Control Engineering Practice 86 (2019) 105–117

Contents lists available at ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems

Ilya Kovalenko [*], Dawn Tilbury, Kira Barton

Department of Mechanical Engineering, University of Michigan, Ann Arbor 48109, USA

ARTICLE INFO

Keywords:
Intelligent products
Multi-agent systems
Manufacturing control
Software architectures
Intelligent manufacturing systems

ABSTRACT

The multi-agent control strategy has been previously shown to improve the flexibility of complex, dynamic manufacturing systems. One key component of this strategy is the product agent. The product agent is responsible for autonomously guiding a physical part in the manufacturing system based on its production goals. Though the product agent has been described in previous works, a fully developed software architecture for the product agent that uses a model-based optimization approach has not been proposed. In this work, a product agent architecture with the capabilities to explore the local environment, plan and schedule events based on its knowledge, and request desired actions from the resources in the system is presented and tested.

1. Introduction

The current manufacturing paradigm is shifting toward more personalized production due to the introduction of a variety of novel technologies on the plant floor. Additive manufacturing systems, smart material handling machines, and the Internet of Things, among other technologies, have the potential to allow manufacturing systems to produce a wide array of products based on customer needs (Barton, Maturana, & Tilbury, 2018; Hu, 2013; Tilbury, 2019). Manufacturing systems are integrating decentralized control strategies to utilize the flexibility of these technologies to produce new, customized products in a profitable manner (Brettel, Friederichsen, Keller, & Rosenberg, 2014; Lu, Riddick, & Ivezic, 2016; Monostori, Váncza, & Kumara, 2006). One type of decentralized control strategy that has been previously proposed to improve the flexibility and responsiveness of manufacturing systems is multi-agent control (Leitão, 2009; Vogel-Heuser, Lee, & Leitão, 2015).

In the multi-agent control strategy, a number of software agents make high-level decisions for various manufacturing system components (e.g. machines, physical parts, product orders, etc.). These decisions are determined based on an agent's goals, communication with other agents in the system, and information available from the physical system. The high-level decisions of the various agents drive the performance of the entire manufacturing system (Barbosa, Leitão, Adam, & Trentesaux, 2015; Nilsson & Darley, 2006). Thus, the design of these software agents plays an important role in understanding and improving the performance of multi-agent manufacturing system controllers.

An important agent that exists in a majority of multi-agent manufacturing architectures is the product agent (PA) (Vrba, Tichý, Mařík, Hall, Staron, Maturana, & Kadera, 2011). The PA is a software component that is responsible for fulfilling production requirements for an associated physical part through interactions with other agents in the system (McFarlane, Sarma, Chirn, Wong, & Ashton, 2003). Multiple PAs can exist in a multi-agent controlled manufacturing system, each guiding a unique physical part to fulfill its production requirements.

In order to effectively communicate with other agents in the system, the PA should understand the complex manufacturing environment and the constraints posed by both the environment and the production requirements. To capture this information, the PA can use a systematic, model-based optimization approach during its decision making process. However, to enable PA adaptability and flexibility, the models should be built and updated based on the current state of the manufacturing system. Therefore, to effectively implement a model-based approach, a well-defined software architecture must be defined for the PA (McFarlane, Giannikas, Wong, & Harrison, 2013).

The main contribution of this paper is the design and testing of a model-based PA software architecture. The proposed adaptive PA autonomously makes intelligent decisions in an unknown manufacturing environment and cooperates with existing resources to schedule and execute actions to guide parts through the manufacturing system. Additionally, the software architecture is designed for integration and implementation into existing multi-agent controllers. This work builds on the previously proposed PA architecture presented in Kovalenko, Barton, and Tilbury (2017). The primary contributions of this paper over the previous work include: (1) a description of exploration components that learn about the associated physical part's local environment,

[*] Corresponding author.
E-mail addresses: ikoval@umich.edu (I. Kovalenko), tilbury@umich.edu (D. Tilbury), bartonkl@umich.edu (K. Barton).

https://doi.org/10.1016/j.conengprac.2019.03.009
Received 5 September 2018; Received in revised form 13 March 2019; Accepted 15 March 2019
Available online 27 March 2019

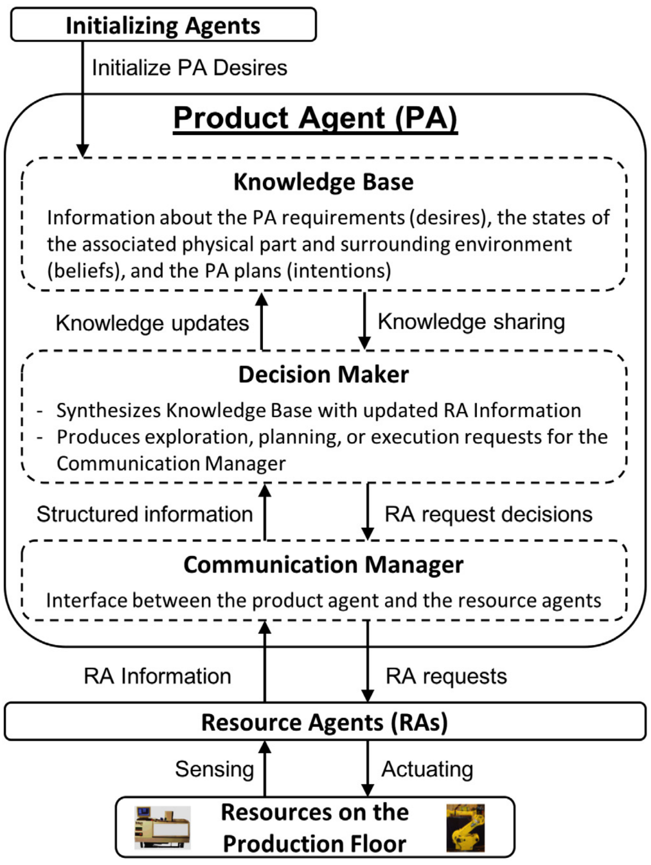

Initializing Agents
Initialize PA Desires
Product Agent (PA)
Knowledge Base
Information about the PA requirements (desires), the states of the associated physical part and surrounding environment (beliefs), and the PA plans (intentions)
Knowledge updates
Knowledge sharing
Decision Maker
- Synthesizes Knowledge Base with updated RA Information
- Produces exploration, planning, or execution requests for the Communication Manager
Structured information
RA request decisions
Communication Manager
Interface between the product agent and the resource agents
RA Information
RA requests
Resource Agents (RAs)
Sensing
Actuating
Resources on the Production Floor

Fig. 1. An abstracted software architecture for the product agent with general descriptions of each component and the shared information.

(2) the utilization of optimization-based planning components to schedule future resource actions, and (3) a demonstration of the PA behavior within a simulated manufacturing facility.

The rest of the paper is organized as follows. Background information regarding the utilization of product agents in manufacturing systems is discussed in Section 2. A high-level overview of the PA is presented in Section 3. Section 4 describes the desires, beliefs, and intentions of the PA. The PA's decision making and communication structure are discussed in Section 5. In Section 6, a simulation case study with the proposed PA architecture is presented. Concluding remarks are presented in Section 7.

2. Background

A wide variety of multi-agent architectures have been introduced to control industrial systems (Leitão, 2009; Leitão, Mařík, & Vrba, 2013; Shen, Hao, Yoon, Norrie, Joong, & Norrie, 2006). Most of these proposed system-level architectures contain an instance of the product agent (PA) (Vrba et al., 2011). This section describes the representation and functionality of the PAs for some of these system-level architectures.

PROSA (Product-Resource-Order-Staff Architecture) is one of the first examples of a decentralized, distributed control architecture developed to improve manufacturing system flexibility (Van Brussel, Wyns, Valckenaers, Bongaerts, & Peeters, 1998). In this system-level control architecture, the Order Holon, the component most resembling the PA, is responsible for tracking the state of the physical product and initiating work requests. The Order Holon stores the state of the physical product, the progress of the current tasks (i.e. events), and the historical progression of accomplished tasks. Once initialized, the Order Holon is able to request a process plan, request an optimal schedule from a scheduling agent, and cooperate with other holons (i.e. agents) in the system. While the general behavior and communication of the Order Holon is described, only a brief, general description of the stored information and decision making of the Order Holon is provided.

Another example of a multi-agent architecture for manufacturing is PABADIS (Plant automation based on distributed systems) (Lüder, Peschke, Sauter, Deter, & Diep, 2004). The proposed system-level control architecture was developed to improve the flexibility and scalability of manufacturing systems by making some agents, such as the PA, more autonomous and active during the decision making process of the system controller (Lüder, Klostermeyer, Peschke, Bratoukhine, & Sauter, 2005). In Lüder et al. (2005), the PA life-cycle is identified as (1) creation, (2) scheduling, (3) migration, (4) execution, and (5) termination. This life-cycle is adopted for our proposed, adaptive PA architecture. Similar to PROSA, while the general behavior of the PA is described, the internal architecture and decision making of the PA are not described in detail.

A recently proposed system-level control architecture that was designed in a formal manner is ADMARMS (Farid & Ribeiro, 2015). ADMARMS used qualitative design principles to identify the basic knowledge and communication necessary for agents in reconfigurable manufacturing systems. In Farid and Ribeiro (2015), the requirements for the stored information, functionality, and communication capabilities of the ADMARMS PA are provided. However, while the required PA data structures are presented in detail, a decision making strategy for the PA that uses this information was not provided. The data structures and PA-RA (resource agent) communication requirements introduced in ADMARMS were taken into consideration when developing our proposed, adaptive PA.

ADACOR (ADAptive holonic COntrol aRchitecture), is one of the first architectures to provide a formal specification to describe the behavior of its proposed agents (Leitão & Restivo, 2006). The behavior of the Product Holon, the component similar to a PA in the ADACOR architecture, is specified using a Petri Net (Leitão, Colombo, & Restivo, 2006). Additionally, a general structure and an algorithm to stabilize the behavior of a holon is described in ADACOR^2, a future iteration of the multi-agent architecture (Barbosa, Leitão, Adam, & Trentesaux, 2013; Barbosa et al., 2015). Although the general guidelines of the Product Holon were provided, it appears that the decision making of the Product Holon was driven by a set of rules.

These four multi-agent system-level architectures (PROSA, PABADIS, ADMARMS, and ADACOR) focus on developing the required agents and the necessary communication to effectively control manufacturing systems. Some more examples and implementations of system-level multi-agent architectures with PAs can be found in Leitão, Rodrigues, Barbosa, Turrin, and Pagani (2015), Marchetta, Mayer, and Forradellas (2011), Meyer, Hans Wortmann, and Szirbik (2011), Ribeiro, Ocha, Veiga, and Barata (2015), Tang, Li, Wang, and Dong (2017). Since the focus of these architectures is to construct all of the agents in the system, the decision making of specific agents (e.g. the PA) is usually rule-based reasoning. While rule-based reasoning is a viable control strategy, it is difficult to scale these reasoners to larger systems with numerous constraints (e.g. a complex manufacturing systems). An alternative control strategy is to use a model of the system and optimization techniques to drive the decision making of the PA.

A PA with a Markov Decision Process (MDP) model has been previously proposed in Ansola, García, de las Morenas, and de las Morenas (2015). In De Las Morenas, Garcia-Higuera, and Garcia-Ansola (2017), this MDP-based PA is implemented to guide a part in a realistic manufacturing testbed. The proposed PA makes decisions using a policy iteration algorithm on a provided MDP model. While the MDP-based PA seems to effectively guide a part through the system, the model of the entire system must be initially provided to the PA. Therefore, for large-scale manufacturing systems, the building of the model becomes a non-trivial task. In addition, the construction of the software architecture for the PA was not described in detail, as the data structures and communication interface for this PA were not fully provided.

A PA with a Finite State Machine (FSM) model was proposed in our previous work (Kovalenko et al., 2017). In this work, the simplified software architecture for the PA was provided. In addition, a decision making function was developed based on the cost of each state in the FSM. The FSM is compiled from information provided to the PA by other agents in the system. This composition was based on a previously existing algorithm described in Gouyon, Pétin, Morel, Et, and al. (2007).

While various multi-agent architectures have been proposed for control of manufacturing systems, these works have not focused on the development of the decision making of individual components for the system. However, the communication and behavior requirements for each of the agents was specified in detail. A few works have focused on developing model-based PAs that use optimization to make decisions. These model-based architectures can be used to systematically design the decision making of PAs. However, the software architecture of the proposed model-based PAs are not outlined in full detail. This work provides a software architecture for model-based, adaptive PAs that fulfills the existing communication and behavior requirements identified in other works.

3. Product agent overview

The product agent (PA) is the controller that is responsible for fulfilling production requirements for a physical part in the manufacturing system. Due to the lack of sensing and actuation components on the physical part, the PA must obtain information and request actions through communication with Resource Agents (RAs). The RAs serve as high-level controllers for various machines and devices on the factory floor for multi-agent control of manufacturing systems (Lepuschitz, Zoitl, Vallee, & Merdan, 2011).

An architecture for multi-agent, manufacturing controllers has been previously developed for Resource Agents in the manufacturing system (Lepuschitz et al., 2011). This RA architecture contained a Communication Management component, a Decision Making module, a World Mode Repository, and a Low Level Interface. The RA architecture was described in more detail and implemented in Rehberger, Spreiter, and Vogel-Heuser. The Communication Management, Decision Making, and World Model Repository components were adapted and redesigned for the PA architecture proposed in this work.

The proposed PA architecture consists of the following components: the Knowledge Base, the Decision Maker, and the Communication Manager. The relationship between these modules is shown in Fig. 1. The Knowledge Base contains beliefs, desires, and intentions of the PA. The Communication Manager provides the link between the PA and the RAs in the system. The Decision Maker integrates the information from the Knowledge Base and the Communication Manager to make decisions regarding action requests. The proposed PA architecture goes through the following lifecycle:

1. Creation and goal initialization
2. Operation (i.e. exploration, planning, and execution)
3. Archiving

When the PA is created, the PA goals must be initialized to match the goals of the associated physical part. These production goals should include the manufacturing processes to be completed. The goals are initialized by another intelligent agent or entity (e.g. a Manufacturing Execution System or a human operator) that understands the requirements of the associated physical part for the specific manufacturing system.

Once an RA sends a message to the PA about the initial state of the associated physical part (e.g. the part has been moved to a certain location), the PA begins the operation phase. During this phase, the PA takes initiative and cycles through the following three tasks:

Table 1
Nomenclature for the PA Knowledge Base.

Desires	
p_d	A desired physical property
P_d	Process plan
RA_{exit}	Agent to remove part from the system
e_{exit}	Exit event to be queried
Beliefs	
M_h	The product history
M_e	The environment model
X	A set of states of the physical part
x	One state of the physical part
E	Set of manufacturing system events
e	One manufacturing system event
Tr	State transition function
Prp	Map of states to physical properties
Ag	Map of events to the associated agent
x_c	Current state of the physical part
s_h	Sequence of past events for the part
Met_e	Set of mappings of events to metrics
$T(e)$	Time duration of an event
Intentions	
$Plan$	The agent plan
s_p	The string of planned events
Te_p	Map of events to start and end times

- Exploring the system: Query local RAs to understand if a manufacturing process can be accomplished
- Planning: Find a sequence of desired resource actions and request appointments for resource utilization
- Execution: Request desired events from RAs

The decision making process for choosing whether the PA should explore, plan, or execute is described in Section 5.1. The specific details regarding each of the three tasks are described in Section 5.2 to 5.4.

Finally, the PA is archived once it is removed from the manufacturing system. It can be removed from the manufacturing system if all of the desired physical properties in the process plan have been completed or if the PA decides that it cannot find a feasible solution for the next step in the process plan.

A detailed implementation of the proposed PA architecture, including specific components and component-to-component information exchange, is shown in Fig. 2.

4. Product agent knowledge base

Intelligent agents must be able to perceive and respond to a changing environment (reactivity), take initiative to achieve their individual goals (proactiveness), and interact with other agents in the system (social ability) (Wooldridge, 2009). A number of functional architectures have been proposed to design intelligent agents (Russell & Norvig, 2009; Wooldridge, 2009). One widely-used functional architecture that has been used in various agent applications is the belief-desire-intention (BDI) architecture (Howden, Rönnquist, Hodgson, & Lucas, 2001). The BDI architecture provides a modular framework to design intelligent agents based on the ideas behind human practical reasoning (Georgeff, Pell, Pollack, Tambe, & Wooldridge, 1998). The architecture proposes partitioning the agent's information of the surrounding environment (beliefs), the goals that the agent would like to achieve (desires), and the plans that the agent creates (intentions) into separate modules. Utilizing these modules, a decision making algorithm is built for the agent to intelligently interact with the surrounding environment. In this work, a BDI framework is used to develop the PA Knowledge Base.

The Knowledge Base is composed of the desires, beliefs, and intentions modules. This section describes each of these modules in more detail. Table 1 provides the nomenclature used in this section.

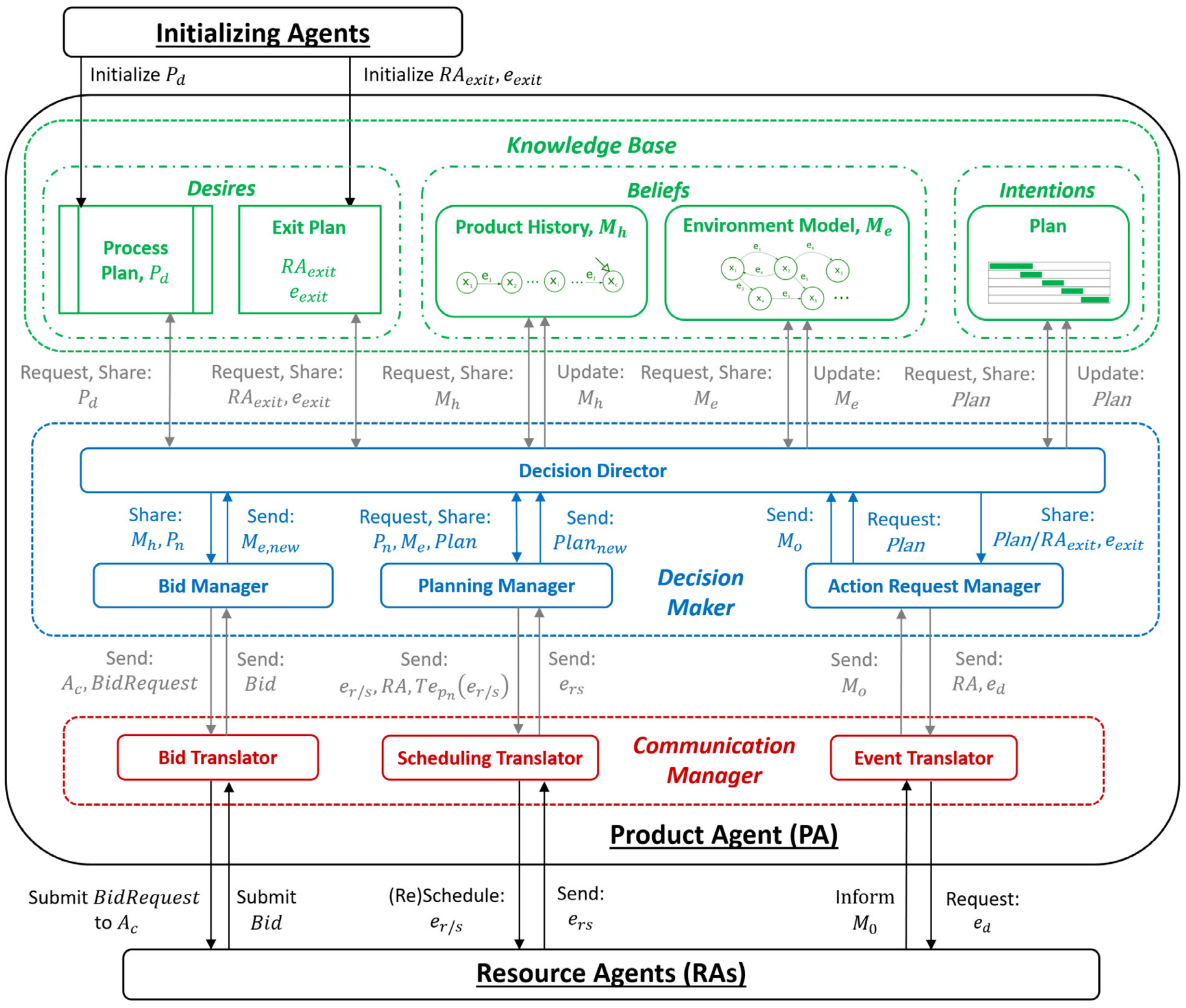


Fig. 2. The full implementation of the product agent architecture. The components of the Knowledge Base, Decision Maker, and Communication Manager are identified. In addition, the figure illustrates the communication between each of the components. The information transfer between the product agent and other agents is also displayed.

4.1. Desires

The desires represent the requirements for the associated physical part. As shown in Fig. 2, the PA desires include a process plan and an exit plan. The process plan contains all of the desired manufacturing processes that must be performed on the associated physical part as well as the locations that must be visited. The exit plan contains the information that the PA uses to make the physical part leave the manufacturing system. Both the process plan and exit plan can be accessed, but not modified, by the Decision Director.

4.1.1. Process plan

A process plan, P_d, is an ordered list of sets of desired physical properties. The PA is responsible for requesting actions from the RAs that will help the physical part attain these desired physical properties. A physical property can be a location in the manufacturing system or a completed manufacturing process. Thus, the process plan can be represented as the following ordered list:

$P_d = (P_{d1}, P_{d2}, \dots, P_{dn})$, where

$$P_{d1} = \{p_{d11}, \dots, p_{d1a}\}$$
$$P_{d2} = \{p_{d21}, \dots, p_{d2b}\}$$
$$\vdots$$
$$P_{dn} = \{p_{dn1}\}$$

where p_d represents a desired physical property. In this representation, the sets of physical properties are ordered. For example, the physical part must accomplish all of the physical properties in the first set, P_{d1}, before trying to achieve the second set of properties, P_{d2}. The physical properties in each set (e.g. $p_{d11}, \dots, p_{d1a}$) can be done in any order. The final set of physical properties, $\{p_{dn1}\}$, will usually represent the final location for the physical part to leave the manufacturing system.

4.1.2. Exit plan

The exit plan represents an alternative goal for the PA if the process plan cannot be accomplished. The exit plan consists of the name of an agent, RA_{exit}, and an exiting event call, e_{exit}. The PA should be able to request e_{exit} from RA_{exit} if it is unable to find a feasible plan based on the explored environment. More information regarding the decision to exit the system is discussed in Section 5.1.

4.2. Beliefs

The beliefs represent the PA's current understanding of the physical world. The beliefs consist of the product history and an environment model. The product history is the representation of the past and current states of the physical part. The environment model is an abstraction of the capabilities of the current local manufacturing environment. Both the product history and the environment model are represented

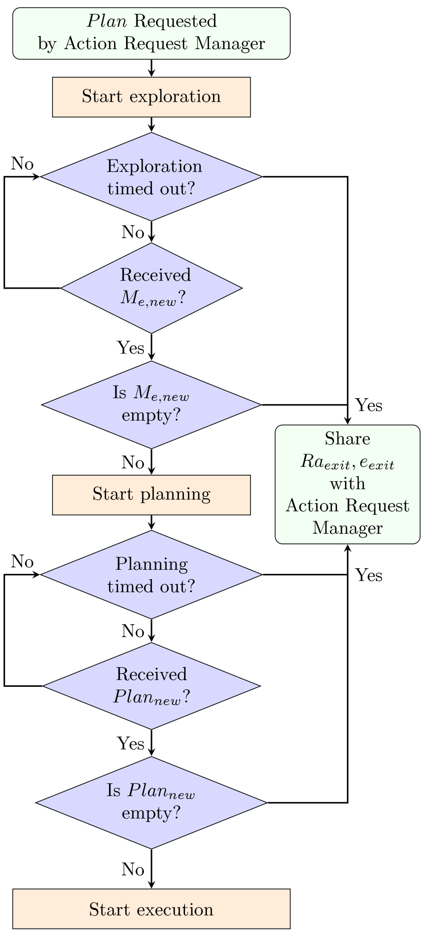


Fig. 3. The flowchart showing how the Decision Director chooses whether to explore, plan, or execute.

via finite state machines (Cassandras & Lafortune, 2009). The product history and environment models consist of states that can be mapped to the physical properties of the associated part (e.g. "Part at Storage", "Part with Rounded Corners", etc.) and events that represent resource actions (e.g. "Move to Machine", "Run Milling Program", etc.). An example of a similar model can be found in Kovalenko et al. (2017). To build and update both of these models, the fusion operation is utilized.

4.2.1. *Product history*

The product history contains information regarding the previous and current states of the associated physical part. The product history model, M_h, is updated based on continuous information provided by the RAs through the PA's execution components.

M_h is defined as the following tuple:

$M_h = (X_h, E_h, Tr_h, Prp_h, Ag_h, x_c, s_h)$, where

$X_h = \{x_0, \dots, x_n\}$: a set of states of the physical part
$E_h = \{e_0, \dots, e_m\}$: a set of events
$Tr_h : X_h \times E_h \to X_h$: a state transition function
$Prp_h : X_h \to P_x$: a function that maps the state to its physical properties
$Ag_h : E_h \to RA$: an event-agent association function between each event and the RA that performs that event
$x_c \in X_h$: the current state of the physical part
$s_h = e_0 e_1 \dots e_l$: a string (sequence of events) that represents events that have occurred

4.2.2. *Environment model*

The PA's unique representation of the current reachable states of the physical part is the environment model. The model of the local environment, M_e, is constructed using the exploration components of the architecture. M_e is updated as the physical part traverses the manufacturing system using the PA execution components. More details regarding the exploration and execution components are discussed in Sections 5.2 and 5.4, respectively.

The M_e is defined as the following tuple:

$M_e = (X_e, E_e, Tr_e, Prp_e, Ag_e, Met_e, x_c)$, where

$X_e, E_e, Tr_e, Prp_e, Ag_e, x_c$ follow the definitions of the product history, M_h
$Met_e = \{f_1, \dots, f_n\}$, where $f_i : E \to \mathbb{R}, 1 \le i \le n$. Met_e is a set of functions that map events to numerical metrics (e.g. time duration, quality, etc.)

Note that the time duration, $T(e)$, is required to be one of the functions in Met_e for the PA's planning components, as discussed in Section 5.3.

Both the product history and environment model are used by the PA to explore the local environment, plan, and execute desired events in the manufacturing system.

4.2.3. *Fusion operation*

A fusion operation (Pétin, Gouyon, & Morel, 2007) is used by the PA to update both the product history and environment model. Given two tuples, M_1 and M_2, that contain the following elements (X, E, Tr, Prp, Ag), the fusion operation performs the following steps to create a new tuple, M_{new}, in the following manner:

1. Remove all events from M_2 that are part of M_1, i.e. remove all $e \in E_2$ from E_2 if $e \in E_1$
2. Perform the following fusion of M_1 and M_2:

$$X = X_1 \cup X_2 \text{: a union of all the possible states}$$
$$E = E_1 \cup E_2 \text{: a union of all of the events, with } E_1 \cap E_2 = \emptyset$$
$$Tr : \begin{cases} Tr_1(x,e) & \text{if } x \in X_1 \text{ and } e \in E_1 \\ Tr_2(x,e) & \text{if } x \in X_2 \text{ and } e \in E_2 \end{cases}$$
$$Prp : \begin{cases} Prp_1(x) & \text{if } x \in X_1 \\ Prp_2(x) & \text{if } x \in X_2 \end{cases}$$
$$Ag : \begin{cases} Ag_1(e) & \text{if } e \in E_1 \\ Ag_2(e) & \text{if } e \in E_2 \end{cases}$$

Output the new tuple: $M_{new} = (X, E, Tr, Prp, Ag)$.

The fusion operation is utilized during PA exploration and execution. During exploration, the fusion operation is used to combine bids from RAs to create the environment model. During execution, this operation is used to incorporate information about the state of the physical part into the PA's product history. Specific details are described in Sections 5.2 and 5.4, respectively.

4.3. *Intentions*

The intentions of the PA are represented using a plan. The plan consists of a sequence of events that the PA has scheduled based on its desires and beliefs. It is defined as follows:

$Plan = (s_p, Ag_p, Te_p)$, where

$s_p = e_0, \dots, e_n$: the planned string (sequence of events)
$Ag_p : E_p \to RA$: the event-agent association function
$Te_p : E_p \to (\mathbb{R}_+, \mathbb{R}_+)$: a function that maps events to start and end times

Note that the start and end times in the agent plan are based on a clock that is available to all of the agents in the system.

This plan is constructed using the planning components of the PA architecture. Following its construction, it is utilized by the PA's execution components to request specific events from the RAs. More information regarding plan construction and utilization can be found in Sections 5.3 and 5.4, respectively.

5. Product agent intelligence

The decision making and communication aspects of the PA are described in this section. Section 5.1 describes how the Decision Director decides whether to explore the surrounding environment, make plans based on its beliefs, or start requesting actions from RAs. Sections 5.2 through 5.4 go into more detail on how each of the three tasks (exploration, planning, and execution) are performed using the components of the architecture (i.e. Knowledge Base, Decision Maker, Communication Manager).

5.1. Decision director

The Decision Director is responsible for making high-level decisions regarding whether the PA should explore, plan, or execute based on its current information. Fig. 3 shows the decision flow chart for the Decision Director

The Decision Director's process begins when a new plan is requested by the Action Request Manager. The request for a new plan can occur when the PA is first created (since there is no initial plan), finishes the current plan, or is notified of an unplanned event. More information regarding each of those possibilities is discussed in Section 5.4. Once this request is received, the Decision Director starts PA exploration.

The PA finishes exploring when the Decision Director receives a new environment model, $M_{e,new}$, or the exploration takes too long to conclude. If $M_{e,new}$ is empty (i.e. no set of RAs are able to take the associated physical part to a desired state) or the exploration timeout is reached, the PA will request to remove the associated physical part from the manufacturing system. This exit strategy is executed by sending the exit plan to the PA's Action Request Manager. If an exit strategy is not employed, the Decision Director starts PA planning.

Similar to exploration, the PA stops planning when the Decision Director receives a new agent plan, $Plan_{new}$, or reaches a planning timeout. If $Plan_{new}$ is empty (i.e. a feasible plan was not obtained) or the planning timeout is reached, the exit strategy is executed. Otherwise, the Decision Director starts PA execution, requesting events from RAs. Thus, if RAs keep updating the PA regarding the states of the associated physical part, the Decision Director will either find a new plan or force the physical part to exit the system.

5.2. Exploration

The goal of PA exploration is to obtain an up-to-date environment model of the local manufacturing system through communication with various RAs. In previous works, bidding has been used effectively to query capabilities of manufacturing systems (Borangiu, Raileanu, Trentesaux, Berger, & Iacob, 2014; Krothapalli & Deshmukh, 1999; Lim, Zhang, & Goh, 2009; Wang, Wan, Zhang, Li, & Zhang, 2016). The proposed PA architecture utilizes a bidding process to obtain the environment model. As shown in Fig. 4(a), the PA leverages the product history and process plan to send a call for bids to the RAs, which are then synthesized into the environment model. The sequence of steps for exploration is illustrated in Fig. 4(b).

5.2.1. Start of exploration

The first step in exploration is to obtain the set of desired physical properties that have yet to be achieved, P_n. Algorithm 1 shows how the PA obtains P_n by comparing its process plan (P_d) to components of its product history (Tr_h, Prp_h, x_c, s_h). Once P_n is calculated, the Decision Director can send P_n and the product history, M_h, to the Bid Manager.

Algorithm 1 Finding A Set of Incomplete, Desired Physical Properties

```
Input: $P_d, Tr_h, Prp_h, x_c, s_h$
Output: $P_n$
Initialize: $Flag = False, X_{check} = (x_c)$
 1: // Obtain the states that have been visited by the PA:
 2: for all $e \in s_h$ do
 3:     Add $Tr_h(e)$ to $X_{check}$ in sequential order
 4: end for
 5: // Populate $P_n$:
 6: for all $P_{di} \in P_d$ do
 7:     Find the first instance of a sequence of states,
 8:     $X_d = x_0 x_1 \ldots \in X_{check}$, that satisfies:
 9:         $\forall p_{di,s} \in P_{di,s}, \exists x \in X_d$, s.t. $p_{di} = Prp_h(x)$
10:     where $P_{di,s} \subseteq P_{di}$ is the largest possible subset that satisfies the
        above condition
11:     if $P_{di,s} \subset P_{di}$ then
12:         Add incomplete, desired physical properties, $p$, to $P_n$ that
            satisfy $p \in P_{di} \backslash P_{di,s}$
13:         return $P_n$
14:     else
15:         Remove $X_d$ from $X_{check}$
16:     end if
17: end for
```

5.2.2. Sending out a bid request

The Bid Manager is responsible for formulating and sending the bid requests to the Bid Translator. A bid request consists of:

PA: the PA requesting for bids
x_c: current state of the associated physical part
P_n: PA's set of incomplete, desired physical properties
t_{bound}: PA's maximum allowable time to complete P_n

where x_c is obtained from M_h and t_{bound} is set by the Bid Manager. t_{bound} represents the longest time that a set of RAs will be allowed to finish P_n.

In addition to the bid request, the Bid Manager must also provide a primary agent to contact, A_c, to the Bid Translator. A_c is responsible for starting the coordination of the RAs. Previous architectures have utilized a bidding supervisor (Borangiu et al., 2014) or one of the RAs in a system (Wang et al., 2016) for RA coordination. For the proposed architecture, the PA contacts the last RA in its product history, $A_c = Ag_h(e_l)$, where e_l is the final event of s_h in M_h.

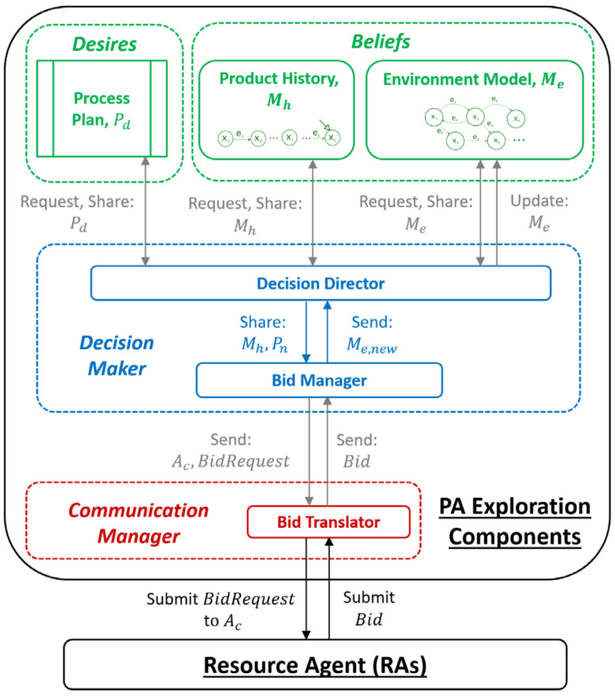
Once the Bid Translator receives a bid request and A_c, it sends out the bid request to the A_c, starting the RA's bid formulation process. The Bid Translator will wait for a certain period of time, $t_{timeout}$, for bids to come in.

5.2.3. RA bid formulation

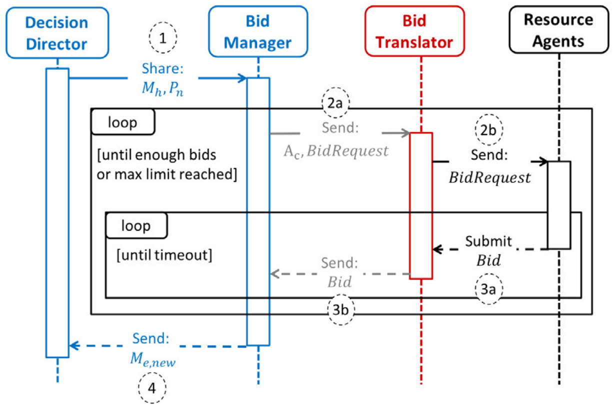
A bid represents how a set of RAs can take the physical product from its current state to a state with desired physical properties. A bid is defined as $Bid = (X_b, E_b, Tr_b, Prp_b, Ag_b, (Met_e)_b, x_c)$. Each of those elements has the same definition as the environment model described in Section 4.2.2.

The coordination of bids is left to the RAs in the multi-agent architecture. In this section, one framework for the coordination of RAs for bid formulation is presented. Existing multi-agent architectures can leverage this framework to send back bids to the proposed PA architecture.

The proposed framework for coordinating bids is through the propagation of bids to "neighboring RAs". In this framework, an RA finds a neighbor, updates all elements of the bid, and passes the bid request and an updated bid to the neighbor. A neighboring RA is a resource that has access to the same state of the physical part. For example, a material handling RA (e.g. robot RA) and a manufacturing machine



(a)



(b)

Fig. 4. (a) shows the architecture components utilized during exploration. (b) displays the sequence diagram for exploration. When the product history and set of desired physical properties are sent to the Bid Manager (1), the PA starts to query (2a, 2b) and obtain bids from the local RAs (3a, 3b). Once the PA is satisfied with the number of bids, a new environment model is created and updated (4).

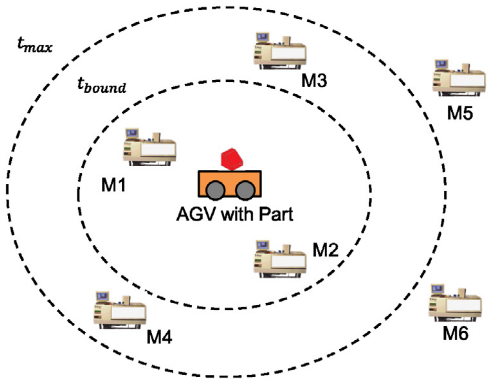


Fig. 5. An example of the exploration of a PA in a manufacturing system with Autonomous Guided Vehicles (AGV) and multiple manufacturing machines. Note that t_{bound} includes both the transportation time and manufacturing operation time.

RA (e.g. mill RA) are neighbors if the material handling resource can move the physical part to the manufacturing machine. Once a neighbor is found, the RA will update the elements of the Bid based on its own capabilities. The events and states of the Bid are updated only if they are within the t_{bound} from x_c. If the neighbor RA can be reached within the t_{bound} constraint, the RA passes the bid request and the bid to this neighbor. This process continues until all of the RAs in the t_{bound} are contacted. During this time, if an RA can accomplish P_n, that RA contacts the requesting PA and submits the full bid representing the capabilities of a set of RAs.

5.2.4. PA's synthesis of RA bids

The Bid Translator passes a submitted bid to the Bid Manager. Once $t_{timeout}$ is reached, the Bid Manager decides if there are enough satisfactory bids. The number of satisfactory bids is a tunable parameter defined during PA creation (e.g. the Bid Manager needs at least 1 or 2 bids, the Bid Manager needs bids from 10 different agents, etc.). If the Bid Manager is satisfied with the number of bids, it compiles these bids into a new environment model, $M_{e,new}$. $M_{e,new}$ is compiled by iteratively performing the fusion operation described in Section 4.2.3 to combine all of the bids (i.e. first two bids are fused to create $M_{e,new}$, the third bid is fused with $M_{e,new}$, …). The final $M_{e,new}$ is then sent to the Decision Director.

If there are not enough satisfactory bids obtained by the Bid Manager, it can start to increase the number of RAs to contact by increasing the t_{bound}. The Bid Manager can increase t_{bound} until a maximum limit, t_{max}, is reached. If there are no bids submitted when $t_{bound} = t_{max}$, the Bid Manager will send an empty $M_{e,new}$ to the Decision Director. This signifies that the exploration process was unable to find a set of RAs that can perform the set of incomplete, desired physical properties.

5.2.5. Exploration example

An example scenario of PA exploration is shown in Fig. 5. In this scenario, the PA's associated physical part is located on an Autonomous Guided Vehicle (AGV) in a manufacturing system containing various machines (M1, M2, etc.). M1 and M2 are part of the current "local neighborhood" of the PA. M3 and M4 can be added to the "local neighborhood" if t_{bound} is increased. The bid request from this particular PA cannot reach M5 or M6 as they fall outside t_{max} (i.e. the largest possible t_{bound}). If a bid request is sent out in the current configuration, then an RA bid may contain information from the AGV, M1, and/or M2.

5.3. Planning

The PA's planning components, shown in Fig. 6(a), create plans for the PA using the process plan and the environment model. The PA planning and re-planning sequences are shown in Figs. 6(b) and 6(c), respectively.

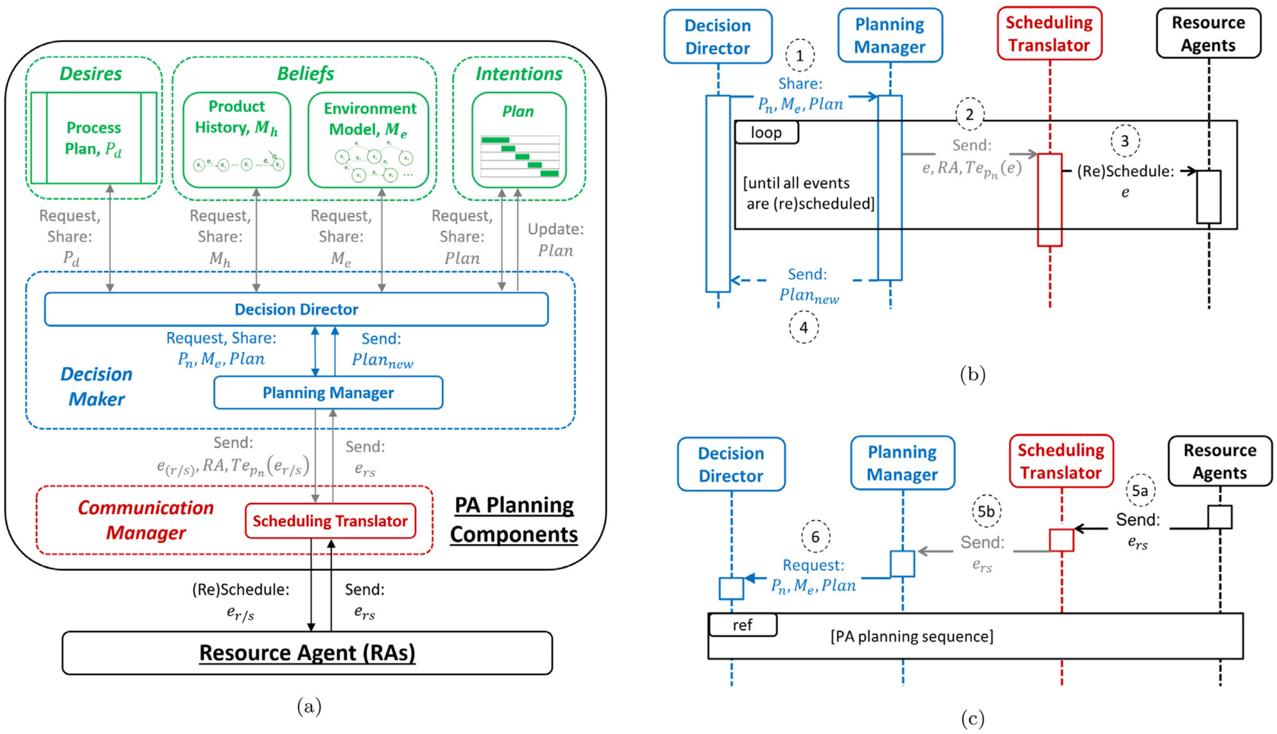


Fig. 6. (a) shows the architecture components for planning. (b) displays the planning sequence diagram. Once the environment model, desired physical properties, and old plan are passed to the Planning Manager (1), a new plan is formulated. After the Planning Manager identifies events to schedule (2), the Scheduling Translator contacts various RAs (3). When all of the RAs have been contacted, the new plan is passed to the Decision Director (4). (c) provides the sequence diagram for re-planning. When an RA wants to reschedule an event, the Planning Manager is contacted (5a, 5b) and a new Plan is requested (6).

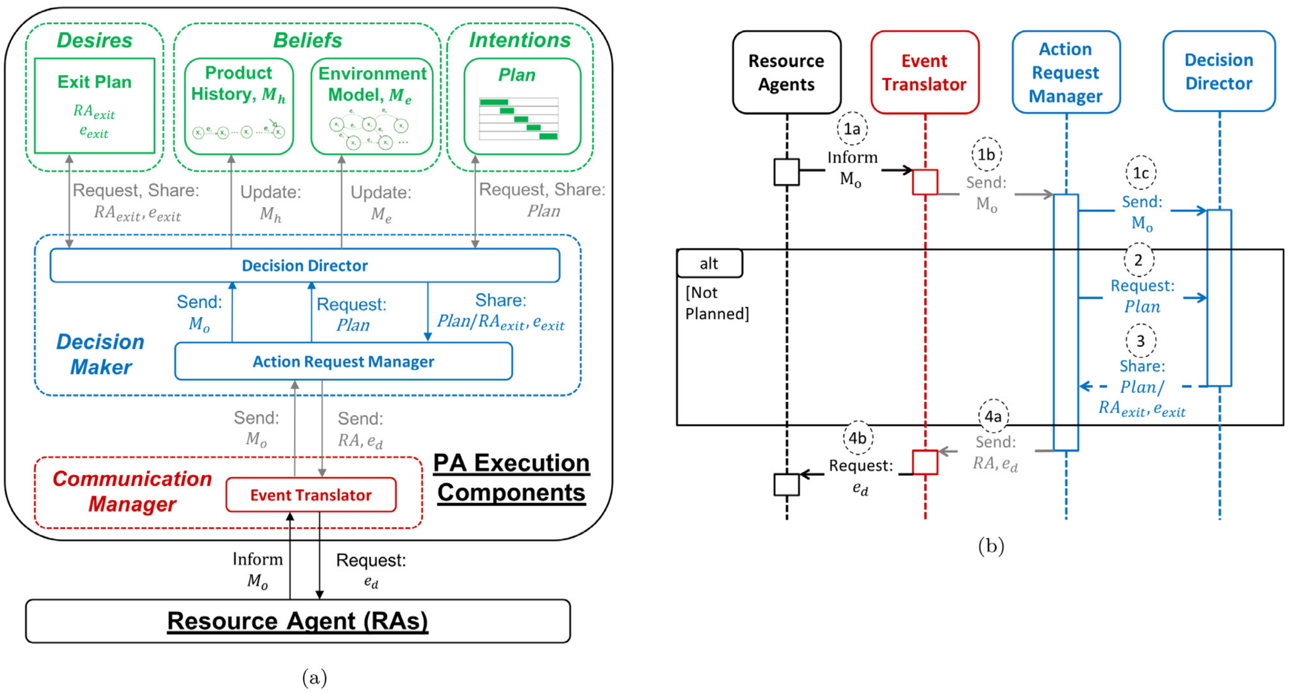


Fig. 7. (a) displays the architecture components involved in execution. (b) shows the sequence diagram for execution. RA information regarding states of the physical part is passed to the Decision Director (1a, 1b, 1c). If an unexpected event occurs, the Action Request Manager requests (2) and receives (3) a new plan. Once the PA has a feasible plan, the PA continues to request actions from the RAs (4a, 4b).

5.3.1. Plan formulation

After the Decision Director obtains P_n using Algorithm 1, it sends P_n, M_e, and the current plan to the Planning Manager. The Planning Manager uses P_n to mark states in M_e. The set of marked states, X_d, is:

$$X_d := \{x : x \in X_e \text{ and } Prp(x) \in P_n\}$$

where X_e is the set of states of M_e and Prp is the function that maps a state to its physical properties.

Utilizing the updated M_e, the following multi-objective, event planning optimization problem can be formulated to find the set of events that need to be scheduled:

$$s^* \in \arg\min_{s_k} \sum_{i=1}^{|s_k|} \alpha_1 f_1(e_{ki}), \sum_{i=1}^{|s_k|} \alpha_2 f_2(e_{ki}), \dots \tag{1}$$

$$\begin{aligned} \text{s.t.} \quad & f_i \in Met_e \text{ and } \alpha_i \in \{-1, 0, 1\} \\ & e_{k0}e_{k1}\dots e_{kn} = s_k \\ & x_0 = x_c \text{ and } x_{i+1} = Tr(x_i, e_i), \text{for } 1 \le i \le n \\ & X_d \subseteq \{x_i : 0 \le i \le n+1\} \end{aligned}$$

where s^* is the desired string (sequence of events) that starts at physical part's current location and visits all $x \in X_d$. The PA will find the strings that maximize ($\alpha = -1$), ignore ($\alpha = 0$), or minimize ($\alpha = 1$) functions from Met_e of the environment model. Note that to solve Eq. (1), the event planning optimization problem, an appropriate optimization algorithm must be used. The development and implementation of algorithms to solve these types of optimization problems is out of scope of this paper. In Section 6, the case studies use Dijkstra's shortest path (Dijkstra, 1959) to solve a specific optimization problem. Note that a time-out for the running time of the optimization algorithm is required to prevent the PA from stalling during planning. If a time-out of the optimization is reached and an s^* is not found, an empty $Plan_{new}$ is sent to the Decision Director.

If $s^* = e_0e_1\dots e_m\dots e_{m+n}$ is found, the Planning Manager compiles a new agent plan, $Plan_{new} = (s_{p_n}, Ag_{p_n}, Te_{p_n})$, with each component defined similarly to the PA's $Plan$. Each component is obtained in the following manner:

$$\begin{aligned} & s_{p_n} = e_0e_1\dots e_m \\ & Ag_{p_n}(e_i) = Ag_e(e_i), \text{ where } 0 \le i \le m \\ & Te_{p_n}(e_i) = \Big(T(e_0) + T(e_1) + \dots + T(e_{i-1}), \\ & T(e_0) + \dots + T(e_{i-1}) + T(e_i) + \epsilon\Big), \text{ where } 0 \le i \le m \end{aligned}$$

The agent association function, Ag_e, and event time duration, T, are obtained from the environment model. ϵ is a small amount of time added to the plan to allow for small changes in the time duration of e_i. Note that the length of s_{p_n}, a substring of s^*, defines how long into the future the PA will plan events. After $Plan_{new}$, the PA begins to communicate with the RAs to schedule these events.

5.3.2. Event scheduling

First, the Planning Manager collects any future events from the current plan, $Plan$, by computing the set:

$$E_r = \{e_i \mid Te_{p,1}(e_i) > t_{current}, \forall e \in s_p, s_p \in Plan\}$$

where $Te_{p,1}(e)$ is the start time of the event and $t_{current}$ is the current time. Using this information, the Planning Manager sends each future event that needs to be removed, $e_r \in E_r$, the RA that performs the event, $Ag_{p_n}(e_r)$, and the event's time duration, $Te_{p_n}(e_r)$, to the Scheduling Translator. The Scheduling Translator uses this information to contact the corresponding RA to remove e_r from its schedule. By removing scheduled events, an RA can use its previously booked time for other tasks (e.g. schedule other PAs or maintenance activities).

After removing all of the future events from $Plan$, the Planning Manager schedules the events in $Plan_{new}$ with the RAs. For each $e_s \in s_{p_n}$, the Planning Manager sends the event, the RA that performs the event, $Ag_{p_n}(e_s)$, and the time duration of that event, $Te_{p_n}(e_s)$, to the Scheduling Translator. The Scheduling Translator uses this information to contact the corresponding RA to add e_s to each RA's schedule.

Finally, the Planning Manager sends $Plan_{new}$ to the Decision Director to update the PA's plans. Once it receives $Plan_{new}$, the Decision Director updates the existing plan with $Plan_{new}$.

5.3.3. Re-planning based on RA information

The Scheduling Translator is also responsible for listening to rescheduling requests from the RAs. A rescheduling request is sent to the PA if the RA cannot accomplish a scheduled action due to some unforeseen change in the manufacturing environment (e.g. unexpected machine malfunction). Once notified by the RA, the Scheduling Translator sends the event to reschedule, e_{rs}, to the Planning Manager. The Planning Manager requests information that is necessary to start the planning sequence shown in Fig. 6(b). Afterwards, the PA starts to follow the same planning methodology as described previously in this section. However, when computing s^* in Eq. (1), the Planning Manager adds a constraint, $e_{rs} \notin s^*$, in the optimization problem to ensure that a new plan is found.

5.4. Execution

The PA execution components, shown in Fig. 7(a), receive RA information regarding completed events in the manufacturing system and send event execution requests to the RAs. The sequence of events for the execution process is shown in Fig. 7(b).

The execution process starts when one of the RAs informs the PA of an event that occurred in the physical system by sending a manufacturing system output, M_o, to the Event Translator. The system output, $M_o = (X_o, E_o, Tr_o, Prp_o, Ag_o, x_{c,o}, s_o)$, is defined similarly to the product history of the PA. After being informed about M_o, the Event Translator passes M_o to the Action Request Manager.

The Action Request Manager checks M_o to see if the product agent is still following the desired plan. The Action Request Manager obtains the last event, e_{ol}, from the system output's string of occurred events, s_o. Then, using each of the components of $Plan$, (i.e. s_p, Ag_p, Te_p), the Action Request Manager performs the following actions:

1. The planned start and end times of e_{ol} are checked against the current time using $Te_p(e_{ol})$
2. The next desired event, e_d, is obtained from the string of planned events, s_p
3. The start time of the next desired event is calculated using $Te_p(e_d)$
4. e_d and the RA associated with the event, $Ag_p(e_d)$, are sent to the Event Translator once the start time is reached

Once the Event Translator receives e_d and $Ag_p(e_d)$, a request is sent to $Ag_p(e_d)$ asking for e_d.

Note that the Action Request Manager will request a new plan from the Decision Director if:

- The last event was not planned, $e_{ol} \notin s_p$
- The current time did not fall between the planned start and end times of e_{ol}
- There is no next desired event, e_d (i.e. the current plan is finished)

In addition to figuring out the next desired event, the Decision Director updates the product history and environment model in the Knowledge Base using the system output. The Decision Director performs the fusion operation, described in Section 4.2, on the X, E, Tr, Prp, and Ag components of M_h and M_o to obtain a new M_h. Additionally, the following components of M_h are also updated:

$$\begin{aligned} & x_c = x_{c,o}, \text{ where } x_{c,o} \text{ is obtained from the } M_o \\ & s_h = s_h s_o, \text{ where } s_o \text{ is appended to the end of } s_h \end{aligned}$$

The current state, x_c, of the environment model, M_e, is similarly updated using $x_{c,o}$.

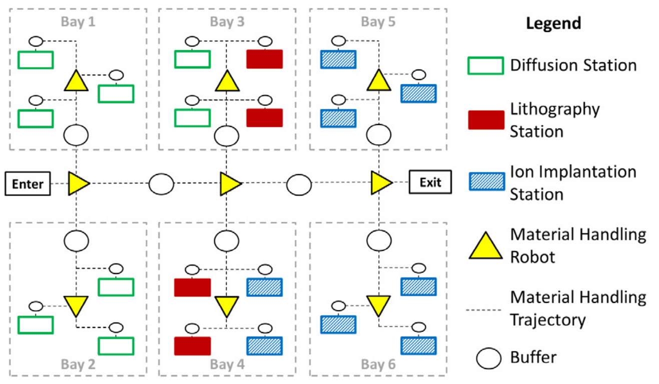


Fig. 8. The set-up for the PA simulation case studies.

Table 2
Manufacturing processes times for each station.

Process number	Process time
P1 (Diffusion)	150 ticks/lot
P2 (Ion implementation)	60 ticks/lot
P3 (Lithography)	110 ticks/lot
P4 (Ion implementation)	100 ticks/lot
P5 (Diffusion)	170 ticks/lot
P6 (Lithography)	20 ticks/lot

6. Product agent case studies

To test the feasibility and performance of the PA architecture, a simulation of a manufacturing system under multi-agent control[1] was evaluated. In this section, the set-up of the system and the PA are described and the results from various case studies of the PA are provided.

6.1. Case-study set-up

The Repast Symphony (RepastS) platform (North, Collier, Ozik, Tatara, Macal, Bragen, & Sydelko, 2013) can be used to model and simulate the behavior of manufacturing systems that are controlled via multi-agent strategies (Barbosa & Leitao). Due to the high modeling power of the Repast environment (Macal & North), this platform was chosen to simulate the complex behavior of manufacturing systems. In this work, the RepastS software was used to create a scaled-up version of the Intel Mini-Fab (Yoon & Shen, 2008), a semiconductor manufacturing facility. The simulated facility contains 20 stations that are connected via a network of 9 material handling robots, as shown in Fig. 8. Buffers without size constraints are placed between each of the robots. Similar, infinite-sized buffers are placed at each of the machining stations. The robots take around 2–5 ticks (RepastS unit of time) to move the lots between the various buffers.

The robots, machines, and buffers have an RA that makes high-level control decisions for its associated resource. Each RA is able to satisfy all of the communication (i.e. exploration, planning, and execution) requests of the PA. The RAs have a schedule that keep track of the scheduling request of the PAs. If necessary, the RAs relay information regarding any scheduling or execution conflicts back to the PA.

Wafer lots are deposited at the facility via the entrance. After completing a set of desired production steps, the lots leave the facility through the exit, as shown in Fig. 8. There are six different processes (P1–P6) that are provided by the machines in this facility. The specific process times and stations are shown in Table 2. The order of the manufacturing processes depends on the lot type, as described in the following separate case-studies.

[1] https://github.com/ikovalenko92/SemiconductorSimulation.

6.2. Product agent architecture implementation

A product agent is created when a lot enters the manufacturing facility. Once the PA is created, its process plan is initialized by providing an ordered set of manufacturing system processes that the lot must accomplish. In addition, an exit plan is generated to enable the lot to exit the production system under specified conditions.

The product history and environment model are created using a custom, weighted, directed graph from the Java Universal Network/Graph (JUNG) Framework (O'Madadhain, Fisher, White, & Boey, 2003). Initially, the product history only contains the location of the lot at the entrance and the environment model is empty. In addition, an empty agent plan is created using a customized Java class. The beliefs and intentions of the PA are updated as the PA explores, plans, and executes actions in the simulated environment.

Once the PA is initialized, it begins the operation phase. In this phase, the Decision Director cycles through exploring the system, planning, and executing actions. The Decision Director sets the exploration and planning time-out times to 1 tick.

During exploration, the initial t_{bound} is set to 300 ticks and the t_{max} is set to 10,000 ticks. t_{bound} is increased by 500 ticks until the PA receives a set of feasible Resource Agents that can take the lot to the next desired physical property or t_{bound} reaches t_{max}. The PA combines the bids from the RAs to update the environment model.

During planning, the PA minimizes the amount of time that it will take for the lot to arrive at the next desired manufacturing process. For this problem, the PA applies Dijkstra's shortest path algorithm with event times as edge weights (Dijkstra, 1959) to find the shortest path between the PA's current state and a desired state. Using the shortest path, the PA constructs a plan of events (maximum of 20) and schedules these events with the associated RAs. After the plan is constructed, the PA continues its execution steps.

6.3. Case-studies for the PA architecture

In this section, the results from three case studies of the PA architecture are described in detail. The results from the case studies are shown in Fig. 9.

6.3.1. One lot type, fully functioning system

Case study 1 demonstrates the production paths of the PAs through the simulated system. The study considers 150 wafer lots entering the system every 200 ticks. The initialized process plan, P_{d0}, is P1 → P2 → P3 → P4 → P5 → P6.

The case study was performed five times. On average, 135.4 lots were produced by the system and 14.6 exited the system. The average flow time for the completed parts was around 1700 ticks/part. Fig. 9(a) shows the decisions that were made by the PAs as they traversed the simulated semiconductor fab. Most of the wait times for the PAs are due to waiting on the robots to move them between the various points in the system, as illustrated by the larger red circles in Fig. 9(a).

6.3.2. One lot type, two machines breakdown

Case study 2 demonstrates the production paths of the PAs through a manufacturing system with machine breakdowns. The number of lots, the frequency of lots, and the initialized process plan are the same as case study 1. In this case study, two lithography stations are taken down 2000 ticks into the simulation. As shown in Fig. 9(b), the PAs chose not to go to the broken down resource based on RA information. The PAs autonomously chose to visit different stations to complete their process plans.

The case study was performed five times in the simulated environment. On average, 134.6 lots were produced by the system and 15.4 exited the system. The average flow time for the completed parts was around 2500 ticks/part. While the PA architecture was able to produce around the same number of finished parts, the amount of time taken to produce the parts was significantly longer.

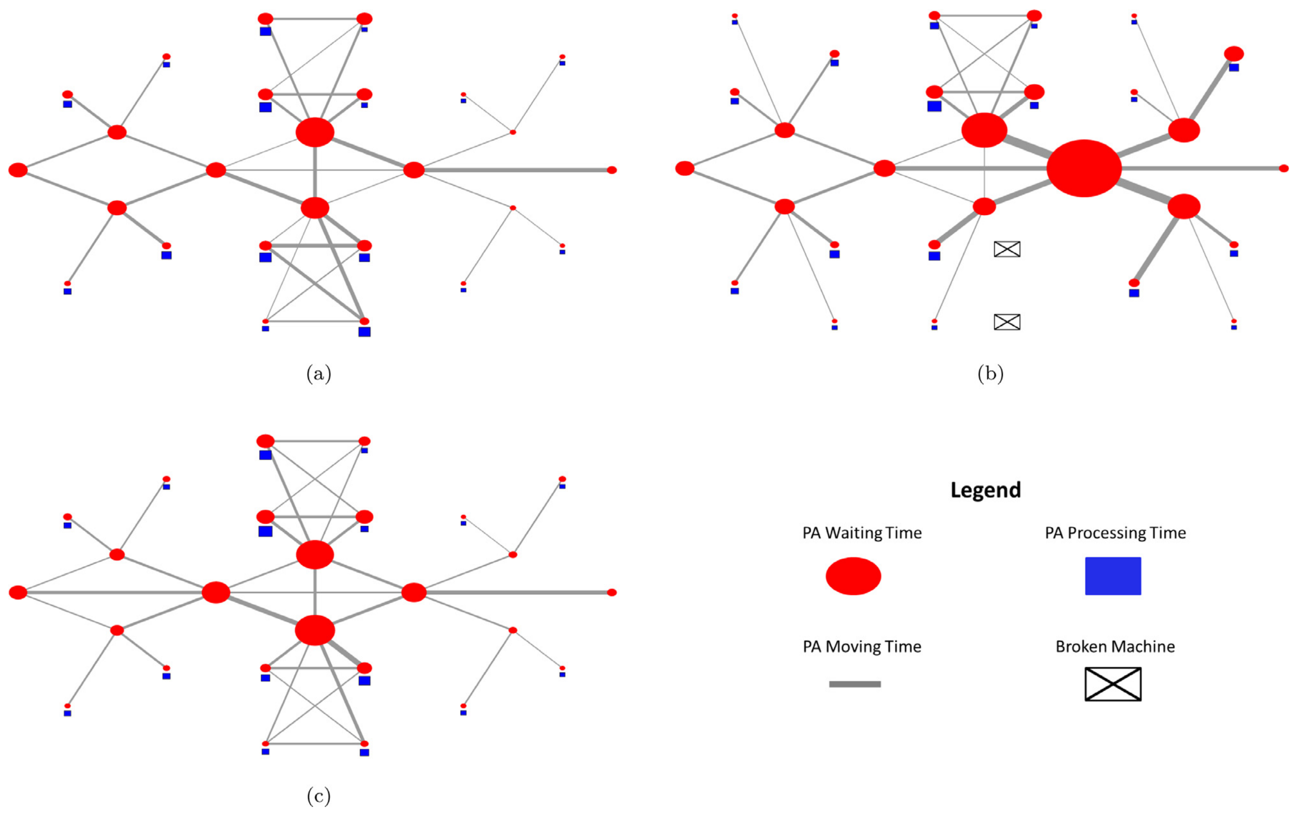


Fig. 9. Figures (a)–(c) represent the behavior of the PAs for Case Studies (1)–(3). Larger nodes (red circles and blue squares) represent longer times that the PAs spent at that location. Similarly, thicker edges (gray lines) represent the number of times that the particular path was taken for the completed parts. The two, independent scale factors (for node size and edge thickness) are the same across all three cases.

6.3.3. *Three lot types, fully functioning system*

Case study 3 demonstrates the production paths of PAs with various process plans. The number of lots and the frequency of lots are the same as case studies 1 and 2. Two other process plans, from Yoon and Shen (2008), are introduced into the system. The first new process plan, P_{d1}, is P2 → P1 → P3 → P5 → P4 → P6. The second new process plan, P_{d2}, is P4 → P5 → P6 → P1 → P2 → P3. For new lots, the initialization of process plans alternates between P_{d0}, P_{d1}, and P_{d2}. Thus, each process plan should be followed by 50 different lots.

Similarly, the case study was performed five times in the simulated environment. On average, PAs initialized with P_{d0} completed 42.8 and exited 7.2 lots, PAs initialized with P_{d1} completed 40.6 and exited 9.4 lots, and PAs initialized with P_{d2} completed 44.6 and exited 5.4 lots. The average flow time for the completed parts was around 1800 ticks/part. Thus, the PA architecture was able to follow each of the process plans and adapt to different process plan sequences.

6.4. *Insights from case studies*

The case studies presented in this section showcase the feasibility of using a model-based, adaptive PA to drive the behavior of products in a multi-agent manufacturing system. Each individual PA was able to systematically construct an environment model of the system that contains the starting location of the product, the capabilities of individual resources, and the most up-to-date schedule of each resource. The environment model, combined with the PA's process plan, product history, and intentions, was used to make an individually optimized decision. In most cases, these types of model-based, goal-based software architectures allow for greater agent flexibility and scalability to larger systems when compared to rule-based architectures (Russell & Norvig, 2009).

In the case studies, the goal of every PA was to minimize the time spent in the system, while completing all of the events in its process plan. However, information regarding other factors, such as product quality or energy expenditure of the system, can be incorporated into the model. Note that different, individualized objectives (e.g. minimizing energy usage, meeting production deadlines, etc.) can be integrated in the PA optimization function.

Additionally, the inclusion of the exit plan in the proposed PA guarantees that all of the PAs either complete their process plan or exit the system. Due to the modularity of the software architecture, the bounds on the PA exploration, and the use of the Decision Maker's time-outs, a PA's associated physical part will not occupy a single resource for an unreasonable amount of time. This property of the PA software architecture is beneficial if the PA is unable to adapt to an unexpected occurrence or a conflict in the system, as observed in each of the case studies.

The RepastS case studies show that a model-based PA can be used for manufacturing systems. However, to implement the PA in a physical manufacturing system, an agent-based developmental platform should be used (Kravari & Bassiliades, 2015). The PAs would communicate with other agents using existing communication protocols. An example platform to incorporate the model-based PAs into existing multi-agent manufacturing implementations is the Java Agent Development Framework (JADE) (JAD; Leitão, Karnouskos, Ribeiro, Lee, Strasser, & Colombo, 2016).

7. Conclusion

Manufacturing system flexibility can be improved through the utilization of multi-agent control strategies. One important component of a multi-agent controller for manufacturing systems is the product agent. In this work, the design and testing of a software product agent architecture that uses a model-based optimization approach is presented. The proposed product agent contains a Knowledge Base, Decision Maker, and Communication Manager. These components are used by the product agent to explore the capabilities of surrounding resources, formulate plans based on this knowledge, and request actions to be taken by various resources based on the production goals of its associated physical part. This product agent software architecture can be used to create customized products and guide parts through dynamically changing manufacturing systems. These product agent capabilities are showcased in a simulated semiconductor manufacturing environment.

To fully realize the potential of this product agent, a variety of other planning algorithms should be tested to find the optimal path of the associated physical part. In addition, negotiation between product agents must be explored to coordinate and improve resource utilization. By exploring these areas, it is possible to improve the performance of the product agent, allowing for faster, more efficient production of products in flexible manufacturing systems.

Acknowledgments

This work was supported in part by an National Science Foundation, USA Graduate Fellowship and a gift from Rockwell Automation.

Conflict of interest

None declared.

References

Jade Site | Java Agent DEvelopment Framework, URL http://jade.tilab.com/.

Ansola, P. G., García, A., de las Morenas, J., & de las Morenas, J. (2015). Aligning decision support with shop floor operations: A proposal of intelligent product based on BDI physical agents. In T. Borangiu, A. Thomas, & D. Trentesaux (Eds.), *Service Orientation in Holonic and Multi-Agent Manufacturing* (pp. 91–100). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-15159-5_9, https://doi.org/10.1007/978-3-319-15159-5{_}9.

Barbosa, J., & Leitao, P. (2011). Simulation of multi-agent manufacturing systems using Agent-Based Modelling platforms. In 2011 9th IEEE International Conference on Industrial Informatics (pp. 477–482), http://dx.doi.org/10.1109/INDIN.2011.6034926.

Barbosa, J., Leitão, P., Adam, E., & Trentesaux, D. (2013). Structural self-organized holonic multi-agent manufacturing systems. *Industrial Applications of Holonic and Multi-Agent Systems SE - 6*, *8062*, 59–70. http://dx.doi.org/10.1007/978-3-642-40090-2_6, http://dx.doi.org/10.1007/978-3-642-40090-2{_}6.

Barbosa, J., Leitão, P., Adam, E., & Trentesaux, D. (2015). Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution. *Computers in Industry*, *66*, 99–111.

Barton, K., Maturana, F., & Tilbury, D. (2018). Closing the loop in IoT-enabled manufacturing systems: Challenges and opportunities. In *2018 Annual American Control Conference (ACC)* (pp. 5503–5509). IEEE.

Borangiu, T., Raileanu, S., Trentesaux, D., Berger, T., & Iacob, I. (2014). Distributed manufacturing control with extended CNP interaction of intelligent products. *Journal of Intelligent Manufacturing*, *25*(5), 1065–1075. http://dx.doi.org/10.1007/s10845-013-0740-3.

Brettel, M., Friederichsen, N., Keller, M., & Rosenberg, M. (2014). How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 perspective. *International Journal of Information and Communication Engineering*, *8*(1), 37–44. http://dx.doi.org/10.1016/j.procir.2015.02.213.

Cassandras, C. G., & Lafortune, S. (2009). Introduction to discrete event systems, (2nd ed.). Springer Science & Business Media.

De Las Morenas, J., Garcia-Higuera, A., & Garcia-Ansola, P. (2017). Shop floor control: A physical agents approach for PLC-controlled systems. *IEEE Transactions on Industrial Informatics*, *13*(5), 2417–2427. http://dx.doi.org/10.1109/TII.2017.2720696.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271.

Farid, A. M., & Ribeiro, L. (2015). An axiomatic design of a multiagent reconfigurable mechatronic system architecture. *IEEE Transactions on Industrial Informatics*, *11*(5), 1142–1155. http://dx.doi.org/10.1109/TII.2015.2470528.

Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1998). The belief-desire-intention model of agency. *Intelligent Agents V: Agents Theories, Architectures, and Languages. 5th International Workshop, ATAL'98.*, 1–10. http://dx.doi.org/10.1007/3-540-49057-4_1.

Gouyon, D., Pétin, J. -F., Morel, G., Et, A., & al., E. (2007). A product-driven reconfigurable control for shop floor systems. *Studies in Informatics and Control*, *16*(1).

Howden, N., Rönnquist, R., Hodgson, A., & Lucas, A. (2001). JACK Intelligent agents-summary of an agent infrastructure. *Management*, 6, http://dx.doi.org/10.1.1.133.8934.

Hu, S. J. (2013). Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Procedia CIRP*, *7*, 3–8. http://dx.doi.org/10.1016/j.procir.2013.05.002.

Kovalenko, I., Barton, K., & Tilbury, D. (2017). Design and implementation of an intelligent product agent architecture in manufacturing systems. In *Emerging Technology and Factory Automation* (pp. 1–8). http://dx.doi.org/10.1109/ETFA.2017.8247652.

Kravari, K., & Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, *18*(1), 11.

Krothapalli, N. K. C., & Deshmukh, A. V. V. (1999). Design of negotiation protocols for multi-agent manufacturing systems. *International Journal of Productions Research*, *37*(7), 1601–1624. http://dx.doi.org/10.1080/002075499191157.

Leitão, P. (2009). Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, *22*(7), 979–991. http://dx.doi.org/10.1016/j.engappai.2008.09.005.

Leitão, P., Colombo, A. W., & Restivo, F. (2006). A formal specification approach for holonic control systems: The ADACOR case. *International Journal of Manufacturing Technology and Management*, *8*(1–3), 37–57. http://dx.doi.org/10.1504/IJMTM.2006.008790.

Leitão, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T., & Colombo, A. W. (2016). Smart agents in industrial cyber-physical systems. *Proceedings of the IEEE*, *104*(5), 1086–1101. http://dx.doi.org/10.1109/JPROC.2016.2521931.

Leitão, P., Mařík, V., & Vrba, P. (2013). Past, present, and future of industrial agent applications. *IEEE Transactions on Industrial Informatics*, *9*(4), 2360–2372. http://dx.doi.org/10.1109/TII.2012.2222034.

Leitão, P., & Restivo, F. (2006). ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, *57*(2), 121–130. http://dx.doi.org/10.1016/j.compind.2005.05.005.

Leitão, P., Rodrigues, N., Barbosa, J., Turrin, C., & Pagani, A. (2015). Intelligent products: The grace experience. *Control Engineering Practice*, *42*, 95–105. http://dx.doi.org/10.1016/j.conengprac.2015.05.001.

Lepuschitz, W., Zoitl, A., Vallee, M., & Merdan, M. (2011). Toward self-reconfiguration of manufacturing systems using automation agents. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *41*(1), 52–69. http://dx.doi.org/10.1109/TSMCC.2010.2059012.

Lim, M. K., Zhang, Z., & Goh, W. T. (2009). An iterative agent bidding mechanism for responsive manufacturing. *Engineering Applications of Artificial Intelligence*, *22*(7), 1068–1079. http://dx.doi.org/10.1016/j.engappai.2008.12.003.

Lu, Y., Riddick, F., & Ivezic, N. (2016). The paradigm shift in smart manufacturing system architecture. In I. Nääs, O. Vendrametto, J. Mendes Reis, R. F. Gonçalves, M. T. Silva, G. von Cieminski, & D. Kiritsis (Eds.), *Advances in Production Management Systems. Initiatives for a Sustainable World* (pp. 767–776). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-51133-7_90.

Lüder, A., Klostermeyer, A., Peschke, J., Bratoukhine, A., & Sauter, T. (2005). Distributed automation: PABADIS vs. HMS. *IEEE Transactions on Industrial Informatics*, *1*(1), 31–38. http://dx.doi.org/10.1109/INDIN.2003.1300283.

Lüder, A., Peschke, J., Sauter, T., Deter, S., & Diep, D. (2004). Distributed intelligence for plant automation based on multi-agent systems: The PABADIS approach. *Production Planning and Control*, *15*(2), 201–212. http://dx.doi.org/10.1080/09537280410001667484.

Macal, C. M., & North, M. J. (2006). Introduction to Agent-based Modeling and Simulation, In MCS LANS Informal Seminar.

Marchetta, M. G., Mayer, F., & Forradellas, R. Q. (2011). A reference framework following a proactive approach for product lifecycle management. *Computers in Industry*, *62*(7), 672–683. http://dx.doi.org/10.1016/j.compind.2011.04.004.

McFarlane, D., Giannikas, V., Wong, A. C. Y., & Harrison, M. (2013). Product intelligence in industrial control: Theory and practice. *Annual Reviews in Control*, *37*(1), 69–88. http://dx.doi.org/10.1016/j.arcontrol.2013.03.003.

McFarlane, D., Sarma, S., Chirn, J. L., Wong, C. Y., & Ashton, K. (2003). Auto ID systems and intelligent manufacturing control. *Engineering Applications of Artificial Intelligence*, *16*(4), 365–376. http://dx.doi.org/10.1016/S0952-1976(03)00077-0.

Meyer, G. G., Hans Wortmann, J. C., & Szirbik, N. B. (2011). Production monitoring and control with intelligent products. *International Journal of Productions Research*, *49*(5), 1303–1317. http://dx.doi.org/10.1080/00207543.2010.518742.

Monostori, L., Váncza, J., & Kumara, S. R. T. (2006). Agent-based systems for manufacturing. *CIRP Annals - Manufacturing Technology*, *55*(2), 697–720. http://dx.doi.org/10.1016/j.cirp.2006.10.004.

Nilsson, F., & Darley, V. (2006). On complex adaptive systems and agent-based modelling for improving decision-making in manufacturing and logistics settings: Experiences from a packaging company. *International Journal of Operations & Production Management*, *26*(12), 1351–1373. http://dx.doi.org/10.1108/01443570610710588.

North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., & Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling*, *1*(1), 3. http://dx.doi.org/10.1186/2194-3206-1-3.

O'Madadhain, J., Fisher, D., White, S., & Boey, Y. (2003). The JUNG (Java Universal Network/Graph) framework. (pp. 3–17). UCI-ICS.

Pétin, J. F., Gouyon, D., & Morel, G. (2007). Supervisory synthesis for product-driven automation and its application to a flexible assembly cell. *Control Engineering Practice*, *15*(5), 595–614. http://dx.doi.org/10.1016/j.conengprac.2006.10.013.

Rehberger, S., Spreiter, L., & Vogel-Heuser, B. (2016). An Agent Approach to Flexible Automated Production Systems Based on Discrete and Continuous Reasoning. In 2016 IEEE International Conference on Automation Science and Engineering (CASE) (pp. 1249–1256), http://dx.doi.org/10.1109/COASE.2016.7743550.

Ribeiro, L., Ocha, A., Veiga, A., & Barata, J. (2015). Collaborative routing of products using a self-organizing mechatronic agent framework - a simulation study. *Computers in Industry*, *68*, 27–39. http://dx.doi.org/10.1016/j.compind.2014.12.003.

Russell, S., & Norvig, P. (2009). Artificial intelligence: A modern approach, (3rd ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Shen, W., Hao, Q., Yoon, H. J., Norrie, D. H., Joong, H., & Norrie, D. H. (2006). Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, *20*(4), 415–431. http://dx.doi.org/10.1016/j.aei.2006.05.004.

Tang, H., Li, D., Wang, S., & Dong, Z. (2017). CASOA: An architecture for agent-based manufacturing system in the context of industry 4.0. *IEEE Access*, *6*, 12746–12754. http://dx.doi.org/10.1109/ACCESS.2017.2758160.

Tilbury, D. M. (2019). Cyber-physical manufacturing systems. *Annual Review of Control, Robotics, and Autonomous Systems*, *2*(1), http://dx.doi.org/10.1146/annurev-control-053018-023652.

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., & Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, *37*(3), 255–274. http://dx.doi.org/10.1016/S0166-3615(98)00102-X.

Vogel-Heuser, B., Lee, J., & Leitão, P. (2015). Agents enabling cyber-physical production systems. *At-Automatisierungstechnik*, *63*(10), 777–789. http://dx.doi.org/10.1515/auto-2014-1153.

Vrba, P., Tichý, P., Mařík, V., Hall, K. H., Staron, R. J., Maturana, F. P., & Kadera, P. (2011). Rockwell automation's holonic and multiagent control systems compendium. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *41*(1), 14–30. http://dx.doi.org/10.1109/TSMCC.2010.2055852.

Wang, S., Wan, J., Zhang, D., Li, D., & Zhang, C. (2016). Towards smart factory for industry 4.0: A self-organized multi-agent system with big data based feedback and coordination. *Computer Networks*, *101*, 158–168. http://dx.doi.org/10.1016/j.comnet.2015.12.017.

Wooldridge, M. J. (2009). An introduction to multiagent systems, (2nd ed.). West Sussex UK: John Wiley & Sons.

Yoon, H. J., & Shen, W. (2008). A multiagent-based decision-making system for semiconductor wafer fabrication with hard temporal constraints. *IEEE Transactions on Semiconductor Manufacturing*, *21*(1), 83–91. http://dx.doi.org/10.1109/TSM.2007.914388.