Tampere University of Technology

ASE-9426 Distributed Automation System Design

Assignment on

Multi Agent Systems

Prasun Biswas
267948
prasun.biswas@student.tut.fi

Salman Azim
245038
salman.azim@student.tut.fi

Shambhu Mishra
267600
shambhu.mishra@student.tut.fi

Introduction

The objective of this assignment is to understand the principle of Multi-Agent Systems(MAS). The knowledge of MAS was studied and implemented on the FASTory line of Tampere University of Technology. This line consists of 12 Robot stations each having their own conveyor, different zones, stopper for stopping a pallet, sensors for sensing a pallet and RFIDs for pallet recognition. This line is used for drawing mobile phones on a paper. It has different options and color choices to choose from having 729 variations (3 screen shapes* 3 colors * 3 keyboard shapes* 3 colors * 3 frame shapes* 3 colors). A simulator interface of the FASTory line was provided which mimics the original line. This interface is very helpful for study and research purpose. Multi-Agent System is the most researched topic in the field of Computer Science. Applying the MAS principle on FASTory included identification of the Agents, designing their behavior, providing autonomy to the system, and establishing communication between the agents.

Tools & Environment

- Webstorm IDE for programming
- Node.js framework for Backend
- Plain HTML as the UI for taking orders
- Visual Paradigm Community Edition for UML Sequence Diagram
- FASTory Simulator Interface for testing the solution

Multi-Agent System

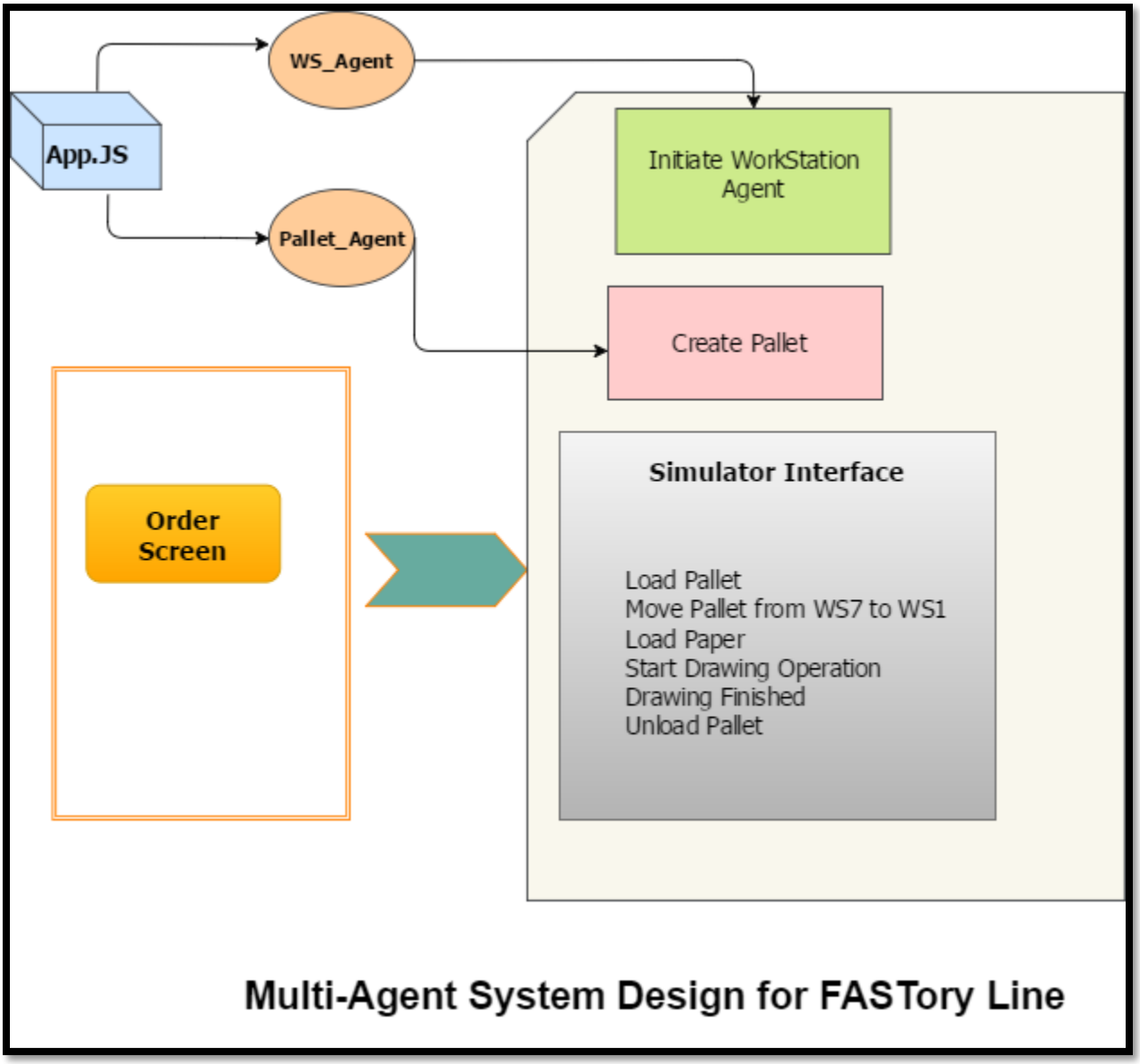
Multi Agent System consists of multiple agents and their environment. Every agent in the society can do some work autonomously without any human interaction. They work to fulfil the goal of the whole society. One agent's function is unknown to other agents and they can only communicate between them about their status. Agents can interact directly or indirectly. Indirect process meaning acting on its environment and direct meaning communicating through messages and protocols. Agents can have conflicting goals. They can negotiate between them and can decide autonomously for completion of the goal of the whole system. A multi agent system can also contain human agents. An agent can be of different types. Active agents with simple goals, Passive agents without any goals and Cognitive agents with complex algorithm. Agents can be said to have these abilities.

1. Autonomous
2. Social
3. Reactive
4. Proactive

Application of MAS on FASTory:

In our approach to solve the assignment with multi agent system we considered two type of agent's workstations are active agent. Here a workstation, which has robotic hand and conveyer belt that can operate independent of each other, are one entity. Every workstation from WS1 to WS12 has some capability and only perform within their limit. Pallets are considered passive agent, because pallet itself doesn't communicate with each other, whereas the workstation agent are capable of communicating with their next neighbor agent.

Our approach to the design of the solution looks like this:



Pallet Agent:

Every pallet has a pallet ID that make it possible to identify in the factory line. This palletID is generated while a pallet is loaded on FASTory line. It has some property which are assigned from customer order, those are frame type, screen type, key type, frame color, screen color, key color. It also has a property named status which indicate if the work on pallet is completed by robot. The last property of pallet agent is path, which is an array that contains the name of required workstation which has the same capability as the work to be done on the paper.

Workstation Agent:

Every workstation has a capability. In our assignment, we declared the capabilities as color for WS2, WS3, WS4, WS5, WS6, WS8, WS9, WS10, WS11, WS12. Each or the workstation is capable of drawing all the parts, that is frame, screen and key, but only of one color for each workstation. WS7 is capable of loading and unloading pallet in the system. Capability of WS1 is to load paper on the pallet. This agent is also ability to unload the paper from pallet but we didn't use that feature in our assignment as we are unloading the whole pallet with paper after all the parts are drawn on it.

Description of Solution:

WS7 is the first workstation that start working at the beginning of the process. The system starts with running app.js file. The other Javascript files are WS_agent and Pallet_agent which are exported and can be accessed through app.js file. When app.js file runs, it creates an express server which follows http protocol and it listens to localhost on port: 4007. This is also the common for WS7, which has capability of loading and unloading pallet, and the workstation ready for communication in this moment.

```
app.listen(port, hostname, function(){
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

When app.js file is running all the other workstations are initiated as objects of commonly defined prototype, and ready for interaction, that is receiving commands, deciding its job, communicating with neighbor workstations, and executing commands.

The html file named order.html has form that can post its various value to the localhost:4007 on app.post(/spec). It sends the required specification for mobile which is posted after user clicks submit button.

```
<form target="_blank "action="http://localhost:4007/spec" method="post">
    <h1> Place Your Mobile Phone Order</h1>
```

Upon submission of the form the specifications of mobile phone are initiated as global variable. At the same time a command is posted to "/WS7notifs" to load a pallet in the FASTory line and a new pallet object is created according to definition of pallet agent.

```
var pallet = new Pallet_Agent(palletID,oFrame,oFrame_color,oScreen,oScreen_color,oKeyboard,oKeyboard_color,0);
//  var pallet = new Pallet_Agent(palletID,1,'BLUE',4,'RED',7,'GREEN',0);
 pallet.setPath(searchCapability(pallet.getFrameColor(),pallet.getScreenColor(),pallet.getKeyColor()));
 setPallet(pallet);
```

Just after a pallet is loaded with certain parameter, "setPath" method is called to insert path as an array to go through workstations, which takes a function "searchCapability" as parameter. This function take 3 parameter frameColor, screenColor, keyColor, checks which workstations has these capabilities get the names of workstations and push to insert in to array. Now this path is assigned as a property of the loaded pallet. Then "setPallet" method sets the value of the object to a variable currentPallet.

```
function simRequest(url) {
    request({
        url: url,
        method: "POST",
        body: JSON.stringify({destUrl:'http://hostname'}),
        headers:{'Content-Type':'application/json'}
    },function (err, res, body) {});
}
```

```
function palletRequest(url) {
    request({
        url: url,
        method: "POST",
        body: JSON.stringify(getPallet()),
        headers:{'Content-Type':'application/json'}
    },function (err, res, body) {});
}
```

This workstation is capable of sending the pallet to the next workstation or unloading the pallet if work is finished on this with “simRequest” method and sending the pallet data to the next workstation with “palletRequest” method. When the properties of the currentPallet is sent to the next station the “resetPath” method is called to clear any value for frame, screen, and key in the path array and only the WS1 is set as the current value.

After leaving from WS7 pallet enters to WS8 and proceed to WS6 after passing through all the workstation where except for WS1 the logic of operation is same for all the workstations. After entering in the WS8 it generates an event with id, senderID, from which a variable is generated using “substr” method to find out only the numerical value of sender (1-12). If the destination status is not busy and the payload.PalletID is showing “not leaving” status, the pallet is allowed to enter the workstation and upon entering using “setStatus” method the current state is set to busy.

```
case "Z1_Changed": {
    if((getStatus() == "free")||(WS_ID=='WS1')){
        if (req.body.payload.PalletID != -1) {
            setStatus("busy");
            console.log(WS,'Z1changed',currentPallet, getStatus());
            var path = currentPallet.path_[0];
            if(currentPallet.path_.length!=0 ){
                for(var i=0; i<path.length; i++){
                    destination[i] = path[i];
                }
```

It also gets all the property for the currentPallet. if the path is not empty it sets the variable destination using the path array that were generated during insertion of the pallet in the system.

If WS_ID is WS1 the using javascript built-in array shift method the value of element inside array is shifted one place and request to transfer the pallet from zone 1 to zone 2 is posted. if the event is “Z2_changed” and payload.PalletID status is not leaving then the next command is posted to transfer the pallet to zone3. Here also depending upon the capability of the stations and required work to be done on the pallet it allows to pallet to enter to the zone 2 of the workstations.

```
switch (palletStatus){
    case 0 : {
        url = 'http://localhost:3000/RTU/SimROB1/services/LoadPaper';
        simRequest(url);
        break;
    }
    case 1: {
        var frameType = currentPallet.frameType_;
        url = 'http://localhost:3000/RTU/SimROB'+WSnum+'/services/Draw'+frameType;
        simRequest(url);
        break;
    }
    case 2: {
        var screenType = currentPallet.screenType_;
        url = 'http://localhost:3000/RTU/SimROB'+WSnum+'/services/Draw'+screenType;
        simRequest(url);
        break;
    }
    case 3: {
        var keyType = currentPallet.keyType_;
        url = 'http://localhost:3000/RTU/SimROB'+WSnum+'/services/Draw'+keyType;
        simRequest(url);
        break;
    }
    default:{
        palletRequest(WS_Neighbour,currentPallet);
        url = 'http://localhost:3000/RTU/SimCNV'+WSnum+'/services/TransZone35';
        simRequest(url);
    }
}
```

When the workstation event is “Z3_changed” and payload.PalletID status is not zone leaving the here some case checking take place. If palletStatus is 0 the command to load pallet is execute by “simRequest” method. Depending on the match of frametype, screedtype, keytype required command is posted to draw as necessity.

If the event is “Z4_changed” the using “getPallet” method variable currentPallet is set to the new status after execution. Also, using “palletRequest” function which takes WS_neighbour, currentPallet as argument pallet property is sent to the next workstation with currentPallet properties.

If the event is “DrawEndExecution” the depending on the case currentPallet.FrameType_ is changed to (done +recipe) which indicates that a process has been done on the paper as per requirement.

```
case "DrawEndExecution":{
    var recipe = parseInt(req.body.payload.Recipe);
    switch (recipe){
        case 1:
        case 2:
        case 3:{
            currentPallet.frameType_ = "done" + recipe;
            break;
        }
        case 4:
        case 5:
        case 6:{
            currentPallet.screenType_ = "done" + recipe;
            break;
        }
        case 7:
        case 8:
        case 9:{
            currentPallet.keyType_ = "done" + recipe;
            break;
        }
    }
    currentPallet.status_++;
    path = currentPallet.path [0];
```

After execution of the process currentPallet.status_ is increased by 1 each time. In this case if all requirement is done on a pallet the currentPallet.status will be 4, which indicate paperloaded and frame, key, screen is drawn on the pallet.

```
            var screenStat = currentPallet.screenType_.toString();
            var keyStat = currentPallet.keyType_.toString();
            if((screenStat.substr(0,4) == "done")&&((keyStat).substr(0,4) == "done")||(keyStat).substr(0,4) == "done"){
                url = 'http://localhost:3000/RTU/SimCNV'+WSnum+'/services/TransZone35';
                simRequest(url);
            }
        }
        else{
            url = 'http://localhost:3000/RTU/SimCNV'+WSnum+'/services/TransZone35';
            simRequest(url);
        }
        palletRequest(WS_Neighbour,currentPallet);
        break;
```

Every status ex: screenStat, keyStat if done the command to move to zone5 is executed and using “palletRequest” function WS_neighbour, currentPallet is sent to the next workstation. This process is repeated throughout the whole process. When pallet reaches to WS7 if the currentPallet.status_ is 4 the command to unload the pallet from is executed using “simRequest” function.

In this code one of the most used method is request.post

```
request.post('http://localhost:3000/RTU/SimROB7/events/PalletLoaded/notifs',{form:{destUrl:"http://localhost:"+port+"/WS7notifs"}}, function(err,httpResponse,body){});
request.post('http://localhost:3000/RTU/SimROB7/events/PalletUnloaded/notifs',{form:{destUrl:"http://localhost:"+port+"/WS7notifs"}}, function(err,httpResponse,body){});
```

Where the first url is for subscription to a notification in an event of any workstation and second url is the destination where the data received from the notification is sent to.

Participation

Through the whole process, we worked as a group. As the process was complex participation and intellectual involvement of every group member was necessary. To complete the whole assignment, it was a rough journey in different step we faced lots of obstacle that we had to solve by lots of trial and test run.

Name & Contribution (%)	Procedures
Salman (34), Prasun (33), Shambhu (33)	Planning
Salman (30), Prasun (50), Shambhu (20)	Algorithm Development
Salman (30), Prasun (50), Shambhu (20)	Coding
Salman (35), Prasun (35), Shambhu (30)	Testing & Troubleshooting
Salman (30), Prasun (30), Shambhu (40)	Reporting