applied sciences

MDPI

Article

Multi-Agent Simulation Environment for Logistics Warehouse Design Based on Self-Contained Agents

Takumi Kato * and Ryota Kamoshida

Center for Technology Innovation—Artificial Intelligence, Hitachi, Ltd. Research & Development Group, 1-280, Higashi-Koigakubo, Kokubunji-shi, Tokyo 185-8601, Japan; ryota.kamoshida.vw@hitachi.com

* Correspondence: takumi.kato.uw@hitachi.com

Received: 9 August 2020; Accepted: 23 October 2020; Published: 27 October 2020

Abstract: We propose a multi-agent simulation environment for logistics warehouses. Simulation is a crucial part of designing industrial systems, such as logistics warehouses. A warehouse is a multi-agent system (MAS) that consists of various autonomous subsystems with robots, material-handling equipment, and human workers. It is generally difficult to analyze the performance of a MAS thus, it is important to model a warehouse and conduct simulations to design and evaluate the possible system configurations. However, the cost of modeling warehouses and modifying the models is high because there are various components and interactions compared to conventional multi-agent simulations. We proposed a self-contained agent architecture and message architecture of a multi-agent simulation environment for logistics warehouses to reduce the simulation-model development and modification costs. We quantitatively evaluated our environment in terms of development costs by comparing such costs of our environment and a widely used multi-agent simulation environment.

Keywords: multi-agent simulation; multi-agent system; self-contained; logistics warehouse; autonomous; robots; material-handling equipment; modifiability

1. Introduction

Simulation is a crucial part of designing industrial systems such as logistics warehouses. A logistics warehouse is modeled as a multi-agent system (MAS) that consists of various autonomous subsystems with robots and material-handling equipment. It is generally difficult to analyze the performance of a MAS; thus, it is important to model a warehouse and conduct simulations [1–3] to design and evaluate the system configuration. There exist several multi-agent simulation environments, of which NetLogo [4] is one of the most widely used.

There are an increasing number of logistics warehouses that have introduced the use of several robots and material-handling equipment. Since e-commerce is growing rapidly, the order volume of logistics warehouses has increased [5–7] worldwide. Despite the workload increase in logistics warehouses, there is a labor-shortage problem [8] in several countries because of the decrease in the number of working-age people. Therefore, there is a growing demand for logistics warehouses to introduce robots and material-handling equipment to automate processes to increase productivity without introducing additional human workers.

The development costs of a logistics warehouse simulation model are high because it includes the development of various entities, such as different agents (e.g., material-handling equipment, robots, and humans), the physical space that agents operate in, and the items and boxes that are processed and shipped by the agents. Generally, a multi-agent simulator focuses on the simple agents and an environment for analyzing the emergent behavior of agent systems. The development of a simulation

model of a logistics warehouse is more complicated than ordinary multi-agent simulation in terms of variation in behavior, complexity of the environment, etc.

The simulation model of a logistics warehouse requires high modifiability. In the process of designing a logistics warehouse, it is necessary to estimate the productivity of the various possible designs of the warehouse and choose the appropriate equipment in a reasonable configuration (cooperation of subsystems, location of equipment, etc.) to achieve the design goals. Since prototyping [9] and evaluation of a system configuration is key to effective designing of logistics warehouses, low-cost development of simulation models and their modifiability are important.

There is a tradeoff between low-cost development and modifiability [10]. It is common to create modules of simulation models and use/reuse them to lower development costs. Using large modules that have several functionalities can greatly reduce the development burden. However, it is difficult to anticipate the simulation requirements because the developers of the logistics warehouses need to try several scenarios of warehouse designs. As the modules become larger, the possibility of the modules cannot fit into the requirements become higher. However, smaller modules have less impact on development cost reduction.

This research aims to provide support for designing and optimizing logistics warehouses by reducing the burden on simulating and evaluating the possible designs to choose the best design options. To realize the objective, we set this study's goal to achieve both low-cost development and high modifiability for logistics warehouses. We propose a simulation modeling environment to realize the goal. To improve the ease of developing logistics warehouse simulation models, a simulation modeling environment is needed that provides the common functionalities for logistics warehouse modeling cost reduction, as well as the architecture that enhances modifiability for design considerations of logistics warehouses.

The remainder of this paper is organized as follows. Section 2 describes the related work of this paper. Section 3 explains the design and concept of our simulation modeling environment. Section 4 describes the prototypical implementation of the proposed simulation modeling environment. The comparison of development and modification cost of logistics warehouse simulation model is described in Section 5. Section 6 concludes this paper.

2. Related Work

There are multi-agent simulation modeling environments [4,11–13] for general multi-agent simulations. These environments provide users with the functions to define the simple data structure and behaviors of agents. The focus of general multi-agent simulation modeling environments is analyzing relatively simple multi-agent systems with emergence characteristics that consist of several small agents. If developers try to simulate logistics warehouses using these environments, it is necessary for the developers to have comprehensive knowledge of logistics warehouses, because it is difficult to abstract the system of a logistics warehouse to analyze the desired characteristics. Since the appropriate abstraction is key to realizing effective simulation models [14], the existing environments are not useful for the non-experts of logistics warehouses.

There are designated simulation environments [6,7,15,16] for logistics warehouses. Some environments (e.g., [6,7]) provide users with the functions to evaluate the performance of a particular subsystem with high accuracy to evaluate and design the configuration of these subsystems. These environments focus on subsystems, not an inter-subsystem's overall performance in a logistics warehouse. There are environments for simulating logistics warehouse with high accuracy [15], however, the modeling requires highly detailed configuration with 3D CAD. Additionally, it is difficult to incorporate the existing API (Application Programming Interface) for particular material-handling equipment.

There exists research [17,18] on optimizing logistics warehouse based on multi-agent simulations. For example, in [17], they focus on optimizing logistics warehouse, especially the number of automated guided vehicles (AGVs) used to unload items from tracks. They propose a simulation framework to

evaluate each scenario of the unloading task with a different number of AGVs. In the experiment, they discovered that the performance is affected by the warehouse layout, not only the number of AGVs. As this study implies, the simulated system's key characteristics are often unclear at the beginning of consideration. It is typically found out later in the experiment, and it induces the new modification requirements. Since they do not provide a technique to ensure modifiability and low-cost development capability, there is still a need to investigate both of them. There are studies based on logistics network simulation using multi-agent simulation technologies [19,20]. However, the system architecture of a logistics network is different from warehouses and difficult to apply to warehouse simulation and optimization.

Given the related work on logistics warehouse simulation, we focused on the following features:

- Prototypical and modifiable implementation of inter-subsystems: Current simulation environments focus on individual material-handling equipment behavior, the logistics warehouse's inter-subsystem simulation in detail, or prototypical multi-agent simulation of partial logistics warehouse. We focused on prototypical (low development costs) and modifiable implementation of a simulation model with multiple subsystems and material-handling equipment by incorporating pre-defined messages and data structure.
- Not necessary to learn new syntax and interfaces: We only used the Python programming language's syntax for rule and data description. In practice, developers often choose a programming language to model the target logistics warehouse from scratch. From a survey of simulation modeling [21], the largest portion of users choose this approach. One of the most significant reasons is that developers do not want to learn the new programming language's syntax.
- Third party functions (APIs, software development kits (SDKs)) can be easily integrated into the simulation model: As long as the desired API library or SDK is usable with the Python programming language, our simulation environment can use the library without developing a designated wrapper.

We summarize the basic comparison among the simulation environments in the above-mentioned investigations. The environment [4] provides basic functions to implement multi-agent simulation models, a visualization function, visual editing of a map, and a built-in function to visualize the value transition of variables without extra programming but with GUI configurations. The environment [11] also provides basic functions to implement multi-agent simulation models, visualization function of simulation space, and function to visualize the simulation model's values with a little programming. It allows programming models in Java. The environment [12] supports the BDI architecture of agents, and the representation of simulation space covers multiple application domains. The environment [13] has a rich GUI to model the multi-agent system, and the functions are mostly focusing on the simulation of society. The designated simulation modeling environments for logistics warehouses [6,7] simulate a mobile fulfillment system (a.k.a. picking system), which is a part of a logistics warehouse. The environment [15] is a commercial simulator for logistics warehouses that provide a rich 3D CAD function to model the logistics warehouse in detail. The investigation [16] uses [6] for simulating a multi-agent system of autonomous robots. The investigation [17] uses a Jason [22] that is a logic-based agent-oriented programming language written in Java. It also supports BDI agent architecture.

3. Proposed Multi-Agent Simulation Modeling Environment for Logistics Warehouses

3.1. Strategy of Proposed Simulation Environment

As mentioned in Section 1, we focus on reducing development costs and achieving modifiability of a multi-agent environment in a logistics warehouse simulation. We designed the multi-agent simulation environment with following features to achieve the above objectives. The features are explained in detail in the following subsections.

- **Self-contained architecture of agents representing entities in a logistics warehouse**: This architecture makes it easy for the model developer to add, replace, and modify the agents in the environment. We made each agent self-contained and individually executable without the need of installing a specific software development kit (SDK) and sharing information only with messages.
- **Unified message architecture and message primitives for a logistics warehouse**: To reduce the development costs of a logistics warehouse simulation model, we designed a unified architecture of agent messages and pre-defined primitives to make the description of message easier. Our simulation environment has a template of agents that has basic functionalities to process messages with the pre-defined primitives, which reduces the development costs.

3.2. Multi-Agent Simulation Environment Architecture

The multi-agent system architecture of the proposed simulation environment is shown in Figure 1. There are three types of agents, i.e., simulation, spatial-temporal and logging agent. The only communication method among these agents is exchanging messages written in the agent communication language. These agents can be running in different computing environments that are networked together via TCP/IP. Since the idea of a performative is effective, the format of agent message is based on KQML (Knowledge Query and Manipulation Language) [23] to ensure ease of description, uniformity, and extensibility. FIPA-ACL [24] can also be utilized as the basic idea of using a performative is inherited in the protocol.

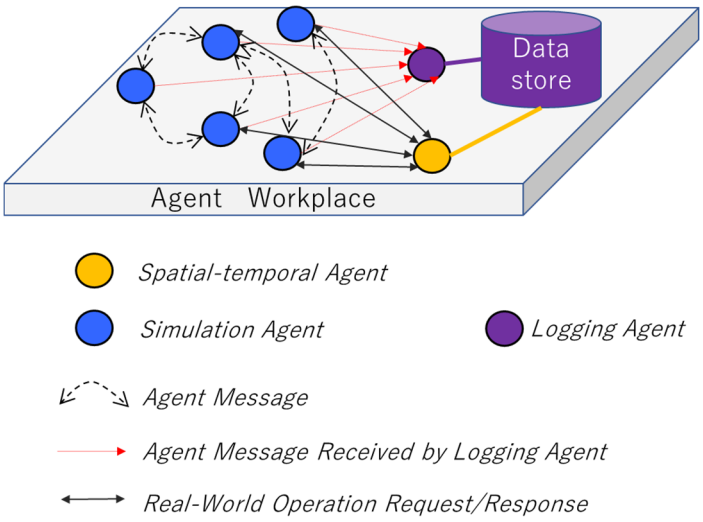

Figure 1. Overview of proposed multi-agent simulation environment.

The simulated entities, such as an automated guided vehicle (AGV), material-handling equipment, and human workers are abstracted as simulation agent. The simulated entities without physical embodiment, such as a warehouse management system (WMS), can also be implemented as a simulation agent. The space and time of the simulation target is represented as a spatial-temporal agent that stores the information of logistics warehouse layouts, each area's attributes, and the physical location of the simulation agents mentioned above. Simulation agents send messages to the spatial-temporal agent to simulate operations (e.g., moving an object) and move themselves.

All the messages exchanged among simulation agents and the spatial-temporal agents are intercepted by a logging agent. Simulation agents can also log the desired information by sending a message to the logging agent in the designated format at any given simulation time. This method of logging simulation helps users to collect and analyze the simulation results in a distributed environment.

The lifecycle of a simulation is as follows. A simulation starts when the agent workplace, logging agent, simulation agents and then spatial-temporal agent are activated in that order. The simulation ends when the simulation agents terminate themselves according to the embedded knowledge of the simulation, and then the spatial-temporal agent, logging agent, and the agent workplace terminate themselves in that order.

All the agents communicate with the agent workplace to be a part of the simulation. The agent workplace stores each agent's ID, IP address, port number, and TCP/IP connection for queuing and transferring messages among the agents. This configuration allows developers to model the simulation agents with high heterogeneity, e.g., requiring different operating systems due to an API's restriction. Despite this heterogeneity, agents are able to exchange information, log, and analyze the simulation results in a unified manner by introducing a unified format of messages. This configuration establishes the modifiability of the simulation model by achieving transparency in the runtime environment and APIs used in each simulation agents.

3.3. Agent Architecture

We designed a self-contained agent architecture in which an agent can be executed as a single program and the internal variables, functions and methods are not directly referred by the other programs. In existing multi-agent platforms, some agents' source codes are referenced by the other agents (e.g., class inheritance and rule import). In our architecture, there is no information shared between agents unless there are message exchanges. All agents have their entry point (e.g., main function) and act as separate programs. This architecture makes it easier to implement the different data structures and processing functionalities in each agent with heterogeneous APIs and runtime environments.

There are two parts to a simulation agent, agent head and agent body. Figure 2 shows the self-contained agent architecture of the proposed simulation environment. The agent head carries out rule evaluations and message exchanges based on the declarative definition of rules to define in what situation an action should be taken. A rule is described by a conditional part and an action part. The working memory in the agent head is a list that stores the received messages and the facts that are derived from the rule evaluations. The rule evaluation module checks whether the conditional part of a rule and facts in working memory are matched. If all the conditions in a rule are matched, the rule is fired, and the action part of the fired rule will be executed (i.e., general if–then-based rules). In the action part, the operations to facts (generation, deletion, or modification of facts, etc.) are defined. The action part also defines commands such as sending messages to the other agents or executing action functions in the agent body.

The agent body is a set of procedural definitions of an agent's actions referred by the rules in the agent head. The procedural definitions are described as a set of functions and methods. For example, in the Python programming language, a set of functions with the action names are defined as the agent body. The necessary APIs and corresponding descriptions are included in the agent body. With the separation of the agent head and agent body, the description of rules (defining when to do what) is not directly affected by the changes in the APIs and external environment of the agent, which could potentially improve modifiability.

The spatial-temporal and logging agents are implemented in this architecture. A spatial-temporal agent always includes the fact in the working memory representing the spatial-temporal information in the defined format explained later. A logging agent's working memory stores the exchanged messages among the other agents, and the messages are exported to a logging file before the logging agent terminates itself.

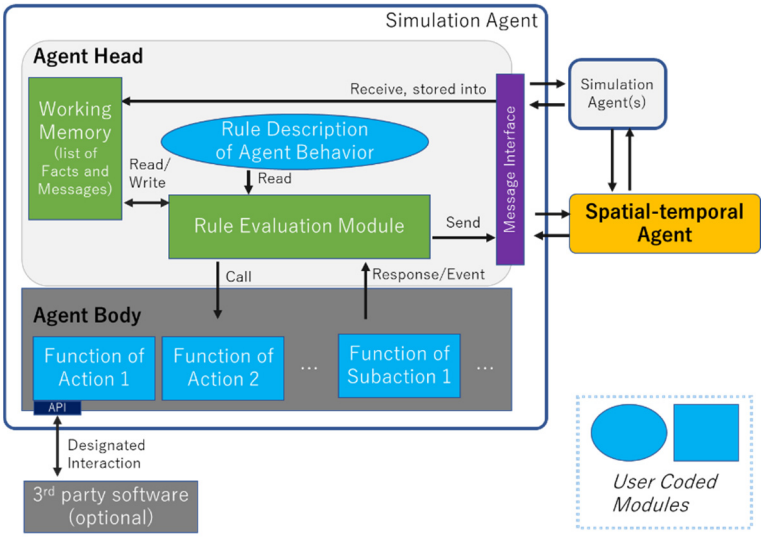

Figure 2. Self-contained architecture of agent in proposed simulation environment.

3.4. Message and Fact Architecture

Our message architecture is shown in Figure 3. Every message has a keyword called "performative" [23], which represents the semantics of what it means to send a message. A performative is also called a "speech act", meaning that sending a message is a particular action of an agent, not just an information exchange. Each performative requires specified information as a property of the message to represent the act of sending the performative. Table 1 lists the pre-set performatives in the proposed simulation environment. The agent core library includes the functionalities to process messages with these performatives to minimize the developer's burden. For ease of explanation, the examples are written in the syntax of a Python dictionary datatype but can be translated into JSON and other equivalent data formats if the essential structure is kept.

Message Examples

MSG1 {'_pf': 'handover', 'to': 'ICR1', 'from':'racrew1', 'object': ['order1', 'ITM1', 2] }

MSG2 {'_pf': 'order_carry_shelf', 'to': 'racrew2', 'from':'WMS', 'shelf': ('SH11', [6, 4, 0],)}

MSG3 {'_pf':'tick_start', 't':10, 'to':'*', 'from':'STAgent' }

MSG4 {'_pf':'tick_done', 't':10, 'from':'racrew2'}

MSG5 {'_pf':'move_request', 'from':'ICR1', 'to':'STAgent', 'node':[10, 10, 0], 'coordinate':[9.6, 10, 0]}

MSG6 {'_pf':'move_request_accepted', 'to':'ICR1', 'from': 'STAgent'}

MSG7 {'_pf':'move_request_rejected', 'to':'racrew1', 'from': 'STAgent'}

Figure 3. Message architecture (examples written in Python syntax).

Table 1. Performatives for logistics warehouse.

Performative.	Argument	Description
tick_start	't':int	Spatial-temporal agent sends this performative to start an interaction for a designated time frame.
tick_done	'from':<AgentID>	Simulation agent sends this performative to notify spatial-temporal agent that there is no more messages to send regarding the current time frame.
move_request	'from':<AgentID>, 'node': [int, int, int], 'coordinate': [float, float, float]	Simulation agent sends this performative to notify spatial-temporal agent where the simulation agent is assigned to move in the next time frame.
move_request_accepted	'to':<AgentID>	Spatial-temporal agent sends this performative to notify a simulation agent that the previously sent move_request is accepted.
move_request_rejected	'to':<AgentID>	Spatial-temporal agent sends this performative to notify a simulation agent that the previously sent move_request is rejected and it needs to stay where it is.
handover	'to':<AgentID>, 'object': <ObjectData>	Simulation agent 1 sends this performative to anther simulation agent 2 to handover the object carried by agent 1 to agent 2.
order_carry_shelf	'to':<AgentID>, 'content': <ShelfData>	Simulation agent 1 (WMS agent) sends this performative to another simulation agent 2 that can carry a shelf to a picking station in the AGV picking system.
ready	'to':<AgentID>	Simulation agent 1 (agents of AGVs) sends this performative to another simulation agent 2 (WMS agent) to notify that the agent 1 is waiting for some commands.
pick	'to':<AgentID>, 'content': <OrderData>	Simulation agent 1 (WMS agent) sends this performative to another simulation agent 2 (a human/robot for picking) to pick the designated items from near objects.

The description of these example messages are as follows:

- MSG1: sent from a simulation agent of the Racrew (Racrew is a registered Japanese trademark owned by Hitachi Industrial Products, Ltd.) AGV to iCarry AGV (Racrew and iCarry are AGVs for logistics warehouses, explained in detail later) to transfer the data value of the "object". This handover primitive triggers the sender to automatically update the fact representing the carrying object in its working memory. The primitive also triggers the receiver to update the fact of carrying an object to add the handed over object data.
- MSG2: sent from the WMS to an agent of the Racrew AGV to order the AGV to carry the shelf with the ID "SH11" at the node location [6, 4, 0] to one of the available picking stations.
- MSG3: sent from the spatial-temporal agent to all the simulation agents to start the interaction of the next time frame (t = 10).
- MSG4: sent from a simulation agent of the Racrew AGV to the spatial-temporal agent to notify the spatial-temporal agent that there is no more information to send in this time frame (t = 10).

- MSG5: sent from a simulation agent of the iCarry AGV to the spatial-temporal agent to move to the coordinate of the values "node" and "coordinate".
- MSG6: sent from a simulation agent of the iCarry AGV to the spatial-temporal agent to notify the spatial-temporal agent that the previously sent move_request is accepted. When this primitive is sent from the spatial-temporal agent to a simulation agent, the fact representing map information is automatically updated both in the spatial-temporal agent and simulation agent.
- MSG7: sent from a simulation agent of the iCarry AGV to the spatial-temporal agent to notify the spatial-temporal agent that the previously sent move_request is rejected. When this primitive is sent from the spatial-temporal agent to a simulation agent, the fact representing map information is automatically updated both in the spatial-temporal agent and simulation agent.

The characteristics of a performative suits logistics warehouse simulation because a warehouse generally has several agents exchanging physical objects such as items to ship. With the performative representing the physical act of passing object from one agent to another, developers can easily represent cooperation among the agents in the simulation.

We defined the architecture and pre-defined keywords of the facts in the working memory and prepared the pre-defined functions to process the facts. In the previous subsection, we explained that every agent has a working memory to store the facts used in the rule evaluations and actions. The working memory is a list of facts as shown in Figure 4. Facts stored in the working memory include the key "_s" and the characterizing keyword as a key. For example, almost every agent in a logistics warehouse carries objects; therefore, the agent core library prepares a function to retrieve and update the facts that include {"_s":"carrying_object"}. Figure 5 shows the fact of map information stored in the spatial-temporal agent. The fact includes {"_s":"map_info"} and the current time frame (e.g., "t":0). This data format represents the physical space of a logistics warehouse as a collection of nodes labeled (<vertical node location>, <horizontal node location>). Each node has a type, such as wall, floor, and some specific types like racrew_area that the Racrew AGVs (the AGVs that can carry shelves) can move around. Each node also has the key "has", and the key value is an empty list by default. The list can contain keywords representing the objects on the nodes, such as the IDs of AGVs, and those of inventory shelves. The spatial-temporal agent sends the whole map to the logging agent to log the map configuration in each time frame. The visualization program parses the "map_info" facts to recreate the location of objects in the time series to visualize the simulated logistics warehouse.

Fact Examples

```
{'_s': 'carrying_object', 'order': ['order1', 'ITM1', '2']}
{'_s': 'path_to', 'toward': 'SH11', 'path': [(6, 4, 0), (7, 4, 0), (7, 5, 0), ..., (6, 3, 0)]}
{'_s':'current_time', 't':0 }
{'_s': 'orders', 'labels': ['orderID', 'itemID', 'pcs'], 'data': [ ['order1', 'itemA', 4], ['order2', 'itemB', 2], ··· ]}
{'_s': 'inventory', 'content': { 'SH1': {'itemA':20, 'itemB':20, ··· } ··· }}
```

Figure 4. Fact architecture (example written in Python syntax).

```
{'_s':'map_info', 't':0,'map_name':'warehouse1','size_vertical':5,'size_horizontal':5,
 (0,0): {'type':'wall', 'has': []}, (0,1): {'type':'wall', 'has': []},
 (0,2): {'type':'wall', 'has': []}, (0,3): {'type':'wall', 'has': []},
 (0,4): {'type':'wall', 'has': []}, (1,0): {'type':'wall', 'has': []},
 (1,1): {'type':'racrew_area', 'has': ['racrew1', "shelf1"]},
 (1,2): {'type':'racrew_area', 'has': ['racrew2', "shelf2"]},
 (1,3): {'type':'racrew_area', 'has': ['racrew3', "shelf3"]},
 (1,4): {'type':'wall', 'has': []}, (2,0): {'type':'wall', 'has': []},
 (2,1): {'type':'racrew_area', 'has': []}, (2,2): {'type':'racrew_area', 'has': []},
 (2,3): {'type':'racrew_area', 'has': []}, (2,4): {'type':'wall', 'has': []},
 (3,0): {'type':'wall', 'has': []}, (3,1): {'type':'racrew_area', 'has': []},
 (3,2): {'type':'racrew_area', 'has': []}, (3,3): {'type':'racrew_area', 'has': []},
 (3,4): {'type':'wall', 'has': []}, (4,0): {'type':'wall', 'has': []},
 (4,1): {'type':'picking_station', 'has': ['picker1']},
 (4,2): {'type':'picking_station', 'has': ['picker2']},
 (4,3): {'type':'wall', 'has': []}, (4,4): {'type':'wall', 'has': []},}
```

Figure 5. Fact of map information stored in spatial-temporal agent (example).

The overview of the messaging sequence of agents in simulation models is shown in Figure 6. In every time frame, the spatial-temporal agent first broadcasts the performative of tick_start to the simulation agents. Simulation agents then make decisions at the current time frame by evaluating rules, executing actions, and/or exchanging messages (e.g., messages with handover performative, pick, etc.). Next, simulation agents send messages with the move_request performative (not applicable if the simulation agent does not have a physical location) to move to the next location in the next time frame. The spatial-temporal agent then evaluates the requests and replies to the messages with move_request_accepted performative to grant the requests. If the move_request messages conflict with the next physical location or the next physical location is unacceptable (e.g., the requested location is a wall), the spatial-temporal agent replies with move_request_rejected performatives. The simulation agents that received the messages with move_request_rejected then make the decisions again and sends the move_request until they are accepted. This procedure repeats in every time frame.

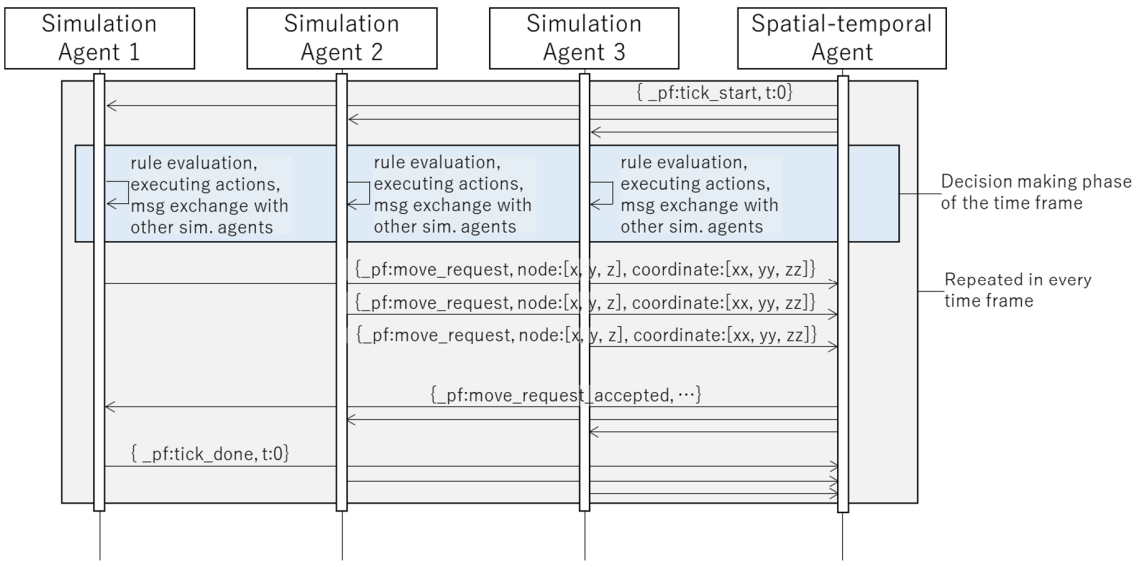

Figure 6. Overview of message exchange sequence among simulation agents and spatial-temporal agent.

Each agent is free to accept or reject a request from the other agents in the decision-making phase of the time frame, while they are also free to request the other agents perform a certain task. The agents make decisions on individual behavior by their states and rules. For example, when the WMS agent (the management system of a logistics warehouse) requests the Racrew agents (the agent representing

an AGV in the picking area) to process a customer order, each Racrew agent evaluates its state and decide whether it accepts the customer order processing task, or not. Once the Racrew agent accepts a customer order processing task, it decides its path to go to the target shelf using a path planning algorithm. The Racrew agent requests the spatial-temporal agent to provide the map of the picking system area to determine the path.

4. Prototypical Implementation of Proposed Simulation Environment

We used the Python programming language for the prototypical implementation of the proposed simulation environment. Our implementation does not include external libraries other than the standard packages included in Python 3. Figure 7 shows the configuration of the prototyped multi-agent simulation environment. To implement the self-contained agent architecture mentioned earlier, we developed a template called agent core library as described earlier. This template includes the necessary functions for messaging, connecting to the agent workplace, rule evaluation, working memory data structures, and so on, thus developers only need to write rules and actions of each agent. In this study, the rule evaluation module is manually developed from scratch that works as a general production system, but it can be implemented using other inference engines. The message interface is implemented using network socket-based communication.

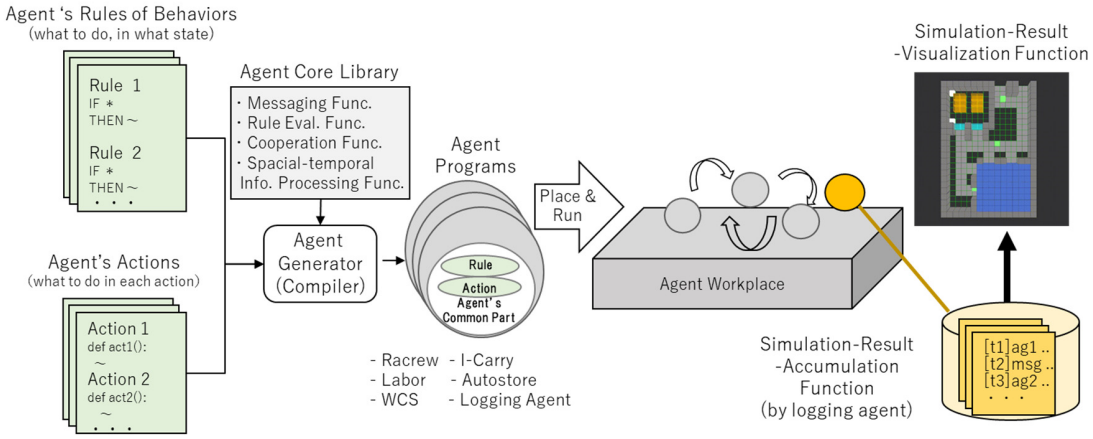

Figure 7. Composition of prototyped multi-agent modelling environment.

Users describe the agent's rules of behaviors and actions, then use the agent generator (compiler) to merge the described rules and actions with the Agent Core Library into a self-contained executable agent program (<agentID>.py). The logging and spatial-temporal agents have, as mentioned above, the same architecture as the simulation agents. We also prepared a template for the logging and spatial-temporal agents, which is an extension of the agent core library for simulation agents. The basic functionality for these two agents is prepared in the templates that are independent from the simulation targets. We prepared the agent workplace as a single Python script that is executable in the platform where the basic Python runtime environment is available.

The developed agents are executed based on the agent workplace, and the simulation logs are exported by the logging agent before the simulation ends. We prepared a visualization program to import and visualize the logistics warehouse's layout and the location of agents and objects (e.g., AGVs, items and shelves) in each time frame. We prepared basic visualization models, such as for shelves, AGVs in a simple rectangular shape, walls, and floors. The visualization program is implemented using a processing programming language.

Figure 8 shows the runtime configuration of the simulation model using the prototype of our simulation environment. Each agent is an independently executed Python script, and all agents are connected to the agent workplace via TCP/IP connections to send messages.

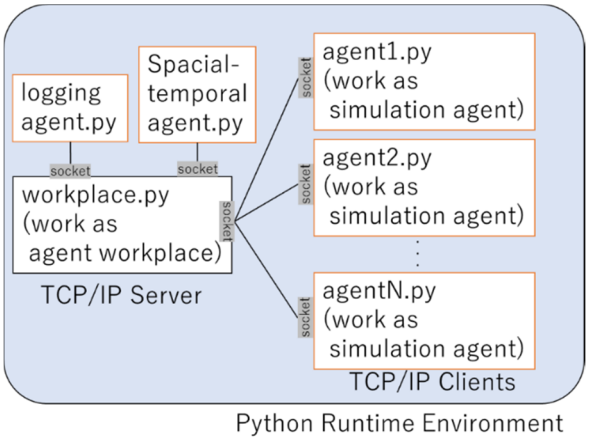

Figure 8. Example structure of multiagent simulation model using prototyped simulation modeling environment.

Figure 9 shows the network configuration of agents. Since agents and the workplace are connected via TCP/IP connections, it is possible to run agents in distributed PCs to run a simulation if the network is a properly configured local area network (LAN). Except the security settings, such as firewall settings, agents can be moved from one PC to another without modification to the program because each agent has an address of the workplace and can connect to the workplace even if the running PC is changed. This characteristic helps users distribute computing loads. The modifiability of runtime configuration (which PC to run which agent) is established based on these characteristics.

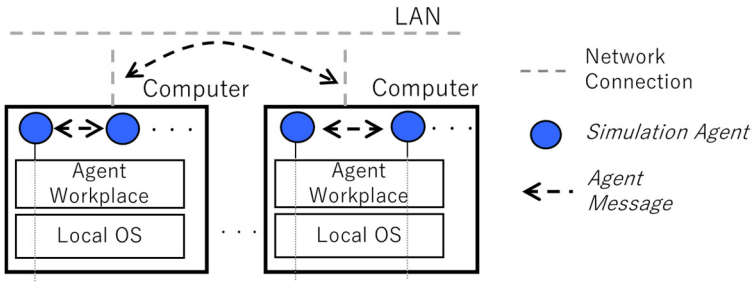

Figure 9. Composition of agents, agent workplaces and network environment.

5. Experiment to Compare Development Costs

5.1. Purpose of Experiment

The purpose of this experiment is to examine the features of development cost reduction and architectural modifiability, by developing the simulation model of logistics warehouse using the existing simulation modeling environment and the environment implemented based on the proposed architecture. This section explains the experimental result of development cost comparison between the proposed simulation environment and the widely used multi-agent simulator NetLogo.

We have chosen NetLogo as a compared multi-agent simulation environment. Because NetLogo is expected to reduce the amount codes that need to be written compared to other MAS platforms, as these features are built-in:

(i) A GUI with a simulation space visualization function. Basically, there is no need to add visualization descriptions to the code.
(ii) Default primitives for manipulating simulation space and objects in the grid world, which is highly compatible with warehouse simulations.

(iii) A function to output graphic charts of the simulation model's values without additional descriptions.

For fairness, we outsourced the development of the simulation model using NetLogo to a software developing company. We only conveyed the simulation model's requirements, input, and output requirement of the simulation model. We did not specify the internal data structure or an agent's communication sequence. We validated the developed model and ensured the validity of experiment in the following manner:

1. First, when we outsourced the simulation model development, we only provided a description of the simulated system and the requirements of the simulation model, as well as the specifications of the inputs and outputs.
2. Second, based on the requirements, we asked the software developing company to design how to implement the agents using NetLogo, i.e., how to implement the internal states and how to make them cooperate with each other.
3. Third, we reviewed the presented design by the software developing company and asked them to modify the functional discrepancies to the developed model using our modeling environment, so that the functions of the material-handling equipment can be the same. We did not specify how to write the code. We ensured the validity of the experiment by asking them to write the code based on their standard method so that we did not tamper with the experimental results. In addition, we did not disclose our intention to use the outsourced model for development cost comparison.
4. Finally, we checked the delivered source codes to see if there was any unfairness in comparing the development cost.

5.2. Developed Simulation Model of Logistics Warehouse with Various Robots and Material-Handling Equipment

The modeling target was an area of a logistics warehouse where several cooperating robots and material-handling equipment were introduced. The area is in charge of the shipping process of picking, sorting, and carrying items from inventory shelves to the shipping area based on the customer's orders. The model includes the function to calculate the productivity of the shipping process. The shipping process consists of the following subsystems:

- Racrew AGV Picking System: At the beginning of the shipping process, it is necessary to pick items specified by the customer orders from the inventory shelves. The classic configuration to pick items is that people carry a cart with them, walk to the shelf storing the desired items, pick the items from the shelf, then put the items into the sorting bucket in the cart. Finally, the cart goes through the rest of the shipping process, such as item inspection and packing. However, this way of picking is not efficient because people need to walk long distances to finish their picking tasks. Therefore, a novel automated logistics warehouse uses an AGV picking system. Instead of people walking to the shelves, AGVs carry the shelves to the people, and the people pick items from the shelves then put the items into the sorting bucket near them.
- iCarry AGV Transport System: In a logistics warehouse, it is necessary to carry items from area to area to process the logistics tasks. This AGV transport system can carry objects by having AGVs move in a pre-defined path.
- Automatic Storage System: In a logistics warehouse, storing items in a buffer is sometimes efficient because the picking task does not always finish right before the shipping time. This system stores items so that the items can be retrieved when the rest of the shipping process is ready.
- WMS: In a logistics warehouse, there is almost always a WMS that has the information of customer's orders and record of inventory. It is also in charge of controlling the picking system regarding which items to pick, when to pick, etc.

Figure 10 shows an overview of the target logistics warehouse of our simulation modeling. The blue lower-right area represents the Auto Storage System, green box represents the AGV Transport System's AGV, white box represents the AGV Picking System's AGV, yellow transparent box represents an inventory shelf, and light blue box close to the inventory selves represents a human picker. The target simulation model was developed and tested using a single computer with Intel Core i7-7800X (3.5 GHz, 12 CPU), 16GB RAM, Windows 10 Pro (64 bit). The authors prepared the order data and parameters of robots and material-handling equipment based on the actual orders of logistics warehouse and machine's specifications. Statistical variation of agent's behaviors is implemented in each agent's action. In the action that simulates statistical variation, the time taken by the behavior is calculated by multiplying average task processing time and a random value generated from Gaussian distribution using Python's default library. The calculated time is passed onto the agent head.

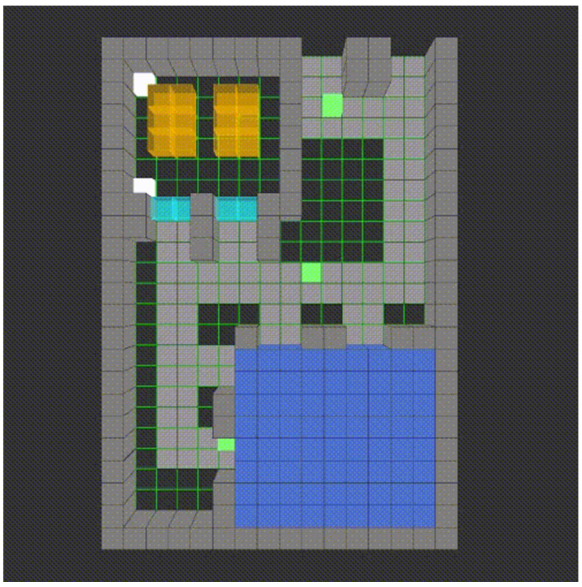

Figure 10. Overview of target distribution warehouse for modeling experiment (a snapshot of simulation result visualization interface).

The developed agents are:

- Racrew Agent: represents an AGV in the Racrew AGV Picking System.
- iCarry Agent: represents an AGV in the iCarry AGV Transport System.
- Picker Agent: represents a human picker in the Racrew AGV Picking System.
- AutoStorage Agent: represents an auto storage system in the simulated logistics warehouse.
- WMS Agent: represents a WMS in the simulated logistics warehouse.

Using NetLogo, we defined the same agents with the identical rules of behaviors and map information. For fairness, we outsourced the development of the simulation model using NetLogo to a software development company. We only conveyed the simulation model's requirements, input, and output requirement of the simulation model so as not interfere with the design policy of the model that changes the development costs. We did not specify the internal data structure or an agent's communication sequence but specified the actual agents' behaviors, e.g., conditions to trigger actions, where to take certain actions, etc.

5.3. Comparison of Development Costs

Compared to using the existing simulation modeling environment (NetLogo), our simulation environment can reduce the development cost by 31.5 %. Figure 11 shows the experimental

results regarding the development costs using the proposed simulation environment and NetLogo. These development costs are calculated by counting the number of lines of codes except the comments and blank lines.

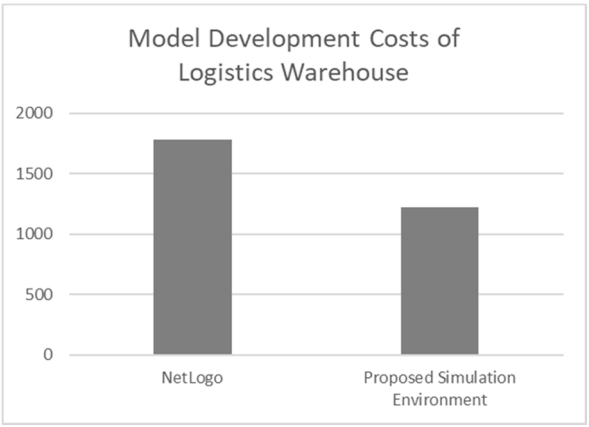

Figure 11. Cost of modeling distribution warehouse with NetLogo and prototyped simulation environments.

We evaluated the modifiability of the developed simulation models. We used the architectural modifiability analysis (ALMA) [10] concept to examine the modifiability of developed multi-agent model. We designed the evaluation in reference to the existing work [25] that applied the ALMA concept to evaluate the modifiability of multi-agent systems. The ALMA allows the developers to elicit multiple cases of modification on the developed software and evaluate the ease of the modification in each case. In the process of designing logistics warehouses, most cases involve reconfiguration and/or replacement of material-handling equipment to assess the productivity. We elicited the practical cases by interviewing a company's employees designing and operating logistics warehouses. Table 2 shows an essential part of the result of modification cost evaluation. LoC (Line of Codes) is the affected part of the source codes in the modified programs. The rest of the result is explained in the Appendix A.

Case 1 (C1) is a modification case of replacing the iCarry transport system with another transport system. In the case of NetLogo, developers first need to review the procedures and global variables, then delete the procedures for iCarry transport system, modify the map information and finally develop the codes for new transport system. In the case of our environment, developers first delete the iCarry agents, then modify the map information in spatial-temporal agent, and finally develop and add the agents of the new transport system using the agent core library. Developing simulation model of the new transport system is necessary in both NetLogo and our environment, but since the previous experimental result shows that the model development costs is relatively low in our environment, overall, using our environment is easier cope with case 1.

Case 2 (C2) is a modification case of replacing the Racrew picking system with another picking system. In the case of NetLogo, developers first need to review the procedures and global variables, then delete the procedures for Racrew picking system, modify the WMS simulation part, modify the map information using a designated editor, and finally develop and add the procedures for the new picking system. In the case of our environment, developers first need to delete the Racrew agents, review and modify the WMS agent and spatial-temporal agent, and finally develop and add the agents of the new picking system using the agent core library. Developing simulation model of the new transport system is necessary in both NetLogo and our environment, but since the previous experimental result shows that the model development costs is relatively low in our environment, overall, using our environment is easier cope with case 2.

Table 2. Modification cost evaluations on essential modification cases.

Case ID	Modification Case Description	Conventional Simulation Environment			Proposed Simulation Environment		
		Required Modification Procedures	Reviewed and Modified Parts	LoC	Required Modification Procedures	Reviewed and Modified Parts	LoC
C1	Replacing iCarry with another transport system	1. Reviewing the global/shared variables and procedures 2. Deleting procedures for iCarry transport system 3. Modifying the map information with a designated editor 4. Developing and adding the procedures for the new transport system	• Global/shared variables • All procedures, focusing on the ones operating the agents of iCarry AGVs	501	1. Deleting the iCarry agents 2. Modifying the map information in spatial-temporal agent 3. Developing and adding agents of the new transport system using the agent core library	• Spatial-temporal agent	351
C2	Replacing Racrew with another picking system	1. Reviewing global/shared variables and procedures 2. Deleting procedures for Racrew picking system 3. Modifying the procedures simulating WMS 4. Modifying the map information with a designated editor 5. Developing and adding the procedures for the new picking system	• Global/shared variables • All procedures, focusing on the ones operating the Racrew AGVs and WMS	630	1. Deleting the Racrew agents 2. Reviewing and modifying the related interaction in WMS agent 3. Reviewing and modifying the map information in spatial-temporal agent 4. Developing and adding agents of the new picking system using the agent core library	• Spatial-temporal agent • WMS agent	421
C3	Replacing automatic storage with another storage system	1. Reviewing the global/shared variables and procedures 2. Deleting procedures for automatic storage system 3. Modifying the procedures simulating WMS 4. Modifying the map information with designated editor 5. Developing and adding the procedures for the new storage system	• Global/shared variables • All procedures, focusing on the ones operating the automatic storage and WMS	358	1. Deleting the AutoStorage agent 2. Reviewing and modifying the related interaction in WMS agent 3. Reviewing and modifying the map information in spatial-temporal agent 4. Developing and adding agents of the new storage system using the agent core library	• Spatial-temporal agent • WMS agent	241

Table 2. *Cont.*

Case ID	Modification Case Description	Conventional Simulation Environment			Proposed Simulation Environment		
		Required Modification Procedures	Reviewed and Modified Parts	LoC	Required Modification Procedures	Reviewed and Modified Parts	LoC
C4	Introducing a picking system in addition to Racrew picking system	1. Reviewing the global/shared variables and procedures 2. Reviewing and modifying the iCarry agents 3. Reviewing and modifying the procedures simulating WMS 4. Modifying the map information with designated editor 5. Developing and adding the procedures for the new picking system	• Global/shared variables • All procedures, focusing on the ones operating iCarry AGVs and WMS	681	1. Reviewing and modifying the iCarry agents 2. Reviewing and modifying the WMS agent 3. Modifying the map information in spatial-temporal agent 4. Developing and adding agents of the new picking system	• Spatial-temporal agent • WMS agent • iCarry agent	486

Case 3 (C3) is a modification case of replacing the automatic storage system with another storage system. In the case of NetLogo, developers first need to review the procedures and global variables, then delete the procedures for automatic storage system, modify the procedures for simulating WMS, modify the map information, and finally develop and add the procedures for the new storage system. In the case of our environment, developers first need to delete the AutoStorage agent, then review and modify the WMS and spatial-temporal agents, and finally develop and add the agents of the new storage system using the agent core library. Developing simulation model of the new storage system is necessary in both NetLogo and our environment, but since the previous experimental result shows that the model development costs is relatively low in our environment, overall, using our environment makes it easier cope with case 3.

Case 4 (C4) is a modification case of introducing the picking system that concurrently works together with the Racrew picking system. In the case of NetLogo, developers first need to review the procedures and global variables, then modify the procedures for iCarry, WMS and the map information, and finally develop the procedures for the new picking system. In the case of our environment, developers first need to review and modify the iCarry and WMS agents, then modify the map information, and finally develop the agents for the new picking system. Developing simulation model of the new storage system is necessary in both NetLogo and our environment, but since the previous experimental result shows that the model development costs is relatively low in our environment, overall, using our environment makes it easier to cope with case 4.

Based on these analysis results, our environment has better architectural modifiability compared to NetLogo. Since every agent in our environment is implemented as a single program file, it is intuitive to understand what to delete. In addition, the process of reviewing the shared variables is not necessary in our environment. Although NetLogo has better support functions such as graphical editor of map, our environment requires less model development costs to create a simulation model, and also requires less burden on elicited modification cases.

We have confirmed the following features of our proposal: (i) our proposed environment can reduce the model development cost of a logistics warehouse, (ii) our proposed environment enables the self-contained architecture of simulation models that realize the architectural modifiability. As mentioned in Section 1, our goal was to realize both low-cost development and high modifiability for logistics warehouses. With the features (i) and (ii), we confirmed that we achieved the goal of this study. We confirmed that we had contributed to designing industrial systems such as a logistics warehouse by reducing the burden on simulating and evaluating the possible designs to choose the best design options.

6. Conclusions

We proposed a multi-agent simulation environment for logistics warehouses. This research aims to provide support for designing industrial systems such as a logistics warehouse by reducing the burden on simulating and evaluating the possible designs to choose the best design options. To achieve the objective, we made the following contributions: To realize the objective, we set this study's goal to achieve both low-cost development and high modifiability for logistics warehouses. To simultaneously reduce the model development costs and achieve architectural modifiability, we made the following contributions:

(i) Designed self-contained architecture of agents representing entities in a logistics warehouse.
(ii) Designed unified message architecture and message primitives for logistics warehouses.
(iii) Empirically compared the model development costs of logistics warehouses with a current multi-agent simulation environment and confirmed that the proposed environment reduces such costs while maintaining architectural modifiability.

For future work, we will further investigate our modeling environment's architectural flexibility based on the simulation model modification and update the scenarios in actual logistics warehouse

development. In the current experiment, the unit of development cost is LoC. Since the characteristics of a programming language and the respective programming style are difficult to consider using LoC, the method of evaluating development costs can be further investigated and improved.

Author Contributions: Conceptualization, T.K.; methodology, T.K.; software, T.K.; validation, T.K.; formal analysis, T.K.; investigation, T.K. and R.K.; resources, T.K.; data curation, T.K.; writing—original draft preparation, T.K.; writing—review and editing, T.K. and R.K.; visualization, T.K.; supervision, R.K.; project administration, R.K.; funding acquisition, R.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

We elicited the practical modification cases by interviewing a company's employees responsible for designing and operating logistics warehouses. Table A1 shows the modification cases on the agent environment and the cost evaluations. Since the change of environment is generally anticipated at the design phase of the simulation model, NetLogo and our simulation environment mostly do not need the modification of codes. Table A2 shows the modification cases on agent behaviors and cost evaluations. Since the change of the agent behaviors is also generally anticipated in the design phase of the simulation model, NetLogo and our simulation environment do not need the modification of codes. In these modification cases, the cost of modification is almost the same in both environments.

Table A1. Modification cost evaluations on modification cases of agent environment.

Case ID	Modification Case Description	Conventional Simulation Environment		Proposed Simulation Environment	
		Required Modification Procedures	Reviewed and Modified Parts	Required Modification Procedures	Reviewed and Modified Parts
C5	Changing the route of iCarry AGVs	• Editing the values in route configuration files	N/A	• Editing the values in route configuration files	N/A
C6	Changing the route of Racrew AGVs	• Editing the values in route configuration files	N/A	• Editing the values in route configuration files	N/A
C7	Changing the area of Racrew picking system	1. Editing map using map editor built in NetLogo 2. Editing the values in route configuration files	N/A	1. Editing the values in the fact of map in spatial-temporal agent 2. Editing the values in route configuration files	Spatial-temporal agent
C8	Changing the items stored in shelves	• Editing the item inventory values in configuration files	N/A	• Editing the item inventory values in configuration files	N/A
C9	Changing the area of iCarry transport system	1. Editing map using map editor built in NetLogo 2. Editing the values in route configuration files	N/A	1. Editing the values in the fact of map in spatial-temporal agent. 2. Editing the values in route configuration files	Spatial-temporal agent

Table A2. Modification cost evaluations on modification cases of agent behaviors.

Case ID	Modification Case Description	Conventional Simulation Environment		Proposed Simulation Environment	
		Required Modification Procedures	Reviewed and Modified Parts	Required Modification Procedures	Reviewed and Modified Parts
C10	Changing the number of AGVs in the iCarry transport system	• Editing the parameters of the number of agents and their locations in configuration files	N/A	1. Copying/deleting the iCarry agents 2. Editing the parameters of the agent locations in configuration files	N/A
C11	Changing the speed of iCarry AGVs	• Editing the parameters in configuration files	N/A	• Editing the parameters in configuration files	N/A
C12	Changing the number of AGVs in the Racrew picking system	• Editing the parameters of the number of agents and their locations in configuration files	N/A	1. Copying/deleting the Racrew agents 2. Editing the parameters of the agent locations in configuration files	N/A
C13	Changing the battery charging threshold of the AGVs in Racrew picking system	• Editing the parameters in configuration files	N/A	• Editing the parameters in configuration files	N/A
C14	Changing the speed of AGVs in Racrew picking system.	• Editing the parameters in configuration files	N/A	• Editing the parameters in configuration files	N/A
C15	Changing the box import speed of Auto Storage System.	• Editing the parameters in configuration files	N/A	• Editing the parameters in configuration files	N/A

References

1. Hofmann, E.; Rusch, M. Industry 4.0 and the Current Status as well as Future Prospects on Logistics. *Comput. Ind.* **2017**, *89*, 23–34. [CrossRef]
2. Pokahr, A.; Braubach, L.; Sudeikat, J.; Renz, W.; Lamersdorf, W. Simulation and Implementation of Logistics Systems based on Agent Technology. In Proceedings of the Hamburg International Conference on Logistics 2008: Logistics Networks and Nodes, Hamburg, Germany, 4–5 September 2008; pp. 1–18.
3. Barbosa, J.; Leitão, P. Simulation of Multi-agent Manufacturing Systems using Agent-Based Modelling Platforms. In Proceedings of the 9th IEEE International Conference on Industrial Informatics, Lisbon, Portugal, 26–29 July 2011; pp. 477–482.
4. NetLogo. Available online: https://ccl.northwestern.edu/netlogo/references.shtml (accessed on 1 April 2020).
5. Boyes, H.; Hallaq, B.; Cunningham, J.; Watson, T. The Industrial Internet of Things (IIoT): An Analysis Framework. *Comput. Ind.* **2018**, *101*, 1–12. [CrossRef]
6. Merschformann, M.; Xie, L.; Li, H. RAWSim-O: A Simulation Framework for Robotic Mobile Fulfillment Systems. *Logist. Res.* **2018**, *11*, 1–11.
7. Hazard, C.J.; Wurman, P.R.; D'Andrea, R. Alphabet Soup: A Testbed for Studying Resource Allocation in Multi-vehicle Systems. In Proceedings of the AAAI Workshop on Auction Mechanism for Robot Coordination, Boston, MA, USA, 16–17 July 2006; pp. 1–8.
8. World Population Prospects 2019, United Nations. Available online: https://population.un.org/wpp/ (accessed on 24 July 2020).
9. Ståhl, I. Simulation prototyping. In Proceedings of the Winter Simulation Conference, San Diego, CA, USA, 8–11 December 2002; pp. 572–579.
10. Bengtsson, P.; Lassing, N.; Bosch, J.; Vliet, H. Architecture-level modifiability analysis (ALMA). *J. Syst. Softw.* **2004**, *69*, 129–147. [CrossRef]
11. Luke, S.; Cioffi-Revilla, C.; Panait, L.; Sullivan, K.; Balan, G. MASON: A Multi-Agent Simulation Environment. *Simul. Trans. Soc. Modeling Simul. Int.* **2005**, *81*, 517–527. [CrossRef]

12. GAMA Platform. Available online: https://www.media.mit.edu/tools/gama-platform/ (accessed on 1 July 2020).
13. Artisoc4. Available online: https://mas.kke.co.jp/en/ (accessed on 1 July 2020).
14. Brooks, R.J.; Tobias, A.M. Choosing the best model: Level of detail, complexity, and model performance. *Math. Comput. Model.* **1996**, *24*, 1–14. [CrossRef]
15. RaLC. Available online: http://ralc.cec-ltd.co.jp/ (accessed on 1 April 2020).
16. Xie, L.; Li, H.; Thieme, N. From Simulation to Real-World Robotic Mobile Fulfillment Systems. *Logist. Res.* **2019**, *12*, 9.
17. Cossentino, M.; Lodato, C.; Lopes, S.; Ribino, P. Multi Agent Simulation for Decision Making in Warehouse Management. In Proceedings of the Federated Conference on Computer Science and Information Systems, Szczecin, Poland, 18–21 September 2011; pp. 611–618.
18. Maka, A.; Cupek, R.; Wierzchanowski, M. Agent-based Modeling for Warehouse Logistics Systems. In Proceedings of the UKSim 13th International Conference on Modelling and Simulation, Cambridge, UK, 30 March–1 April 2011; pp. 151–155.
19. Wang, Y.; Ye, S.; Yan, G. Multi-agent System Developed for the Logistics Supply Chain Coordination and Risk Management. In Proceedings of the International Conference on E-Business and E-Government, Guangzhou, China, 7–9 May 2010; pp. 3243–3246.
20. Wang, Y. Flexible and Responsive Multi-agent Based Logistics Coordination Management. In Proceedings of the 2nd IEEE International Conference on Information Management and Engineering, Chengdu, China, 16–18 April 2010; pp. 43–47.
21. Heath, B.; Hill, R.; Ciarallo, F. A Survey of Agent-Based Modeling Practices (January 1998 to July 2008). *J. Artif. Soc. Soc. Simul.* **2009**, *12*, 1–9.
22. Bordini, R.; Hubner, J.; Wooldridge, M. *Programming Multi-Agent Systems in AgentSpeak Using Jason*; John Wiley & Sons, Inc.: West Sussex, UK, 2 October 2007; pp. 1–294.
23. Finin, T.; Fritzson, R.; McKay, D.; McEntire, R. KQML as an Agent Communication Language. In Proceedings of the Third International Conference on Information and Knowledge Management, Gaithersburg, MD, USA, 29 November–2 December 1994; pp. 456–463.
24. FIPA ACL Message Structure Specification. Available online: http://www.fipa.org/specs/fipa00061/SC00061G.html (accessed on 23 September 2020).
25. Kato, T.; Takahashi, H.; Kinoshita, T. Multiagent-based Autonomic and Resilient Service Provisioning Architecture for the Internet of Things. *Int. J. Comput. Sci. Netw. Secur.* **2017**, *17*, 36–58.