Cyber-Physical Multiagent-Simulation in Production Logistics

Christoph Greulich(✉), Stefan Edelkamp, and Niels Eicke

Institute for Artificial Intelligence, University of Bremen, Bremen, Germany
{greulich,edelkamp,neicke}@tzi.de

Abstract. A growing network of technical systems, embedded and autonomous, influence our daily work. Among them, cyber-physical systems establish a close connection between the virtual and the real world. In this paper we show how an existing multiagent system that controls the physical production of goods on a monorail is virtualized by extracting the agents as black boxes and by integrating them into a multiagent simulation system. As a result, the exact same agents run in physical and cyber world. Towards this end, the physical environment has been mapped and visualized. Experiments show that the modeling and simulation error is small, such that scenarios can be varied, tested, debugged, and scaled, saving huge amounts of labor.

1 Introduction

Production logistics has undergone a significant transformation in recent years. While in the past, mechanization and automation were clearly marked, nowadays there is a rising interest in autonomous and interconnected software solutions. The political and economic significance of this development has been associated with the name *Industry 4.0* [14].

Cyber physical systems (CPS) have been identified as transformative technologies for managing interconnected systems between its physical assets and computational capabilities [3]. In our setting, the CPS maps the digital into a real factory. Therefore, we aim at integrated planning, evaluation and continuous improvement of the essential structures, processes and resources in a real factory. We observe an increasing complexity of such systems and an increasing number of time-consuming tasks in their practical evaluation. Analytical methods are only partly sufficient for the study of such systems. Instead, *simulations* are preferred, because they often show a better mapping of the real behavior of the systems.

Even though no unified definition of *agents* exists in the literature, most authors agree that agents are autonomous software programs with certain social abilities [7]: agents can use sensors to perceive the world around them, control actuators to manipulate their environment and are able to communicate with other agents or even human users if necessary [28]. Agents use the perceived information to make decisions on their own and change the world around them

J.P. Müller et al. (Eds.): MATES 2015, LNAI 9433, pp. 119–136, 2015.
DOI: 10.1007/978-3-319-27343-3_7

to their advantage. The degree of autonomy an agent has is only restrained by its software program. The biggest advantage of agent technology in comparison to equation-based modeling is the capability of *multiagent systems* (MAS) to solve problems locally and react dynamically to occurring events [19].

Despite the growing body of research in Industry 4.0 applications, accessible MAS that run on real hardware for in-door production are rare. One of the few successful real-world implementations of a MAS is the so called Z2 production floor unit that has been developed at BIBA[1]. As individual production steps are performed at different stations, the stations are interconnected by a monorail transport system. The structure of the transport system is shown in Fig. 2. The transported goods are autonomous, which means that each product decides on its own which variant it aims at and which station to visit. This way, a decentralized control of the production system is possible [25].

At the University of Bremen, the Platform for Simulation with Multiple Agents (PlaSMA) has been developed [24]. It has mainly been applied to the simulation of macro-logistical processes. PlaSMA expands the functionality of the Java Agent Development toolkit (JADE) to a discrete-event simulation. Due to the need for simulation for CPS in production logistics, in this work we investigate, whether PlaSMA is able is accurately simulate existing production logistics scenarios by providing a model of the physical environment and by extracting existing agents without any change.

In this paper we provide a mapping of the Z2 monorail production unit to PlaSMA, leading to a cyber-physical system scenario of already existing agents and new ones; and an experimental study that shows how much executing the simulated and the real-world systems differ. The main contribution is the successful mix of existing agents as black boxes that operate seamlessly together with additional ones, substituting the hardware units by agents in the simulated world. Aspects of the agent models are detailed.

The text is structured as follows. First, we give insights to the architecture of the system and analyze the components of the real, the virtual system, as well as the agent model. We analyze the properties of the available agents from the existing physical system and introduce agents which represent the products, the stations, as well as the readers and the plant management. For the technical transformation we first explain how PlaSMA and JADE had to be adapted, and the communication unit had to be integrated. Then, we adapt the model into simulation and implement and visualize the according agents. For the validation of our approach we test the implementation in terms of concept adequateness and operational consistency. Both the physical and the simulated systems are ran and the according results are compared. Finally, we conclude and provide a brief outlook to possible future research avenues.

[1] Bremer Institut für Produktion und Logistik GmbH.

2 Multiagent System

The JADE-based agent model of the Z2 monorail production unit was designed and implemented within *Collaborative Research Center 637* [11,17]. Unlike most MAS representations of in-house logistics, the Z2 software was developed to control a real-world hardware implementation as a decentralized, heterarchical approach to achieve *positive emergence* and adaptability to changes in the environment and the goals of various stakeholders. During the development of Z2, the authors identified similarities between the *Internet of Things*[6], *Intelligent Products*[16] and MAS.

Z2 is a monorail-based assembly network for automotive tail-lights. The modular system consists of six different workstations, each is operated manually by a human worker and dedicated to one specific production step. At production steps III and V, different parts can be used to assemble different variants of the tail-lights as illustrated in Fig. 1.

At the first station, the basic metal cast parts enter the monorail on a dedicated shuttle. The monorail connects all stations, each station is assigned to one specific task, such as adding bulbs or diffusers. Each tail-light is transported from station to station until it is assembled completely. The scenario is illustrated in Fig. 2.

Low frequency RFID tags are embedded into the metal cast parts in order to identify and locate every tail light automatically. Morales-Kluge et al. [17] emphasize the dedicated *Hardware Abstraction Layer* of their implementation, which provides interfaces to various hardware machines of the scenario, including the transport shuttles.

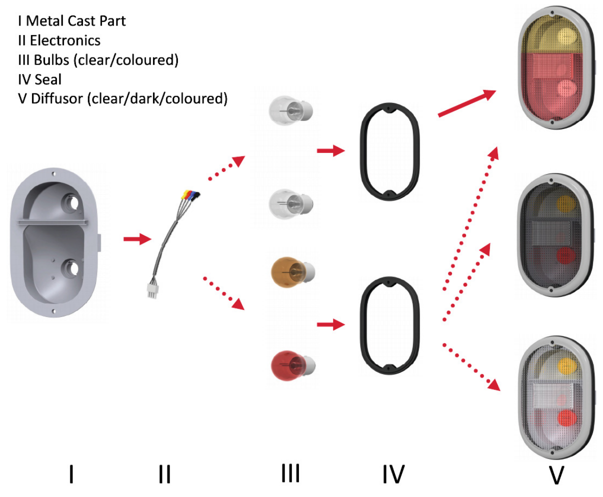

Fig. 1. Assembly states of tail lights [11].

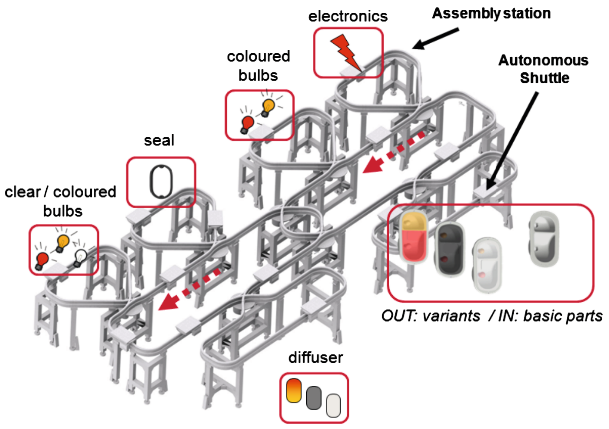

Fig. 2. Assembly scenario for tail-lights [17].

To add autonomy to this scenario, agents are applied to represent the individual stakeholders of the assembly line. A number of simple reflex agents [22] represent entities such as RFID readers which keep track of the traffic and each product's current location, information providers and even a Graphic User Interface (GUI) to supervise the agents' activities and to manually manipulate the current order status of each tail-light type variant. However, two particular agent types stick out, as they are the most relevant for the system's overall performance. The *Station Agent*, which provides an interface between the MAS and the individual worker at the respective station and the *Product Agent* which represents each individual product on the assembly line.

Each Station Agent represents one of the six workstations. The agent provides information regarding estimated waiting time for the service provided at its station and handles reservation requests by product agents. Furthermore, each Station Agent autonomously negotiates entrance and departure of product agents into the processing area where the manufacturing step takes place.

2.1 The Product Agent

The most interesting agent, however, is the Product Agent. The Product Agent is directly linked to the RFID chip embedded in the metal cast part of a tail light. When the cast is placed on an autonomous shuttle, both entities establish a connection, so that the Product Agent can order the shuttle which station to approach next.

Based on the current utilization of assembly stations and the overall order status, the Product Agent chooses the next production step and targets a type variant. After every production step, the plan is reconsidered to ensure that the current plan is still the optimal choice.

The Product Agent is a Finite State Machine (FSM) as illustrated in Fig. 3. Its behavior strongly depends on its current state. While transferring between

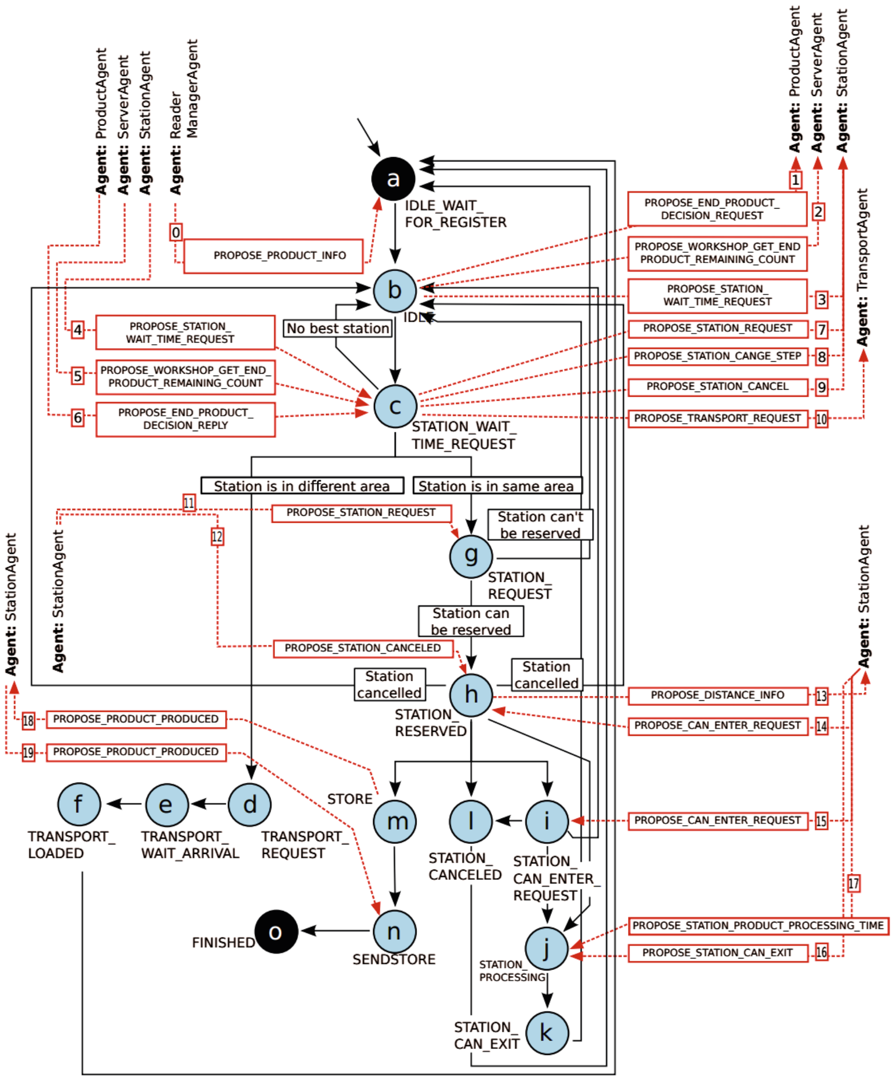
Agent: ProductAgent
Agent: ServerAgent
Agent: StationAgent
Agent: Reader ManagerAgent
a
IDLE_WAIT_ FOR_REGISTER
0 PROPOSE_PRODUCT_INFO
1 PROPOSE_END_PRODUCT_ DECISION_REQUEST
2 PROPOSE_WORKSHOP_GET_END PRODUCT_REMAINING_COUNT
3 PROPOSE_STATION_ WAIT_TIME_REQUEST
b
IDLE
No best station
4 PROPOSE_STATION_ WAIT_TIME_REQUEST
5 PROPOSE_WORKSHOP_GET_END_ PRODUCT_REMAINING_COUNT
6 PROPOSE_END_PRODUCT_ DECISION_REPLY
7 PROPOSE_STATION_REQUEST
8 PROPOSE_STATION_CANGE_STEP
9 PROPOSE_STATION_CANCEL
10 PROPOSE_TRANSPORT_REQUEST
Agent: TransportAgent
c
STATION_WAIT_ TIME_REQUEST
Station is in different area
Station is in same area
11 PROPOSE_STATION_REQUEST
12
Station can't be reserved
g
STATION_ REQUEST
Station can be reserved
PROPOSE_STATION_CANCELED
Station cancelled
Station cancelled
h
STATION_ RESERVED
13 PROPOSE_DISTANCE_INFO
14 PROPOSE_CAN_ENTER_REQUEST
Agent: StationAgent
18 PROPOSE_PRODUCT_PRODUCED
19 PROPOSE_PRODUCT_PRODUCED
f TRANSPORT_ LOADED
e TRANSPORT_ WAIT_ARRIVAL
d TRANSPORT_ REQUEST
STORE
m
l STATION_ CANCELED
i STATION_ CAN_ENTER_ REQUEST
15 PROPOSE_CAN_ENTER_REQUEST
17
PROPOSE_STATION_PRODUCT_PROCESSING_TIME
16 PROPOSE_STATION_CAN_EXIT
o FINISHED
n SENDSTORE
j STATION_ PROCESSING
k STATION_ CAN_EXIT

Fig. 3. Architecture of the product agent.

stations, the agent is in the state IDLE_WAIT_OF_REGISTER, listening for messages from RFID readers until the destination is arrived. The navigation is executed by the shuttle itself, the Product Agent only registers at RFID readers to keep track of its own location.

In the state IDLE, the Product Agent acquires all necessary information to decide which production step to take next. First, the agent decides which tail-light variant its product should become. The decision is based on the current order status for each type variant and on the previously applied production steps which may limit the amount of possible choices.

Once a decision is made, the agent decides which production step to take next and which station to approach. Therefore, the agent requests the current waiting time and queue length at each considered station and waits for replies in the state `STATION_WAIT_TIME_REQUEST`. Once the response messages were received, a MATLAB server is invoked by a direct socket connection to determine the next production step for this individual agent. If no satisfying solution could be determined, the decision-making process will be repeated.

Once a destination is chosen, the Product Agent attempts to make a reservation at the respective station and waits in the state `STATION_REQUEST` until the reservation is either confirmed or denied. In the latter case, the agent returns to the state `IDLE`.

In the state `STATION_RESERVED`, the agent constantly keeps track of the waiting time at the destination and estimates the remaining spatial distance. Furthermore, the agent handles incoming messages by the station, which provide information whether the agent is allowed to enter the processing area upon arrival or is supposed to wait in line. In the latter case, the agent continues to wait for the message in `STATION_CAN_ENTER_REQUEST` after the product arrived at the station. Furthermore, while being in the state `STATION_RESERVED`, the agent handles reservation cancellations messages, which may be sent by the station if the service cannot longer be provided.

Once the agent is allowed to enter the area where the production step is applied, the state `STATION_PROCESSING` is reached and the shuttle takes the product into the production area. The worker now applies the production step to the product while its agent waits for a message which allows the product to leave the area.

In the state `STATION_CAN_EXIT`, the Product Agent is informed by the Station Agent, whether the production step was applied or not. The Station Agent gains that information from the worker. In both cases, the Station Agent returns to the state `IDLE` and, therefore, decides again how to proceed.

After all parts are assembled, only storage stations can be chosen for reservation. In this special case, the agent registers its final tail-light type variant at the respective storage in the state `STORED`. Afterwards, the agent waits in the state `SENDSTORE` for a message which confirms that the product was removed from the assembly line by a worker. The agent then switches to the state `FINISHED` and terminates permanently.

2.2 Decision Making

The research presented in this paper was partly motivated by our interest in the application of alternative decision-making and optimization methods for this particular setup, since we consider it an interesting testbed for real-world application of AI methods. However, because of time and personnel required to run the hardware, the need for a simulation model compatible to the hardware was indispensable.

Currently, the decision-making process in this agent model is based on *hierarchical aggregation* as presented in [20]. During the process, the agent splits his overall goal into sub-goals and evaluates the effect of every possible action on each sub-goal. Depending on the current context, different sub-goals may have a different weight for the overall utility function. The overall utility value is calculated by solving a *multi-criteria evaluation problem*, where each criterion corresponds to the effect the action will have on one specific sub-goal.

In this model, knowledge is represented by a set of rules. As *decision support systems* like *analytic hierarchy process* and *weighted average* lack the ability to handle linguistic variables, *fuzzy logic* is used to express information.

Fuzzy sets and fuzzy logic are a solid research branch of computational intelligence, allowing to draw reliable inferences about vague concepts that are inherent in linguistic terms. The main idea is to have a partial membership of elements in the set, or propositions that are only partially true. In difference to probabilities, where truth is only uncertain, in such a possibilistic world, truth itself is a fractional term.

There are many applications of *fuzzy decision-making*. E.g., in Robotics, such reasoning can be used to trade conflicting behaviors like wall-following and obstacle avoidance. Fuzzy decision-making goes back to work of Bellmann and Zadeh [1]. As surveyed by Carlsson and Fuller [5], after some first doubts on the impact of fuzzy decision-making by French [8], it has shown considerable impact in the improved working of rational agents, as predicted by Gaines et al. [10].

Interestingly, in the Z2 MAS, the decision-making process based on fuzzy reasoning is completely separated from all other agents' programs and made available via a dedicated MATLAB server through a TCP/IP socket connection. The Product Agents communicate their individual perception of the environment and their respective product's states to this MATLAB server. Furthermore, the agents provide a set of available decision options to the server and receive an answer on what to do next.

3 From Reality to Simulation

The original implementation of the Z2 monorail system consists of three layers, as illustrated in Fig. 4. On top of the architecture lies the agent model layer which contains the MAS including all its agents. The agent layer is set upon a virtual model of the hardware system. The lowest level consists of the hardware itself. Since the MAS and the hardware have already been described in detail in the previous sections, we now focus on the virtual model layer.

The virtual model is a simplified model of the real-world hardware system. From the agent's perspective, the virtual model is its environment, since the agent only percepts and manipulates the virtual model and does not directly interact with the hardware layer. The virtual model encapsulates all dynamic and static features of the system in data structures accessible by the MAS.

The static features, which cannot be modified at runtime, describe the infrastructure of the system as well as its purpose. The infrastructure of the system consists of Radio Frequency Identification (RFID) reader entities, Intelligent Routing Module (IRM) sensor entities, and RFID/IRM sensor types, the various assembly stations and areas of the monorail system, the shuttles and their

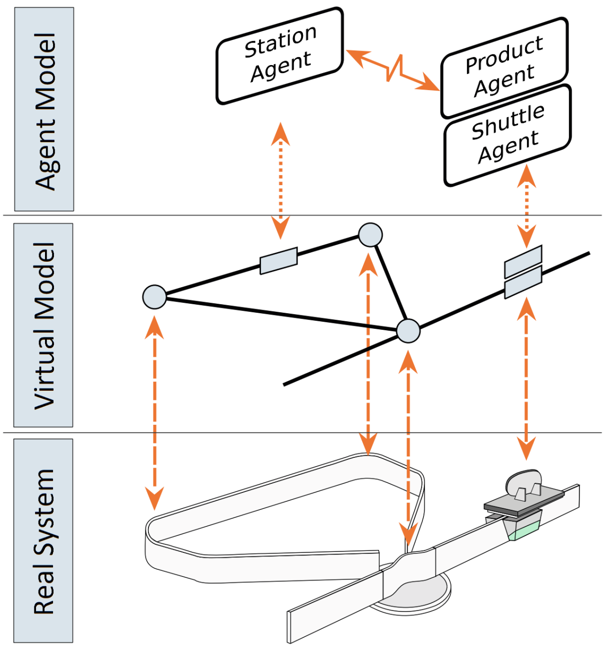

Fig. 4. The different abstraction layers.

respective IDs. Some of the IRM sensors, which are referred to as *stop modules*, require the shuttles to stop on arrival. Stop modules are interconnected by railways, which are referred to as *edges*. Furthermore, fixed routes are defined between all stop modules, given that the shuttles are not capable of executing any shortest-path search algorithms.

Since the purpose of the system is the manufacturing of tail-lights, the virtual model additionally contains information about the different parts and production steps, the interdependencies between production steps, possible part combinations (type variants) and variant groups. Furthermore, the virtual model stores information regarding the initial order status of each type variant and the interface towards the MATLAB decision-making server. The static features of the system are stored in an Extensible Markup Language (XML) file, which is made available to each agent at the beginning of its lifecycle.

Far more interesting, however, are the dynamic features of the system, which encourage application of agents and decentralized control. The dynamic features are the current order status of the different variants, the varying product states, waiting times at the various stations, and the position of products and shuttles.

While shuttles are located by the IRM system, products are located by the RFID system. Both systems update the virtual model but do not depend on each other. Hence, if a worker decides to manually remove one product from its shuttle and attach it to another one, the product will determine its new location and continue its work from the new position. Respectively, shuttles adapt to

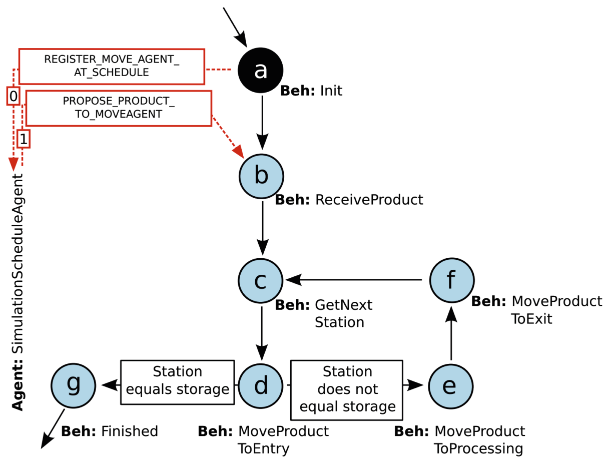
REGISTER_MOVE_AGENT_ AT_SCHEDULE
PROPOSE_PRODUCT_ TO_MOVEAGENT
0
1
Agent: SimulationScheduleAgent
a
Beh: Init
b
Beh: ReceiveProduct
c
Beh: GetNext Station
f
Beh: MoveProduct ToExit
g
Station equals storage
d
Station does not equal storage
e
Beh: Finished
Beh: MoveProduct ToEntry
Beh: MoveProduct ToProcessing

Fig. 5. Architecture of the move agent.

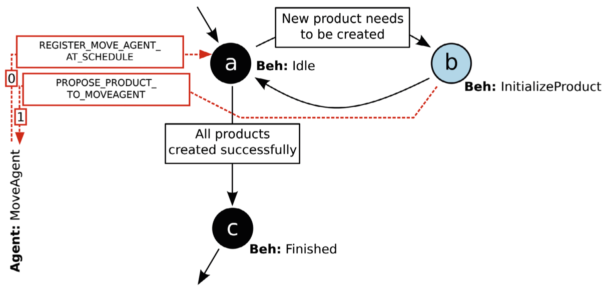
REGISTER_MOVE_AGENT_ AT_SCHEDULE
PROPOSE_PRODUCT_ TO_MOVEAGENT
0
1
Agent: MoveAgent
New product needs to be created
a
Beh: Idle
b
Beh: InitializeProduct
All products created successfully
c
Beh: Finished

Fig. 6. Architecture of the simulation scheduling agent.

a change of carriage and establish a connection with the new product. It is worth mentioning, that the positions of products and shuttles are not constantly tracked. Instead, the discrete model is updated whenever a shuttle or a product registers or deregisters at a reader.

Since dynamic reactions on changing order situations are considered one of the major advantages of digital factories [2], the order situation within the system can be manipulated by a human operator at runtime. A dedicated agent provides a GUI for the operator and updates the virtual model when the situation changes. The GUI allows to increase or decrease the amount of orders for each tail-light variant separately. The information is used by the Product Agents during their decision-making processes.

3.1 Supporting Simulation Agents

To run this MAS in a simulation environment, the underlying hardware layer has to be replaced by a software system, which adapts the interfaces between hardware layer and virtual model. The data exchange between both layers covers discrete time updates of RFID and IRM data. Therefore, a simulation model of the monorail infrastructure is of particular interest.

We adapted the hardware infrastructure into a PlaSMA-compatible graph, where nodes represent IRM sensors and RFID readers and edges represent direct railway connections between the nodes. More precisely, we derive the required information from the XML data scheme used in the original system to generate our own graph infrastructure. Each edge is weighted by the physical length of the corresponding hardware railway. Nodes representing RFID readers or railway switches are tagged with additional information regarding the waiting time per shuttle. The original hardware routing treats every connection between two nodes as a one-way railway. We adapted this behavior by defining the infrastructure graph to be directed. Furthermore, we model shuttles as virtual physical objects to be placed on nodes or edges.

Since shuttles are autonomous entities in the hardware layer, we implemented a new Move Agent type to represent one unique shuttle. The agent controls movement and speed of the virtual physical object. While pathfinding is only mocked in the original hardware system by a number of fixed paths to choose from, the simulation Move Agent applies shortest path search as presented in [13].

Like the Product Agent, the Move Agent can be seen as a FSM as illustrated in Fig. 5: After initialization, the agent receives a product to carry and moves it around to various stations depending on the Product Agent's requirements until a storage was reached and the product unloaded. Then, the agent waits for a new product to receive. The agent architecture does not allow manual displacement of the product. However, since the simulation obviously does not contain any human workers, manual displacement is obsolete.

The Simulation Scheduling Agent instantiates virtual shuttles and the corresponding Move Agents as well as Product Agents according to the simulation configuration. While shuttles are created immediately when the simulation starts, batches of new products can be created at regular intervals. The Simulation Scheduling Agent is illustrated in Fig. 6.

In order to simulate the behavior of the various RFID/IRM units, a third agent is applied, which keeps track of all sensors within the system. The agent maintains the order of waiting shuttles (first-in-first-out (FIFO)) and simulates waiting times at railway switches if the switch needs to change its routing direction.

4 Multiagent Simulation Model

As in other types of concurrent systems, issues like dead- and livelocks are often met while analyzing MAS. Concurrent process models enable reasoning the support for dynamic change and parallel execution.

There have been different formalizations of an MAS that are available in the literature. Burkhard [4] uses a formal language approach to represent a MAS M as a quadruple (A, T, τ, L), where A is a set of agents, T is a set of actions, and τ is a mapping from A to 2^T, so that for all a in A we have that

$T_a = \{t \in T \mid t \in \tau(a)\}$ is the set of actions executable in a. The set of actions sequences (solutions, plans) is denoted as $L \subseteq T^*$.

Recall that by definition for a homomorphism h we have $h(x \circ y) = h(x) \circ h(y)$. For strings $x, y \in \Sigma^*$ a homomorphism $\circ$ is the concatenation $x \circ y = xy$. The identity for this homomorphism is $\epsilon \in \Sigma^*$, the empty string.

In the formalization, homomorphisms are used as selection operators. In the overall plan each agents applies a homomorphism to select its own actions. Additionally, another string denotes which turn it is.

The *behavior* of each agent $a \in A$ is described by a homomorphism $h_a(L)$ with $h_a(l) = t$, if $t \in T_a$, and ϵ otherwise. $h_A(L) \subseteq A^*$ and *the behavior of a MAS* is characterized by a homomorphism h_A acting on the set L.

We exemplify the formalization for the Z2 demonstrator. Let $P = \{P_1, \ldots, P_k\}$ be the set of RFID locations for the shuttles, and $\texttt{move}(P_i, P_j)$ the action for moving the vehicle via the shortest path from $\langle P_i = P_{\pi(1)}, P_{\pi(2)}, \ldots, P_{\pi(n)} = P_j \rangle$ from P_i to P_j. Action $\texttt{move}(P_i, P_j)$ decomposes into $\texttt{step}(P_l, P_k)$, with P_l and P_k being adjacent.

Assume that we have two shuttle agents a_1 and a_1 and a global plan that consists of steps $S_{l,k} = \texttt{step}(P_l, P_k)$ from P_l to P_k. The empty string ϵ correspond to an action wait. We may assume, that each agent is asked in turn to resume, and answers ϵ if there is nothing to do.

Let $L = S_{1,2}, S_{1,2}, S_{2,3}, S_{3,4}, S_{2,5}, \ldots$ be the observed overall plan. The MAS behavior $h_A(L) = a_1 a_2 a_1 a_1 a_2 \ldots$ is sequence of agents defining the order to execute the actions. Hence $h_{a_1}(L) = S_{1,2}, S_{2,3}, S_{3,4}, \ldots$ is the behavior of agent a_1, and $h_{a_2}(L) = S_{1,2}, S_{2,5}, \ldots$ is the behavior of agent a_2.

It is also possible to add communication to the model. A (rendezvous) communication activity is a pair of actions (often a reading and a writing event), so that the entire communication is a string of the communication activities, modeled as a homomorphism $h_k(L)$ into $A \times A$. Communication activities can also be thought of mandatory synchronization points. While so far the order of actions is inherited by L it may be that there are several interleavings of agent executions in form of MAS behaviors that lead to the same result. For example, $L' = S_{1,2}, S_{1,2}, S_{2,3}, S_{2,5}, S_{3,4}, \ldots$ together with $h_A(L') = a_2 a_1 a_1 a_2 a_1$ yields the same outcome as $L = S_{1,2}, S_{1,2}, S_{2,3}, S_{3,4}, S_{2,5}, \ldots$ together with $h_A(L) = a_1 a_2 a_1 a_1 a_2 \ldots$.

Usually, the interpretation is unit time, so that each action takes one time step. However, this is not necessary, since we may associate a time stamp with each plan step, such that time extend and concurrency of actions can be modeled.

The $\mathcal{LORA}$ MAS formalization provided by Wooldridge [26,27] origins in model checking, with labeled transition and Kripke systems characterizing the behavior of the agents (their belief, their desire and their intention), and temporal logics expressing their required interplay, as well as the progression of knowledge. Alternatives consider an MAS as a game, in which agents –either in separation or cooperatively– optimize their individual outcome [23]. Communication between the agents is available via writing to and reading from channels, or via common access to shared variables. Other formalization approaches include work in the context of the MCMAS tool by Lomuscio[2]. Recently, there has been some approaches by Nissim and Brafman to formulize MAS as planning problems [18].

In the present paper, we define a multiagent system as an arbitrary complex concurrent computer program (that can be best thought of an ensemble of C or Java program threads). By the virtue of the Theorem of Rice [21], every non-trivial condition in such MAS (even while considering only one agent) is, therefore, already undecidable, so that essentially, we are bound to simulate MAS to get definite insights to their working. Such simulation leads to the concept of *simulation time*, which is measured in *ticks* or *cycles*, sometimes mapped to a more realistic time scale.

As in our case, quite often birth-giver agents create other agents according to some random process. We have one initial agent running: the environment agent (which can be thought as the main thread in the execution of a computer program), but as agents can be created or deleted in the course of the simulation, the number k of currently acting agents is not known in advance and has to be adapted dynamically.

Every agent (program) is further partitioned into *finite state machines* (FSM) of subprograms, whose states are called *behaviors*. Edges taken depend on the outcome of evaluating these behaviors. To ease the implementation, we assume that agents are parameterized, such that one FSM can be instantiated to many different individual agents. In object-oriented terms, each agent schema is a class, and each agent is a class object.

Together with a discrete event queue Q an MAS may now be exploited to simulate the evolution of time. The data structure Q offers the traditional set of priority queue operation of inserting, finding, and (minimum) deletion of events. Ordered wrt. timestamps it can be implemented in form of a heap. An element in Q is a triple of $(key, agent, state)$ where the key is the current timestamp of the simulation, and $(agent, state)$ is the information about where a particular agent's execution has to be resumed.

After extracting the event with minimum timestamp, we know the agent's behavior to be resumed. There might be several agents that have to execute code at the same point in time. Physically, all those agents run in different threads.

Each resumed agent knows its FSM state, where it was suspended and first looks into his inbox for messages that have arrived. After committing to all incoming messages, by calling the *action* method of the behavior, the agent tries progressing his individual task unless he reaches a point where it suspends (with each suspension a time interval is fixed, where no resuming is foreseen). While each action execution takes physical time, its simulation time is zero, so that each individual progression from a resume to a point in time for suspension requires no tick. Under certain assumptions the event-based simulation will progress, and with an upper bound on the maximum number of ticks eventually terminate.

In *multiagent simulation*, we distinguish the following three notions of time [9]: *physical time* (alias *real time*) refers to the time in the real world, i.e., at which

[2] http://vas.doc.ic.ac.uk/software/mcmas/

simulated events would happen in reality; *simulation time* models physical time in the simulation; and *wallclock time* that is consumed by the simulation system in order to execute the simulation. While physical time is continuous in multiagent-based simulation [15] simulation time is discrete, so that simulated events are mapped to timestamps. In *multiagent communication*, for the flow of information between agents, effective message handling is essential [12]. To ensure quality criteria for message handling we require

Time Model Adequacy. It is important to choose an appropriate granularity of time progression.

Causality. Due to the autonomy in accessing the message inbox, agents may process messages early. To ensure causality, message visibility has to be controlled: processing earlier messages is deferred, until local time has progressed.

Reproducibility. The ordering of messages may still depend on the system scheduling of the respective senders. For the sake of traceability, additional orderings (besides message arrival time) like the *agent's ID* are imposed.

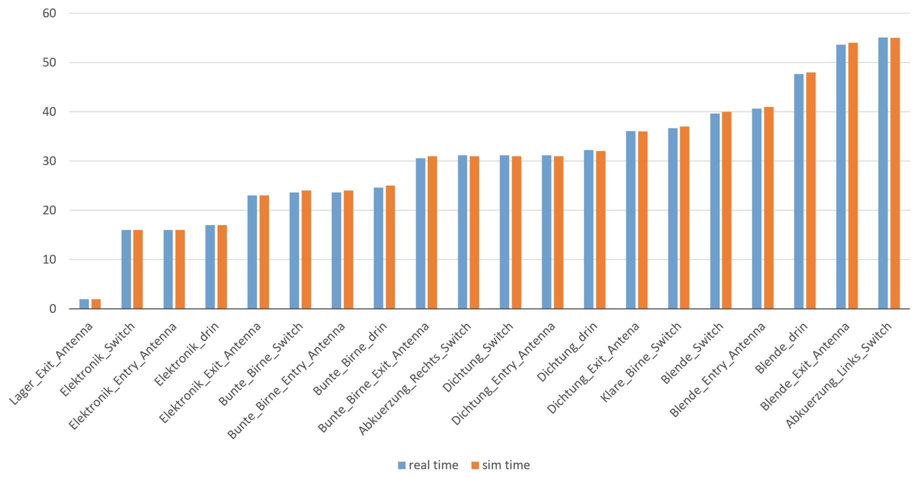

Fig. 7. Time distance between start position departure and arrival at nodes without waiting times.

5 Evaluation

The original hardware implementation runs on the Z2 demonstrator, for which the timing results are measured manually. All simulations are executed with the latest release of the PlaSMA software on a laptop computer, equipped with an Intel Pentium i7 processor and 16 GB RAM. Each agent is a Java thread, and can be profiled individually using appropriate tools. Performance indicators of

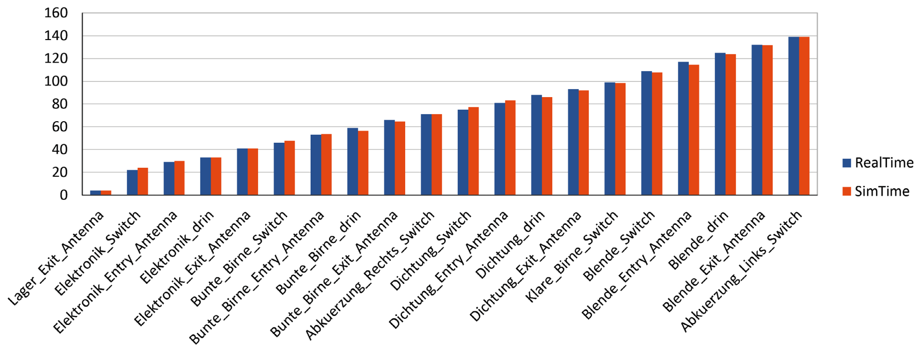

Fig. 8. Time distance between start position departure and arrival at nodes with waiting times.

the simulation have been selected and stored in a database. A virtual machine[3] contains the MATLAB server. The PlaSMA software includes a graphical user interface, where the progress of the simulation can be visualized.

5.1 Simulation Accuracy

We evaluated the simulation layer by comparing the performance of the simulation and the original hardware implementation. In both cases, we traced one of the shuttles in various target variant setups and measured physical time and simulation time distance at every stop module. In the real world implementation as well as the simulation, the shuttles move at a speed of 3 km/h.

In the first series of experiments, we neglect the time consumed by waiting at readers and switches and concentrate solely on the traveling times of the agent. The average deviation between real world physical time and simulation time is 0.216 s with a peak of 0.4 s. However, since some routes have positive and others have negative deviation, the difference between the physical and simulation time of arrival at the final station is 0.8 s. Figure 7 illustrates the results.

To validate the correct waiting behavior at sensors and switches, we conduct a second series of experiments. When waiting times are included, the total duration increases to 2.3 times the duration of the first experiment series on average. However, the deviation between simulation and the real hardware system is exactly the same as in the first series. The results lead us to believe that the deviation emerges during the acceleration of the shuttle, which is not considered in the simulation model. Figure 8 illustrates the deviation in one example setup.

In a third series of experiments, we evaluated planning and routing behavior of the Product Agents, which lead to the same results in hardware and simulation environments, given the same requirements and the same target variant. We set up check points at every hardware station and measured the physical time distance between the start of a shuttle and the arrival time at the respective check

[3] http://www.virtualbox.org

point. We repeated this experiment for every type variant and compared the results with results from simulation runs. Since we could not install a hardware measurement system in the simulation software, we recorded entering and exiting times of the product at the given station's first and last sensor. Table 1 shows an excerpt from our experiment results, indicating minor deviations, which are caused by the hardware itself. In simulation, results are deterministic, as shown in Table 2. Results show, that in any case, the check point within a station in the real world is reached at a physical time instant which lies between the simulation time instants of arrival at the entrance node and arrival at the exit node of the corresponding station in simulation. Consequently, we conclude that even though we have minor modeling errors in our simulation, they are insignificant to the simulation outcome.

Table 1. Real-world results for one single product agent per target variant.

	Target variant	Electronics	L1 (Colored Lamps)	L2 (Clear Lamps)	Seal	Cover
Exp. 1	Colored Lamps	0:32	1:00	-	1:31	2:09
Exp. 2	Colored Lamps	0:33	1:01	-	1:33	2:15
Exp. 3	Colored Lamps	0:32	1:01	-	1:33	2:12
Exp. 4	Colored Lamps	0:33	1:00	-	1:31	2:11
			⋮			
Exp. 1	Clear Lamps	0:33	-	1:47	1:15	2:19
Exp. 2	Clear Lamps	0:32	-	1:44	1:13	2:17
Exp. 3	Clear Lamps	0:32	-	1:46	1:13	2:18
Exp. 4	Clear Lamps	0:33	-	1:45	1:14	2:19
			⋮			

Table 2. Simulation results for one single product agent per target variant.

Target variant	Electronics		L1 (Col. Lamps)		L2 (Cl. Lamps)		Seal		Cover	
	Enter	Exit	Enter	Exit	Enter	Exit	Enter	Exit	Enter	Exit
Colored Lamps	00:31	00:45	00:57	01:11	-	-	01:29	01:41	02:03	02:23
Clear Lamps	00:31	00:45	-	-	01:33	01:48	01:10	01:22	02:03	02:23
					⋮					

5.2 System Performance

As mentioned before, this research was partly motivated by our interest in evaluation of other planning, decision making and optimization methods in the given manufacturing scenario. Additionally, hardware resources in the real-world setup are limited to a certain amount of shuttles and product parts. In simulation, however, the number of physical entities is only limited by the computer's physical limitations regarding RAM and CPU. Therefore, the whole manufacturing system can be simulated, tested and improved on one single machine with each component still being an individual agent program.

We conducted a series of experiments over a limited simulation time of 30 min to explore the performance of the simulation system. With each experiment, we increased the number of shuttles and/or available products to investigate the wallclock time of the simulation as well as the changes in capacity utilization of station agents and the production duration of each tail light.

Table 3. Simulation performance with increasing agent numbers (Rail and Idle refer to time spent on the rail and in idle mode, respectively, Lifecycle is averaged over the products, WC is wallclock time and TP is throughput).

Agents		MAT	Sim	Time spent processing						Lifecycle		Performance	
Shuttles	Prod.	LAB	Time	Electr.	L1	Seal	L2	Cover	Avg.	Rail	Idle	WC	TP
1	1	2	30:00	0:12	0:12	0:10	0:00	0:18	0:13	2:42	0:19	0:40	1
1	30	2.7	30:00	1:39	1:14	1:22	0:25	2:09	1:22	3:52	0:21	3:05	7
2	30	3.5	30:00	3:18	1:14	2:43	2:16	4:54	2:53	3:30	0:21	3:45	15
4	30	3.3	30:00	4:58	1:52	4:25	3:18	6:08	4:08	3:55	0:21	5:59	28
6	30	3.1	30:00	4:45	3:06	4:25	1:54	6:29	4:08	4:10	0:21	6:01	30
8	40	3	30:00	6:17	4:20	5:30	2:16	6:00	4:53	4:44	0:21	10:18	34
10	50	3	30:00	7:40	5:10	6:13	3:09	8:48	6:12	4:53	0:21	12:58	43
15	70	3.1	30:00	7:58	5:10	7:13	4:20	8:45	6:41	5:28	0:21	24:32	54
20	100	3	30:00	8:52	7:14	7:55	4:50	9:11	7:36	6:03	0:22	43:32	61
30	110	3	30:00	9:17	5:36	7:20	6:12	9:57	7:40	6:38	0:22	57:22	64

Table 3 presents an excerpt of our simulation results. The numbers show, that with an increasing number of shuttles, the various stations spend more time processing and, therefore, less time in idle mode. Furthermore, the average production time of each product increases as the products have to wait in front of the stations. Interestingly, the average number of MATLAB invocations per agent hardly increases and the maximum never exceeded 5.

With an increasing number of agents, the wallclock time of our experiments also increases. Consequently, with a certain amount of agents acting in parallel, the wallclock time actually exceeds the simulation time and, therefore, the physical time. However, since the number of available shuttles in the real world environment is limited to 12, physical time is only exceeded in experiments which could not be conducted on the hardware system. Furthermore, the time required to set up the system (30 min. approx.) and the number of human operators required to conduct experiments on the hardware system indicate that a slightly exceeding simulation time is insignificant to the overall advantage of the simulation.

6 Conclusion

In this paper we have presented a multiagent simulation of a production unit that integrates already existing agents as black boxes with a few additional virtual agents that drive the simulation. We showed that the implementation of intelligent products as agents in such cyber-physical system design is a viable option for controlling and simulating smart factories.

The decision-making process is done by frequently calling an external software server that applies advanced fuzzy reasoning methods. The obtained close match between the real and the simulated system is remarkable, given that, e.g., shortest paths are computed differently.

During the implementation process, we fixed a number of bugs both in the real and the simulated system, showing that running a simulation is also a means to improve multiagent software quality. We provided experiments showing that the implementation scales (to a rising number of shuttles and products).

In future work, we are interested in exploring distributed decision making and optimization strategies suitable for similar production units. We consider the given system as a testbed for further research in simulation and real world application and the performance of the original Z2 system as an interesting benchmarking baseline.

Acknowledgement. This research was partly funded by the International Graduate School for Dynamics in Logistics, University of Bremen, Germany.

References

1. Bellmann, R.E., Zadeh, L.A.: Decision-making in a fuzzy environment. Manage. Sci. **17**(4), 141–164 (1970)
2. Bracht, U., Geckler, D., Wenzel, S.: Digitale Fabrik: Methoden und Praxisbeispiele. Springer, Heidelberg (2011)
3. Broy, M.: Cyber-physical systems. In: Broy, M. (ed.) acatech DISKUTIERT. Springer, Heidelberg (2010)
4. Burkhard, H.D.: Liveness and fairness properties in multi-agent systems, pp. 325–330, August 1993
5. Carlsson, C., Fullér, R.: Fuzzy multiple criteria decision making: Recent developments. Fuzzy Sets Syst. **78**, 139–153 (1996)
6. Fleisch, E., Mattern, F. (eds.): Das Internet der Dinge - Ubiquitous Computing und RFID in der Praxis. Springer, Heidelberg (2005)
7. Franklin, S., Graesser, A.: Is it an agent, or just a program?: a taxonomy for autonomous agents. In: Müller, J.P., Wooldridge, M.J., Jennings, N.R. (eds.) Intelligent Agents III Agent Theories, Architectures, and Languages. Lecture Notes in Computer Science, vol. 1193, pp. 21–35. Springer, Heidelberg (1997)
8. French, S.: Fuzzy decision analysis: some criticisms. In: Zimmermann, H.J. (ed.) TIMS/Studies in the Management Sciences, vol. 20, pp. 29–44. Elsevier Science Publishers, Amsterdam (1984)
9. Fujimoto, R.: Parallel and Distributed Simulation Systems. Wiley, New York (2000)

10. Gaines, B., Zadeh, L.A., Zimmermann, H.J.: Fuzzy sets and decison analysis - a perspective. In: Zimmermann, H.-J. (ed.) TIMS/Studies in the Management Sciences, vol. 20, pp. 3–8. Elsevier Sciences Publishers, Amsterdam (1984)
11. Ganji, F., Morales-Kluge, E., Scholz-Reiter, B.: Bringing agents into application: intelligent products in autonomous logistics. In: Schill, K., Scholz-Reiter, B., Frommberger, L. (eds.) Artificial Intelligence and Logistics (AiLog) - Workshop at ECAI 2010, pp. 37–42 (2010)
12. Gehrke, J., Schuldt, A.: Incorporating knowledge about interacting for uniform agent design for simulation and operation. In: AAMAS, pp. 1175–1176 (2009)
13. Greulich, C., Edelkamp, S., Gath, M., Warden, T., Humann, M., Herzog, O., Sitharam, T.G.: Enhanced shortest path computation for multiagent-based intermodal transport planning in dynamic environments. In: Filipe, J., Fred, A. (eds.) ICAART 2013, vol. 2, pp. 324–329. SciTePress, Barcelona (2013)
14. Kagermann, H., Wahlster, W., Helbig, J.: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0. Abschlussbericht des Arbeitskreises Industrie (2013)
15. Klügl, F.: Multiagentensimulation - Konzepte, Werkzeuge. Addison-Wesley, Munich (2001). Anwendung
16. McFarlane, D., Sarma, S., Chirn, J.L., Wong, C., Ashton, K.: Auto ID systems and intelligent manufacturing control. Eng. Appl. Artif. Intell. **16**(4), 365–376 (2003)
17. Morales Kluge, E., Ganji, F., Scholz-Reiter, B.: Intelligent products - towards autonomous logistic processes - a work in progress paper. In: 7th International Product Lifecycle Management Conference PLM 2010, Bremen (2010)
18. Nissim, R., Brafman, R.I.: Cost-optimal planning by self-interested agents. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14–18, 2013, Bellevue, Washington, USA (2013)
19. Van Dyke Parunak, H., Savit, R., Riolo, R.L.: Agent-based modeling vs. equation-based modeling: a case study and users' guide. In: Sichman, J.S., Conte, R., Gilbert, N. (eds.) MABS 1998. LNCS (LNAI), vol. 1534, pp. 10–25. Springer, Heidelberg (1998)
20. Rekersbrink, H., Ludwig, B., Scholz-Reiter, B.: Entscheidungen selbststeuernder logistischer objekte. Industrie Manage. **23**(4), 25–30 (2007)
21. Rogers, H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)
22. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach, 3rd edn. Pearson Education, New Jersey (2010)
23. Saffidine, A.: Solving Games and All That. Ph.D. thesis, University Paris-Dauphine (2014)
24. Warden, T., Porzel, R., Gehrke, J.D., Herzog, O., Langer, H., Malaka, R.: Towards ontology-based multiagent simulations: the plasma approach. In: Bargiela, A., Azam Ali, S., Crowley, D., Kerckhoffs, E.J.H. (eds.) European Council for Modelling and Simulation ECMS 2010, pp. 50–56 (2010)
25. Windt, K., Böse, F., Philipp, T.: Autonomy in production logistics: Identification, characterisation and application. Robot. Comput. Integr. Manuf. **24**(4), 572–578 (2008)
26. Wooldridge, M.: Reasoning about Rational Agents. The MIT Press, Cambridge (2000)
27. Wooldridge, M.: An Introduction to Multi-Agent Systems. Wiley, Chichester (2002)
28. Wooldridge, M., Jennings, N.R.: Intelligent agents: theory and practice. Knowl. Eng. Rev. **10**(02), 115–152 (1995)