hasPaper value to true and this triggers the first negotiation for drawing a frame. The first round of conflict resolution interaction is presented in figure 6.

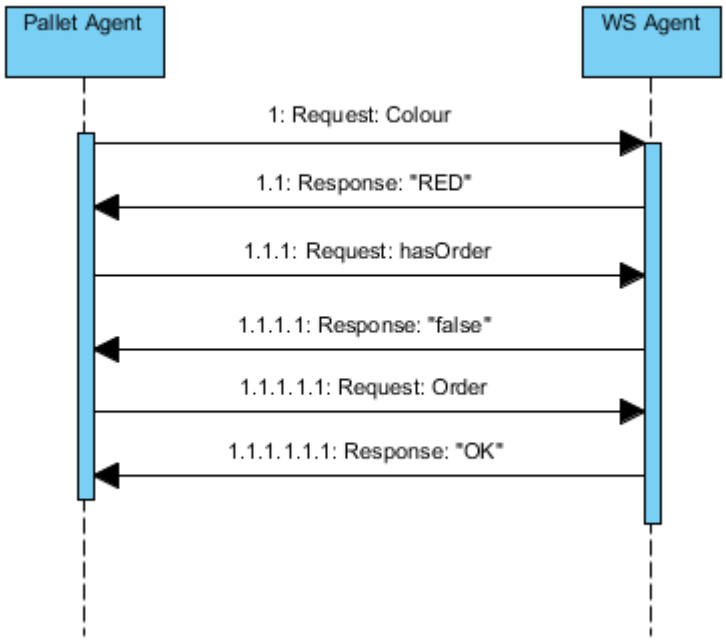

Figure 6. *First round of evaluation*

In plain language the negotiation goes as follows: pallet agent asks color of robot; if response is same as the desire of pallet agent then it asks if workstation is free of order; if workstation has no order then pallet agent makes order to reserve workstation for itself. If any of the responses differ it means that pallet agent stops negotiation and continues to negotiate with the next one. Pallet agent amount of workstation to negotiate with is limited according to its hardness value. If hardness is 1 then it will negotiate with all workstation agents, if 2 then will negotiate to WS 9 and if 3 then it will negotiate to WS 4. If the first round of negotiation fails then pallet agent lowers it expectations and start a new round of negotiations as presented in figure 7.

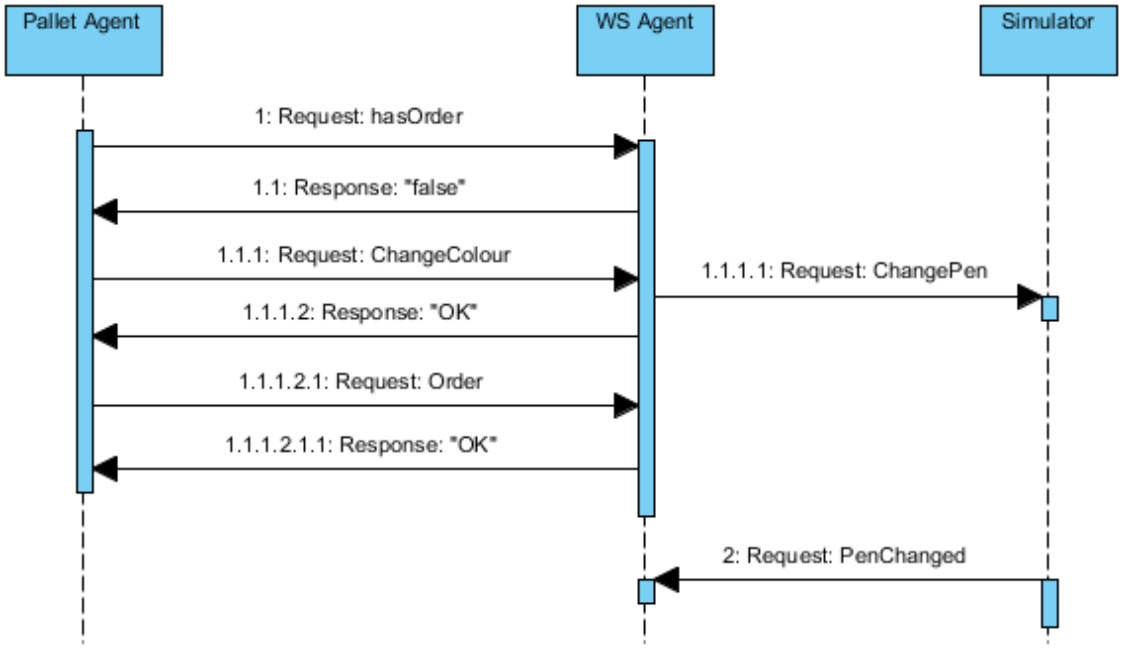

Figure 7. *Second round of evaluation*

In plain text the negotiation goes as follows: Pallet agent asks if WS is free; if is, then it requests for color change and continues to reserve WS for itself. If any response differs, then pallet agent starts to negotiate with next WS agent and so on.

After order request the pallet agent changes its destination to workstation in question and WS agent change its Order status to true and is unable to take a new order until the pallet has been served. After pallet reaches the destination workstation the following interaction occurs to get the job done.

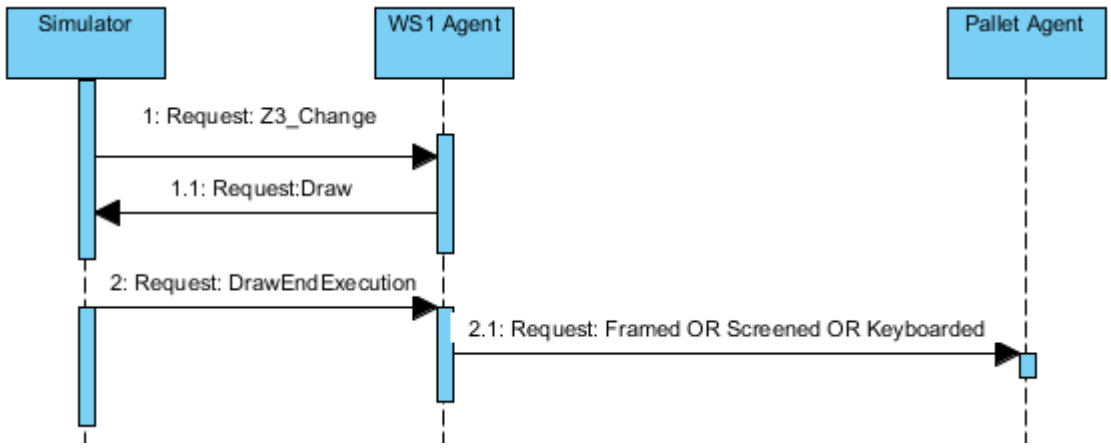

Figure 8. *Drawing*

Naturally, in figure 8 the draw request is made only after the pallet in Z3 is known to be the orderer. In this case the request to pallet agent would be “Framed”. This would first be checked by pallet agent to be correct drawing and then it would trigger new negotiations for screen. Negotiations would go as earlier except in case the screen color is same as color of frame. If this is true then pallet agent makes a new order to workstation in which it is and there are no negotiations at all. After finishing screen the request for pallet agent is “Screened” (figure 8). This triggers negotiation for keyboard. First round of negotiations are as in figure 6. Difference is that there is no second round of negotiations, but pallet agent requests ChangeColor and Order straight from the workstation agent that it is in. In this case request for pallet agent is “Keyboarded” (figure 8), which sets pallet’s destination to 1. After arriving to WS 1 interactions of figure 9 happen.

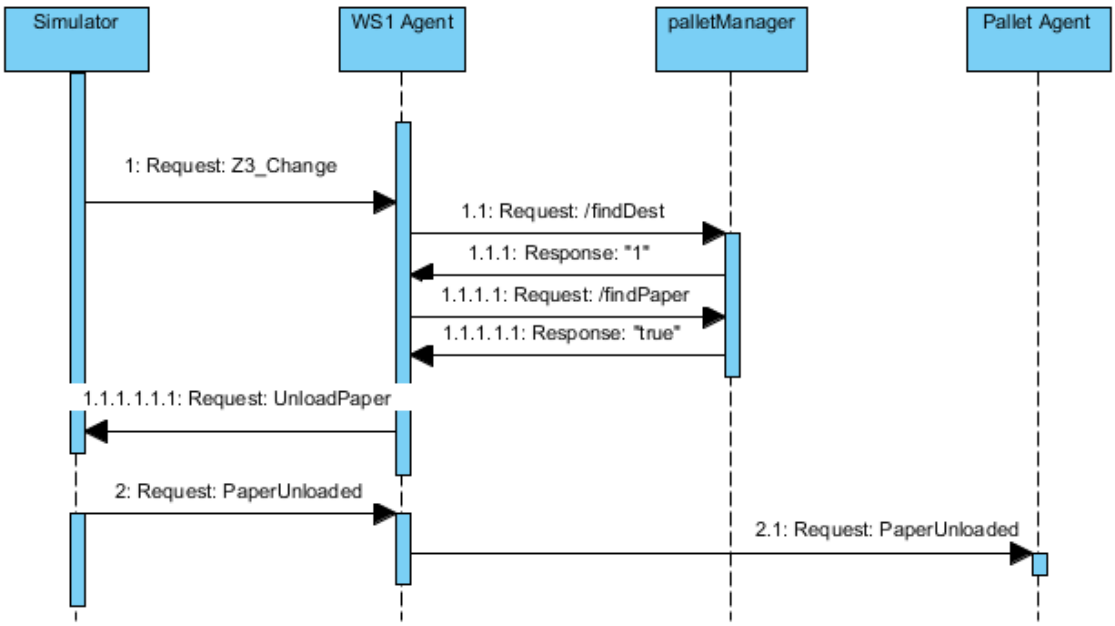

Figure 9. *Paper unload*

Request PaperUnloaded to pallet agent triggers the agent change its destination to 7. After getting to WS 7 interactions of figure 10 happen.

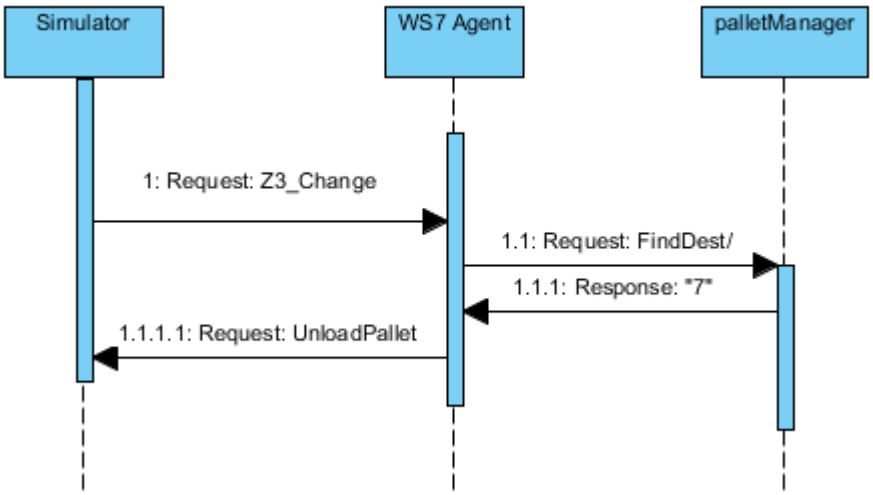

Figure 10. *Pallet unload*

Because event PalletUnloaded does not work the information of pallet unloading does not get into pallet manager. On the other hand this is not a problem after simulator reuses pallet IDs, so a new pallet will inherit the old agent. This is the life of an order.

Involvement of groupies

Involvement was equal.