Int J Adv Manuf Technol (2011) 52:1079–1090
DOI 10.1007/s00170-010-2784-2

ORIGINAL ARTICLE

An approach for agent modeling in manufacturing on JADE™ reactive architecture

Venkateswara Rao Komma · Pramod K. Jain · Narinder K. Mehta

Received: 2 November 2009 / Accepted: 15 June 2010 / Published online: 29 June 2010

Abstract Java Agent DEvelopment framework (JADE™) is a leading platform for the development of agent-based systems that are complaint with Foundation for Intelligent Physical Agents specifications. Due to the complexity, concurrency, and dynamic nature of manufacturing, it has been an important application of agent-based systems. Application of multi-agent concept in simulation leads to the agent-based simulation. Modeling the elements of manufacturing system (such as part, machine, and AGV) in reactive agent architecture is a better way of modeling for achieving discrete-event agent-based simulation. This paper focuses on modeling of different agents in manufacturing domain on JADE reactive architecture. Modeling of different agents on a shop floor in JADE reactive architecture led to the development of a simulator known as an agent-based shop floor simulator (ABSFSim). In the modeling process, different agents in the manufacturing domain have been identified by physical and functional decomposition. Internal architecture of individual agents is finalized based on their behavioral requirements. Modeling of the agents is an important development step of ABSFSim. A randomly generated sample manufacturing system has been used for testing and demonstration of ABSFSim. The modeling details provided in this paper are useful for development of agent-based systems in manufacturing domain as well as other discrete systems.

Keywords Agent-based modeling · Reactive architecture · FIPA · JADE · AGVS

V. R. Komma (✉)
Department of Mechanical Engineering, Motilal Nehru National Institute of Technology,
Allahabad 211004, Uttar Pradesh, India
e-mail: vrkomma@gmail.com

P. K. Jain · N. K. Mehta
Department of Mechanical and Industrial Engineering,
Indian Institute of Technology Roorkee,
Roorkee 247667, Uttarakhand, India

P. K. Jain
e-mail: pjainfme@iitr.ernet.in

N. K. Mehta
e-mail: mehtafme@iitr.ernet.in

1 Introduction

Multi-agent system (MAS), an area of distributed artificial intelligence, has gained attention of researchers due to its rich set of capabilities. Manufacturing has been a leading application of MAS due to its complexity and dynamic nature [1]. Agents are autonomous, computational entities that can be viewed as perceiving their environment through sensors and acting upon their environment through effectors. A few common properties of an agent are (1) autonomy, (2) cooperativity, (3) reactivity, and (4) proactiveness [2]. Autonomy is considered as a prerequisite to be an agent by many researchers. An agent becomes intelligent due to its cooperativity, reactivity, and proactiveness.

Agent architecture is essentially a map of the internals of an agent—its data structures, the operations that may be performed on these data structures, and the control flow between these data structures. According to the characteristics of concrete agent architectures, they are classified as four classes as given below [2, 3]:

(a) Logic-based agents: In this type of agents, decision making is realized through logical deduction. These agents are useful where more reasoning is required for decision making.

(b) Reactive agents: In this type of agents, decision making is implemented in some form of direct mapping from situation to action. Behaviors of the reactive agents are implemented as situation to action with simple reasoning, and these behaviors can present in several levels. Behaviors in lower level get higher priority over higher-level behaviors of the agent.
(c) Belief–desire–intention (BDI) agents: In this type of agents, decision making depends on the manipulation of data structures representing beliefs, desires, and intentions of the agent. These agents are useful for systems that need human-like behavior.
(d) Layered agents: In this type of agents, decision making is realized via various software layers. To have reactive and proactive behaviors for an agent, an obvious decomposition involves creating separate subsystems to deal with these different types of behaviors in a hierarchy of interacting layers. Two types of control flow among the layers can be observed in layered architectures, viz., horizontal layering and vertical layering.

Suitability of agent architecture depends on the type of application. Reactive agent architecture is suitable for modeling and simulation of agents in discrete-event systems such as manufacturing domain. Reactive agents have several advantages such as simplicity, computational tractability, economy, robustness against failure, and elegance. However, the disadvantages of this type of agents are (1) an agent must have sufficient local information about its environment to take decision; (2) it is difficult to design agents that learn from experience to improve its performance over time; and (3) as number of levels of behaviors increases, it is difficult to understand the dynamics of interactions between different behaviors. An example of such agent architecture is Brooks subsumption architecture [4, 5], which is arguably the best known reactive agent architecture [2].

In agent-based modeling, in contrast to traditional top-down approach, the emphasis is on capturing the individual, together with all its limitations (both cognitive and computational) and the interaction of individuals. Agent-based simulation became an accepted methodology for developing plausible explanations for emergent phenomena (i.e., change in global behavior of the system with change in local interactions). Agent-based simulation is also used for verifying MAS design in a number of fields from social to natural sciences [6].

For developing agent-based systems, many multi-agent programming languages and platforms were developed over years. Bordini et al. [7] reported a comparison for some of the programming languages and platforms along several criteria. Selecting an agent programming language and platform is important for a specific application, since they are the limiting factors for building specific features of agents. Agent platforms differ widely in architecture and implementation, thereby impeding interoperability of agents from different software environments. To standardize the abstract architecture and communication among agents, specifications for MAS have been tried by a few organizations. Foundation for Intelligent Physical Agents (FIPA) [8] is such a non-profit organization, and its specifications are widely accepted. A few platforms such as FIPA-OS, Java Agent DEvelopment Framework (JADE) [9], Jack Intelligent Agents, ZEUS, and Grasshopper were reported in the literature, which are compliant with FIPA specifications. Among these platforms, JADE is the leading agent development platform that helps in developing FIPA-compliant agent-based systems.

The agent modeling details presented in this paper were used in the development of an agent-based shop floor simulator (ABSFSim) [10]. As the development of agent-based systems from the scratch is a tedious and time-consuming task, selecting an existing platform as a middleware is a better choice. However, many existing software platforms for multi-agent development were either specific to an application area or they did not have compliance with any agent development specifications. Reuse or extension of the reported platforms was scarce due to non-compliance with agent development specifications. It is observed in the literature that reactive architecture is used for entities in manufacturing entities followed by BDI architecture. However, modeling details of individual agents in the manufacturing domain were seldom reported in the literature.

After conducting a detailed literature survey by the authors for the capabilities of various available platforms, flexibility, ease of development, compliance with standards, and active support from its developer, JADE platform was selected for the development of ABSFSim. The main features that justify the selection of JADE platform are (1) JADE is fully compliant with FIPA specifications; (2) JADE directly supports reactive agent architecture, which is useful in modeling discrete-event entities in a manufacturing domain; and (3) JADE has a good support from its developing team. Due to simplicity and ease of development, reactive architecture is used in this paper for modeling agents in discrete-event simulation of manufacturing domain. However, individual agents are simple in design; the interaction among these agents is complex to meet the complexity of manufacturing domain. Therefore, in this paper, details of reactive agent modeling of common entities of manufacturing domain for agent-based simulation are presented which run on the top of JADE platform. To the best of our knowledge, modeling details of individual agents in the manufacturing domain which run on JADE platform were not reported in the literature. The modeling details reported in this paper are useful for developing commercial agent-based simulation tools for manufacturing domain. Availability of agent-based simulation tools for manufacturing will standardize the agent modeling and enhance the agent-based research by reducing the effort needed for modeling manufacturing systems.

2 Literature review

Since research is being carried out aggressively in agent technology, a vast literature is available on the development of concepts, architectures, interaction techniques, general approaches to the analysis, and specification of MAS. However, these contributions, which are sometimes formal but often informal, have been quite fragmented, without any clear way of "putting it all together" and thus completely inaccessible to practitioners [7]. Due to these facts, many of the published books are just a collection of several research articles.

Wooldridge and Jennings [11] gave the most general definition of agent, which also contains an extensive review of agent architectures and programming languages. A collection of answers for the question "what is an agent" can be found in an article by Müller et al. [12]. Comparison of object and agent can be found in articles by Parunak [13] and Odell [14]. Extension of Unified Modeling Language (UML) from objects to agents and representation of agent interaction protocols in UML can be found in an article by Odell et al. [15]. A well-known book in the area of MASs is by Wooldridge [2], for its wide coverage and sufficiently enlightening details. Articles by Weiss [13] and Jennings and Wooldridge [16] are also good collection of literature in the agent technology. The development of agent-based technology in different applications and its past, present, and future developments of roadmap is given in an article by Luck et al. [1]. A later article by Luck et al. [17] presents the updated status of the agent technology, its application in various areas, and several issues with statistics such as deliberative Delphi Study of impact of agent technology in different areas. In recent years, the focus of the researchers is towards the standardization of the developments in agent technology, which are essential for making them available to the practitioners [7].

A good review of literature in the area of agent-based manufacturing system (ABMS) was presented by Shen and Norrie [18], and the review was updated by Shen et al. [19]. Another article aimed at reviewing ABMS is by Monostori et al. [6]. Among the several agent architectures and platforms developed over the years, Manufacturing Agent Simulation Tool (MAST) [20] is an important agent-based simulation tool for manufacturing systems designed mainly for the simulations of material handling systems. MAST was developed on the top of JADE platform. Although modeling details of agents were not reported, MAST revealed the feasibility of building agent-based simulators for manufacturing systems on JADE platform. It is observed in the literature that reactive architecture is used for entities in manufacturing entities followed by BDI architecture.

In spite of the vast literature available in agent-based modeling, the contributions are often informal and quite fragmented. Due to the lack of compliance and non-availability of modeling details of agent development tools, their extension or reuse was scarcely reported. After conducting a detailed literature survey by the authors for the capabilities of various available platforms, flexibility, ease of development, compliance with standards, and active support from its developer, JADE platform was selected for the development of ABSFSim. Due to simplicity and ease of development, reactive architecture is used in this paper for modeling agents in discrete-event simulation. However, individual agents are simple in design; the interaction among these agents makes them meet the complexity of manufacturing domain. To the best of our knowledge, modeling details of individual agents in the manufacturing domain which run on JADE platform were not reported in the literature. Thus, this paper presents the details of reactive agent modeling of common entities of manufacturing domain for agent-based simulation to run on the top of JADE platform. The modeling details reported in this paper are useful for developing commercial agent-based simulation tools for manufacturing domain.

Agents in manufacturing domain are referred here as "manufacturing agents." Modeling details of the reactive manufacturing agents to execute on JADE platform are presented in the following sections.

3 Defining manufacturing agents and their architecture

A manufacturing system is perceived as a set of several agent types, which are defined based on physical and functional roles in it. The manufacturing agents used in ABSFSim are given in Table 1. Some of these agents such as resource agents (machine-agents and AGV-agents) and functional agents exist in the system from the start of the simulation to the end of the simulation, while others such as a part-agent start at the arrival time of the part and terminate after departure of the part.

An abstract view of JADE reactive architecture with proactive goal for a manufacturing agent is shown in Fig. 1. The architecture can be adapted for any specific type of manufacturing agent. JADE agent contains a set of "behavior" objects, which are designed for specific tasks. These behavior objects are dynamically added or removed from the agent's behavior list to make them active and

Table 1 Manufacturing agents in ABSFSim and their roles

Agent	Purpose
AGV-agent	Represents a typical AGV
Arrival-queue-agent	Represents a system arrival-queue
Departure-agent	Represents the queue at the system departure place of parts
Machine-agent	Represents a typical machine with its input and output buffers on the shop floor
Node-agent	Represents a node or control point on the AGV guidepath
Part-agent	Represents a typical job or part on the shop floor
Part-generator-agent	Represents an agent that facilitates the arrival of part-agents at the shop floor
Segment-agent	Represents a segment on the AGV guidepath

inactive, respectively. This dynamic addition or removal provides specific characteristics to the agent. When there is no active task, then these behavior objects are "blocked" (inactivated) to avoid active utilization of CPU. When an agent receives a message from another agent in its environment, a suitable behavior object is activated to handle the message. In the corresponding behavior, the content of the message is extracted, and its meaning is ascertained with the help of the corresponding ontology. While considering the agent's current state and local knowledge, a suitable action is performed by the agent that helps in reaching the goal of the agent. Result of the action is sent as a reply message to the agent that initiated the interaction protocol. Thus, manufacturing agent on JADE is reactive and proactive towards its goal.

Manufacturing agents are generally modeled by extending java class of JADE library jade.gui.GuiAgent, which facilitates GUI implementation also for the agents. A typical manufacturing agent is modeled to contain a set of behavior instances, which are extended from the JADE behavior classes. The behavior instances required for an agent depends upon its functionality and its surrounding environment. In modeling the manufacturing agents, JADE-specialized behaviors such as OneShotBehaviour, SimpleBehaviour, FSMBehaviour (Finite State Machine Behavior), and ParallelBehaviour are used to ease the development process.

4 Modeling of different manufacturing agents

The agents in the developed ABSFSim are classified under the following three categories:

1. Agent types with single instances such as part-generator-agent (functional agent), arrival-queue-agent, and departure-agent;
2. Agent types with multiple instances and long life span such as machine-agent, AGV-agent, node-agent, and segment-agent; and
3. Agent types with multiple instances and short life span such as part-agent.

Before dealing with the modeling of individual agent types, a brief idea of the attributes and behavior instances, which serve similar purpose in different manufacturing agents, is beneficial. Therefore, for a typical manufacturing agent, a few common attributes or identifiers with their purposes are given in Table 2 followed by common behavior instances with their purposes.

Common behavior instances and their role:

(a) TimerBehavior
This behavior is extended from SimulatorBehavior, which helps the agent to synchronize with the simulation clock and other agents. The TimerBehavior updates the local virtual time of the agent in-line with the global virtual time (i.e., simulation clock time) as and when required.

(b) StatisticsBehavior
This behavior internally collects and updates the required statistics of the agent in either of the following cases: (1) at regular time intervals, (2) when there is a change of state of the main behavior of the agent, and (3) when an event occurred in the agent. Variants of this behavior are used in different manufacturing agents for collecting the statistics.

(c) SimulationSpecialBehavior
This behavior handles the occasional events during the simulation such as reaching the warm-up time of the simulation, which needs the agent to clear the collected statistics, and end of the simulation, which needs to finalize the collected statistics and store them to files.

In addition to the above common attributes and behaviors (i.e., behavior objects), manufacturing agents

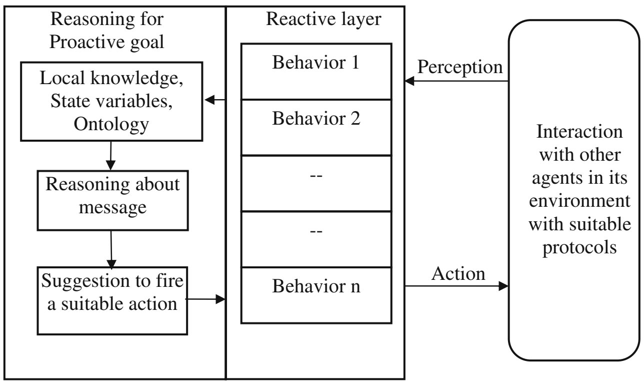

Reasoning for Proactive goal
Local knowledge, State variables, Ontology
Reasoning about message
Suggestion to fire a suitable action
Reactive layer
Behavior 1
Behavior 2
--
--
Behavior n
Perception
Action
Interaction with other agents in its environment with suitable protocols

Fig. 1 An abstract view of JADE reactive architecture with proactive goal for manufacturing agent

Table 2 Common attributes of a typical manufacturing agent and their purposes

Agent attribute	Purpose
codec	This attribute represents the message content language codec. FIPA-SL0 is used as content language codec while communicating with platform domain agents such as AMS or DF. In all other cases, FIPA-SL is used. FIPA-SL is superset of SL0, SL1, and SL2 and supports complex content expressions such as "Queries"
jadeOntology	This attribute represents the JADE-Agent-Management-Ontology, which can be used to request JADE specific operations to AMS, DF, and JADE tool agents such as Sniffer and Introspector
fipaOntology	This attribute represents the FIPA-Agent-Management-Ontology, which can be used for management of agents on FIPA agent platforms
agvsOntology	This attribute represents the AGVs-Ontology, which is specifically developed for ABSFSim and is used for communicating with the manufacturing agents on the shop floor
localEventList	This attribute represents a list that contains events that occur in the agent and is useful in achieving agent-based event-driven simulation
logger	This attribute helps in tracing the execution process of the agent and stores the information in a file that is useful in analyzing and debugging the model. Each agent stores the logged information in a different file. The level of logging the information can be set during the start of the simulation

contain several agent-specific attributes and behaviors, which are discussed below in the modeling details of different agent types.

4.1 Part-generator-agent

Part-generator-agent is a functional agent and is responsible for the arrival of different part types at the shop floor in the given part mix, volume mix, and arrival distribution. Whenever simulation time reaches the scheduled arrival time of a new part, a part-arrival-event occurs in the part-generator-agent. Consequently, the part-generator-agent sends a request to AMS agent with suitable part information (such as part type) to start a part-agent. In the next state of the part-generator-agent, it queries the newly started part-agent to inform about its complete initialization, which includes the execution of some of the basic steps such as request to place the part in arrival-queue. On receiving the confirmation from the part, the started part-arrival event is completed by the part-generator-agent. Once the part-arrival event is completed, the agent checks whether the cumulative number of parts arrived is less than the maximum permitted number of arrivals. If parts are due for their arrival, then the agent waits for the arrival time of the next part. The agent's key behavior, known as PartsLaunchFSM, is shown as State Machine Diagram of UML in Fig. 2.

4.2 Queue agents

When parts arrive at the input location, they are placed in arrival-queue. The agent that represents the arrival-queue is arrival-queue-agent. Functions of the arrival-queue-agent are (1) keep the arrived parts in queue, (2) remove the parts from the head of the queue whenever the agent receives suitable requests from the corresponding parts, and (3) notify the position of parts in the queue as simulation time advances. The arrival-queue-agent maintains two lists of agent identifiers (AIDs), one representing the parts in queue and the other representing the parts, which subscribed the arrival-queue-agent for their current position in the queue. By default, parts in queue are sorted based on their arrival times with the help of user-defined comparator to follow first-in-first-out (FIFO) discipline. Parts can also be sorted based on the part priority or any other user-defined comparators (i.e., queue disciplines). Parts subscribe the arrival-queue-agent with the help of FIPA-subscription interaction protocol.

An arrival-queue-agent has a responder behavior that receives requests from part-agents to add or remove from the queue. This behavior uses FIPA-request interaction protocol. Arrival-queue-agent has a subscription responder behavior that handles the part-agents' subscription for their position in the queue.

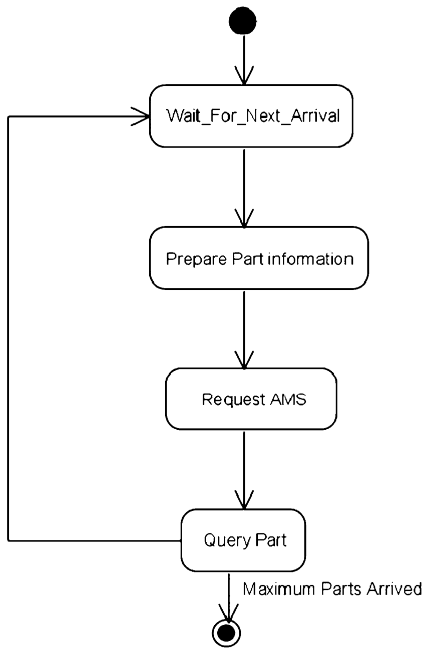


Fig. 2 PartsLaunchFSM behavior of part-generator-agent

Whenever current position of a part changes in the queue, then the corresponding part-agent is duly notified. In addition, when a part-agent cancels its subscription, it will be withheld from the notifications by removing the part's AID from the list of subscribed parts.

Similar to arrival-queue-agent, a departure-agent exists at the departure location. The departure-agent responds to the requests from the finished parts to place them in departure-queue. The agent also collects the relevant statistics of finished parts such as total number of finished parts and job completion time.

4.3 Machine-agent

A machine-agent represents a workstation that includes a machine with its input (IB) and output (OB) buffers. When a part arrives at the workstation and requests to add in IB, the machine-agent suitably places the part in the slot reserved for it in IB or directly loads it on the machine. In a typical case, where a part is placed in IB, machine-agent notifies the change in position of the part as a response to the part-agent's subscription. Working of the subscription responder behavior of machine-agent is a variant of arrival-queue-agent subscription responder behavior.

When a part is loaded on the machine, state variable of the machine-agent is set from idle to busy. After completing the operation on the part, it will be placed in OB (if space in OB is available), and the machine state is changed from busy to idle and looks for another part in IB to load on to the machine. If OB is full, then the part waits on the machine, which causes the state of the machine from busy to blocked state. When an AGV arrives at the machine to pick up a part, the part-agent requests the machine-agent to remove its identity from its current location at the machine.

Besides simulating the activities at the machine, machine-agent responds to the queries of parts about the availability of space in IB. The machine-agent reserves empty buffer slots for the parts committed for arrival to avoid any deadlocks at later stages. If an empty buffer slot is available in the IB, a part is selected from the parts waiting for the machine and is called to occupy the empty space. This pulling of parts is performed in the machine-agent by a specially developed behavior known as PullBehavior.

Some of the important attributes and their functions of the machine-agent, which is a key facility on the shop floor, are briefly discussed here. An important attribute of the machine-agent is the state variable representing machine state at any given point of time. A special attribute known as "service" is used to represent the type of operations that the machine can perform such as drilling and boring. These services are registered with the DF (Directory Facilitator) agent of the JADE platform. The DF aids part-agents to discover the machine AIDs dynamically, based on the type of services (operations) offered by the machines. Machine-agent maintains a few lists of parts' details corresponding to the parts present in IB, OB, and machine; parts committed to arrive but are in transit; parts processed at the machine; and parts waiting for the machine. Statistics of the machine, such as machine utilization and amount of blocking time, are maintained separately which will be finalized at the end of simulation.

The behaviors in machine-agent that play important role are machine finite state machine (FSM) behavior, responder behavior that places parts in IB or remove from OB, machine subscription responder behavior, pull behavior, and a special behavior known as HandlePartCNP that has a set of PartCNResponder behaviors as sub-behaviors to handle parts' bidding process for available empty slots at the machine.

Details of some of these behaviors are discussed below.

(a) MachineFSMBehavior

This is the principle behavior of the machine-agent representing three basic finite states, namely, Machine-Idle, Machine-Process, and Machine-Blocked, in the current implementation of ABSFSim. When the machine-agent enters a specific finite state, some of the machine-agent's behaviors may be restarted (activated) or reset, and other behaviors may be blocked (inactivated) or terminated to suit the current state of the machine.

(b) HandlePartCNP

This behavior is extended from the JADE ParallelBehaviour, and several PartCNResponder behavior instances are registered as the sub-behaviors. The PartCNResponder suitably proposes or declines the bids to call for proposal (CFP) from parts seeking operation at the machine under FIPA-contract net protocol (FIPA-CNP). The number of PartCNResponder sub-behavior instances required is one more than the sum of IB and machine capacities. In the current implementation of ABSFSim, machines have unit capacity.

(c) MachineSubscriptionResponder

The machine subscription responder behavior provides the flexibility of logically rearranging and informing the parts about their new position according to the selected machine scheduling. In the current implementation of ABSFSim, first-come first-served (FCFS) is used as machine scheduling policy.

(d) PullBehavior

The PullBehavior is a parallel behavior and is responsible for requesting the parts waiting for the process at the machine whenever any of the IB slots becomes empty. The behavior has HandleEmptyIB-Slots (extension of SimpleBehaviour), as a moderator behavior, which adds or removes the instances of PartPullFSM behavior according to the number of empty IB slots available. Logically, the PartPullFSM is supposed to be the initiator of FIPA-propose interaction protocol; however, to simplify the communication process, a selected waiting part is requested to restart its contract net bidding process.

4.4 AGV-agent

Modeling of AGV-agent is an important task in the development of ABSFSim due to complexity in its operation. AGV-agent modeling in the ABSFSim is specific to the assumptions made. However, the modeling approach can also be extended to any other design and operation conditions of AGV System (AGVS). The considerations made in the system are briefly introduced before the modeling details.

The ABSFSim considers unit load vehicles, which travel on a conventional guidepath layout to complete the requested transport assignment. The layout of the guidepath consists of several nodes (control points), and any two successive nodes are connected with segments (i.e., single lane guidepath). The layout can have all unidirectional or all bidirectional or mixed uni/bidirectional segments. Several decision-making issues arise during the operation of AGVs. An AGV is initially parked at a node. Under typical conditions, when a request for transportation is received form a part, the AGV starts from its current node and travels along a selected path to reach the pickup node. When the AGV reaches the pickup node, it enters the parking at the node and indicates about its arrival to the part. As most of the modern AGVs can perform automatic loading and unloading operations, the part requests the AGV for loading operation. After loading the part, the vehicle starts moving along a selected path until it reaches delivery node of the part. When the AGV reaches the delivery node, it enters the parking at the resource and indicates to the part. Consequently, the part requests the AGV for unloading operation. This sequence of steps is repeated for different transportation assignments. However, modeling this behavior of the AGV has to consider different other conditions such as vehicle traffic congestion along the segment of a path, and is the current node of AGV same as the pickup node?

This AGV behavior can be correlated with a car/cab hire service system, in which a car picks the passenger from his/her current location (on receiving a request via telephone or so) and drops him/her at their destination.

As far as the AGV-agent modeling details are concerned, an important attribute of the AGV-agent is the list of transportation tasks. AGV-agent contains several behavior instances such as AGVFSMBehavior, NegotiateWithPartResponder, CanIGo, RequestToGetReady, LeaveSegment, LeaveSidingOrParking, and LoadAndUnload. Among these behaviors, AGVFSMBehavior is the key behavior of the AGV-agent. The above behaviors are dynamically added or removed from the agent's behavior list based on the AGV state. Details of some of these behaviors and their role in the AGV-agent are discussed below.

(a) AGVFSMBehavior

The AGVFSMBehavior has several finite states registered with suitable behavior instances to obtain the required characteristics in the corresponding state of the AGV. The UML State Machine Diagram of the behavior is shown in Fig. 3. When the AGV is empty and idle, as it does not have any task to execute, the state is represented with Empty-Idle. When an empty AGV is executing a transportation request but is waiting for traffic clearance, the state is represented with Empty-Wait. The state of a moving empty AGV is represented with Empty-Move. When the AGV is loaded, the corresponding move and wait states are represented with Loaded-Move and Loaded-Wait. The states of the AGV during the part loading and unloading operation are represented with pickup and

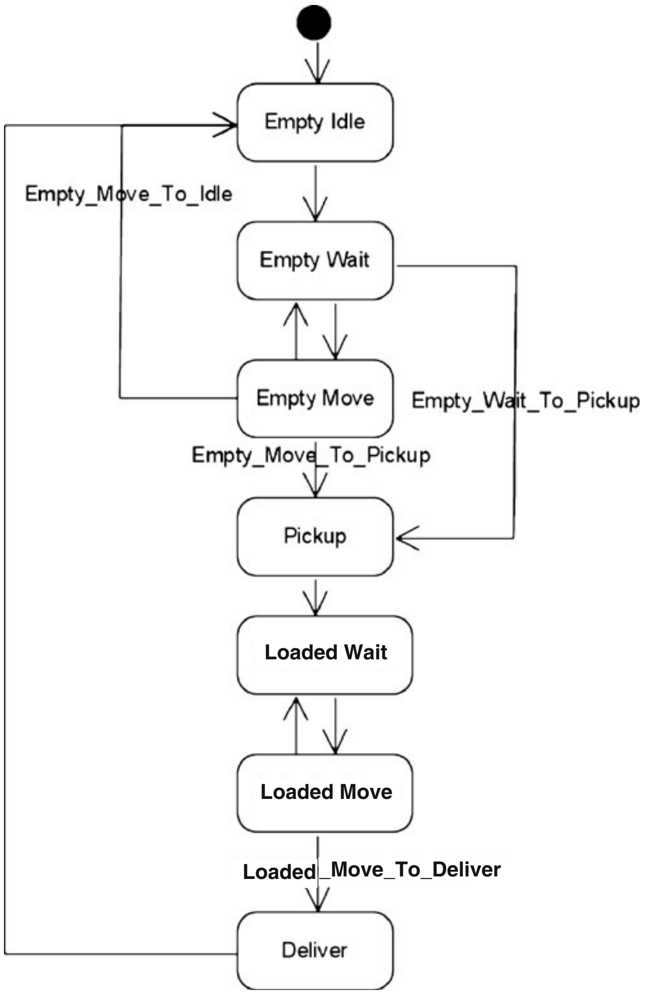


Fig. 3 State Machine Diagram of AGVFSMBehavior

delivery, respectively. The state transitions are obvious from Fig. 3, among them, some are default state transitions, and others are conditional state transitions.

(b) NegotiateWithPartResponder

This is an important behavior and handles the transportation requests from the parts.

(c) CanIGo

This behavior is used to ask permission from the nearest control point (node) to move along the segment in the selected path. Based on the vehicular congestion on the segment, a suitable response will be received.

(d) RequestToGetReady

This is the behavior used to inform a part when the AGV reaches either pickup location or delivery location.

(e) LeaveSegment

This behavior is used to inform a segment when the AGV is leaving the currently occupied segment.

(f) LeaveSidingOrParking

This behavior is used to inform a segment (if the AGV is in siding of a segment) or node (if the AGV is parked at node) when the AGV is leaving the occupied resource.

(g) LoadAndUnload

This behavior handles the loading and unloading of part at pickup and delivery locations, respectively.

(h) RespondToSegment

When an AGV is standing in either a siding or a parking and waiting for traffic clearance to move ahead in its selected path, the segment corresponding to the path may give green signal for the movement of the AGV. This behavior handles the message from the segment.

4.5 Node-agent and segment-agent

Node-agent acts as a control point to the flow of AGVs and helps in avoiding head-on collisions of the AGVs. Node at the location of a resource has a parking to accommodate the AGVs arriving at the resource for pickup or delivery purposes. It is generally considered that the parking at a node is located off the layout to avoid blocking of the traffic. A segment connecting a node will have a siding at the node. This acts as a buffer for vehicles travelling along the segment for traffic clearance; therefore, siding capacity is also a design parameter of the AGVS. Finite states of the segment-agent are unoccupied, occupied, and blocked. Blocking can occur in a single direction (partially blocked) or in both the directions (completely blocked).

When an AGV approaches a node, it seeks permission from the node-agent to move ahead in its direction. The node-agent forwards the request to the corresponding segment for further processing. The segment-agent checks the constraints, for the proposed motion of AGV, based on direction of AGV movement and current status of the segment. The segment-agent conveys the suitable decision, i.e., either permission or denial to the AGV through the node-agent. The set of behaviors of the segment-agent and the finite state transitions of the segment are shown in Fig. 4.

4.6 Part-agent

A part-agent represents any part type on the shop floor. Several part types exist together on the shop floor. While part-agent is started on JADE platform, information specific to part type such as number of processing stages, machine or process sequence, and processing times are retrieved from the input data supplied to the simulation model. In the present context, state of a part-agent represents the physical condition of the part at a particular time, while stage of the part represents the number of operations completed in the part's process sequence. A part-agent is started by AMS on the request of part-generator-agent. Based on the type of part, tailor-made behaviors are started in the part-agent. Part-agent contains several sequential and parallel behaviors such as PartBehavioralFSM, NegotiateWithMachines, CNPWithMachines, NegotiateWithAGVs, and Subscribe-Machine. These behaviors are dynamically added or removed from the part-agent's behavior list according to the corresponding state of the part-agent. Among these behaviors, PartBehavioralFSM plays an important role in the activity of the part on the shop floor.

The sequence of states that a part enters in different stages of its manufacturing is similar. However, the number of stages required to obtain a finished part depends on the part type. Therefore, the PartBehavioralFSM behavior is prepared as a set of states through which part-agents pass during their manufacturing cycle. UML state machine diagram of PartBehavioralFSM is shown in Fig. 5. Many of these finite states are registered with multi-step simple behaviors, which are themselves finite state behaviors.

Behavior classes of different manufacturing agents were derived from the JADE behavior classes. Table 3 presents the list of important behavior classes of different manufacturing agents and their super classes for understanding their basic characteristics.

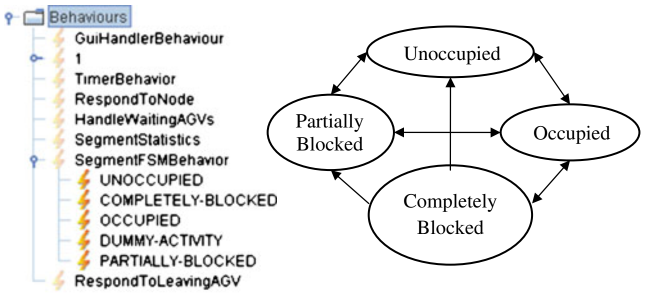


Fig. 4 Segment-agent behaviors and finite state transitions of the agent

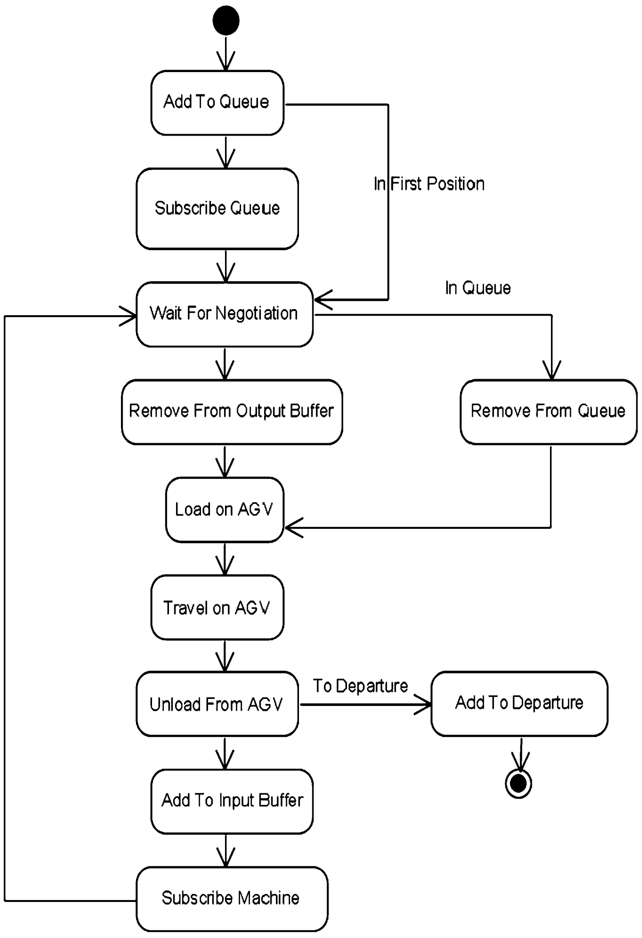


Fig. 5 State Machine Diagram of PartBehavioralFSM

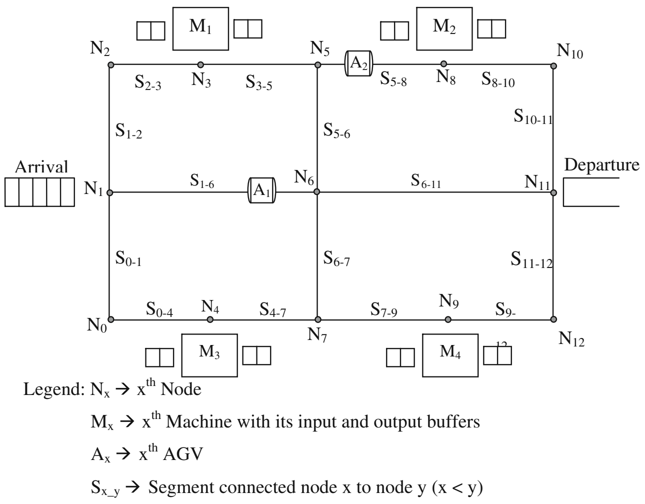


Fig. 6 Sample model of a manufacturing system for agent-based simulation on ABSFSim

5 Testing the agent-based simulator

A randomly generated sample manufacturing that was considered in Komma et al. [21] is considered here for agent-based simulation on ABSFSim, which is shown in Fig. 6. The sample manufacturing system has four machines, two AGVs, one arrival, and one departure location. Parts arrive at part arrival-queue in unit quantity at discrete points of time. Three part types were considered, and the sequence of arrival of part types is considered to take place in a cyclic manner (i.e., part types 1-2-3-1-2-3

Table 3 Some of the developed behavior classes and their super classes

Name of agent class	Name of behavior class	Name of super class (es)
AGVAgent	AGVFSMBehavior	jade.core.behaviours.FSMBehaviour
	CanIGo	jade.proto.AchieveREInitiator
	RequestToGetReady	jade.proto.SimpleAchieveREInitiator
	LeaveSegment	jade.proto.SimpleAchieveREInitiator
	NegotiateWithPartResponder	jade.proto.ContractNetResponder
	RespondToSegment	jade.proto.SimpleAchieveREInitiator
	AGVAgent.TimerBehavior	abms.simulator.SimulatorBehavior ← jade.core.behaviours.SimpleBehaviour
MachineAgent	AddRemoveResponder	jade.proto.SimpleAchieveREInitiator
	HandlePartCNP	jade.core.behaviours.ParallelBehaviour
	MachineSubscriptionResponder	jade.proto.SubscriptionResponder
	MachineFSMBehavior	jade.core.behaviours.FSMBehaviour
	PullBehavior	jade.core.behaviours.ParallelBehaviour
PartAgent	NegotiateWithMachines	jade.core.behaviours.ParallelBehaviour
	PartBehavioralFSM	jade.core.behaviours.FSMBehaviour
	ReplyToPartGenerator	jade.proto.SimpleAchieveREInitiator
	Respond2AGVRequest	jade.proto.SimpleAchieveREInitiator
	ResponderToMachineQuery	jade.proto.SimpleAchieveREInitiator
	PartAgent.TimerBehavior	abms.simulator.SimulatorBehavior ← jade.core.behaviours.SimpleBehaviour

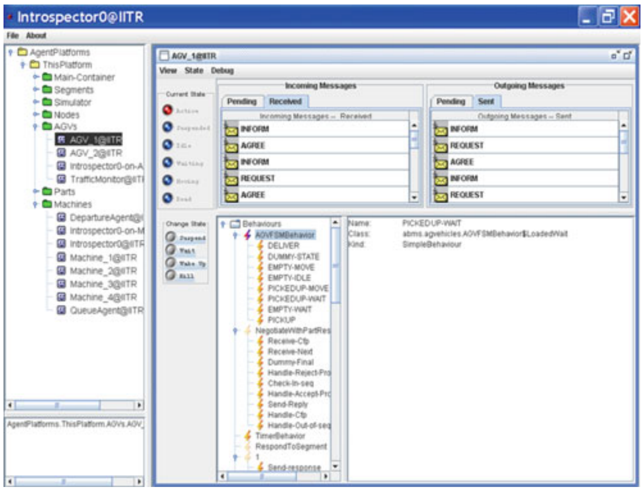


Fig. 7 Introspection of AGV_1 agent during its operation

and so on). The AGV layout has 13 nodes and 16 segments. Simulation results are collected after simulating the system for a period of 2,000 time units.

All possible paths from the current position of AGV to any other nodes in the layout are effectively determined by a recursive path-finding algorithm by searching the entire solution space. The recursive path-finding algorithm effectively discards the paths with loops. The recursive function of the algorithm returns to the calling function when it finds either the destination node or a loop or a node with dead end. All paths between the source node and destination node are sorted in ascending order of their path lengths. The recursive algorithm is repeatedly applied for all pairs of nodes in the network, and all the paths are stored in the form of tables. These tables are look-up tables that reduce the computational time of on-line controller. On failure of any segment, the affected tables can be selectively updated. However, in the current implementation of the simulator,

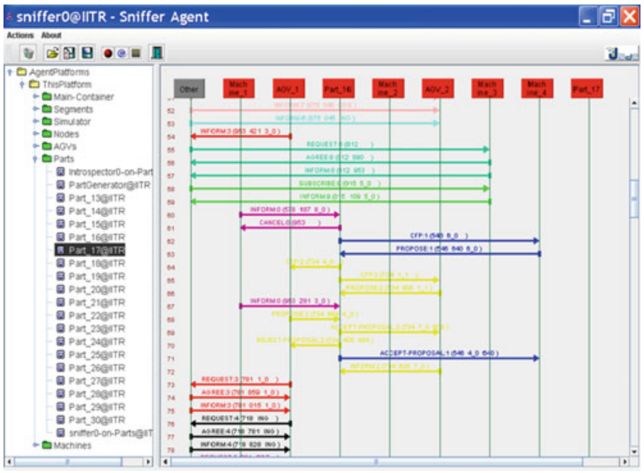
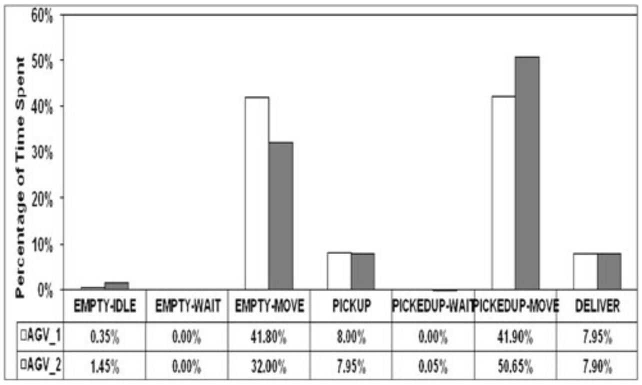


Fig. 8 Monitoring the agent communication among selected agents with JADE Sniffer

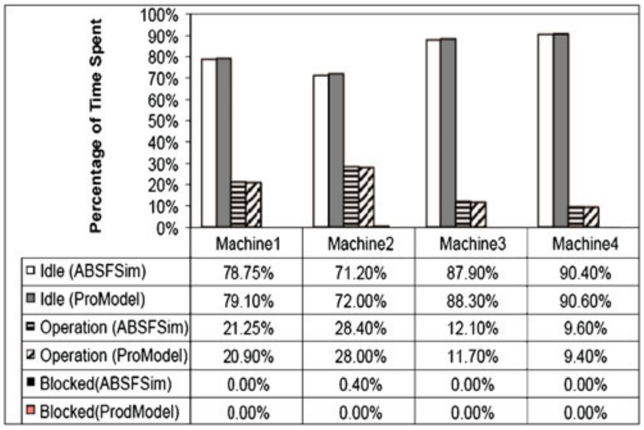


	EMPTY-IDLE	EMPTY-WAIT	EMPTY-MOVE	PICKUP	PICKEDUP-WAIT	PICKEDUP-MOVE	DELIVER
□AGV_1	0.35%	0.00%	41.80%	8.00%	0.00%	41.90%	7.95%
■AGV_2	1.45%	0.00%	32.00%	7.95%	0.05%	50.65%	7.90%

Fig. 9 Percentage of time spent by AGVs in different states in ABSFSim

AGVs follow the shortest path between the starting node and the destination node.

Whenever the simulation clock time reaches the arrival time of a new part, the part-generator-agent decides the part type and requests the AMS agent to generate a new part-agent of the given part type. During the execution, the part-generator-agent passes through the states given in Fig. 2. On arrival, the part-agent requests the queue agent to place in the queue, and the part-agent passes through the states given in Fig. 5. The number of operation stages required for a part depends on the part type. While the part progresses in the manufacturing system, the part interacts with machines and AGVs with the protocols such as Part-Machine-AGV and AGV-Node-Segment protocols reported in [21]. ABSFSim is carefully debugged with JADE tool agents such as Sniffer-agent and Introspector-agent. Snapshot of the introspection of the AGV_1 agent is shown in Fig. 7. The left pane in the snapshot indicates the agents loaded in different containers of the JADE platform. Since the inception of introspection for AGV_1, the messages received and sent by the agent appear in the upper right panes. Individual messages can be visualized



	Machine1	Machine2	Machine3	Machine4
□ Idle (ABSFSim)	78.75%	71.20%	87.90%	90.40%
■ Idle (ProModel)	79.10%	72.00%	88.30%	90.60%
▤ Operation (ABSFSim)	21.25%	28.40%	12.10%	9.60%
▨ Operation (ProModel)	20.90%	28.00%	11.70%	9.40%
■ Blocked(ABSFSim)	0.00%	0.40%	0.00%	0.00%
□ Blocked(ProdModel)	0.00%	0.00%	0.00%	0.00%

Fig. 10 Comparison of different machine states of ABSFSim in ProModel®

by double clicking on them. Active behaviors of AGV_1 agent are dynamically displayed in a lower right pane. Snapshot of the sniffer-agent is shown in Fig. 8, while monitoring the communication among different selected agents. In addition to these graphical tools, Java logging mechanism, which is the most powerful debugging tool, is used for debugging individual agents. Each agent logs the agent's execution details in a text file for debugging and analysis purpose. At the start of the simulation, the user can set the level of logging the details for an agent.

Some of the simulation output measures of ABSFSim are used for estimation of other output measures based on their interrelation. For instance, utilization of machines and average number of deliveries made by the AGVs are estimated from the number of finished parts. The cross-verification of output measures is meant for partial verification of the developed model on the ABSFSim. Similarly, the values of loaded travel time were calculated based on the process sequence and the minimum distance between the machines in the process sequence.

Total number of parts arrived during the simulation was 134. The number of parts finished by the end of the simulation was 24, 23, and 24 of part type-0, part type-1, and part type-2, respectively. Time spent by each AGV in different states of their FSM behavior has been recorded during the simulation period and is shown in Fig. 9. For obtaining higher level of confidence on the model developed in ABSFSim, the output is compared with an equivalent model developed in ProModel®, a conventional simulation environment for manufacturing. The outputs obtained from both the models are quite in good agreement; it is evident from Fig. 10, which represents the percentages of time spent in different finite states by the machines of the agent-based and conventional simulation models. The advantage of ABSFSim over the conventional simulation environments is that it provides higher flexibility for decision making in dynamic environment with the help of agent communication.

6 Concluding remarks

Agent-based modeling has a good potential to model complex systems, which are distributed, dynamic, and having concurrently behaving entities such as manufacturing domain. Agent-based simulation of manufacturing system provides a good insight of the concurrent behaving entities in the system. Agent-based modeling provides greater flexibility for dynamic decision making at different levels of shop floor control system. Better understanding of the concurrent behaving entities and greater flexibility for dynamic decision making play a vital role in enhancing the system performance and flexibility. Due to simplicity and ease of modeling, reactive agent architecture is selected in this paper for modeling of agents in manufacturing domain. The manufacturing agents are developed on JADE platform, which directly supports reactive agent architecture and fully compliant with FIPA specifications.

In this paper, modeling of agents on JADE reactive architecture is presented, emphasizing on different behaviors modeled for common manufacturing agents. Different agents identified in the manufacturing system are part-generator-agent, arrival-queue-agent, departure-queue-agent, machine-agent, AGV-agent, node-agent, segment-agent, and part-agent, and they are suitably modeled to imitate their counterparts in the real systems. The modeling approach helps in standardizing the procedure for modeling these agents. Most of the behavior classes used in manufacturing agents for achieving required characteristics are the specialized behaviors classes of JADE FSMBehaviour, ParallelBehaviour, and SimpleBehaviour (with multi-steps). Finite states of the agents are identified by critically analyzing their operation. The developed ABSFSim uses the modeling details provided in this paper along with the domain-specific ontology and different agent communication protocols including the hybrid protocols reported in [21]. A randomly generated manufacturing system is used for testing the working of ABSFSim. Verification of ABSFSim is done by comparing the model outputs with an equivalent model developed in ProModel® environment.

More research is needed in agent-based modeling of manufacturing systems thereby producing commercial agent-based simulators for manufacturing systems. Commercial agent-based simulators need enriched GUI features and inclusion of omitted states for different agents such as failure of machines and AGVs. The agent modeling approach presented in this paper can be easily extended to any other discrete-event systems, may be within or outside manufacturing domain.

References

1. Luck M, McBurney P, Preist C (2003) Agent technology: enabling next generation computing—a roadmap for agent based computing. AgentLink. http://www.agentlink.org/admin/docs/2003/2003-48.pdf. Accessed 02 Nov 2009
2. Wooldridge M (2001) An introduction to multiagent systems. Wiley, London
3. Weiss G (ed) (1999) Multiagent systems—a modern approach to distributed artificial intelligence. The MIT Press, Massachusetts
4. Brooks RA (1991a) Intelligence without reason. Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, pp 569–595
5. Brooks RA (1991b) Intelligence without representation. Artif Intell 47:139–159
6. Monostori L, Váncza J, Kumara SRT (2006) Agent-based systems for manufacturing. Ann CIRP 55:697–720

7. Bordini RH, Dastani M, Dix J, Seghrouchni AEF (eds) (2005) Multi-agent programming languages, platforms and applications. Springer Science+Business Media, Inc., New York
8. FIPA (2009) Foundation for intelligent physical agents. http://www.fipa.org. Accessed 02 Nov 2009
9. JADE (2009) Java Agent Development framework. http://jade.tilab.com. Accessed 02 Nov 2009
10. Komma VR (2009) Agent-based simulation of distributed automated guided vehicle systems. Ph.D. Thesis, Department of Mechanical and Industrial Engineering, Indian Institute of Technology Roorkee, Roorkee, India
11. Wooldridge M, Jennings NR (1995) Intelligent agents: theory and practice. Knowl Eng Rev 10:115–152
12. Müller J P, Wooldridge M, Jennings N R (eds) (1997) Intelligent agents III. Lecture Notes on Artificial Intelligence (LNAI), Vol. 1193, Springer, Berlin
13. Parunak HVD (1993) Autonomous agent architectures: a non-technical introduction. Report. Industrial Technology Institute, Ann Arbor, Michigan, USA
14. Odell J (2002) Objects and agents compared. J Object Technol 1:41–53
15. Odell J, Parunak HVD, Bauer B (2001) Representing agent interaction protocols in UML. In: Ciancarini P, Wooldridge M (eds) Agent-oriented software engineering. Springer-Verlag, Berlin, pp 121–140
16. Jennings NR, Wooldridge M (eds) (1998) Agent technology: foundations, applications and markets. Springer, Berlin
17. Luck M, McBurney P, Shehory O, Willmott S (2005) Agent technology: computing as interaction—a roadmap for agent based computing. Agentlink. http://www.agentlink.org/roadmap/al3rm.pdf. Accessed 02 Nov 2009. ISBN: 0854328459
18. Shen W, Norrie DH (1999) Agent-based systems for intelligent manufacturing: a state-of-the-art survey. Knowl Inf Syst Int J 1:129–156
19. Shen W, Hao Q, Yoon HJ, Norrie DH (2006) Applications of agent-based systems in intelligent manufacturing: an updated review. Adv Eng Inform 20:415–431
20. Vrba P (2003) MAST: manufacturing agent simulation tool. Proceedings of IEEE Conference on Emerging Technologies and Factory Automation, September 2003, Lisbon, Portugal, pp. 282–287
21. Komma VR, Jain PK, Mehta NK (2007) Agent-based simulation of a shop floor controller using hybrid communication protocols. Int J Simul Model 6:206–217