Multi Agents Systems Approach for Bicycle Production
CSCK504: Multi-agent Systems
Munpreet Doorgah, Muhammad Ali and Samuel Otoo

This report details the development of a multi-agent system (MAS) approach to assembling a bicycle frame consisting of various components (Head Tube, Top Tube, Down Tube, Seat Tube, Seat Stay and Chain Stay). The primary objective is to both efficiently and autonomously weld the frame together with the aim of cutting costs for the bicycle company. The design of the MAS will be analysed and explained with the aid of code, pseudocode and various diagrams. Furthermore, shortcomings of the MAS design with possible improvements will also be discussed to provide a comprehensive critique.

1. Introduction

Multiagent systems are defined as systems that consists of multiple computing elements known as agents which are capable of autonomous action to some degree and are able to interact with other agents (Wooldridge, 2009). The focus of this literature review will be on problem solving in benevolent agents known as **Cooperative Distributed Problem Solving (CDPS**) and a task sharing protocol called **contract Net**; CDPS is essentially the study of how loosely coupled network of problem solvers can work together to achieve goals/solve problems that are beyond their individual capabilities. Each problem-solving node in the network is capable of sophisticated problem-solving with the ability to work independently but the problems faced by the nodes cannot be completed without cooperation (Wooldridge, 2009). Contract net paradigm is an example of loosely coupled distributed problem solving where a heterogeneous group of agents negotiate task sharing (Hsieh, 2006).

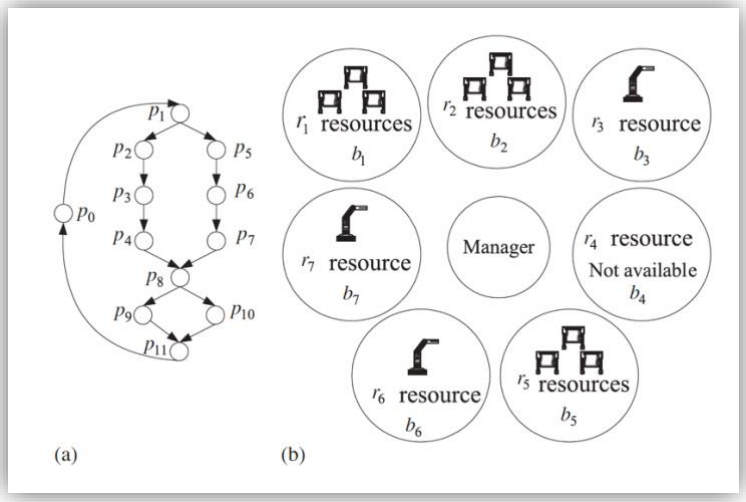

Figure 1: Tasks and resources in a contract net model (Hsieh, 2006).

There are five stages of a contract net protocol: Recognition, Announcement, Bidding, Awarding and Expediting (figure 2) (Wooldridge, 2009). **Recognition** stage involves an agent realising that it has a problem which requires additional help, this could be due to a

complex goal or a preference not to work in isolation based on compromised quality of output. **Announcement** stage involves the agent with the task sending an announcement of the task along with specifications such as description of the task, applicable constraints and meta-task information. Once the announcement is broadcasted, **Bidding** is initiated amongst agents wishing to participate. The factors for bidding involve capability of agents to expedite the task along with determining the quality constraints and price information if deemed relevant. The agents choosing to bid then submit a tender. Once the tenders are submitted, the task announcing agent must choose between bids and award the contract. The result is subsequently communicated to bid submitting agents and the successful contractor is required to expedite the task.

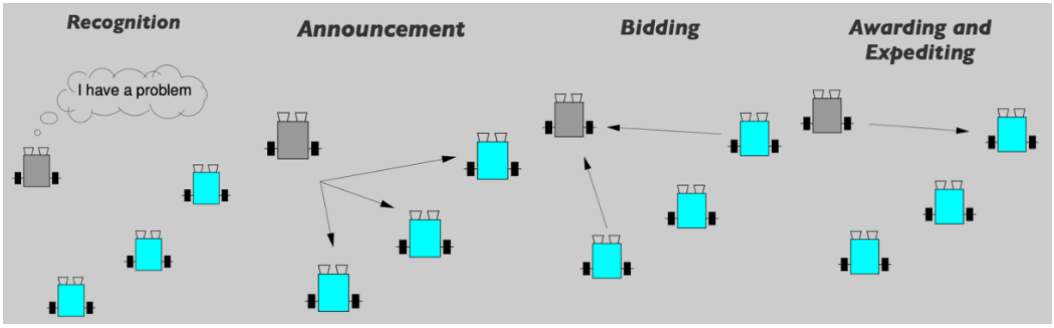

Figure 2: Different stages in a contract net model (Wooldridge, 2009).

Figure 3 shows the implementation of contract net protocol in AgentSpeak with six agents consisting of one contractor, three fully participating agents, one refusing agent and one silent agent.

```
1.  MAS contractNetProtocol {
2.        infrastructure: Centralised
3.
4.   agents:
5.        contractor          // The CNP Initiator
6.             [mindinspector="gui(cycle,html,history)"];
7.        participant #3;     // The 3 service providers
8.        refusenik;          // Participant who always refuse
9.        silentpartner;      // A Participant that doesn't answer
10.
11.  aslSourcePath:
12.       "src/asl";
13. }
```

Figure 3: Example of contract net in AgentSpeak (Wooldridge, 2009).

The assignment brief requires six components welded together as shown in figure 4 in order to form the bicycle frame. The following specialised agents are utilised for the proposed contract net protocol of this project:

- *Initiator Agent (IA)*: The manager of the contract net protocol which handles interactions between different types of agents in the system.
- *Robotic Arm Agent (RA)*: Agent responsible for picking a single piece out of bin and moving it in to position on the welding surface.
- *Welding Agent (WA)*: Agent tasked with welding joints.
- *Holding Agent (HA)*: Agent responsible for holding. A piece firmly in place on the assembly surface.
- *Moving Agent (MA)*: Agent tasked with moving the entire bicycle safely away from the assembly surface.

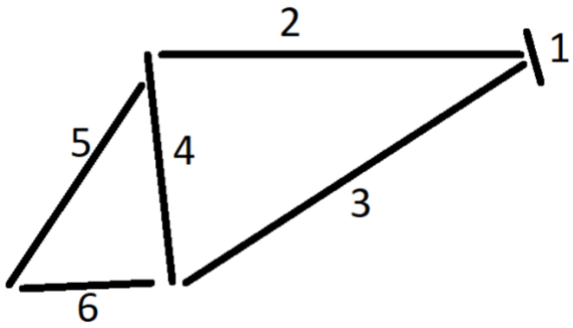

Figure 4: Arrangement of components in bicycle's frame.

After an extensive literature review, contract net protocol (CNP) was selected for implementation in this project; both (Beer, et al., 1999) and (Hsieh, 2006) argue that the application of CNP and MAS in manufacturing systems has been extensively studied with uses ranging from modelling a factory to a market-driven contract net protocol where agents are connected between the shop floor and the marketplace.

2. System Design and Configuration

In this section, MAS system design for assembly of bicycle frame will be explained in detail with the aid of snippets of pseudocode; class and sequence diagrams will also be included. Complete code for the MAS implementation (5 ASL files and 1 java file) will be attached separately.

The proposed MAS system uses a centralised architecture which consists of agents and the environment interacting with one another and is executed on top of an execution platform running on a single processor such as Java VM (Mertens & Holvoet, 2005).

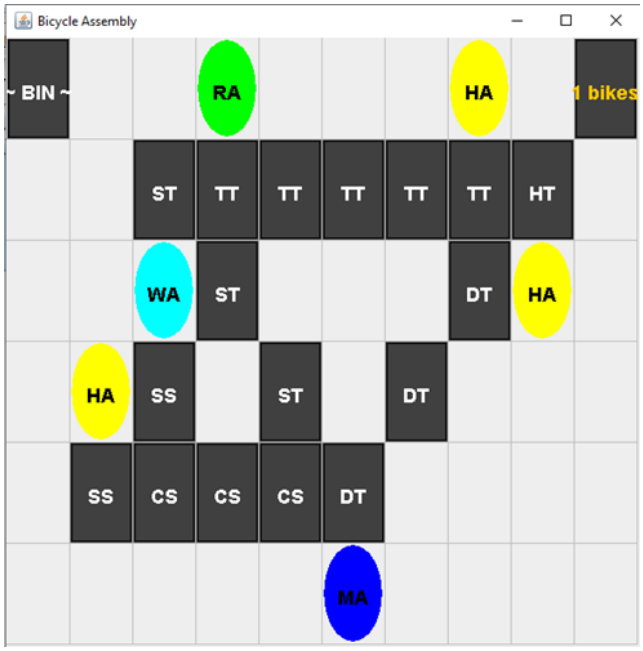

Figure 5a: MAS environment for Bicycle Assembly.

The MAS runs on a ProductionEnv environment (figure 5) configured as a 10 x 6 grid.It consists of the following elements:

1. Bin: positioned at a fixed point (0,0).
2. Assembly/Welding surface.
3. Bike Stash: A fixed location (9,0) to store assembled bicycle frames.
4. Agents: RA, HA, WA and MA (introduced in the previous section).
5. Bike Parts: HT (Head Tube), TT (Top Tube), DT (Down Tube), ST (Seat Tube), SS (Seat Stay) and CS (Chain Stay).

As shown in figure 5b, we are using the GridWorldModel class, provided by Jason API; this class uses the GridWorldView which allows us to model the bicycle production environment as a 10 x 6 graphical UI grid where each slot may contain either static objects or agents. The environment, i.e., ProductionEnv, defines the literals and terms to be used by the agents while the model, ProductionModel, implements one method for each possible action in the MAS.

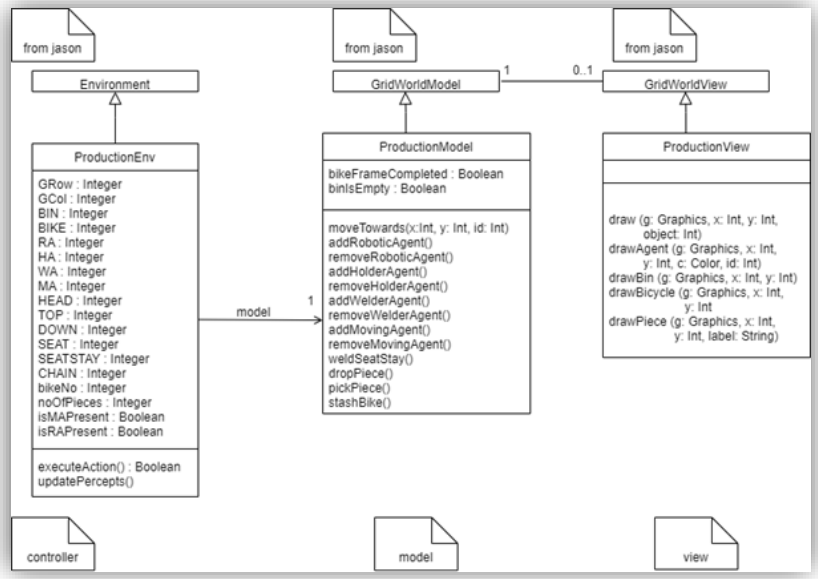

Figure 6b: Java Environment Class Diagram.

Shown below is the architectural diagram showing the communication between the various agents with the Initiator being the intermediate agent interacting with all agents in both directions.

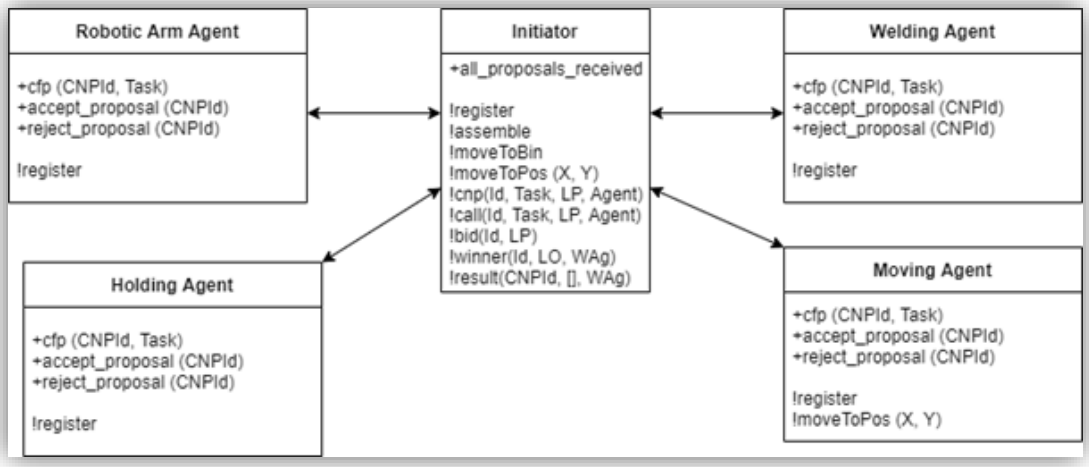

Figure 7c: Agent Architecture Diagram.

The environment and agents are integrated together to produce the entire MAS system in a coherent and logical flow of processes. Figure 5d is the sequence diagram illustrating the main tasks in the Bicycle Production MAS. Further details on the functioning and design of the system, including pseudocode, has been given in the upcoming subsection.

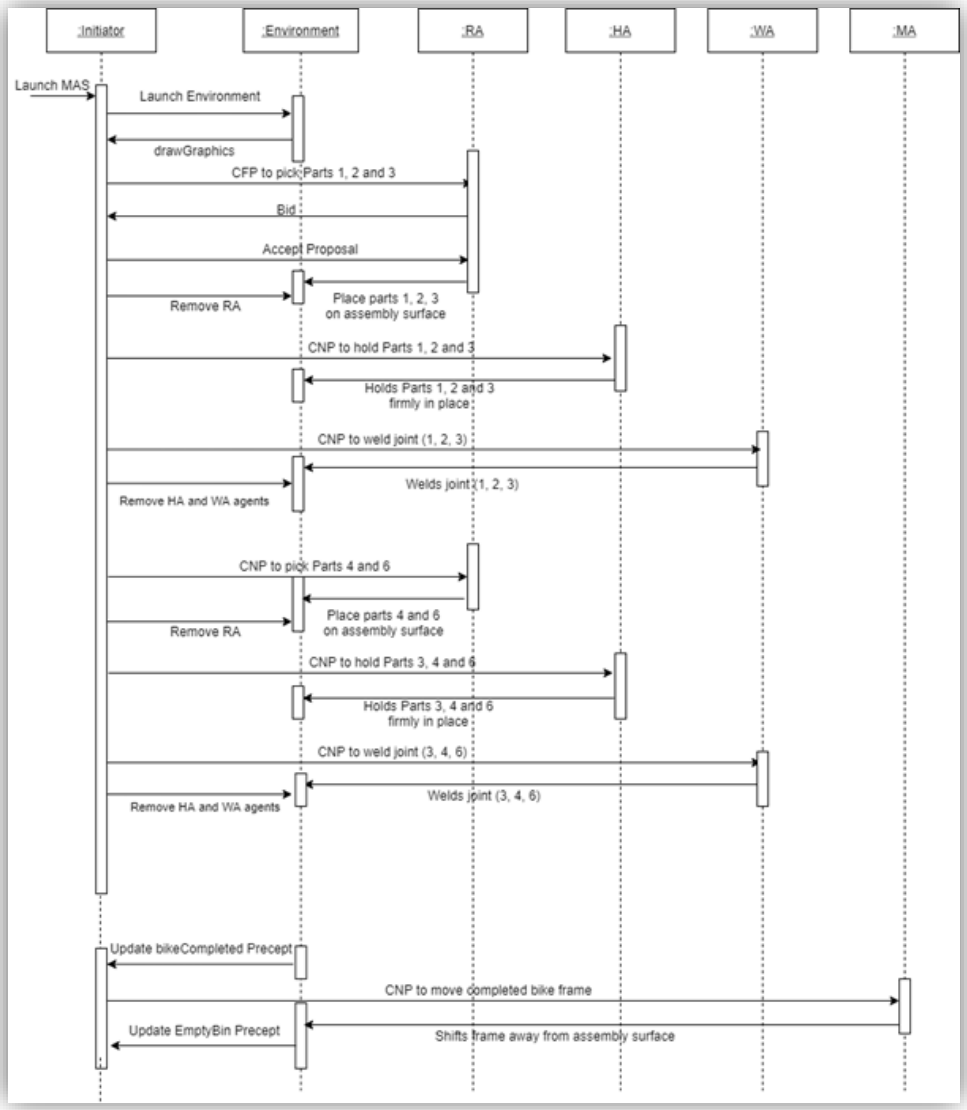

Figure 8d: MAS sequence Diagram

Agent	Registered name in DF
Initiator	initiator
RA	robotic_arm_agent
HA	holding_agent
WA	welding_agent
MA	moving_agent

Figure 9: Agents and their respective names in Directory Facilitator.

At various points throughout the system, a CNP request is established which includes the following steps:

1. *The IA sends a Call for Proposal (CFP) to all participants of a specified agent type registered in the Directory Facilitator (DF) and subscribed to the initiator* (figure 6).

For each agent, there is a separate ASL file in which the register action is used to register each agent to the DF using the assigned name as shown in figure 6. The initiator agent is also registered in the DF so that other agents may subscribe to it.

```
// Register the Initiator Agent to the Directory Facilitator

ACTION register

        // Initiator should be registered for other agents to be able to subscribe to it

        REGISTER "initiator" to Directory Facilitator

END ACTION
```

Figure 10: Registering the initiator agent.

Subsequently, RA agent is also registered in the DF as shown in figure 8.

```
// Register the Robotic Arm Agent to the Directory Facilitator

ACTION register

        REGISTER "robotic_arm_agent" to Directory Facilitator

        SUBSCRIBE TO "initiator"

END ACTION
```

Figure 11: Registering the Robotic Arm agent.

Lastly, CFP are sent to selective agents as seen in figure 9.

```
// Call for CNP Proposal
// Under all circumstances
ACTION call (Id, Task, LP, Agent)
        PRINT "Waiting participants for task " Task
        WAIT(2000)                                  // Time delay to await participants
        // Look for Agent in the Directory Facilitator using its registered name
        CALL ACTION df_search (Agent, LP)
        PRINT "Sending CFP to " LP
        // Send Call for Proposal to available agents
        SEND CFP to LP for given Id and Task
END ACTION
```

Figure 12: Sending call of proposals to agents.

2. *The agents send a bid for the task to the initiator as seen in figure 10.*

```
// Answer to Call For Proposal
ANNOTATED ACTION cfp (CNPId, Task) [source (A)]
        PRE-CONDITION: "initiator" is the provider in source A
        // Remember agent's proposal
        ADD PRECEPT proposal for CNPId and Task with Offer value
        SEND proposal offer to Source A
END ACTION
```

Figure 13: Agents send their offer to initiator.

3. *The initiator checks if all proposals have been received using the all_proposals_received rule, analyses the bids and chooses a winner based on the criteria of minimum offer followed by rejection of remaining bids (figure 11).*

```
// Check if proposals from all agents have been received

RULE all_proposals_received (CNPId, NP)

        STORE count of number of accepted proposals received in NO

        STORE count of number of refusals received in NR

        NP = NO + NR                    // NP is the total number of proposals

END RULE
```

Figure 14: Rule for checking total number of proposals.

Once all the bids are received and analysed, winning agent is chosen as seen in figure 12.

```
// Choose winner to execute task

ACTION winner (Id, LO, WAg)

        PRE-CONDITION: Find all offers for proposals in source A

        Print "Offers are " LO                    // Display all participants' offers in logger

        CALL ACTION min to choose the proposal with the smallest offer

        PRINT "Winner is " WAg "with " WOf        // Display winning agent's offer

END ACTION
```

Figure 15: Action for choosing winning bid.

4. *The initiator announces the results to the participating agents and the winner proceeds to perform the task assigned to it. Both the winning and losing agents are informed about the result as shown in figure 13.*

```
// Announce result to the winner
ACTION result (CNPId, offer, WAg)
        SEND Proposal Acceptance to winning agent        // accept_proposal action for agent
        RECURSIVE CALL to result action
END ACTION

// Announce result to other participants
ACTION result (CNPId, offer, WAg)
        SEND Proposal Rejection to other agents          // reject_proposal action for agent
        RECURSIVE CALL to result action
END ACTION
```

Figure 16: Result Declaration.

Using an empty(bin) precept, the IA checks whether the bin is empty or full. As long as there are bicycle frames in the bin, the following actions are performed:

- An RA is requested via the CNP and added to the environment with its initial position same as the bin, (0, 0).

```
// Request RA
CALL ACTION cnp(1, bring_part123 , LP, "robotic_arm_agent")
// Add RA to the environment at the bin's location by default
CALL ACTION add(ra)
```

Figure 17: Adding the RA Agent.

- The following steps are repeated by the RA for three bicycle parts- HT, TT and DT:
    a) Pick the bicycle part from the bin.
    b) Move to a specified location within the assembly surface of the environment.
    c) Drop the bicycle part on to the surface.
    d) Go back to the bin.

- RA is removed from the environment as seen in figure 15.

```
// Request RA
CALL ACTION cnp(1, bring_part123 , LP, "robotic_arm_agent")
// Add RA to the environment at the bin's location by default
CALL ACTION add(ra)
```

Figure 18: Action to remove the RA agent.

- IA requests 3 HAs to hold each part from previous step and 1 WA via the CNP and adds the agents to the environment.

```
// Initiator requests for a holding agent to hold the Head Tube
CALL ACTION cnp(2, hold_part1 , LP1, "holding_agent")
// Holding agent 1 is added to the environment
CALL ACTION add(h1)
// Initiator requests for a holding agent to hold the Top Tube
CALL ACTION cnp(3, hold_part2 , LP2, "holding_agent")
// Holding agent 2 is added to the environment
CALL ACTION add(h2)
// Initiator requests for a holding agent to hold the Down Tube
CALL ACTION cnp(4, hold_part3 , LP3, "holding_agent")
// Holding agent 3 is added to the environment
CALL ACTION add(h3)
// Initiator requests for a welding agent to weld joint (1, 2, 3)
CALL ACTION cnp(5, weld_parts123 , LP4, "welding_agent")
// Welding agent is added to the environment
CALL ACTION add(w1)
```

Figure 19: Requesting holding and welding agents via CNP.

- Joint (1,2,3) is welded and all HAs and WA are removed from the environment.

- Steps 2-5 are repeated for the following:

  a) Assembly of the ST and CT and welding of joint (3, 4, 6) followed by the welding of joint (2, 4) using 3 HAs and 2 HAs respectively.
  b) Assembly of the SS and welding of joint (5, 6) followed by the welding of joint (4, 5), each using 2 HAs in the process.

An example of this process is the RA agent being requested again to pick up the seat and chain parts as seen in figure 17.

```
// Initiator requests RA agent again to bring parts 4 and 6
CALL ACTION cnp(6, bring_part46 , LP5, "robotic_arm_agent")
// RA is added back to the environment
CALL ACTION add(ra)
// RA picks Seat Tube and drops it to a specific position on the assembly surface
CALL ACTION pick(seat)
CALL ACTION moveToPos(3, 3)
CALL ACTION drop(seat)
// RA picks Chain Stay from bin and drops it to a specific position on the assembly surface
CALL ACTION moveToBin
CALL ACTION pick(chain)
```

Figure 20: RA being recalled.

- At this point, a Boolean flag initialized to false is updated to indicate that the bicycle frame assembly is completed. A complete(bike) precept is added. Defined in the environment model, the bikeFrameCompleted flag is set to true upon the welding of the last joint which is when the seat stay is welded to the seat tube.

```
METHOD weldSeatStay
        UPDATE bikeFrameCompleted flag TO TRUE
END METHOD



Check if bikeFrameCompleted defined in model is TRUE
        If yes, add percept complete(bike)
```

Figure 21: Boolean flag updated to indicate assembly completion followed by percept addition.

- A move action, which is executed under the circumstance that the complete(bike) precept is present, is triggered.
- An MA is requested and added to the environment.
- MA picks the bike frame from the assembly surface and moves to the (9, 0) location to store the bike.
- MA is removed from the environment as seen in figure 19.

```
// Execute the task and report to the initiator
ANNOTATED ACTION accept_proposal (CNPId)
        PRE-CONDITION: Proposal precept for CNPId, Task and Offer exists
        CALL ACTION add(MA)                          // Add Moving Agent to the environment
        // MA picks the assembled bike frame from the welding surface
        CALL ACTION pick(bike)
        CALL ACTION moveToPos(9, 0)                  // Move MA to fixed location to stash bikes
        // MA carefully places assembled bike in specified location
        CALL ACTION stash(bike)
        PRINT "Bike moved away from assembly surface"    // Logger message
        CALL ACTION remove(ma)                       // Remove MA from environment
END ACTION
```

Figure 22: Moving bicycle action initiated by moving agent.

- After each pick(item) action, the system updates the number of pieces in the bin. If zero, the empty(bin) precept is added. Otherwise, Steps 1 – 11 are recursively performed until the bin is empty.

```
METHOD pickPiece
        RESET bikeFrameCompleted flag to FALSE
        DECREMENT noOfPieces by 1
        CHECK IF noOfPieces is equal to 0
                If yes, update binIsEmpty Boolean flag to TRUE
END METHOD
```

Figure 23: Updating bin to indicate status.

```
Check if binIsEmpty defined in model is TRUE
If yes, add percept empty(bin)
```

Figure 24: Adding the percept for an empty bin.

```
// In the event that the agents need to assemble a bike frame

// Under the circumstance that the bin is not empty

ACTION assemble

        PRE-CONDITION: empty(bin) precept does not exist
```

Figure 25: Bike assembly pre-condition.

The proposed system has some distinct advantages; the model enables an agent to be removed from the environment during a state of inactivity, e.g., the robotic arm agent is removed during the holding and welding actions. Additionally, it also allows for multiple bicycles to be assembled based on the number of bicycle frame parts in the bin with each frame being moved away from the assembly surface after completion.

However, it has been assumed that all bicycle parts are always available for each bike frame, e.g., there are 6, 12, 18, etc. pieces and not 8, 13, 22, etc. Otherwise, the system abruptly stops executing after all pieces have been used up.

Even though it provides an organised and established approach for communication, CNP might not be entirely effective since it consists of 5 stages. For cases where only one agent of a particular type is required, e.g., a robotic arm agent, the request for proposal, bidding, acceptance and declaration processes are time-consuming.

3. Future Work

There were a few key points noted for further improvements during the design stages with the goal of making future implementations more efficient. They are as follows:

- A new percept can be added to check whether any bicycle frame is left incomplete due to missing pieces so that the system may terminate correctly. As bottlenecks create delays which in turn increase production costs, streamlining the assembly line will help ensure lean operation for the company (Hajmirfattahtabrizi & Song, 2019).
- When removing agents from the environment, currently the design system directly removes agent from its last position. In order to mimic a real-life scenario, it is argued by authors that moving the agent to the end of the environment before exiting will be ideal.
- Currently, once an agent's task is over for a particular iteration, it is removed from the environment. As part of future improvement, it can analysed whether allowing the agents to be in resting position with the environment such as shifting idle agents to the bottom of the grid could be a more time effective approach as opposed to removing an agent and requesting it again when required in later processes.

4. Conclusion

The scenario described in the assignment excerpt suggests exploring the approach of designing an MAS for the automated assembly of bicycle frames to cut costs. As supported by (Beer, et al., 1999) and (Hsieh, 2006), MAS systems have been thoroughly analysed for their role within the manufacturing industry with a positive review on their efficiency. The functioning of the designed CDPS system which also implements the CNP is in line with this analysis since it enables collaborative communication among multiple agents in an organised and sequential manner. These agents, otherwise, would not be capable of single-handedly assembling the bicycle frame.

Initially a challenge to construct a structured, concise and efficient system, the designed MAS achieves the goals set in the scenario as required by processing the tasks via different agents in a specified order. For future optimisation in terms of performance, allocation of memory resources and time taken for execution, the authors would like to explore other alternatives or protocols for the communication between various agents of the system**.**

References:

Wooldridge, M., 2009. *An Introduction to MultiAgent Systems.* 2nd ed. Chichester: Wiley.

Hsieh, F.-S., 2006. Analysis of contract net in multi-agent systems. *Automatica,* Issue 42, pp. 733-740.

Beer, M. et al., 1999. Negotiation in multi-agent systems. *The Knowledge Engineering Review,* 14(3), pp. 285-290.

Hajmirfattahtabrizi, M. & Song, H., 2019. Investigation of Bottlenecks in Supply Chain System for Minimizing Total Cost by Integrating Manufacturing Modelling Based on MINLP Approach. *Applied Sciences,* 9(6).

Mertens, K. & Holvoet, T., 2005. *An Adaptive Distributed Layout for Multi-agent Applications.* United States, International Workshop on Software Engineering for Large-Scale Multi-agent Systems.