Intelligent Product Agents for Multi-Agent Control of Manufacturing Systems

by

Ilya Kovalenko

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in the University of Michigan
2020

Ilya Kovalenko
ikoval@umich.edu
ORCID iD: 0000-0003-4764-9117

ACKNOWLEDGEMENTS

I would like to thank the Department of Mechanical Engineering, Rackham Graduate School, and the University of Michigan for giving me the opportunity to pursue the doctoral degree and providing me with financial support during my time at the university. In addition, I would like to thank the National Science Foundation and Rockwell Automation for providing financial assistance.

Most importantly, I would like to thank Professors Kira Barton and Dawn Tilbury. They have been extremely helpful in helping me grow and develop my technical, professional, and personal skills since meeting them five years ago. My graduate school experience has been greatly enhanced in large part due to their support and guidance throughout my time at the University of Michigan. I would also like to thank each of my committee members: Professor Stéphane Lafortune, Professor Dimitra Panagou, and Professor Birgit Vogel-Heuser for providing me their support and feedback that improved the dissertation.

In addition to providing unwavering support, I appreciate the culture and environment that Professor Barton and Professor Tilbury have fostered in their labs. This work in this dissertation would not have been possible without support from each and every member of the two groups. Specifically, I would like to thank Efe Balta, Mingjie Bi, Gary Zheng, Issac Spiegel, Felipe Lopez, Miguel Saez, and Yassine Qamsane for helping me develop the agent architectures described in this work. I would also like to thank the Secure Cloud Manufacturing Multidisciplinary Design Project Team at the University of Michigan for helping me implement the developed architectures, methods, and algorithms in the various testbeds at the University of Michigan.

Outside of the University of Michigan, I would like to thank Professor Vogel-Heuser and her lab at the Technical University of Munich for hosting me for several months during the course of my dissertation. This opportunity was immensely helpful in the design and implementation of the multi-agent controller described in this dissertation. Specifically, I would like to thank Felix Ocker, Daria Ryashentseva, and Juliane Fischer for helping me navigate the lab, the university, and the city during this time. In addition, I would like to thank Professor Jun Ueda at Georgia Tech for growing my interest in research as an undergraduate student.

Last, but not least, I would like to thank my family and friends for supporting me during the past several years. I will fondly remember my time at the University of Michigan and in Ann Arbor because of all of the people I encountered, friends I made, and experiences I had. My parents have been supportive of my pursuit of my degree and I am eternally grateful for their ever-present guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The current manufacturing paradigm is shifting toward more flexible manufacturing systems that produce highly personalized products, adapt to unexpected disturbances in the system, and readily integrate new manufacturing system technology. However, to achieve this type of flexibility, new system-level control strategies must be developed, tested, and integrated to coordinate the components on the shop floor. One strategy that has been previously proposed to coordinate the resources and parts in a manufacturing system is multi-agent control.

The manufacturing multi-agent control strategy consists of agents that interface with the various components on the shop floor and continuously interact with each other to drive the behavior of the manufacturing system. Two of the most important decision-making agents for this type of control strategy are product agents and resource agents. A product agent represents a single product and a resource agent represents a single resource on the plant floor. The objective of a product agent is to make decisions for an individual product and request operations from the resource agents based on manufacturer and customer specifications. A resource agent is the high-level controller for a resource on the shop floor (e.g., machines, material-handling robots, etc.). A resource agent communicates with other product and resource agents in the system, fulfills product agent requests, and interfaces with the associated resource on the plant floor.

While both product agents and resource agents are important to ensure effective performance of the manufacturing system, the work presented in this dissertation improves the intelligence and capabilities of product agents by providing a standardized product agent architecture, models to capture the dynamics and constraints of the manufacturing environment, and methods to make improved decisions in a dynamic system. New methods to explore the manufacturing system and cooperate with other agents in the system are provided. The proposed architecture, models, and methods are tested in a simulated manufacturing environment and in several manufacturing testbeds with physical components. The results of these experiments showcase the improved flexibility and adaptability of this approach. In these experiments, the model-based product agent effectively makes decisions to meet its production requirements, while responding to unexpected disturbances in the system, such as machine failures or new customer orders. The model-based product agent proposed in this dissertation pushes the fields of manufacturing and system-level control closer to realizing the goals of increased personalized production and improved manufacturing system flexibility.

CHAPTER I

Introduction

1.1 Motivation

The manufacturing sector is an important part of the global economy, being responsible for 16% of the Gross World Product [75]. In the United States, manufacturing accounts for 11% of the Gross Domestic Product, provides 12 million jobs, and supports many other services across the economy [89]. Recent technological advancements in the areas of sensing, computation, and communication have led to the promise of a new manufacturing paradigm known as Smart Manufacturing in the United States and Industry 4.0 in Europe [54, 68]. The goal of this new paradigm is to increase personalized production, improve system flexibility, and enhance manufacturing productivity by connecting the different stages of the product lifecycle, gathering data from every stage, and using this data to dynamically adapt the system to variations in production demands and operating conditions [19, 40, 129]. The design, implementation, and analysis of a fully connected manufacturing enterprise presents a number of scientific and technical challenges [54].

To improve manufacturing system flexibility and enable customized production, manufacturers are looking to incorporate more advanced manufacturing technology on the plant floor. Current manufacturing systems are starting to integrate traditional manufacturing technology, such as CNC (Computer Numerical Control) machines and conveyor systems, with recent manufacturing technology, e.g additive manufacturing, automated guided vehicles (AGV), smart industrial robots [33, 51], and the Industrial Internet of Things [114]. As both the capabilities of individual components and customer expectations for highly personalized products increase, the control and coordination of the components on the plant floor become increasingly more complicated. Coordination of the components on the plant floor, also known as system-level control, presents a number of interesting fundamental research problems [11, 119].

One of the major challenges in this area of system-level control is the development of control strategies that can rapidly adapt to unexpected disturbances on the shop floor. A system-level controller has to handle unexpected machine faults or breakdowns, respond to changing customer orders, and quickly integrate new machines into the shop floor, to name a few objectives [11]. One strategy for responding to these system disturbances is through the use of a centralized, hierarchical control architecture [67, 119]. For this control strategy, the disturbance is handled by a single controller that has access to all of the information in the manufacturing system. However, finding new schedules and coordinating all of the system components becomes more difficult as the complexity of the manufacturing system increases [10]. Therefore, distributed strategies for manufacturing system-level control have been proposed to increase the flexibility of the manufacturing system [18, 69, 83].

One particular distributed control strategy that can be leveraged to address the challenge of coordinating complex systems is multi-agent control [128]. Agent-based control of manufacturing systems has been a growing area of research over the past 30 years [56, 60, 61, 112, 122]. In this control strategy, a number of software agents make high-level decisions for various manufacturing system components (e.g., machines, physical parts, product orders, etc.) [56, 122]. 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

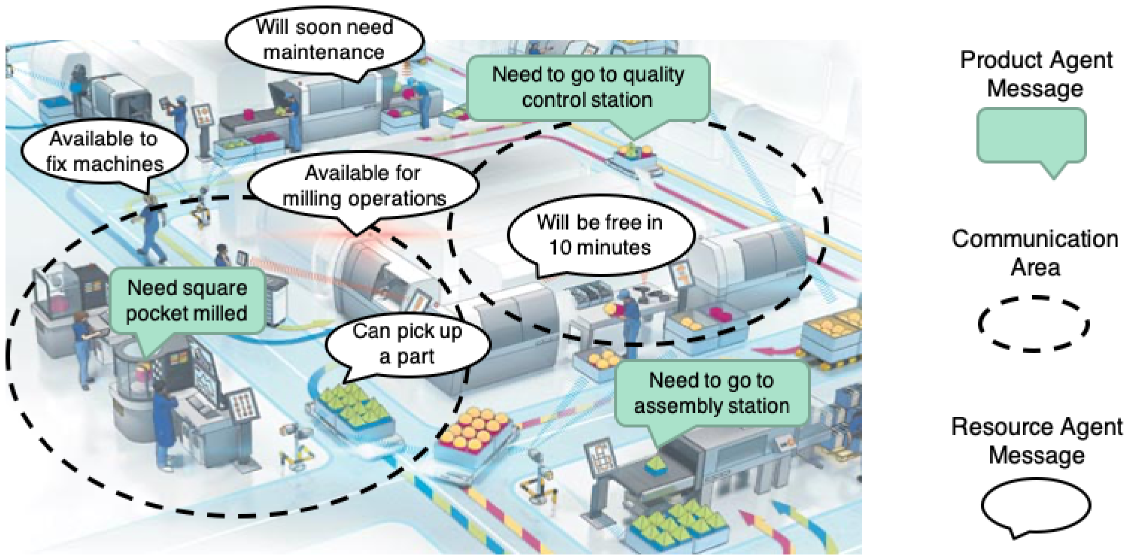

Will soon need maintenance
Need to go to quality control station
Available to fix machines
Available for milling operations
Will be free in 10 minutes
Need square pocket milled
Can pick up a part
Need to go to assembly station
Product Agent Message
Communication Area
Resource Agent Message

Figure 1.1: An overview of the communication between product agents and resource agents for multi-agent control of manufacturing systems. The background picture of the smart manufacturing system is taken from [115].

determine the performance of the entire manufacturing system [10, 90].

The multi-agent architectures proposed in existing works contain various combinations of agents, with each agent having its own purpose and objective [123]. These agents are representations of different manufacturing system components, such as resources on the plant floor, parts in the manufacturing system, and customer orders, among others. The product agent (or a similarly named agent) is a key part of a majority of these multi-agent architectures. A product agent makes decisions for a single part in the manufacturing system [29, 78, 123]. The objective for the product agent (PA) is to schedule and request various operations from the system resources based on customer specifications. The PA accomplishes its objective through communication and negotiations with other product agents and resource agents in the system. A resource agent (RA) is a high-level controller for a resource (e.g. robot, machine) on the shop floor. RAs attempt to safely and efficiently schedule and complete logistic or manufacturing tasks in the system.

Figure 1.1 provides a high-level example and overview of the communication between product and resource agents. In the multi-agent control strategy, the PAs need to communicate with RAs to accomplish required customer specifications. Therefore, the PAs make decisions and send requests (shown by the filled, green message boxes) in a limited communication area. Meanwhile, RAs respond to the relevant messages and make scheduling and operation decisions based on the capabilities, availability, and status of the associated resource. These decisions are relayed to both product and resource agents, as shown in Figure 1.1 via the white, unfilled message boxes. This type of communication and coordination occurs in the entire manufacturing system, as multiple product agents and resource agents communicate to fulfill the customer and manufacturer requirements. More details and definitions of both product and resource agents are provided in Chapter II.

While both both product and resource agents are important components of the multi-agent control strategy, this dissertation focuses on the development of an intelligent, autonomous, and adaptive product agent. In fact, there remain a number of challenges that must be addressed in the design, development, and implementation of PAs to fulfill the Industry 4.0 goals of customized production and improved system flexibility [78]. A majority of existing PAs use rule-based reasoning during decision making. While rule-based reasoning is easy to develop and implement in real world systems, it reduces the flexibility and adaptability of the PA, especially in the presence of unexpected disturbances.

A few model-based approaches to PA intelligence have been proposed to improve the performance of PAs in manufacturing systems [5, 23, 102, 104, 108]. The model-based approaches described in [5, 23, 108] do not describe and define a detailed architecture that can be integrated with existing multi-agent controllers for manufacturing systems. The work in [102, 104] describes and uses an architecture for PA decision making. However, the architecture in [102, 104] does not contain the components and methods to dynamically build and update models used during decision making.

In addition, all of the previously proposed model-based PAs [5, 23, 102, 104, 108] use passive cooperation techniques. This passive cooperation methodology does not allow the PA to identify and negotiate over conflicting constraints in the system, reducing the flexibility of this control strategy.

This dissertation presents new methods to develop autonomous, adaptive, and cooperative PAs. Specifically, this dissertation proposes a novel software architecture for the PA and provides methods that can be used by the PA to improve its performance. The developed methods enable the PA to efficiently build a model of the system capabilities in a dynamic environment, use this model to autonomously plan and execute actions through communication with other agents in the system, and cooperate with other agents in the system during planning and scheduling. Overall, this dissertation work can be used as a blueprint for developing intelligent PAs that can fulfill highly personalized customer orders, while meeting the various constraints found in the manufacturing environment.

1.2 Contributions

The core contributions of the dissertation are described in this section.

1) A model-based architecture for product agents:

In most existing multi-agent architectures, PAs use rule-based reasoning to make decisions in a manufacturing system [29, 59, 71, 76, 80, 106, 116, 120]. While rule-based reasoning is a viable control strategy, it is difficult to scale this approach to a large, complex manufacturing system with numerous constraints and incorporate flexible decision making algorithms to improve the performance of the agent. An alternative control strategy is to use a continuously updated model of the system and optimization techniques for PA decision making. A few works have focused on developing model-based PAs, but the software architecture of the proposed model-based PAs are not outlined in full detail [5, 23, 104, 108]. Therefore, the **first contribution** of this dissertation is a model-based PA architecture that autonomously makes intelligent decisions in an unknown manufacturing environment and cooperates with RAs to schedule and execute actions to guide parts through the manufacturing system. The model-based architecture is presented in Chapter III and in [44, 47].

2) An exploration methodology for efficient product agent model creation in a dynamic manufacturing environment:

To enable effective decision making in a manufacturing environment, a PA must understand the state of the physical part and the capabilities of the surrounding resources. Thus, the PA should be able to dynamically explore the capabilities of the manufacturing environment through communication with other agents in the system. In existing architectures, PA exploration is accomplished by either supplying the PA with a holistic view of system capabilities or by allowing the PA to query all of the resource agents (RAs) in the system. Both of these techniques provide the PA with too much information, either creating unnecessary communication overhead or populating the PA knowledge base with extraneous information. Therefore, the **second contribution** of this dissertation is a novel methodology to enable PA exploration based on a dynamic network of RAs. In the proposed methodology, RAs are able to coordinate and form teams to enable efficient PA exploration. This exploration methodology is described in Chapter IV and in [45].

3) A framework to enable direct and active cooperation for the product agent:

A PA must cooperate with other PAs and RAs when making decisions. Specifically, when planning and scheduling future resource actions, a PA must cooperate with both PAs and RAs to find a sequence of resource actions to complete its associated part's production requirements and not conflict with other parts in the system. Prior work has focused on a passive approach to PA cooperation, where the PA has to find a set of feasible resource actions without negotiating with other agents to resolve scheduling constraints [10, 29, 50, 55, 58, 104, 127]. However, the PA is not always able find a sequence of actions that satisfies all of the existing scheduling constraints and accomplishes its production goals [47]. This limitation of the passive cooperation approach reduces the flexibility of the PA and, in turn, reduces the flexibility and adaptability of the multi-agent control strategy. To improve the flexibility of the PA, a direct, active cooperation approach can be utilized during the PA's planning and scheduling phase. In this type of cooperation, the PA can identify conflicting actions and, through negotiation, find resolutions for these conflicts. Therefore, the **third contribution** of this dissertation is a model-based decision making framework to enable direct, actively cooperative product agents. This framework is presented in Chapter V and further details about the framework and its applications are in [43,48].

4) Integration of the model-based product agent with industrial system controllers:

One major challenge in the development of multi-agent control for manufacturing systems is the integration of agents with industrial system controllers and architectures [61, 123]. To ensure the compatibility of the proposed model-based product agent, the architecture, models, and methods proposed in this dissertation have been tested in several manufacturing system testbeds. Therefore, the **fourth contribution** of this dissertation is the integration of the model-based product agent with existing, standardized system-level controllers for manufacturing systems. Chapter refchap6 presents a description of multi-agent control implementations in three manufacturing testbeds: the Fischertechnik testbed at the University of Michigan [131], the myJoghurt Demonstrator at the Technical University of Munich [2, 121], and the System-level Manufacturing and Automation Research Testbed at the University of Michigan [46].

1.3 Dissertation Overview

This dissertation focuses on the development of intelligent product agents to improve system flexibility. The proposed product agent can explore the local manufacturing environment, make autonomous decisions in a dynamic system, and cooperate with other product agents and resource agents in the system to resolve conflicts. Chapter II provides background information and limitations for existing product agents. Chapter III presents the architecture developed for the product agent. This proposed architecture is a baseline architecture for a model-based product agent that is expanded in the following sections. Chapter IV describes a method for product agent exploration via dynamic resource task negotiation. Chapter V proposes a framework for active and direct cooperation for the product agent. Chapter VI describes implementations of the product agent in physical manufacturing testbeds. Finally, Chapter VII provides concluding remarks for the dissertation and proposes several future directions for this work.

CHAPTER II

Background

In this chapter, an overview of the current state of multi-agent control for manufacturing systems is provided. Then, several sections survey the existing work in the development of intelligence for the product agent, one of the agents commonly found in this control strategy. Specifically, the existing architectures, exploration techniques, and cooperation methods for product agents are presented.

2.1 Multi-agent Control of Manufacturing Systems

The field of agent-based systems and multi-agent control has been developed over the past several decades to tackle problems in a variety of application domains [20, 93, 105, 128]. A number of large, complex systems, from multi-robot systems [30, 95] to power grids [42, 77], have been analyzed and controlled via agent-based technology.

In the area of manufacturing, agent-based technology has been used at various levels of production, from the supply chain level [27, 39, 81] to the factory floor [56, 60, 122]. For system-level control of manufacturing systems, a wide variety of multi-agent architectures have been developed [56, 61, 112, 122]. Most of these architectures identify the roles and responsibilities of the different manufacturing system agents and develop the communication requirements for this control strategy. Therefore, as part of these architectures, a number of agents have been proposed to represent customer

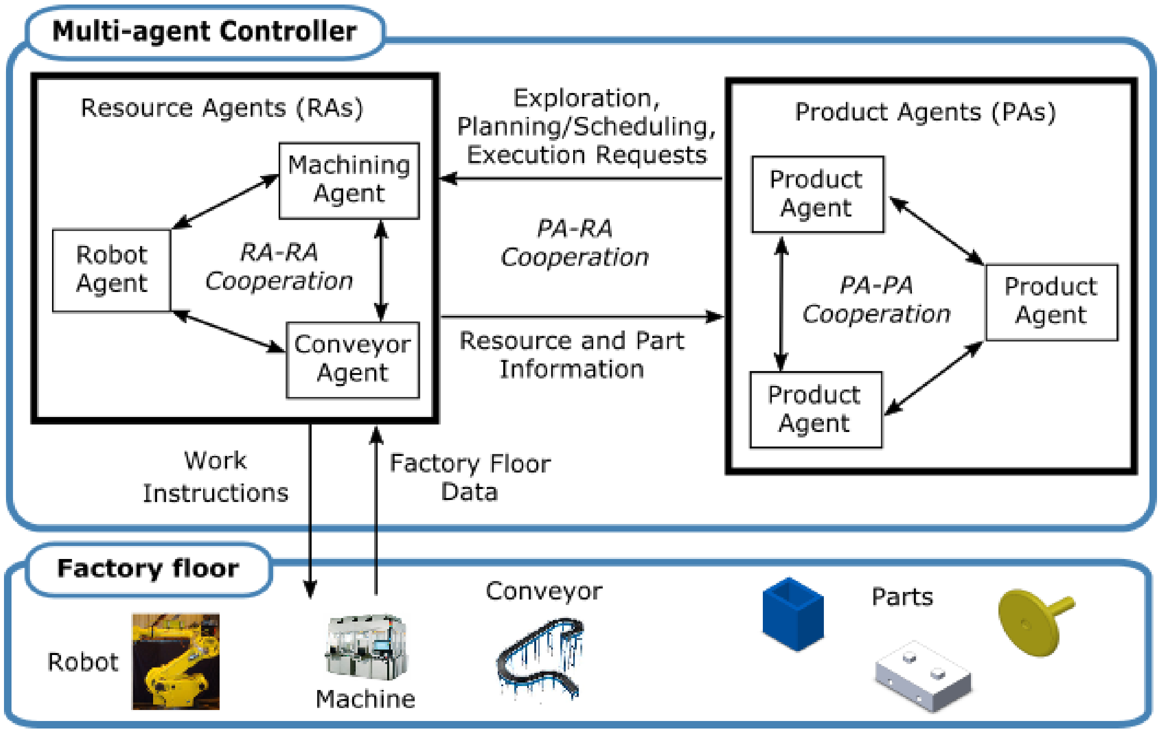


Figure 2.1: The interactions between product agents, resource agents, and the factory floor in a multi-agent architecture.

orders, parts in the manufacturing system, resources (e.g. robots and machines) on the shop floor, among a number of other components [123]. However, for real-time system level control during production, a large majority of the architectures rely on the cooperation and decision making of two types of agents: product agents (PAs) and resource agents (RAs) [119, 123].

A resource agent (RA) is a high-level controller for a resource (e.g. robot, machine) on the shop floor [29, 65]. The RA captures the capabilities and current status of the associated resource, relays that information to other agents in the system, and sends high-level parameters and actuation commands to the resource (e.g. pick up a specific part, start a particular manufacturing operation, etc.). The goal of the RA is to safely and efficiently schedule and complete logistic or manufacturing tasks in the system.

A product agent (PA) makes decisions for a single part in the manufacturing system [29]. The PA receives information about the status of the physical part and the capabilities of the manufacturing system from the RAs. Based on the information received from the RAs and a set of production requirements [113], the PA makes a decision to: (1) obtain the capabilities and plans of other PAs and RAs in the system, (2) plan and schedule future resource actions, or (3) request the execution of a resource action. Figure 2.1 shows the general flow of information for product agents, resource agents, and the factory floor for a multi-agent control strategy.

A number of multi-agent architectures have been integrated into existing manufacturing facilities with promising results [61, 87]. These existing implementations have showcased only the potential of the multi-agent control strategy, as these works have focused on developing agents to make decisions in the presence of very specific disturbances (e.g. a single machine going down) in small-scale manufacturing systems [57]. Thus, there are a number of challenges that must be addressed before this strategy can be used in larger, more complex industrial manufacturing systems. One of the primary challenges is to create more flexible agents that autonomously make decisions in the presence of multiple, different disturbances (e.g., multiple machine failures, addition of new resources to the manufacturing system, unexpected changes to a customer order, etc.). Another challenge is the development of methods and algorithms to allow agents to identify and cooperate with only a subset of the agents in the multi-agent controller. This dissertation focuses on addressing these challenges for only one type of agent: the product agent. The following section provides background information about the product agent's decision making, exploration, and cooperation capabilities and identifies some of the limitations of the existing architectures and methods.

2.2 Intelligent Product Agents

A wide variety of multi-agent architectures have been introduced to control industrial systems. Most of these proposed system-level architectures contain an instance of the product agent (PA) [123]. This section overviews the existing architectures and frameworks used to enable product agent intelligence for the control of parts in a manufacturing system.

2.2.a Rule-based product agents

PROSA (Product-Resource-Order-Staff Architecture) is one of the first examples of a decentralized, distributed control architecture developed to improve manufacturing system flexibility [120]. 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) [72]. 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 [71]. In [71], 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 [29]. ADMARMS uses qualitative design principles to identify the basic knowledge and communication necessary for agents in reconfigurable manufacturing systems. In [29], 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 is not provided. The data structures and PA-RA (resource agent) communication requirements introduced in ADMARMS are 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 [62]. The behavior of the Product Holon, the component similar to a PA in the ADACOR architecture, is specified using a Petri Net [59]. Additionally, a general structure and algorithm that prevents erratic behavior of the holon is described in ADACOR^2, a second iteration of the multi-agent architecture [9, 10]. Although the general guidelines of the Product Holon are provided, the decision making of the Product Holon is 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 [63, 76, 80, 106, 116]. 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 types of reasoners to larger systems. More complex manufacturing systems (e.g. systems with a higher product variety and more technologically advanced resources) require a larger set of rules for the PA. As the number of rules increases, it is more challenging to develop new rules, find conflicting rules, and verify the behavior of the product agent. In addition, as more rules are added, the flexibility of the agent's behavior is reduced. Thus, it is difficult to scale these rule-based PAs to larger, more complex manufacturing systems. One approach that is used to address some of these challenges is model-based reasoning. This approach focuses on developing a model of the system and using optimization techniques for decision making of the PA.

2.2.b Model-based product agents

In the general field of agent-based system and multi-agent control, agents often use a systematic, model-based optimization approach during the decision making process [107]. Hence, model-based reasoning has recently been proposed to improve the autonomy and adaptability of PAs [5, 23, 55, 102, 104]. PAs can leverage various types of models of the manufacturing environment to complete required manufacturing operations and meet the provided specifications.

The most common modeling approach for model-based PAs is the use of finite-state machines (FSMs) to represent the available resources and operations in the manufacturing system [5, 23, 55, 102, 104]. These models encode sequences of logistic and manufacturing operations (i.e., events) that lead to changes in the location or physical composition of the associated part on the plant floor (i.e., states). FSM models use static weights (costs) on events to capture the time required to complete processing, handling, and buffering operations [5, 55]. There have been several extensions of the FSM modeling approach, such as the utilization of Markov decision processes (MDPs) to incorporate the stochastic nature of manufacturing operations on the plant floor [5, 23].

However, even though model-based reasoning has been proposed as a potential alternative to rule-based decision making, a full, detailed architecture for model-based PAs has not been proposed. The work in [5, 23] leverages the belief-desire-intentions

(BDI) architecture to build the model-based product agent. However, while an MDP model of the manufacturing system capabilities and a planning algorithm for this model are described, a full architecture (with models, decision making algorithms, and communication components) that allows the product agent to gather knowledge, schedule future actions, and request actions from RAs is not provided. The work in [102–104] uses the reference architecture developed in [126] for the design of both product and resource agents. The developed product agent architecture contains the following modules: planning and strategy, knowledge base, resource module control, and diagnosis. While a detailed description of the FSM model in the knowledge base and a shortest-path based planning algorithm in the planning and strategy modules is described in detail in [102, 104], only a high-level overview is provided for the other components in this architecture. In addition, the work in [102–104] does not describe the communication and information requirements that enable product agent knowledge gathering in a dynamic manufacturing environment. Chapter III provides a description of a full architecture for model-based PAs that fulfills the communication and behavior requirements set in prior work, e.g., the architectures described in [29, 71, 76, 80, 106, 116, 120, 123]. The product agent architecture presented in Chapter III incorporates models and algorithsm that allow the product agent to gather knowledge, schedule future actions, and request actions from RAs.

In addition, there are limitations to the current modeling frameworks used by the PA. For example, existing PA models encode the scheduling constraints of the PA as *hard constraints*, i.e., these constraints cannot be violated or negotiated by the PA. However, these scheduling constraints in a manufacturing system are sometimes flexible and can be violated through negotiation with other agents in the system. Current model-based PAs do not capture the negotiable actions, i.e., *soft constraints,* in their planning and scheduling models. Therefore, Chapter V describes an extension to existing environment models to capture the soft constraints in a manufacturing system.

2.2.c Product Agent Exploration

In existing multi-agent architectures, the PA gathers knowledge about the manufacturing environment either by utilizing a global view of the system or by querying all of the system RAs.

A global view of system capabilities has been used to help PAs make decisions in some agent architectures [55, 102, 104, 108]. In these works, a formal model of the entire manufacturing environment is generated offline by a manufacturer or central controller. This model is provided to the PA during initialization and is dynamically updated when making planning decisions [55, 108]. Effective implementations of this type of exploration strategy can be found in [102, 104, 108]. While these implementations work well for the case studies provided in those works, this exploration methodology becomes more computationally intensive as the manufacturing systems, and the corresponding models, scale in size and complexity. In addition, the global view models contain information that is not always used during PA decision making. For example, these models may contain resources (e.g. robots and machines) that are not necessary to finish the associated part's production requirements, adding unnecessary complexity for the PA's planning and scheduling. Therefore, to enable more efficient decision making, PA's should use "local models" of the manufacturing environment that are tailored to their production requirements.

Another method for PA exploration is to send operation requests to all RAs in the system, as described in [111]. For example, in [50, 66, 125], the PAs contact all of the RAs in a manufacturing system, allowing the RAs to formulate bids to respond to PA desires. A downside of allowing the PA to contact all of the system RAs is the large amount of communication overhead. Thus, architectures that utilize this type of approach have to manually design a filter that prevents the PA from sending infeasible requests to the RAs. For example, in [66], material handling is ignored to limit the amount of communication overhead. The PA sends a request only to the RAs that control the resources in the same communication zone as the PA's associated physical part. The PA will not be able to find the RAs that are outside the communication zone and, therefore, might not find all of the feasible sequences of actions that will take the physical part to the next desired state.

Both exploration techniques run into problems for larger, more complex manufacturing systems. A single model for the entire system grows in storage space and computational complexity, making it difficult to dynamically update all of the individual states and events. Similarly, the communication overhead necessary to negotiate with all of the RAs increases in size and complexity if a PA tries to communicate with all of the system RAs. However, the PA does not need to know about the status of all of the system resources to make intelligent decisions. The need to store a single model or to communicate with all of the system RAs can be prevented by leveraging the structure of the manufacturing system. Therefore, Chapter IV proposes a technique that uses a connected network of RAs. The exploration technique described in that chapter inherently incorporates dynamic changes to the system, improves PA exploration efficiency, and removes unnecessary PA knowledge and communication overhead.

2.2.d Product Agent Cooperation

During decision making, a PA must cooperate with other PAs and RAs in the system. Specifically, when planning and scheduling future resource actions, a PA must cooperate with both PAs and RAs to find a sequence of resource actions to complete its associated part's production requirements and not come into conflict with other agents in the system.

To put existing PA cooperation strategies into context, the following definitions are provided for direct, indirect, passive, and active cooperation. Using the definition in [96], *direct* cooperation between two agents is defined as the utilization of a one-to-one communication link between the agents for the exchange of information. *Indirect* cooperation between two agents implies that agents pass information and cooperate with each other through other agents or through the environment. Using the general agent cooperation descriptions found in [64, 100], *passively* cooperating agents are defined as a pair of agents that can only request actions that do not conflict with decisions from other agents. On the other hand, *actively* cooperating agents can identify conflicting actions and, through negotiation, find resolutions for these conflicts during decision making. Using these definitions, the cooperation used by PAs in existing multi-agent architectures can be defined.

As shown in Fig. 2.1, PAs have to cooperate with other PAs (PA-PA cooperation) or with RAs (PA-RA cooperation) in the system. PA-RA cooperation is often *direct* and *passive*. PAs directly communicate to the RAs in the system. However, during this direct communication, PAs can only request resource actions that fulfill the scheduling constraints provided by the RA. Examples of this type of cooperation can be found in most multi-agent frameworks for manufacturing systems [10,29,45,47,55, 58,104].

Most of the product agent to product agent (PA-PA) cooperation is *indirect* and *passive* [10, 34, 45, 47, 58, 102, 108]. In these examples, a PA captures the plans of other PAs in the system through information provided by another agent. For example, when queried by a PA, RAs provide other PA schedules as scheduling constraints that cannot be violated. These types of multi-agent architectures are usually first-come-first-serve for the PAs in the system. Therefore, there is never direct communication between PAs in the system and, as previously mentioned, there is no negotiation over the scheduling constraints.

For these passive cooperation approaches, the PA is not always able to find a sequence of actions that satisfies all of the existing scheduling constraints and accomplishes its production goals. This limitation of the passive cooperation approach reduces the flexibility of the PA and, in turn, reduces the flexibility and adaptability of the multi-agent control strategy. *Direct* and *active* cooperation would allow a PA to negotiate with agents when the PA identifies a conflicting event. This type of cooperation provides more flexibility for the PA, allowing the agent to identify previously infeasible solutions that more effectively achieve its goals and react to unexpected disturbances. Chapter V proposes a framework to enable direct and active cooperation for the product agent during planning and scheduling.

CHAPTER III

Model-Based Architecture

This chapter presents the work published in [44, 47] that proposes an architecture for model-based product agents. As described in Section 2.2, a number of multi-agent architectures have been proposed for the control of manufacturing systems. Most of the existing literature on multi-agent control for manufacturing systems focuses on the communication and behavior requirements for each agent, rather than the development of decision-making strategies for each individual agent. As described in Section 2.2.b, a few works focus on developing model-based PAs that use optimization to make decisions. While these works provide high-level overviews of architectures for model-based PAs, a comprehensive architecture that explicitly identifies all of the models, interfaces, and algorithms to enable automatic model creation, autonomous decision making, and adaptation to unexpected disturbances has not been previously developed. This chapter provides such an architecture for model-based, adaptive PAs that fulfills the existing communication and behavior requirements for intelligent products.

The primary contributions of this chapter are: (1) the formulation of the internal components for a model-based product agent based, (2) the utilization of optimization-based planning to schedule future resource actions, and (3) a demonstration of the PA behavior within a simulated manufacturing facility. 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 architecture is designed for integration and implementation into existing multi-agent controllers.

The rest of the chapter is organized as follows. A high-level overview of the PA is presented in Section 3.1. Section 3.2 describes the desires, beliefs, and intentions of the PA. The PA's decision making and communication are discussed in Section 3.3. In Section 3.4, a simulation case study with the proposed PA architecture is presented. Concluding remarks are presented in Section 3.5.

3.1 Overview

An architecture for multi-agent, manufacturing controllers has been previously developed for resource agents in the manufacturing system [65]. This RA architecture contains a Communication Management component, a Decision Making module, a World Model Repository, and a Low Level Interface. The RA architecture is described in more detail and implemented in [102–104]. However, as described in Chapter II, a number of the architecture components and communication requirements are not described in full detail. This chapter uses the high-level descriptions for the resource agent Communication Management, Decision Making, and World Model Repository modules from [65] to develop the components, models, algorithms, and communication requirements for a product agent.

The proposed PA architecture consists of the following components: the Knowledge Base, the Decision Maker, and the Communication Manager. The relationship between these components is shown in Figure 3.1. 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

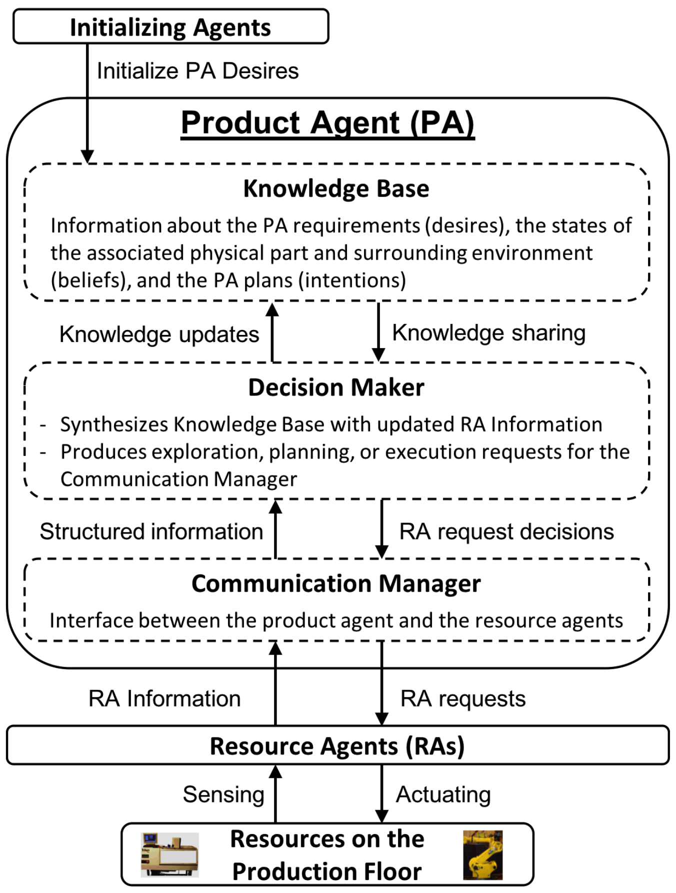

Initializing Agents
Initialize PA Desires
Product Agent (PA)
Knowledge Base
Information about the PA requirements (desires), the states of the associated physical part and surrounding environment (beliefs), and the PA plans (intentions)
Knowledge updates
Knowledge sharing
Decision Maker
- Synthesizes 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

Figure 3.1: An abstracted software architecture for the product agent with general descriptions of each component and the shared information.

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 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 initialized, the PA waits for an update regarding the initial state of the part. The update comes from an RA that senses and identifies the associated physical part and provides this information to the PA.

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:

- 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 3.3.a. The specific details regarding each of the three tasks are described in Sections 3.3.b to 3.3.d.

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 Figure 3.2.

Table 3.1: 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

3.2 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) [128]. A number of functional architectures have been proposed to design intelligent agents [107, 128]. One widely-used functional architecture that has been used in various agent applications is the belief-desire-intention (BDI) architecture [38]. The BDI architecture provides a modular framework to design intelligent agents based on the ideas behind human practical reasoning [32]. The architecture proposes partitioning the agent's information of the surrounding environment (beliefs), the goals that the agent would like to achieve

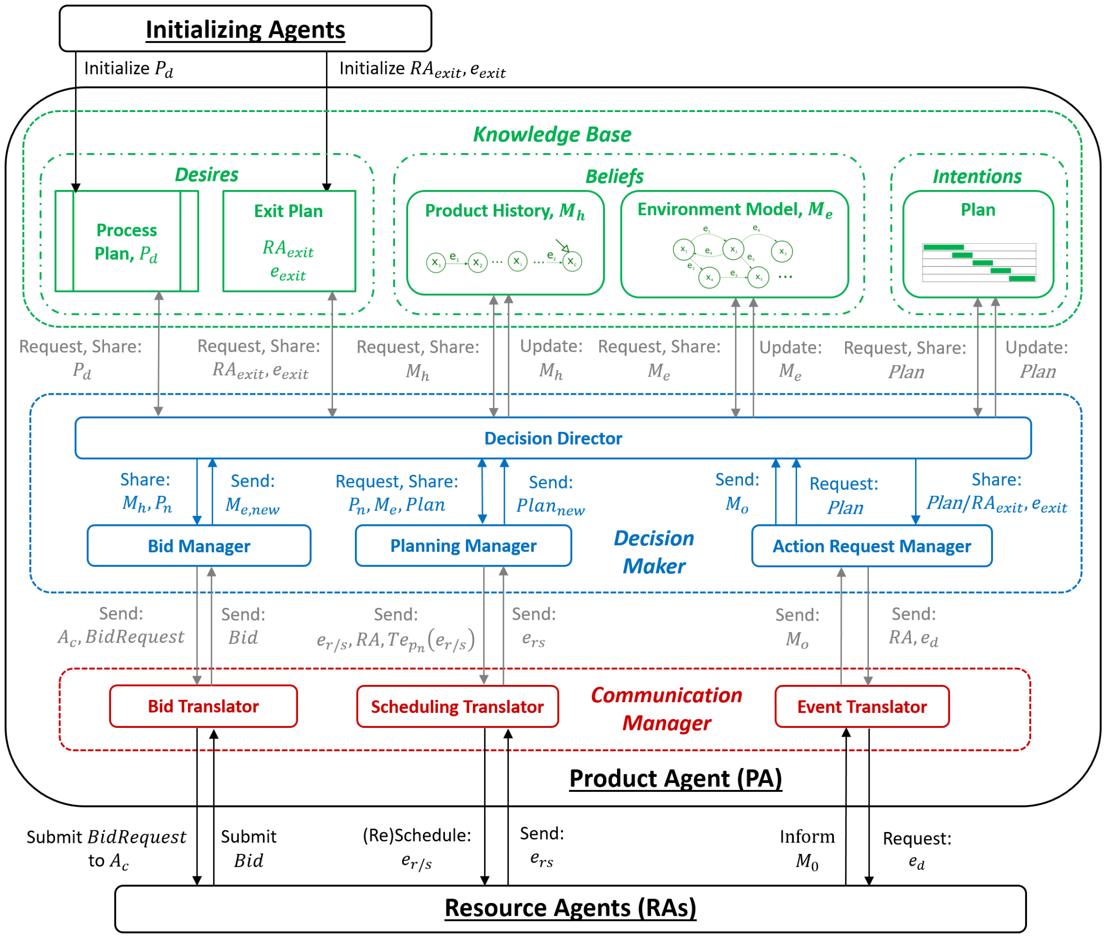


Figure 3.2: The full implementation of the product agent architecture. The components of the Knowledge Base, Decision Maker, and Communication Manager are identified. In addition, the figure illustrates the communication between each of the components. The information transfer between the product agent and other agents is also displayed.

(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 beliefs, desires, and intentions modules. This section describes each of these modules in more detail. Table 3.1 provides the nomenclature used in this section.

3.2.a Desires

The desires represent the requirements for the associated physical part. As shown in Figure 3.2, 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.

3.2.a.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:

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

$$P_{d1} = \{p_{d11}, ..., p_{d1a}\}$$

$$P_{d2} = \{p_{d21}, ..., p_{d2b}\}$$

$$\vdots$$

$$P_{dn} = \{p_{dn1}\}$$

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. $p_{d11}, ..., p_{d1a}$) 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.

3.2.a.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 3.3.a.

3.2.b 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 via finite state machines [22]. 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.). To build and update both of these models, the fusion operation is utilized.

3.2.b.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 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

$X_h = \{x_0, ..., x_n\}$: a set of states of the physical part

$E_h = \{e_0, ..., e_m\}$: 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 one-to-one 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

$s_h = e_0e_1...e_l$: a string (sequence of events) that represents events that have occurred

3.2.b.2 Environment Model

The PA's unique representation of the capabilities of the manufacturing system is the environment model. Unlike the product history, which represents past states of the physical part, the environment model represents the reachable states of the physical part. The model of the local environment, M_e, is constructed using the exploration components of the architecture. The exploration components are discussed in detail in Section 3.3.b. M_e is updated as the physical part traverses the manufacturing system using the PA execution components. More details regarding the execution components are discussed in Section 3.3.d.

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

$Met_e = \{f_1, ..., f_n\}$, where $f_i : E \rightarrow \mathbb{R}, 1 \leq i \leq n$. 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 3.3.c.

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.

3.2.b.3 Fusion Operation

A fusion operation [99] 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:

$X = X_1 \cup X_2$: a union of all the possible states

$E = E_1 \cup E_2$: 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. A description of the exploration methodology and an algorithm for the fusion operation for exploration is provided in Chapter IV and briefly described in Section 3.3.b. During execution, this operation is used to incorporate information about the state of the physical part into the PA's product history, as described in Section 3.3.d.

Note that if the common event between the two tuples, $e \in E_1 \cap E_2$, then Tr, Prp, and Ag in M_{new} are all obtained from M_1. From a practical implementation perspective, the PA should not obtain two tuples with the following properties: $e \in E_1 \cap E_2$ and $Tr_1(x, e) \neq Tr_2(x, e)$ or $Ag_1(e) \neq Ag_2(e)$. This occurrence signifies that there is mismatching information provided to the PA. However, to ensure autonomous decision making, the proposed architecture will set the more recently provided tuple as M_1 to capture the latest available information in M_{new}.

3.2.c 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

$s_p = e_0, ..., e_n$: the planned string (sequence of events)

$Ag_p : E_p \rightarrow RA$: the event-agent association function

$Te_p : E_p \rightarrow (\mathbb{R}_+, \mathbb{R}_+)$: 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 3.3.c and 3.3.d, respectively.

3.3 Product Agent Intelligence

The decision making and communication aspects of the PA are described in this section. Section 3.3.a 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 3.3.b through 3.3.d 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).

3.3.a 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. Figure 3.3 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 3.3.d. 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

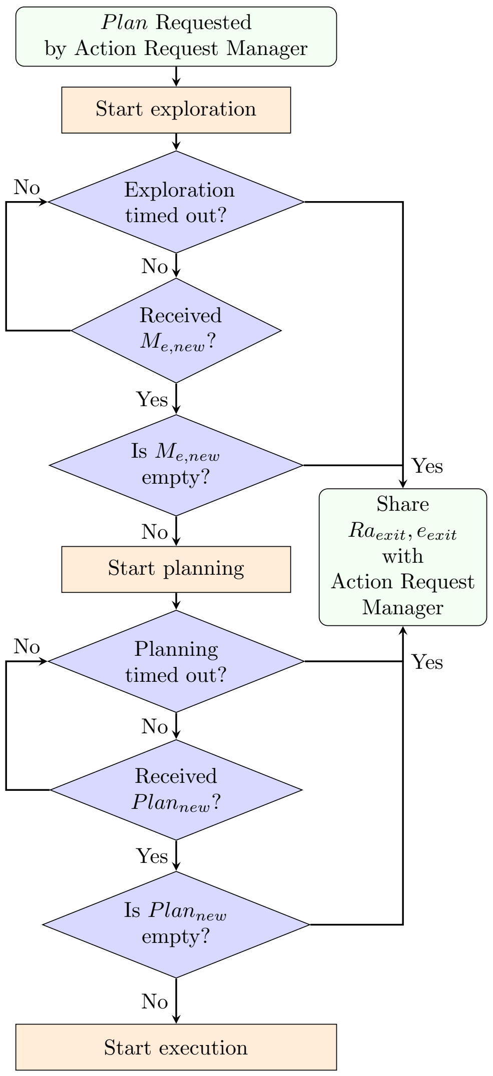
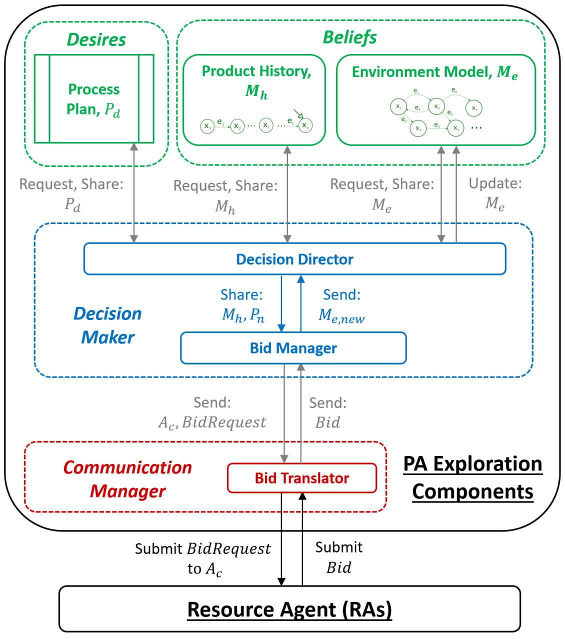


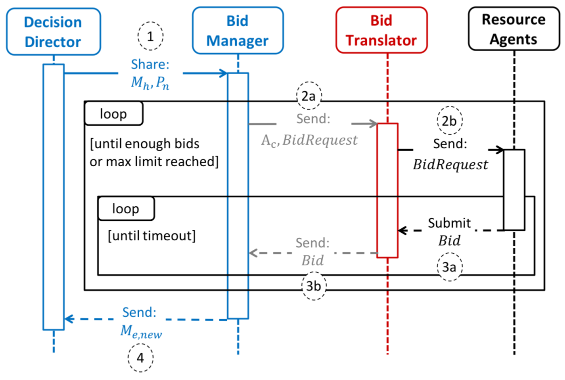
Figure 3.3: The flowchart showing how the Decision Director chooses whether to explore, plan, or execute.

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



(a)



(b)

Figure 3.4: (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).

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.

3.3.b 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. The proposed PA architecture utilizes a novel bidding process to obtain the environment model. As shown in Figure 3.4a, 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.

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:     $X_d = x_0x_1... \in X_{check}$, 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
```

The sequence of steps for exploration is illustrated in Figure 3.4b.

3.3.b.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.

3.3.b.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 [17] or one of the RAs in a system [125] 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.

3.3.b.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 3.2.b.2.

The coordination of bids is left to the RAs in the multi-agent architecture. Chapter IV presents a method to explore the system efficiently. In this section, a brief overview of the exploration methodology from Chapter IV is provided.

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 RA (e.g. mill RA) are neighbors if the material handling resource can move the physical part to/from 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.

3.3.b.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 3.2.b.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 $M_{e,new}$, etc.). Note that to prevent any potential mismatch in information, as described in Section 3.2.b.3, the bids should be ordered in the reverse order that they were received in, i.e. the first bid should be the most recently received one and the last bid should be the first bid received. 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

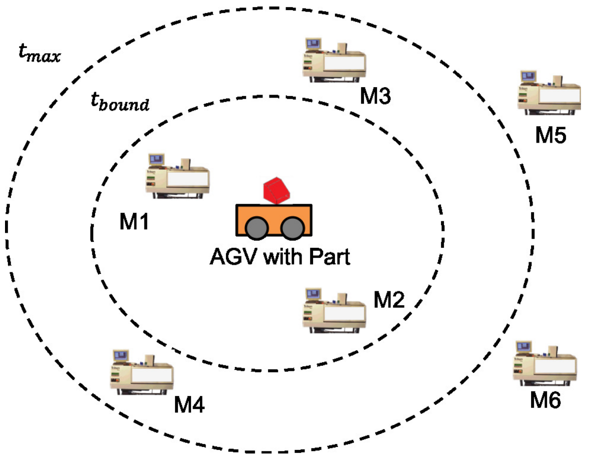


Figure 3.5: An example of the exploration of a PA in a manufacturing system with Autonomous Guided Vehicles (AGV) and multiple manufacturing machines. Note that t_{bound} includes both the transportation time and manufacturing operation time.

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.

3.3.b.5 Exploration Example

An example scenario of PA exploration is shown in Figure 3.5. 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.

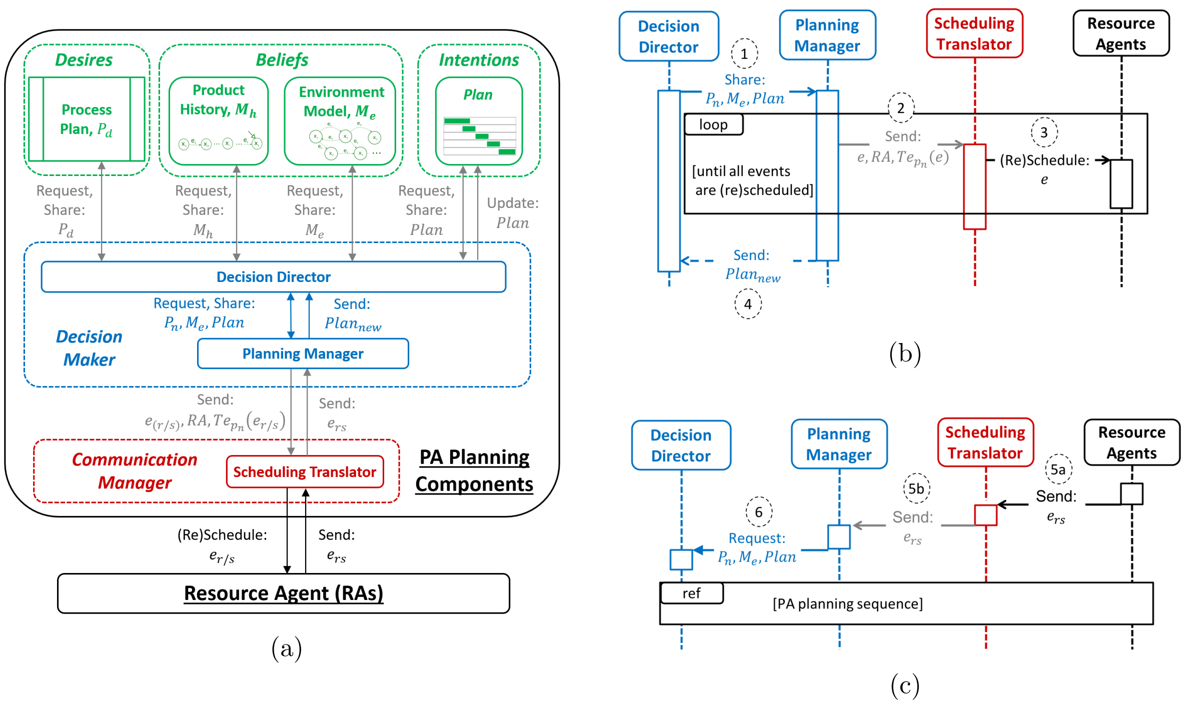


Figure 3.6: (a) shows the architecture components for planning. (b) displays the planning sequence diagram. Once the environment model, desired physical properties, and old plan are passed to the Planning Manager (1), a new plan is formulated. After the Planning Manager identifies events to schedule (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).

3.3.c Planning

The PA's planning components, shown in Figure 3.6a, create plans for the PA using the process plan and the environment model. The PA planning and re-planning sequences are shown in Figure 3.6b and Figure 3.6c, respectively.

3.3.c.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:

$$s^* \in \arg\min_{s_k} \sum_{i=1}^{|s_k|} \alpha_1 f_1(e_{ki}), \sum_{i=1}^{|s_k|} \alpha_2 f_2(e_{ki}), ... \tag{3.1}$$

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

where s^* is the desired string (sequence of events) that starts at the 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.

To solve Eq. 3.1, the event planning optimization problem, an appropriate optimization algorithm must be used. In Section 3.4, the product agent transforms the multi-objective optimization problem into a weighted single-objective problem and uses Dijkstra's shortest path [26] to find a solution. Since the RAs only return bids to the PA if a feasible path for the associated part can be found (see Section 3.3.b and Chapter IV), a solution to the shortest path, weighted single-objective problem will always be feasible. A complexity analysis for a similar environment model and shortest path algorithm can be found in [104]. The results in [104] show that there is a need to reduce the complexity of the problem to allow the product agent to make timely decisions. As described in Section 3.3.b and Chapter IV, this reduction in complexity can be accomplished by developing an efficient exploration methodology. In addition, note that, as part of the proposed product agent architecture, a time-out for the running time of the 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.

Note that the optimization problem in Eq. 3.1 is a local optimization problem for the system. The solution to Eq. 3.1 depends on the individual goals of the product agent and the local environment model built through the exploration methodology proposed in Section 3.3.b and described in detail in Chapter IV. While the solution to the problem is a local optimum, the case studies described in Section 3.4 show that the local solutions found by the PAs will drive the global behavior of the system to finish manufacturing the parts in a timely manner. Future work will look at understanding the difference in system performance when decisions are made by individual agents with a local view of the system and when decisions are made by a centralized controller with a global view of the system.

If $s^* = e_0e_1...e_m...e_{m+n}$ 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:

$$s_{p_n} = e_0e_1...e_m$$

$$Ag_{p_n}(e_i) = Ag_e(e_i), \text{ where } 0 \leq i \leq m$$

$$Te_{p_n}(e_i) = \Big(T(e_0) + T(e_1) + ... + T(e_{i-1}),$$
$$T(e_0) + ... + T(e_{i-1}) + T(e_i) + \epsilon\Big), \text{ where } 0 \leq i \leq m$$

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.

3.3.c.2 Event Scheduling

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

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

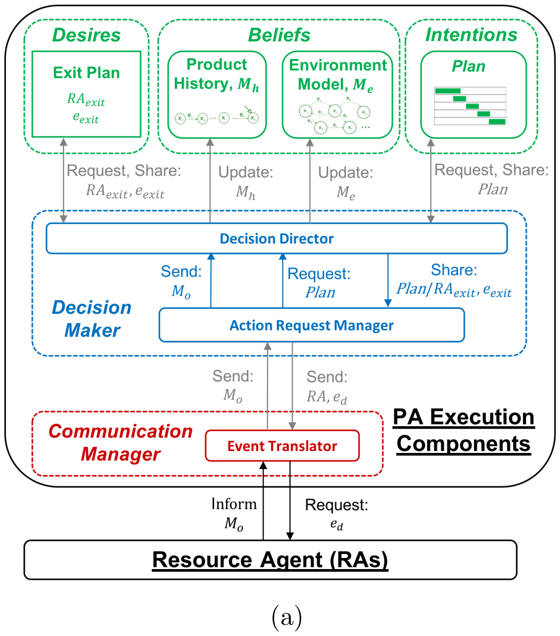
where $Te_{p,1}(e)$ is the start time of the event and $t_{current}$ is the current time. Using this information, the Planning Manager sends to the Scheduling Translator each future event that needs to be removed, $e_r \in E_r$, the RA that performs the event, $Ag_{p_n}(e_r)$, and the event's time duration, $Te_{p_n}(e_r)$. 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 to the Scheduling Translator the event, the RA that performs the event, $Ag_{p_n}(e_s)$, and the time duration of that event, $Te_{p_n}(e_s)$. 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 $Plan_{new}$.

3.3.c.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 can not accomplish a scheduled action due to some unforeseen change in the manufacturing environment



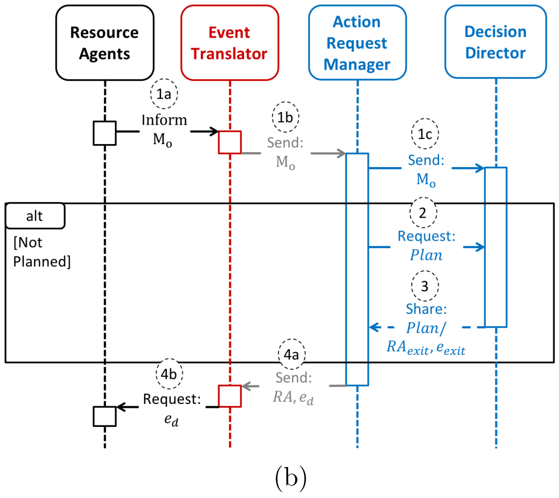


Figure 3.7: (a) displays the architecture components involved in execution. (b) shows the sequence diagram for execution. RA information regarding states of the physical part is passed to the Decision Director (1a, 1b, 1c). If an unexpected event occurs, the Action Request Manager requests (2) and receives (3) a new plan. Once the PA has a feasible plan, the PA continues to request actions from the RAs (4a,4b).

(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 Figure 3.6b. Afterwards, the PA starts to follow the same planning methodology as described previously in this section. However, when computing s^* in Eq. 3.1, the Planning Manager adds a constraint, $e_{rs} \notin s^*$, in the optimization problem to ensure that a new plan is found.

3.3.d Execution

The PA execution components, shown in Figure 3.7a, 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

Figure 3.7b.

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)

Table 3.2: 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

In addition to determining 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 3.2.b, 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:

$x_c = x_{c,o}$, where $x_{c,o}$ is obtained from the M_o

$s_h = s_h s_o$, where s_o is appended to the end of s_h

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

3.4 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.

3.4.a Case-study set-up

The Repast Symphony (RepastS) platform [91] can be used to model and simulate the behavior of manufacturing systems that are controlled via multi-agent strate-

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

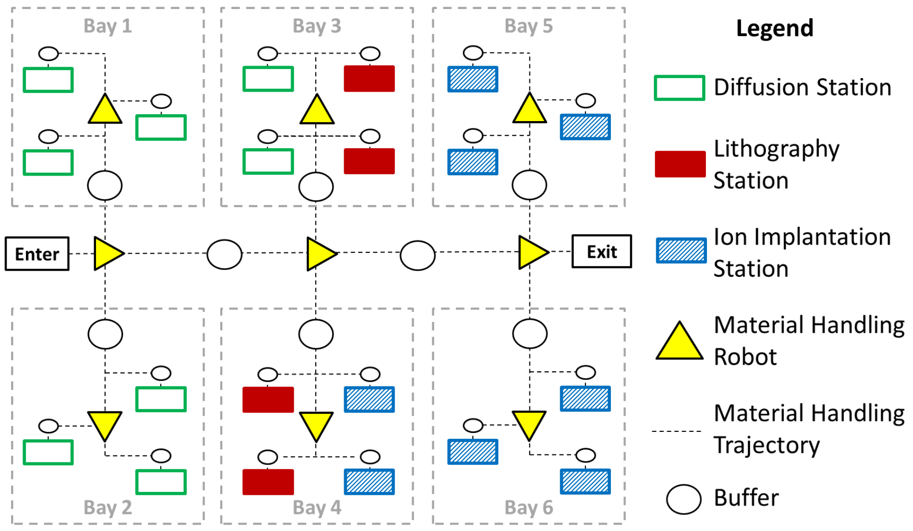


Figure 3.8: The set-up for the model-based PA simulation case studies.

gies [8]. Due to the high modeling power of the Repast environment [74], 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 [130], a semiconductor manufacturing facility. The simulated facility contains 20 stations that are connected via a network of 9 material handling robots, as shown in Figure 3.8. 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.

Each robot, machine, and buffer has 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

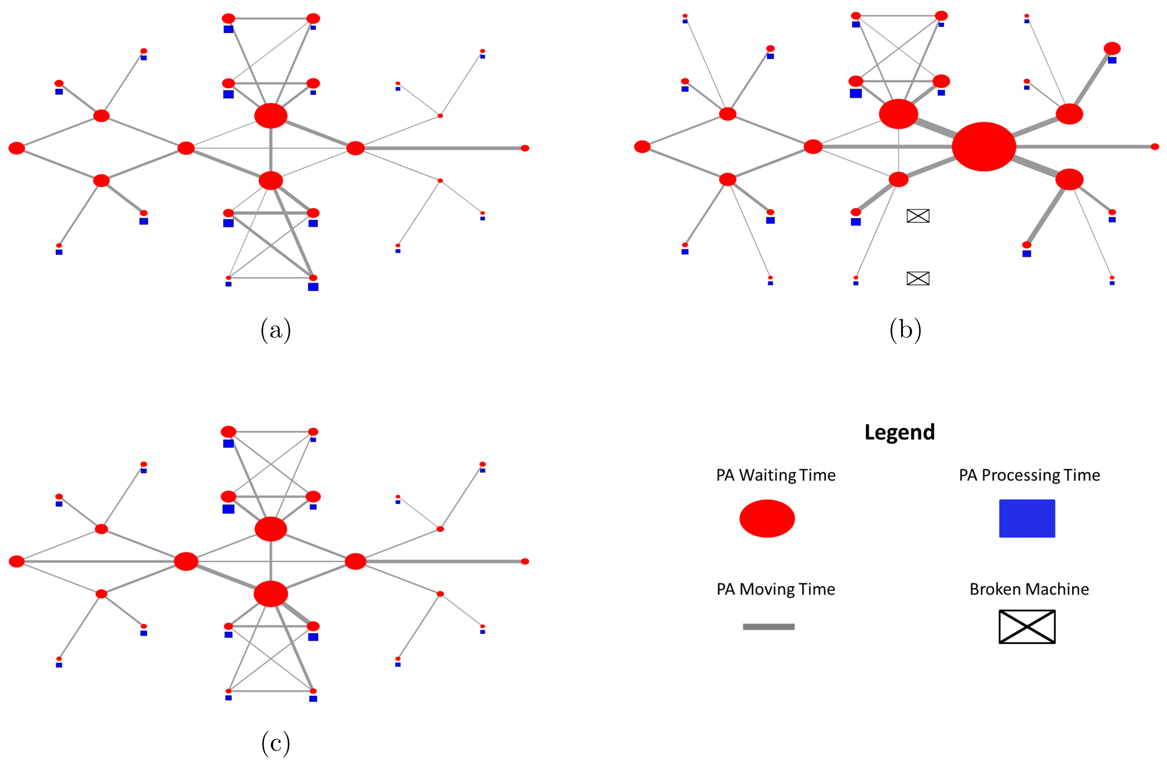


Figure 3.9: Figures (a)-(c) represent the behavior of the PAs for Case Studies (1)-(3). Larger nodes (red circles and blue squares) represent longer times that the PAs spent at that location. Similarly, thicker edges (grey lines) represent the number of times that the particular path was taken for the completed parts. The two, independent scale factors (for node size and edge thickness) are the same across all three cases.

Figure 3.8. 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 3.2. The order of the manufacturing processes depends on the lot type, as described in the following separate case-studies.

3.4.b 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 [94]. 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. Note that for non-simulated testbeds, the time-outs need to be tuned through experimentation. For example, for the myJoghurt testbed described in Chapter VI, the time-out time 150 ms and 50 ms for exploration and planning, respectively.

During exploration, the initial t_{bound} is set to 300 ticks and the t_{max} is set to 10000 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 [26] 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.

3.4.c 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 Figure 3.9.

3.4.c.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 $\rightarrow$ P2 $\rightarrow$ P3 $\rightarrow$ P4 $\rightarrow$ P5 $\rightarrow$ P6.

The case study was performed five times. On average, 135.4 lots were produced by the system and 14.6 exited the system without finishing. The average flow time for the completed parts was around 1700 ticks/part. Fig 3.9a 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 Figure 3.9a.

3.4.c.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 3.9b, 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.

3.4.c.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 [130], are introduced into the system. The first new process plan, P_{d1}, is P2 $\rightarrow$ P1 $\rightarrow$ P3 $\rightarrow$ P5 $\rightarrow$ P4 $\rightarrow$ P6. The second new process plan, P_{d2}, is P4 $\rightarrow$ P5 $\rightarrow$ P6 $\rightarrow$ P1 $\rightarrow$ P2 $\rightarrow$ 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.

3.4.d 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 [107].

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 [49]. 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) [3, 60].

3.5 Conclusions

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 chapter, 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.

CHAPTER IV

Dynamic Exploration

This chapter presents the work published in [45] that proposes a methodology for dynamic resource agent task negotiation to enable product agent exploration. As described in Chapter III, a PA is tasked with making various intelligent decisions, such as requesting an RA to execute a manufacturing process or to schedule future operations. Prior to making this decision, a PA must understand the state of the physical part and the capabilities of the surrounding resources. Thus, the PA should be able to dynamically explore the capabilities of the manufacturing environment through communication with the RAs and build an environment model. As described in Section 2.2.c, PA exploration in existing architectures is accomplished by either supplying the PA with a holistic view of system capabilities or by allowing the PA to query all of the system RAs. Both of these techniques provide the PA with too much information, either creating unnecessary communication overhead or populating the PA knowledge base with extraneous information. Therefore, the primary contribution of this chapter is a novel methodology to enable PA exploration based on a dynamic network of RAs. In the proposed methodology, RAs are able to coordinate and form teams to enable efficient PA exploration.

The rest of the chapter is organized as follows. Section 4.1 provides the set-up for the new PA exploration technique. Section 4.2 explains how a network of RAs can enable PA exploration and Section 4.3 provides attributes of this exploration approach. Conclusions are stated in Section 4.4.

4.1 Multi-agent Architecture

In this section, the resource agent and product agent knowledge needed for exploration via a dynamic resource network is described. The structures used by the PAs and RAs for communication are provided.

4.1.a Resource Agent Knowledge

To utilize the proposed PA exploration, RAs are provided a model of their capabilities and the states that are shared with neighboring RAs.

4.1.a.1 Capabilities Model

A finite state-machine can be used by an RA to represent its resource capabilities [104]. To enable effective communication and cooperation, a resource capabilities model for one RA is defined similarly to the PA environment model in Chapter III. The resource capabilities model is defined as the following tuple:

$M = (X, E, Tr, Prp_p, Prp_{np}, T, T_c, x_i, X_m)$, where:

$X = \{x_0, ..., x_n\}$: a set of states that can be achieved by any physical part utilizing the resource

$E = \{e_0, ..., e_l\}$: a set of events that the RA can trigger that correspond to processes in the physical system

$Tr : X \times E \rightarrow X$: a state transition function

$Prp_p : X \rightarrow P_p$: a function that maps states to P_p, which is the set of physical properties that represent a change in the physical composition of the part

$Prp_{np} : X \rightarrow P_{np}$: a function that maps states to P_{np}, which is the set of physical properties that represent no changes to the physical composition of the part

$T : E \rightarrow \mathbb{R}_+$: the time it takes for the event to occur

$T_c : E \rightarrow Sc_e$: the current schedule for each event

$x_i \in X$: an initial state (updated for every bid request)

$X_m \subseteq X$: a set of marked states (updated for every bid request)

Events can be logistic or manufacturing operations [31]. An example element of Prp_p is an addition of a feature to a part [7]. Example elements of Prp_{np} are the location and orientation of a part. $Sc_e = \{(t_0, t_1), (t_2, t_3), ..., (t_{k-1}, t_k)\}$ represents times when a resource is available for event e, with $0 \leq t_0 < t_1 < t_2 < ... < t_k$. Thus, the event can be scheduled between t_0 and t_1, between t_2 and t_3, etc.

This capabilities model is constantly updated as the RA receives information from the resources on the shop floor (e.g., sensor data that identifies a new state for the physical part) and other agents in the system. T is updated based on the RA's estimation of event duration. The estimation can be improved with data from the manufacturing system. Sc_e is updated as PAs and other agents request to use the associated resource. Note that x_i and X_m are updated based on an incoming bid request, as described in Section 4.2.b.

4.1.a.2 Neighboring RAs

For cooperation with other RAs, each RA is provided the states, $X_s \in X$, shared between itself and other resources. Note that shared states have to be a part of the RA's capabilities model. For example, a machine RA and a material handling RA might share a state with one physical property: the drop-off location inside the

Table 4.1: Resource agent knowledge model example.

X	{"At mill1", "Milled pocket"}
E	{"Mill pocket", "Set for pickup"}
Tr	Tr("At mill1", "Mill pocket") = "Milled pocket" Tr("Milled pocket", "Set for pickup") = "At mill1"
Prp_p	Prp_p("At mill1") = $\emptyset$, Prp_p("Milled pocket") = {(pocket, 5x5x2 cm, ...)}
Prp_{np}	Prp_{np}("At mill1") = Prp_{np}("Milled pocket") = {(mill1)}
T	T("Mill pocket") = 120 sec, T("Set for pickup") = 5 sec
T_c	T_c("Mill pocket") = {(0,120),(500,620)}, T_c("Set for pickup") = {(120,125),(620,625)}
N	N("At mill1") = $\{\text{Robot1 Agent}, \text{Robot2 Agent}\}$

machine. Each RA stores this information in a table that relates the shared states to specific RAs, $N : X_s \rightarrow 2^{RA}$, where 2^{RA} denotes a power set of RA and RA is the set all resource agents in the system. All of the RAs in that table, $RA_{neigh} = \{N(x) : x \in X_s\}$, are the neighboring RAs.

4.1.a.3 Resource Agent Knowledge Example

An example description of a resource capabilities model for a mill is provided in Table 4.1. In this scenario, the mill has pocket milling capabilities and has two neighboring robots, which pick up finished parts and place new parts into the mill.

4.1.b Product Agent Knowledge

The PA contains a process plan and product history, as described in Chapter III. The process plan is provided during PA initialization. As for the product history, the PA keeps track of actions that affect the physical part through communication with RAs.

4.1.b.1 Process Plan

The process plan is a representation of a customer order, defined as the following ordered list: $P_d = (P_1, P_2, ...P_n)$, where $P_1 = \{p_{11}, ..., p_{1a}\}$, $P_2 = \{p_{21}, ..., p_{2b}\}$, ..., $P_n = \{p_{n1}, ..., p_{nm}\}$ and p represents a desired physical property. The desired physical property can be a feature added to a part or a specific location in the manufacturing system, i.e. $p \in P_p \cup P_{np}$. In the process plan, all of the physical properties in a prior set must be complete before trying to accomplish a new set of properties. For example, all of the properties in P_1 and P_2 must be completed before trying to accomplish P_3 properties. The goal of the PA is to request actions from RAs that will enable the physical part to accomplish the entire process plan.

4.1.b.2 Product History

The product history is a representation of the current state of the associated physical part. The product history is defined as: $PH = (X_v, Prp_v, RA_c)$, where $X_v = x_0, x_1, ..., x_m$ is a sequence of achieved states, $Prp_v =: X_v \rightarrow P_p \cup P_{np}$ is a set of completed physical properties, and RA_c is the last RA to inform the PA of a visited state. To keep the PA informed, the RAs match each part to an associated PA via auto-identification technology [79] (e.g. Radio-frequency identification tags). Then, as a resource performs actions on a part, the associated RA sends information to the identified PA to update X_v, Prp_v, and RA_c.

4.1.c Agent Communication

The two structures used for agent communication are bid requests and bids. PAs use the bid request to start the exploration process and receive bids from RAs. In the proposed methodology, the PAs can send multiple bid requests with different desired physical properties to the RAs. As described in Section 4.2, the bids and bid requests are used by the RAs to enable task negotiation and by the PAs to formulate a better

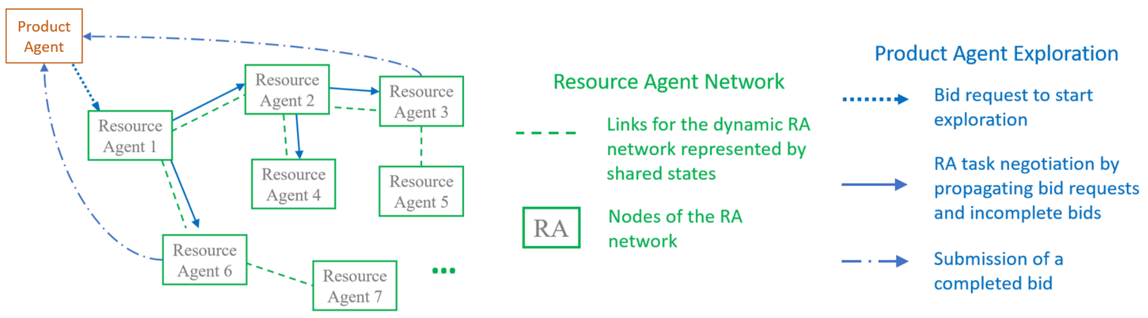

Product Agent
Resource Agent 1
Resource Agent 2
Resource Agent 3
Resource Agent 4
Resource Agent 5
Resource Agent 6
Resource Agent 7
...
Resource Agent Network
Links for the dynamic RA network represented by shared states
RA
Nodes of the RA network
Product Agent Exploration
Bid request to start exploration
RA task negotiation by propagating bid requests and incomplete bids
Submission of a completed bid

Figure 4.1: An overview of the agent communication when the network of RAs is used for PA exploration.

understanding of the local manufacturing environment.

4.1.c.1 Bid request

A bid request is defined as $BR = (PA, x_c, P_{id}, t_{bound})$, where PA is the requesting PA, x_c is a starting state, P_{id} is a set of incomplete, desired physical properties and t_{bound}: the maximum allowable time to complete P_{id}.

4.1.c.2 Bid

The bid is defined as: $Bid = (Str_e, Str_x, Ag_h, Prp_p, Prp_{pn}, T, T_c)$, where $Str_e = e_0e_1...e_{n-1}$ is a sequence of events, $Str_x = x_0x_1...x_n$ is a sequence of states, $Ag_h : e \rightarrow RA$ is an event-agent association function between each event and the RA that performs that event, and Prp_p, Prp_{pn}, T, T_c are defined similarly to the resource capabilities model.

4.2 Resource Agent Task Negotiation

In this section, the development and utilization of a dynamic network of resource agents to enable PA exploration is described. The PAs use a bidding process to obtain the capabilities of surrounding resources. For this process, the RAs form teams dynamically to respond to PA requests.

An overview of the PA exploration approach is shown in Figure 4.1. The goal of PA exploration is to obtain the capabilities and schedules of the resources in the vicinity of the associated physical part. The first step in the exploration is a bid request from the PA to the last RA to inform the PA of a visited state. The bid is then propagated by the RA to its neighbors until a team of RAs that can satisfy the bid request is found. If a team of RAs is formed, then a bid is submitted to the PA. Otherwise, RAs keep sharing the bid and bid request until the propagation conditions are not satisfied. After waiting for RA responses, the PA synthesizes the bids to obtain the model of the local manufacturing environment.

4.2.a Product Agent Bid Request

The PA must create a bid request to start the exploration process. For the bid request, x_c is the last state in the sequence of visited states from the PA's product history. P_{id} is created by comparing the completed physical properties in the product history to the properties required by the process plan. Finally, the PA picks a reasonable t_{bound} to provide a maximum amount of time for the RAs to finish P_{id}. Note that the PA can vary both the P_{id} and t_{bound} to expand or limit the amount of information it will receive from the RAs, as described in Chapter III. The initial bid request is sent to RA_c, which is the last RA to contact the PA based on the PA's product history. If the PA had just entered the system (i.e. no product history), it will wait until an RA contacts it with some initial information regarding the location of the physical product.

4.2.b Resource Agent Task Negotiation

An RA uses its resource capabilities model to submit a bid to the requesting PA or propagate the bid request to its neighboring resources. To decide whether to submit or propagate the bid, the RA checks if it can satisfy the bid request. We use the following definitions from discrete event system theory to describe this process [22] A string is defined as a sequence of events. $\mathcal{L}(M)$, the set of all strings that are allowable in M, denotes the language generated by M. $\mathcal{L}_m(M) \subseteq \mathcal{L}(M)$, the set of strings in $\mathcal{L}(M)$ that end in a marked state, denotes the marked language of M.

The initial state and marked states of the capabilities model are updated using the bid request. The initial state is defined as: $x_i := x_c$. The marked states are defined as the desired states of the bid request:

$$X_m := \{x \in X \mid Prp_p(x) \in P_{id} \text{ or } Prp_{np}(x) \in P_{id}\} \tag{4.1}$$

An RA can satisfy a bid request if it finds a string, $s_e = e_0 e_1 ... e_{n-1}$, and states, $s_x = x_0 x_1 ... x_n$ that satisfy a few conditions. First, the string must be in the marked language of M and s_x must be a feasible sequence of states in the RA's capabilities model:

$$s_e \in \mathcal{L}_m(M) \text{ and } x_{i+1} = Tr(x_i, e_i), \text{for } 0 \leq i \leq n-1 \tag{4.2}$$

In addition, all desired physical properties in the PA's bid request must be satisfied. Any physical properties of $x \in s_x$ that alter the part's composition must be a part of the PA's bid request. Thus, the following constraint must be satisfied:

$$\cup_{i=1}^{n} Prp_p(x_i) = P_{id} \setminus \cup_{i=1}^{n} Prp_{np}(x_i) \tag{4.3}$$

Note that any number of properties that do not alter the part's composition are allowed. However, if the states have extra properties that alter the part's composition, they do not satisfy this constraint. To satisfy Eq. 4.3, all of the properties in P_{id} must be accomplished in s_x. Finally, all of the properties must be accomplished in the requested amount of time:

$$\sum_{i=1}^{n-1} T(e_i) \leq t_{bound} \tag{4.4}$$

A bid is submitted if the RA finds s_e and s_x to satisfy these conditions. Otherwise, the bids are propagated to neighbors.

4.2.b.1 Submitting a Bid

If Eq. 4.2 – 4.4 are satisfied, the RA creates a bid. For the bid, $Str_e = s_e$ and $Str_x = s_x$ are obtained from the above equations. Prp_p, Prp_{pn}, T, T_c are obtained from the resource capabilities model. For the bid's Ag_h, all of the events in Str_e are mapped to the RA creating the bid. The bid is submitted to the PA in the bid request.

4.2.b.2 Bid Request Propagation

If Eq. 4.2 – 4.4 are not satisfied, the RA decides if the bid should be propagated. The marked states are redefined as the set of shared states:

$$X_m := X_s \tag{4.5}$$

The RA will propagate a bid if there exists a string, $s_{e,p} = e_0 e_1 ... e_{n-1}$, and states, $s_{x,p} = x_0 x_1 ... x_n$ that satisfy Eqs. 4.2 and 4.4. Note that $s_{e,p}$ has at least one event to ensure propagation, i.e. $n > 0$. Finally, without the requirement to accomplish all of the physical properties in the process plan, Eq. 4.3 is relaxed. However, the string must not have any physical properties that alter the composition of the part unless part of the desired physical properties:

$$\cup_{i=1}^{n} Prp_p(x_i) \subseteq P_{id} \tag{4.6}$$

If these conditions are satisfied, the RA creates an incomplete bid, Bid_{ic}, where

$Str_{e,ic} = s_{e,p}$, $Str_{x,ic} = s_{x,p}$, $Prp_{p,ic}, Prp_{pn,ic}, T_{ic}, T_{c,ic}$ are obtained from the resource capabilities model, and $Ag_{h,ic}$ maps all of events to the current RA. In addition to Bid_{ic}, a new bid request BR_{ic}, is created in the following manner:

$PA_{ic} = PA$ (no change to the requesting PA)

$x_{c,ic} = x_n$

$P_{id,ic} = P_{id} \setminus \left[\bigcup_{i=1}^{n} \left(Prp_p(x_i) \cup Prp_{np}(x_i)\right)\right]$

$t_{bound,ic} = t_{bound} - \sum_{i=1}^{n} T(e_i)$

where $x_{c,ic}$ is the shared state, $P_{id,ic}$ are the physical properties that must be finished to complete the bid, and $t_{bound,ic}$ is the allowable remaining time to complete those properties. Bid_{ic} and BR_{ic} are sent to $N(x_{c,i})$, the neighboring RAs with the shared state.

4.2.b.3 Creating a Team of Resources

If an RA receives an incomplete bid and a bid request, the RA will try to complete the bid by answering the provided bid request. To complete the bid, the RA tries to find a string, $s_{e,c}$, and a sequence of states, $s_{x,c}$, that satisfy 4.1 - 4.4 for the propagated bid request, BR_{ic}. If these constraints are satisfied, the RA creates and submits a completed bid. To complete the bid, the state and event sequences are appended to the incomplete bid, i.e. $Str_e = e_{0,ic}...e_{n,ic}e_{0,c}...e_{m,c}$, where $e_{i,ic} \in Str_{e,ic}$ for $0 \leq i \leq n$, $e_{j,c} \in s_{e,c}$ for $0 \leq j \leq m$. In this formulation, n and m denote the length of the events in the incomplete bid and the newly found sequence of events, respectively. Similarly, the newly found states are appended to the states of the incomplete bid. Using the RA's capabilities, Prp_p, Prp_{pn}, T, T_c are updated based on the newly added states and events. Finally, for Ag_h, all of the added events are mapped to the RA creating the bid. The complete bid is then sent to the PA in the bid request.

If the RA cannot complete the bid, it will propagate the bid further. If Eqs. 4.2 and 4.4 – 4.6 are satisfied by a sequence of events and states, the RA updates the incomplete bid. To update the bid, the state and event sequences are appended to the $Str_{e,ic}$ and $Str_{x,ic}$ and $Prp_{p,ic}, Prp_{pn,ic}, T_{ic}, T_{c,ic}, Ag_{h,ic}$ are updated. The RA sends the updated, but still incomplete, bid and a new bid request to its neighboring RAs.

4.2.c Product Agent Bid Compilation

After sending the initial bid request, the PA waits for the RAs to reply with a set of bids. If the PA is not satisfied with the number of bids received (e.g. 0 bids), then it would send out another bid request to re-explore the system. To obtain different bids, the PA can change the number of states that it initially wants completed or increase the maximum allowable time in the bid request.

If the PA is satisfied with the number of bids received, then a model of the manufacturing environment can be compiled. For the PA, the model is the following tuple: $M_e = (X_e, E_e, Tr_e, Prp_{p,e}, Prp_{np,e}, Ag_{h,e}, T_e, T_{c,e}, x_c)$, where x_c is the current state of the physical part, $Ag_{h,e}$ is the event-agent association function, and all other elements are defined similarly to the resource capabilities model. Note that the PA can use this model to determine desired states for the part, as described in Chapter III.

The environment model can be compiled using the fusion operator [99] on the RA bids. Algorithm 2 illustrates how to create M_e using the bids and past product states from the product history, X_v. Note that two states, x_a and x_b, or two events, e_a and e_b, are defined as equivalent if and only if:

$$Prp_p(x_a) = Prp_p(x_b) \text{ and } Prp_{np}(x_a) = Prp_{np}(x_b) \tag{4.7}$$

$$e_a = e_b \text{ and } Ag_h(e_a) = Ag_h(e_b) \tag{4.8}$$

In other words, two states are equivalent if they have the same physical properties

Algorithm 2 Compiling all bids into an environment model

```
Input: $Bids = Bid_1, Bid_2, ..., Bid_m, X_v$
Output: $M_e$
Initialize: $X_e, E_e, Tr_e, Prp_{p,e}, Prp_{np,e}, Ag_{h,e}, T_e, T_{c,e}$
 1: // Iterate through each of the bids:
 2: for all $Bid \in Bids$ do
 3:     // Add the states and associated properties
 4:     for all $x \in Str_{s,Bid}$ do
 5:         if $x \notin X_e$ (see Eq. 4.7 for state equality) then
 6:             Add $x$ to $X_e$
 7:             Let $Prp_{p,e}(x) := Prp_{p,Bid}(x)$
 8:             Let $Prp_{np,e}(x) := Prp_{np,Bid}(x)$
 9:     end for
10:     // Add the events and associated properties
11:     for all $e_n \in Str_{e,Bid}$ do
12:         if $e_n \notin E_e$ (see Eq. 4.8 for event equality) then
13:             Add $e_n$ to $E_e$
14:             Let $Tr_e(x_n, e_n) := x_{n+1}$
15:                 where $x_n$ is the $n^{\text{th}}$ element of $Str_{s,Bid}$
16:             Let $Ag_{h,e}(e) := Ag_{h,Bid}(e)$
17:             Let $T_e(e) := T_{Bid}(e)$
18:             Let $T_{c,e}(e) := T_{c,Bid}(e)$
19:     end for
20: end for
21: Let $x_c := x_m$, where $x_m$ is the last element in $X_v$
```

and two events are equivalent if they have the same name and the same associated RA.

With this newly compiled environment model, the PA can plan, schedule, and request the requisite events from RAs. Before sending scheduling requests, the PA uses this model to understand the RA capabilities and schedules. The PA can find paths to less congested machines and work with the RAs to minimally affect existing RA schedules. In addition, if a PA is not able to find a feasible path after thoroughly exploring the system, the PA will ask for assistance.

4.3 Key Attributes of the Proposed Approach

In this section, the dynamic nature of the RA network and the benefits of the proposed approach are discussed.

4.3.a Dynamic Network of Resource Agents

The proposed PA exploration framework is enabled by creating a dynamic communication network of RAs. The nodes in the communication network correspond to individual RAs in the system, while the links represent the shared states between neighboring RAs. Note that a link exists only if both resources have physical access to the shared state.

When a new resource is added into the system, the capabilities model and the table of neighboring resources and states are provided to a newly initialized RA. To establish the link in the communication network, the new resource informs its neighboring RAs about their mutually shared states. Once informed, each neighbor RA can add the new RA to its own table of neighboring resources and states. With the communication link established, bid requests from PAs in the system will be able to reach the new resource agent.

Similarly, an RA can be removed from the communication network (e.g. a resource goes down) by contacting its neighboring RAs. Once the RA is removed from a neighbor's table of shared states, the communication link is broken. Note that another RA might need to take over the removed RA responsibilities if the removal causes a drastic change in the manufacturing system (e.g. unreachable machines).

The connectivity of this network drives the time performance of PA exploration. The speed of PA exploration can be improved when links are removed from the network. While removing links may improve the exploration time performance, RAs can not use the removed connection when responding to PA exploration requests. Thus, the level of connectivity should be optimized based on the capabilities of each resource, the number of PAs and RAs in the system, and the speed of the agent communication and decision making.

4.3.b Benefits of Proposed Negotiation Strategy

4.3.b.1 Flexible part behavior

The physical part can be placed anywhere in the system and, if there exists a way to obtain desired physical properties in a reasonable amount of time, the network of resource agents will submit a bid that satisfies the criteria of the associated PA. The PA does not need to have any prior knowledge of the system to function. To obtain an accurate representation of the physical part's environment, the RAs must continuously keep updating the PA regarding the state of the associated physical part.

4.3.b.2 Response to changes on the plant floor

The dynamic network of RAs can help in improving the flexibility of manufacturing systems. The network can capture changes in the capabilities of the entire manufacturing system as RAs are added, moved, or removed. Modifications of individual resource capabilities can also be captured in the RA models. Using the approach presented, PAs can re-explore the system and change their plans to adapt to dynamic conditions on the plant floor or to changes in customer orders.

4.3.b.3 Security of RA capabilities

The RAs decide how much of their capabilities they share with the PA. The manufacturer can hide some resource capabilities from certain PAs. This provides the manufacturer more control over the amount of shared information between agents. In addition, there is no central storage place that contains a detailed description of all of the resources. This removes a central point of failure from existing agent-based manufacturing controllers.

4.3.b.4 Integration with existing communication protocols

Due to its bidding mechanics, the proposed technique can be implemented using existing communication protocols. For example, for the implementation study in the myJoghurt demonstrator described in Section 6.2, the standardized contract net protocol [111] was used for agent communication.

4.4 Conclusions

The multi-agent control strategy consisting of product and resource agents (PAs and RAs) has been proposed to improve manufacturing system flexibility. For this strategy, the PAs must effectively guide an associated physical part through a manufacturing system. To accomplish this goal, PAs must understand the capabilities of the surrounding environment. In this chapter, a novel approach for PA exploration based on the propagation of bids and bid requests over a dynamic network of RAs is proposed. Using the proposed methodology, RAs form teams to provide PAs with a comprehensive view of the surrounding environment. In addition to enabling RA task negotiation, this approach limits the amount of information available to agents, improving security and reducing communication overhead of agent systems.

The exploration methodology described in this section is used for all of the simulations described in Chapters III – V and the physical testbed implementations presented in Chapter VI. This methodology is scalable, as the model-based PA was able to combine bids from a number of resource agents (up to 20) in a reasonable amount of time. The PA was able to build models with up to 100 states and up to 100 events without scalability issues. Specific exploration times for the physical testbeds are provided in Chapter VI. Future work will look at finding the computation, storage, and time limitations for the exploration methodology.

CHAPTER V

Direct, Active Cooperation

This chapter presents the work published in [43,48] that proposes direct and active cooperation for the PA. PAs must cooperate with other PAs and RAs when making decisions. Specifically, when planning and scheduling future resource actions, a PA must cooperate with both PAs and RAs to find a sequence of resource actions to complete its associated part's production requirements and not come into conflict with other agents in the system. As described in Section 2.2.d, prior work has focused on a passive approach to PA cooperation, where the PA has to find a set of feasible resource actions without being able to negotiate with other agents over existing scheduling constraints. However, as shown in Chapter III the PA is not always able find a sequence of actions that satisfies all of the existing scheduling constraints and accomplishes its production goals. This limitation of the passive cooperation approach reduces the flexibility of the PA and, in turn, reduces the flexibility and adaptability of the multi-agent control strategy.

To improve the flexibility of the PA, a direct, active cooperation approach can be utilized during the PA's planning and scheduling phase. In this type of cooperation, the PA has to identify the *soft constraints* in the system, i.e., the scheduling constraints that can be negotiated through communication with other agents. Then, the PA must choose which, if any, of the soft constraints prevent the agent from accomplishing its individual goals. Finally, the PA needs to communicate and cooperate with the appropriate agents in the system to resolve the identified conflicts.

The main contribution of this chapter is a model-based decision making framework to enable direct, actively cooperative product agents (DA-PAs). Specifically, this chapter proposes a model to capture the soft constraints in the system, an algorithm that enables the DA-PA to identify which of the soft constraints are conflicting with its individual goals, and a communication method to allow the DA-PA to negotiate and coordinate with other agents in the system.

The rest of the chapter is organized as follows. Background information about the priced timed automaton modeling formalism is provided in Section 5.1. Section 5.2 overviews the components of the DA-PA's knowledge base used for the cooperation framework. Section 5.3 describes the framework used for a direct and actively co-operating product agent. Case studies for the cooperation framework are given in Section 5.4. Section 5.5 provides concluding remarks.

5.1 Priced Timed Automata

In existing model-based product agents, both the scheduling constraints and other various metrics (e.g., energy cost) are combined and encoded as edge weights on a finite state machine (FSM) [23, 104]. This encoding approach was utilized for the PA's environment model presented in Chapters III and IV. This encoding makes it difficult to identify the scheduling constraints in the system, separate them into hard and soft constraints, and enable direct, active cooperation. Therefore, this work proposes to leverage the priced timed automaton (PTA) modeling formalism [13, 52] to capture the PA's scheduling constraints.

Here we provide a formal definition of PTA, based on [13], and some additional definitions we use in our work. A PTA is composed of a set of states, X, connected by a set of edges, E. Each edge is labeled with an event, $\sigma \in \Sigma$. A PTA also has a set of clocks, continuous variables $\mathbf{c} \in C = \mathbb{R}^{n_c}_{\geq 0}$, where n_c is the number of clocks in the system. All clocks have the same constant, positive growth rate and an initial value of zero.

A finite set of Boolean indicator functions of clock values, $\mathcal{B}$, represents the constraints. For example, for a clock value $\mathbf{c} \in C$, the Boolean indicator function $\mathcal{I}_{\mathbf{c}^i \leq a}(\mathbf{c}) \in \mathcal{B}$ evaluates to *true* if and only if the i^{th} element of $\mathbf{c}$ is less than or equal to a. Each element of $\mathcal{B}$ is either an invariant for a location, which must evaluate to true for the system to occupy the location, or a guard on an edge, which must evaluate to true to traverse an edge. Reset maps on edges set clocks to predetermined values when the edge is traversed.

Additionally, a PTA has costs on states and edges. Costs on edges are discrete increments added to the total running cost when the edge is traversed, while the states have cost rates, and steadily accumulate cost over time.

Definition V.1 (Priced Timed Automata)**.** A PTA $\mathcal{A}$ is defined as an 8-tuple $\mathcal{A} = (X, C, \Sigma, E, I, R, P, x_0)$, where

- $X = \{x_0, x_1, ..., x_{n^x}\}$ is a finite set of states of the PTA
- $C = \{c_0, c_1, ..., c_{n^c}\}$ is the set of clocks
- $\Sigma = \{\sigma_0, \sigma_1, ..., \sigma_{n_\sigma}\}$ is a finite set of events
- $E \subseteq X \times \mathcal{B} \times \Sigma \times X$ is a finite set of edges
- $I : X \to \mathcal{B}$ is the invariant operator
- $R : E \times C \to C$ is a reset operator
- $\mathcal{K} : X \cup E \to [0, \infty)$ maps the locations and edges to their costs
- $x_0 \in X$ is the initial state

There are several differences between the PTA model described in Definition V.1 and the environment model presented in previous chapters. Note that, to align with existing PTA definitions, the events of the environment model described in Chapters III and IV have been decomposed into the edges and the events in Definition V.1. For Definition V.1, the events, Σ, are the labels for the edges, E, in the priced timed automaton. In addition, the costs of the system are associated with the states of the system, as described in Section 5.2. The addition of the system clocks is the most significant change to the environment model and is described in more detail in Section 5.2.

The guards are embedded in the edge definition, with $\mathcal{B}$ denoting constraints on the clocks for each edge, i.e., the guard conditions. A transition is a formal description of a change in the system state and the transition type. There are two types of transitions, discrete transitions (changes in location) and delay transitions (changes in clock values). Let $t_{y,z}$ denote a transition with $y \in \{s, d\}$ denoting if the transition is a discrete (s) or a delay (d) transition, and $z = \{e, \tau\}$ where $e \in E$ and $\tau \in \mathbb{R}$. We define transitions as $x' \xrightarrow{t_{y,e}} x$, where $x', x \in X$. We provide further definitions on the clock structure before formally defining the two types of transitions.

We utilize a two clock structure in this work. The two clock set is given as $C = \{c_l, c_g\}$ through the rest of the chapter, noting that additional clocks may be added for various applications. The local clock, c_l, represents the time spent at a single state and, thus, is always reset to 0 when entering a state via a transition in the system. The global clock, c_g, represents the absolute time for the system and is never reset in the system. Additionally we use the valuation operator $val(\cdot)$ to denote the value of a clock. The valuation operator captures the values of clocks for a state $x \in X$ at the time when a discrete transition corresponding to an edge $e \in E$ is *taken* from the state x to a new state x'. For example $val(c_g^x)$ denotes the value of the global clock at state x at the time of transition to a new state x' by taking an edge. Then, the two types of transitions for PTA are defined as follows.

Definition V.2 (Discrete Transition)**.** A transition $x' \xrightarrow{t_{s,e}} x$ is a discrete transition if $e = (x', \sigma, x) \in E$ and clock valuations are incremented as $val(c_g^x) = val(c_g^{x'})$ and $val(c_l^x) = 0$. Note that we say a discrete transition is *taken* when an edge in the model is traversed.

Definition V.3 (Delay Transitions)**.** A transition $x' \xrightarrow{t_{d,\tau}} x$ is a delay transition if $x' = x$, the invariant $I(x')$ is true, and clock valuations are incremented as $val(c_g^x) := val(c_g^x) + \tau$ and $val(c_l^x) = \tau$.

Note that by explicitly defining the two transition types with the specific clock valuation updates, we implicitly defined a reset operator, R, for the PTA. We omit R in later definitions for brevity.

5.2 Knowledge Base

This section provides a description of the DA-PA's knowledge base, namely its goals, environment model, and decision making model. The goals and models described in this section are used for the direct, active PA cooperation described in Section 5.3.

5.2.a Goals

During its initialization, a DA-PA is provided a process plan, exit plan, performance weights, and the priority of the associated part with respect to other parts in the system. The process plan and exit plan are similar to the components described in Chapter III. The performance weights have been added to aid in the PA's decision making using the PA's environment model described later in this section.

5.2.a.1 Process Plan

The process plan is formulated based on a customer order and consists of an ordered list of desired physical properties for the associated part, as described in Chapters III and IV. Physical properties include part locations or part features, e.g. "part at exit loading dock" or "part has hole with taper" [7,98]. Formally, the process plan is defined as: $Plan = (PP_d, Dl)$, where $PP_d = (pp_1, pp_2, ...)$ is the ordered list of desired physical properties pp_i. Note that this is a simplification of the process plan presented in Chapter III, as the process plan in Chapter III is an ordered list of sets of desired physical properties. In addition, to enable the DA-PA to finish the process plan per the customer order, deadlines can be added to each of the properties in the process plan. Formally, we set $Dl : PP_d \to \mathbb{R}_{\geq 0}$ as the latest allowable finish time for a property. The PTA-based encoding of the environment model described later in this section enables the addition of deadlines to the process plan described in Chapter III, allowing the PA to better meet customer requirements.

The process plan is developed offline and provided to the DA-PA (e.g. [92]). Since the plan is computed offline and, additionally, to ensure that communication between agents is kept in a local neighborhood as described in Chapter IV, we assume that the DA-PA accomplishes the process plan one physical property at a time. Therefore, the DA-PA will only look to complete the next unfinished property in the process plan. As shown in the case study presented in Chapter III, in most cases, a model-based PA is able to complete its process plan one property at a time. Future work will look at developing a methodology to incorporate multiple physical properties when making decisions and using other types of structures (e.g., temporal logic [101]) to encode the specifications in the process plan. Note that the DA-PA keeps track of the completed physical properties and can identify the next unfinished physical property when required.

5.2.a.2 Exit Plan

The DA-PA is initialized with an exit plan that is used if the DA-PA cannot find a sequence of resource actions to fulfill the process plan. A detailed description and formulation of the exit plan is presented in Chapter III. The DA-PA can exit the manufacturing system by calling an exit agent in the exit plan. For example, an exit agent can be a human agent [131] who can retrieve the part from the manufacturing system. The decision of the DA-PA to exit the system is discussed in detail in Section 5.3.

5.2.a.3 Performance weights

There are various, measurable performance metrics, $PM = \{pm_1, pm_2, ..., pm_n\}$, for resources on the plant floor, e.g., energy and material cost. These metrics are tracked and stored by the associated RAs in the system. The RAs share these performance metrics when they communicate their capabilities with the DA-PA. These metrics are stored in the environment model of the DA-PA. A formal description of their encoding in the environment model is provided in Section 5.2.b.

A set of performance weights is provided during the DA-PA's initialization. These performance weights are computed offline based on the customer order or the manufacturer requirements. Each performance weight corresponds to a metric. Formally, the set of performance metrics is defined as $\{\alpha_{p_i} \in \mathbb{R}_{\geq 0} \mid p_i \in PM\}$. The magnitude of the weight represents the relative importance of the corresponding metric for the DA-PA. For example, if a DA-PA favors minimizing material cost over energy usage, the following relation will hold $\alpha_{material} > \alpha_{energy}$. The DA-PA uses the performance weights and metrics during its decision making to identify desirable resources and resource actions in the system.

5.2.a.4 Agent Priority

The DA-PA is provided an importance value, $pr \in \mathbb{N}$, which represents the priority of the associated physical part when compared to other parts in the system. An importance value of 1 signifies that the part is of the highest priority. Therefore, a part i has a higher priority than part j if $pr_i < pr_j$.

Note that while the process plan, performance weights, and agent priority are all part of the DA-PA's individual goal, they are linked to the overall performance of the manufacturing system. The process plan is based on a customer's order and needs to be designed to accomplish all of the requirements in that order [92, 113]. The performance weights can be set by the manufacturer to prioritize certain actions for the DA-PA (e.g., use less energy or use less material). Finally, the agent priority can be set by the needs of the manufacturer and the importance of the customer order. For example, if a customer sends in a highly profitable order with harsh penalties for production delays, the DA-PAs associated with the order would be provided with a high priority. Future work will look at understanding how to map the needs of the manufacturing system to the goals of the DA-PA.

5.2.b Environment Model

The capabilities and scheduling constraints of the local manufacturing environment are captured by the DA-PA's environment model, as described in Chapter III. The environment model is updated by the DA-PA as other agents (RAs and PAs) provide the capabilities and scheduling constraints for the resources in the local manufacturing environment to the DA-PA. This information is communicated either following a DA-PA request or if there is a disturbance in the manufacturing system. The environment model is defined as the following tuple:
$\mathcal{A}_{EM} = (X, Prp, x_0, \Sigma, E, C, \Gamma, I, \mathcal{K}_{nm}, \mathcal{K}_{sft}, Ag)$:

$X = \{x_1, x_2, ...\}$ is the set of states for the associated physical part

$Prp : X \rightarrow PP$ maps states to physical properties

$x_0 \in X$ is the current state of the part

$\Sigma = \{\sigma_0, \sigma_1, ...\}$ is a finite set of events, where each event represents the start or completion of a resource action (e.g a logistic or manufacturing operation)

$E \subseteq Q \times \Sigma \times Q$ is a finite set of edges

$C = \{c_l, c_g\}$ are the local and the global clocks

$\Gamma : \{\gamma_{x_i,1}, \gamma_{x_i,2}, ...\gamma_{x_{i+n},1}, ...\}$ is the set of slack variables, where $\gamma_{x_i,j} \in \mathbb{R}$ is the j^{th} slack variable for state x_i.

$I : X \rightarrow \mathcal{B}[val(C), val(\Gamma)]$ maps states to their constraints as a function of the clock and slack variables

$Ag : X \cup E \cup \mathcal{B}[val(C), val(\Gamma)] \rightarrow Agents$ maps states, events, and constraints to corresponding agents

$\mathcal{K}_{nm} : X \times PM \times val(C) \rightarrow \mathbb{R}_{\geq 0}$ maps states, performance metrics, and valuations of clocks to nominal cost values

$\mathcal{K}_{sft} : X \times PM \times val(\Gamma) \rightarrow \mathbb{R}_{\geq 0}$ maps states, performance metrics, and valuations of slacks to cost-of-constraint-violation values

where X, Prp, x_0, E are the discrete event dynamics, C, Γ, I encode the time-based state constraints, Ag is the agent association function, and $\mathcal{K}_{nm}, \mathcal{K}_{sft}$ are the state costs for the DA-PA.

The vectors $val(C)$ and $val(\Gamma)$ are valuations of the variables C and Γ for each state in the system. These valuations capture the values of the clocks and associated slack constraints for a state $x \in X$ at the time when a discrete transition corresponding to an edge $e \in E$ is *taken* from state x to a new state x'. To ensure unique clock and slack valuations, we enforce the $\mathcal{A}_{EM}$ to be acyclic, while noting that cyclic graphs can be "flattened" into acyclic graphs efficiently [88].

$\mathcal{A}_{EM}$ is an extension of the PTA modeling formalism presented in Section 5.1. A description of how the manufacturing environment is mapped to each component is discussed in the rest of this section. Note that there are several differences when $\mathcal{A}_{EM}$ is compared to the PTA from Section 5.1. As previously discussed, the reset operator, R, is left out for brevity. Only states (not edges) of $\mathcal{A}_{EM}$ have constraints and these state constraints are encoded in the invariant, I. Finally, there are a number of extensions to the constraints and costs of $\mathcal{A}_{EM}$, which are discussed in the rest of this section.

5.2.b.1 Discrete event dynamics

X, Prp, x_0, E represent the capabilities of the manufacturing system. X are the states of the physical part in the system. Each state is a combination of several physical properties of the part. Physical properties define locations and physical composition of the part (e.g. "part at machine" or "part feature completed") [98] or operations performed on the part (e.g. "moving the part" or "working on a manufacturing process"). Prp maps each state of the environment model to one or more of these physical properties. A current state, x_0, is updated when an RA informs the DA-PA that it has started or finished a resource action. Each event, Σ, represents an instantaneous (takes 0 time) start or completion of a manufacturing or logistic operation [31].

5.2.b.2 Time-based Constraints

The state constraints, $\mathcal{B}$, limit the amount of time that the associated part can be in the state (e.g. how long the part can stay at a location). These constraints are split into hard and soft constraints, $\mathcal{B} = \mathcal{H}[val(C)] \cup \mathcal{S}[val(C), val(\Gamma)]$. Hard constraints,

$\mathcal{H}[val(C)]$, cannot be violated by the DA-PA (e.g. minimum time required to complete a manufacturing operation). Soft constraints, $\mathcal{S}[val(C), val(\Gamma)]$, contain slack variables, Γ. As described in Section 5.3.c, these soft constraints may be violated by the DA-PA through negotiation with other agents in the system. Note that there is no limit for the number of constraints at each state. Therefore, all of the constraints in the model are stored in the invariant mapping, I. This mapping links a state to all of its constraints.

All time constraints, $\mathcal{B}$, are logic expressions [14]. There are two types of constraints used in the environment model: *bound constraint* and *interval gap constraint.* The bound constraint is a time limit that represents an absolute limit when the part can enter a state or leave a state. The gap constraint represents an interval when the part cannot be at a certain state. Formally, the two constraints are defined as follows:

Definition V.4 (Bound Constraint)**.** A bound constraint, $\mathcal{B}_b$, for state, x, is satisfied (i.e. evaluates to *true*) if and only if a clock valuation (either local or global) at the state, $val(c_b^x)$, for $c_b \in C$ satisfies:

$$val(c_b^x) \bowtie t_b \pm a_\gamma \, val(\gamma_b^x) \tag{5.1}$$

where $t_b \in \mathbb{R}_{\geq 0}$ is the time limit for the clock, $a_\gamma \in [0, 1]$ is the hardness coefficient, γ_b^x is a slack variable associated with the constraints on state x, and $\bowtie$ is one of the following logical operators: $<$, $\leq$, $=$, $\geq$, $>$.

Note that a bound constraint is a hard constraint if $a_\gamma = 0$.

Definition V.5 (Interval Gap Constraint)**.** An interval gap constraint, $\mathcal{B}_g$, for state x and interval (t_l, t_u) is satisfied if and only if the local and global clock valuations at the state, $val(c_l^x)$ and $val(c_g^x)$, satisfy:

$$\neg\big[X_1 \vee X_2\big] \wedge \big[X_1 \vee X_3\big] \tag{5.2a}$$

$$X_1 := val(c_g^x) - val(c_l^x) < t_l + a_{\gamma,l}\, val(\gamma_1^x) \tag{5.2b}$$

$$X_2 := val(c_g^x) - val(c_l^x) > t_u + a_{\gamma,u}\, val(\gamma_u^x) \tag{5.2c}$$

$$X_3 := val(c_g^x) \leq t_l + a_{\gamma,l}\, val(\gamma_1^x) \tag{5.2d}$$

where $t_l, t_u \in \mathbb{R}_{\geq 0}$ and $t_u > t_l$, $a_{\gamma_l}, a_{\gamma_u} \in 0, 1$ are the hardness coefficients, γ_l^x, γ_u^x are unique slack variables associated with the constraint, and $\neg, \wedge, \vee$ are logic operators.

Note that the expression $val(c_g^x) - val(c_l^x)$ represents the global clock time when a discrete transition is taken into state x. Therefore, $\big[X_1 \vee X_2\big]$ represents that the part cannot enter the state during a time interval of the global clock, (t_l, t_u). $\big[X_3 \vee X_4\big]$ represents that if the part enters the state before the start of the interval, (t_l), it must also exit the state before the start of the interval.

5.2.b.3 Transitions

The objective of the DA-PA's decision making is to find clock and slack variable valuations that satisfy all of the state constraints in the system. To satisfy these constraints, the DA-PA must choose a sequence of *discrete* transitions from E that transition the agent from one state x to a new state x' and *delay* transitions that keep the agent at the same state x, but take a finite time $\tau \in \mathbb{R}$. The definitions of these two transition modes are provided in Section 5.1. Details about how a DA-PA chooses these transitions is provided in Section 5.3.

Remark V.6. It is always feasible to denote delay and discrete transitions in an alternating sequence since delay transitions are additive, and we can define a zero time delay transition between two consecutive discrete transitions.

5.2.b.4 Agent association function

The DA-PA builds the environment model by requesting capabilities from the RAs in the system [55]. If the RAs provide their capabilities as PTA-based models, the DA-PA can put together these models to create the proposed environment model [99]. However, to enable the proposed cooperation framework, the DA-PA must keep track of the agents associated with the components in the environment model.

The agent association function, Ag, maps states, events, and constraints to an associated agent. The states can be mapped to one or more agents. If a state is mapped to more than one agent, it is a shared state (see Chapter IV). The presence of these shared states allows for the modular composition of the environment model. The events are mapped to a single agent so that the DA-PA can identify which agent to contact when requesting a discrete transition. Similarly, constraints are mapped to a single agent, representing which agent is responsible for the scheduling constraint in the system.

5.2.b.5 State costs

The nominal cost, $\mathcal{K}_{nm}^{p_i}(x_j)$, for performance metric p_i is the cost for a part to stay at a state with respect to the corresponding metric (p_i). Formally, the nominal cost for performance metric p_i for a single state x_j is:

$$\mathcal{K}_{nm}^{p_i}(x_j) = A_{nm} \cdot val(C^x) + b_{nm} \tag{5.3}$$

where $A_{nm} \in \mathbb{R}^2_{\geq 0}$, $b_{nm} \in \mathbb{R}_{\geq 0}$ are the nominal cost parameters provided by the RAs and $val(C^x) = [val(c_g^x), val(c_l^x)]^T$ are the valuations of the clocks for state x.

Similarly, the soft constraint violation cost, $\mathcal{K}_{sft}^{p_i}(x_j)$, is the cost for the part to stay at a state if there is a constraint violation. The soft constraint violation cost for performance metric p_i for a state is:

$$\mathcal{K}^{p_i}_{sft}(x_j) = \sum_{k=1}^{m} a^k_{sft}\, val(\gamma_k^{x_j}) + b^k_{sft}\delta(val(\gamma_k^{x_j})) \tag{5.4}$$

where $m = |I(x_j)|$ is the number of constraints for the state, γ_k is the slack variable associated with the k^{th} constraint, $a^k_{sft}, b^k_{sft} \in \mathbb{R}_{\geq 0}, \forall 1 \leq k \leq m$, and $\delta(\cdot)$ denotes an indicator operator with 1 if the argument is nonzero and 0 otherwise. Note that if $a^k_{sft} = 0$ and $b^k_{sft} = 0$, then there is no penalty for violating constraint k. Similarly, if the associated valuation of the slack variable is 0, then there is no violation cost added, i.e. if $val(\gamma_k^{x_j}) = 0$, then $\delta(val(\gamma_k^{x_j})) = 0$ and $val(\gamma_k^{x_j}) + b^k_{sft}\delta(val(\gamma_k^{x_j})) = 0$. Thus, the sum (5.4) is a summation of slack costs for the nonzero slack valuations.

For the nominal cost, RAs estimate the A_{nm}, b_{nm} for each state and provide this information to the DA-PA when queried. While determining A_{nm}, b_{nm} is out of the scope of this dissertation, there are several methods that can be used to estimate these values. For example, smart sensors can capture information relevant to A_{nm}, b_{nm} and experiments can be run to determine these parameters (e.g., using an energy sensor to determine the energy cost to perform resource actions). Resource models can also be developed and used to determine appropriate values for A_{nm}, b_{nm}. For the soft constraint violation cost, the agent associated with each soft constraint, ag_{sc}, must provide the a^k_{sft}, b_{sft} to the DA-PA. a^k_{sft}, b_{sft} represent how undesirable the constraint violation would be to the agent associated with the soft constraint, ag_{sc}, in terms of performance metric k. Therefore, to ensure desirable performance of all of the other agents, the DA-PA should (1) only violate these constraints if absolutely necessary and (2) minimize the cost of constraint violations, as described in Section 5.3.b.

5.2.c Decision Making Model

A decision making model is represented as the following tuple: $\mathcal{A}_{DM} = (X, Prp, x_0, \Sigma, E, C, \Gamma, I, Ag, X_m, \mathcal{K})$, where $(X, Prp, x_0, \Sigma, E, C, \Gamma, I, Ag)$ are obtained from the environment model, $X_m \in X$ is a set of marked states, and $\mathcal{K} : X \times val(C) \times val(\Gamma) \to \mathbb{R}_{\geq 0}$ denotes the state costs. The set of marked states, X_m, represents desired states for the DA-PA [22]. A discussion of how the decision making model is put together is presented in Section 5.3.a.

Figure 5.1 is an example of a decision making model for a DA-PA in a manufacturing system composed of 2 robots (R1 and R2), 2 machines (M1 and M2), and 2 buffers (B1 and B2). There are 4 RAs in the system, one for each robot and for each machine. The buffers are used by the robots to pick and place parts in the system and do not have an associated agent in this example. B1 is used by R1 and B2 is used by both R1 and R2. The machines complete a manufacturing process (P1) required as part of the DA-PA's process plan. This initial state state, x_0, and marked states, X_m are shown in Fig. 5.1. The dashed rectangles outline which states and events are associated with a specific agent in the agent association function, e.g. states x_1, x_2, x_3 and events σ_1, σ_2 are associated with the R1 agent.

Table 5.1 displays information about the states and constraints for this example. All of the constraints are bound constraints (see Definition V.4), where t is the time limit for the clock at each state.

Soft constraints represent desired, but flexible, operating conditions for the RAs. The only soft constraint for this system is defined on state x_2. This constraint represents the desired minimum operating time, $t^{sft}_{R1,B2}$, for R1 to take the part from B1 to B2. This constraint can be violated if the value of the slack variable is greater than 0 for that state, $\gamma^x_l > 0$. However, there is a penalty that is associated with violating that constraint, $\mathcal{K}^{nrg}_{sft}(x)$. This penalty represents how undesirable this constraint violation would be to R1. The rest of the constraints in Table 5.1 are hard

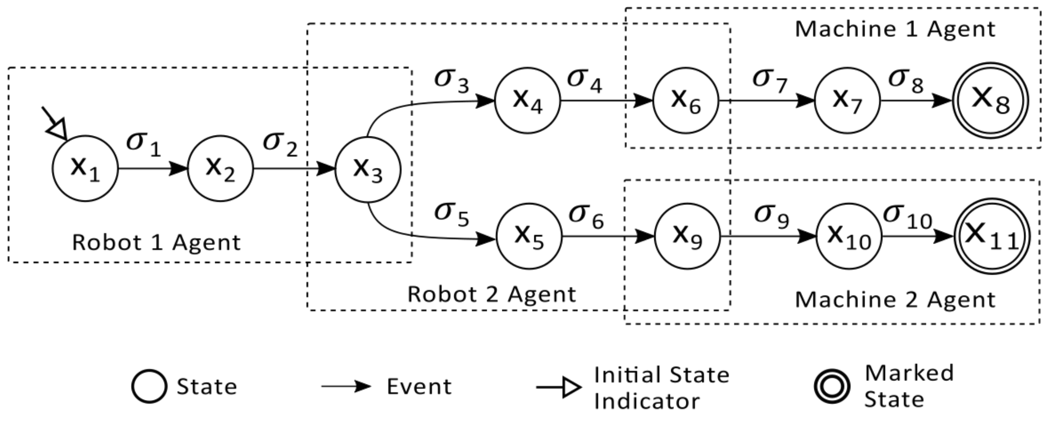


Figure 5.1: A visualization of the decision making model for a system with 6 resources – 2 robots (R1, R2), 2 machines (M1, M2), 2 buffers (B1, B2) – and 4 agents representing R1, R2, M1, and M2. R1 moves the part from B1 (x_1) to B2 (x_3). R2 moves the part from B2 (x_3) to M1 (x_6) or M2 (x_9). M1 and M2 complete manufacturing process 1 (P1), represented by x_8 and x_{11}. The part is at B1 and needs P1 completed (x_8 or x_{11}) per the process plan. See Tables 5.1 and 5.2 for more information about states and events.

constraints. Hard constraints are the boundary conditions for the RA and are defined based on physical and safety limitations. For example, $t^{nm}_{R1,B2}$ is the hard time limit for taking the part between B1 and B2 using the R1 agent, represented by x_2. Note that the desired, operating time for R1 should be greater than the boundary time due to physical or safety limitations, $t^{sft}_{R1,B2} > t^{nm}_{R1,B2}$.

The constraints associated with x_1, x_8, x_{11} are added during model creation, as described in Section 5.3.a. The constraint associated with x_1 is the time limit for the DA-PA to find a new sequence of actions for the DA-PA. t_{opt} is the upper bound for the time to solve the optimization problem and must be provided during initialization or estimated by the DA-PA. Similarly, t_{coord} is an upper bound for coordination time and must be provided or estimated by the DA-PA. In addition, there is a hard constraint for x_8 and x_{11} that represents a time deadline, t_d, to complete the manufacturing process.

An overview of the costs is also provided in Table 5.1. The state costs are a combination of two metrics: processing time and energy expenditure. The processing

Table 5.1: State properties, invariants, and costs for example in Fig. 5.1

State, (x)	Properties, $Prp(x)$	Invariant, $I(x)$	Cost, $\mathcal{K}(x)$
x_1	At B1	$val(c_l^x) > t_{opt} + t_{coord}$	$\alpha_t \mathcal{K}^t(x)$
x_2	Moving to B2	$val(c_l^x) > t_{R1,B2}^{sft} - val(\gamma_l^x)$ $val(c_l^x) > t_{R1,B2}^{nm}$	$\alpha_t \mathcal{K}^t(x) +$ $\alpha_e \Big(\mathcal{K}_{nm}^{nrg}(x) + \mathcal{K}_{sft}^{nrg}(x) \Big)$
$x_3/x_6/$ x_9	At B2/M1/ M2	None	$\alpha_t \mathcal{K}^t(x)$
x_4/x_5	Moving to M1/M2	$val(c_l^x) > t_{R2,M1/M2}^{nm}$	$\alpha_t \mathcal{K}^t(x) + \alpha_e \mathcal{K}_{nm}^{nrg}(x)$
x_7/x_{10}	At M1/M2 P1 Working	$val(c_l^x) > t_{M1/M2,P1}^{nm}$	$\alpha_t \mathcal{K}^t(x) + \alpha_e \mathcal{K}_{nm}^{e}(x)$
x_8/x_{11}	At M1/M2 P1 Finished	$val(c_g^x) < t_d$	$\alpha_t \mathcal{K}^t(x)$

Table 5.2: Event descriptions for example in Fig. 5.1.

Event, (σ)	Event Name	Event, (σ)	Event Name
$\sigma_1, \sigma_3, \sigma_5$	Start moving to B2/M1/M2	$\sigma_2, \sigma_4, \sigma_6$	Finish moving to B1/M1/M2
σ_7, σ_9	Start P1 at M1/M2	σ_8, σ_{10}	Finish P1 at M1/M2

time cost is the amount of time spent in that state: $K^t(x) = val(c_l^x)$. The energy expenditure cost is provided by the RAs and represents the energy used by the RA for the part to stay at the state. For example, the energy cost for moving to B2 is evaluated to the following: $\mathcal{K}_{nm}^{nrg}(x) = c_{nm} \cdot val(c_l^x)$ and $\mathcal{K}_{sft}^{nrg}(x) = c_{sft} \cdot val(\gamma_l^x)$, where c_{nm} and c_{sft} are energy usage coefficients. Note that $c_{sft} > c_{nm}$ to ensure that there is a penalty for violating the soft constraint.

Table 5.2 provides information about the events in the system. Note that the events are associated with a single agent and can be called by the DA-PA to transition

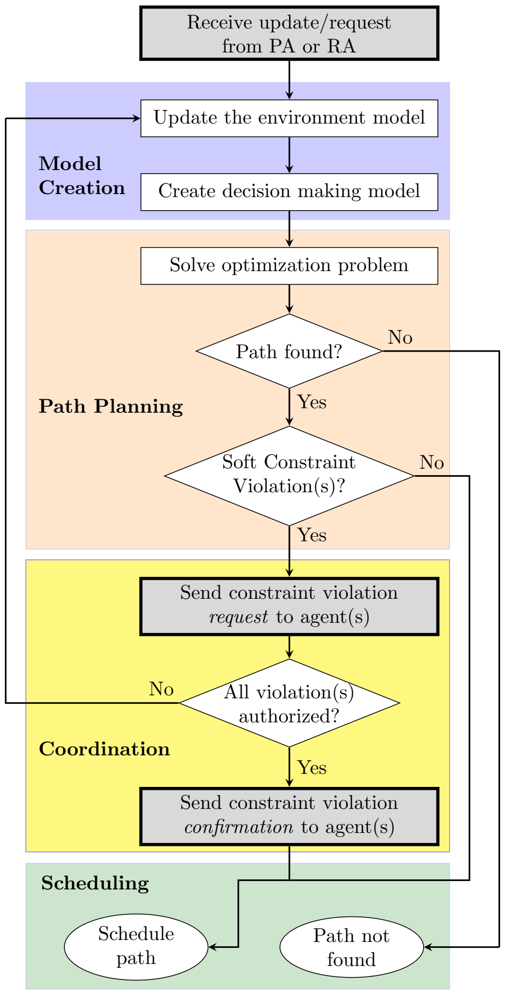

Receive update/request from PA or RA
Update the environment model
Model Creation
Create decision making model
Solve optimization problem
Path found?
No
Yes
Path Planning
Soft Constraint Violation(s)?
No
Yes
Send constraint violation *request* to agent(s)
No
All violation(s) authorized?
Coordination
Yes
Send constraint violation *confirmation* to agent(s)
Scheduling
Schedule path
Path not found

Figure 5.2: A high-level overview of the direct, active cooperation for the product agent. The framework includes 4 steps: model creation, path planning, coordination, and scheduling. Note that the gray boxes indicate instances when the PA communicates with other agents in the system.

between states. For example, the "Start P1 at M1" (σ_7) can be requested from the RA associated with M1 to start the P1 manufacturing process at the machine (x_7).

5.3 Cooperation Framework

The DA-PA uses the goals, environment model, and decision making model described to find a new path, or sequence of resource actions, in the system after receiving an update to its environment model. An update to the DA-PA's environment model occurs either (1) when the DA-PA is provided information about the environment during initialization (2) another agent contacts the DA-PA with new information due to an unexpected disturbance or (3) when RAs reply to a PA query for new information about the environment. Direct, active cooperation is used by the DA-PA to find paths in the system that allow all of the agents to meet their goals, if possible. Once a path is found, the DA-PA will start to schedule these events with the RAs in the system. If a path is not found, the DA-PA will contact an RA to exit the system.

After the DA-PA receives an update to the environment model, the DA-PA goes through the 4 steps in the cooperation framework: (1) model creation, (2) path planning, (3) coordination, and (4) scheduling. A high-level overview of the cooperation framework and the steps are shown in Figure 5.2 and described in detail in this section.

5.3.a Model Creation

The DA-PA goes into the model creation phase after receiving an update to its environment model. In this phase, the DA-PA updates the environment model to capture the current capabilities and constraints of the environment. Then, the DA-PA creates the decision making model used to find the next sequence of resource actions.

5.3.a.1 Update the environment model

The DA-PA's initial step in the model creation phase is to update the environment model using the information provided to the DA-PA by another PA or RA in the system. Other agents can send a message to the DA-PA to update any component of the $\mathcal{A}_{EM}$, i.e. $(X, Prp, x_0, E, Tr, C, \Gamma, I, \mathcal{K}_{nm}, \mathcal{K}_{sft}, Ag)$. These updates represent changes in the manufacturing environment, e.g. new resources, updated scheduling constraints, changes to the associated part's current state, etc.

5.3.a.2 Create the decision making model

The DA-PA combines $\mathcal{A}_{EM}$ and its goals to create the decision making model, $\mathcal{A}_{DM}$, by taking the following steps. To create $\mathcal{A}_{DM}$, the DA-PA computes a set of marked states and updates the constraints and costs of $\mathcal{A}_{EM}$.

The DA-PA computes the set of marked states, X_m by obtaining the next unfinished physical property, pp_d, from its process plan. The DA-PA marks a state (i.e. adds the state to the set of marked states in $\mathcal{A}_{DM}$) if pp_d matches the physical property of a state in the environment model: $pp_d \in Prp(x_i), \forall x_i \in X_m$. Deadlines are incorporated in $\mathcal{A}_{DM}$ by adding a hard constraint, $val(c_g) < Dl(pp_d)$, to the constraints of the marked states, $I(x_i), \forall x_i \in X_m$.

The DA-PA adds a constraint to the initial state in $\mathcal{A}_{DM}$ to allow for path planning and coordination. Thus, the following constraint is added to the initial state: $val(c_l^{x_0}) > t_{opt} + t_{coord}$. This constraint requires the DA-PA to stay in its initial state to complete the steps of the cooperation framework. Note that these times should be significantly less than the time it takes to complete the resource actions.

To account for the priority of other the parts in the system, the DA-PA updates the soft constraint violation costs from $\mathcal{A}_{EM}$ before incorporating these costs into $\mathcal{A}_{DM}$. Let the DA-PA be denoted as a_c and its priority as pr_{a_c}. Since each constraint is mapped to an agent with Ag, the DA-PA obtains a constraint's associated agent and the agent's priority, $a_i, pr_{a_i} = Ag(\mathcal{B}_i)$. Soft constraints with a higher priority DA-PA (i.e. $pr_{a_i} < pr_{a_c}$) are hardened by setting the corresponding $a_\gamma = 0$ (see Eq. 5.1). For soft constraints with a lower priority DA-PA (i.e. $pr_{a_i} > pr_{a_c}$), the penalty for constraint violation is removed by setting $a_{sft}^k = 0$ and $b_{sft}^k = 0$ (see Eq. 5.4). Soft constraints with an equal priority DA-PA (i.e. $pr_{a_i} = pr_{a_c}$) are not changed.

The state cost, $\mathcal{K}(x_j), \forall x_j \in X$ is a function that depends on the valuation of soft constraint violations and the clock valuations. The DA-PA computes the state cost by weighing the state cost for each metric in $\mathcal{A}_{EM}$, $p_1, p_2, ..., p_n$, with its performance weights, $\alpha_{p_1}, \alpha_{p_i}, ..., \alpha_{p_n}$. Formally, the cost for each state is defined as:

$$\mathcal{K}(x_j) = \sum_{i=1}^{n} \alpha_{p_i} \big(\mathcal{K}_{nm}^{p_i}(x_j) + \mathcal{K}_{sft}^{p_i}(x_j)\big), \tag{5.5}$$

where $\mathcal{K}_{nm}^{p_i}(x_j)$ and $\mathcal{K}_{sft}^{p_i}(x_j)$ are given as previously. Note that $\mathcal{K}_{sft}^{p_i}(x_j)$ is a summation of slack costs for nonzero slack valuations. Thus, the state cost is dependant on the soft constraint violation cost only if there is a constraint violation, i.e., nonzero slack variable valuation.

Once the DA-PA completes these steps to create $\mathcal{A}_{DM}$, it will solve an optimization problem to find the path, i.e., sequence of resource actions.

5.3.b Path Planning

A path on the environment model PTA is a sequence of transitions that will take the associated part from its current state to a marked state in the environment model, while satisfying the constraints in the system.

Definition V.7 (Path)**.** A path s on a PTA is an ordered set of discrete and delay transitions in which the post-transition state of each transition is equal to the pre-transition state of the subsequent transition. Thus a path can be notated as

$$s = t_{d,\tau_1}\ t_{s,e_1}\ t_{d,\tau_2}\ t_{s,e_1}\ ...\ t_{d,\tau_n}\ t_{s,e_m} \tag{5.6}$$

and the path's total cost is given by the sum of the costs for all discrete and delay transitions in the path.

As stated in Remark V.6, a path will always start with a delay transition, alternate between discrete and delay transitions, and end with a discrete transition so that $n = m$ in Eq. 5.6.

Let $\xi_i = (x_i, c_i, \gamma_i)$ denote a shorthand for the state, clock values, and slack values respectively at the i^{th} transition of the path s, $s(i)$. Note that $s(i)$ can be either a delay or a discrete transition. To pose the path planning problem as an optimization problem, we additionally define

$$\phi(s, \xi_1) = \{\xi_1 \xrightarrow{s(1)} \xi_2, \xi_2 \xrightarrow{s(2)} \xi_3, \ldots, \xi_{N-1} \xrightarrow{s(N)} \xi_N\}$$

as the *solution sequence*, where $\xi_1 = (x_1, c_1, \gamma_1)$ is the initial state, clock values, and slack values respectively, and $N = 2n$ is the total number or transitions of Eq. 5.6, with $n = m$. We define the projection operator $\Pi_G(\phi(\cdot, \cdot))$ that projects the solution sequence to a subspace $G \subset \mathcal{A}_{DM}$. For example $\Pi_X(\phi(s, \xi_1)) = \{x \in X \mid x \in \phi(s, \xi_1)\}$ projects the solution subspace to the states in $\mathcal{A}_{DM}$. We use $\phi(s)$ instead of $\phi(s, \xi_1)$ for brevity when the arguments are clear from the context.

Thus, the DA-PA solves the following optimization problem to find a cost-optimal path:

$$s^* \in \arg\min_{s_k} \sum_{i=1}^{N} \mathcal{K}(x_i) \tag{5.7a}$$

$$\text{s.t.} \quad x_i \in \Pi_X(\phi(s_k, \bar{\xi})) \tag{5.7b}$$

$$s_k \in \mathcal{L}(\mathcal{A}_{DM}) \tag{5.7c}$$

$$\#(\Pi_\Gamma(\phi(s_k, \bar{\xi}))) \leq \zeta, \tag{5.7d}$$

where $\mathcal{L}(\mathcal{A}_{DM})$ denotes the set of all feasible paths (i.e., the language) of $\mathcal{A}_{DM}$, $\bar{\xi} = (\bar{x}, \bar{c}, 0)$ is the initial state and clock value, $\#(\cdot)$ denotes the number of nonzero elements in the argument, $\zeta \in \mathbb{Z}_{>0}$ is the number of allowable constraint violations for the solution, and s_k is a path as defined in Definition V.7. The solution s^* is a path that minimizes the cost function $\mathcal{K}(x_i)$, where x_i are the states in the solution sequence defined by constraint 5.7b. Constraint 5.7c ensures that the path is in the language of $\mathcal{A}_{DM}$, ensuring that all the constraints in $\mathcal{A}_{DM}$ are satisfied and all the transitions are well-defined. Note that soft constraints and soft constraint violations will satisfy Constraint 5.7c, but the violations are penalized in the cost function for this optimization problem (Eq. 5.7a). Constraint 5.7d defines an upper bound on the number of constraint violations so the solution sequence s^* can violate at most ζ constraints. Note that we recover the total number of constraint violations in the optimal solution using $\#(\Pi_\Gamma(\phi(s^*)))$.

If the number of constraint violations, $\#(\Pi_\Gamma(\phi(s^*)))$, is zero, then the DA-PA can schedule the obtained sequence of actions without the need to negotiate over conflicting events, as described in Section 5.3.d. If a solution contains a nonzero number of constraint violations, the DA-PA has to coordinate with agents to find a solution to the conflicting events prior to scheduling actions. More detail about coordination is discussed in the next section. Finally, if a solution to the above problem cannot be obtained or this optimization problem is not solved within the provided or estimated bounded time (t_{opt}), the DA-PA will request an exit plan, as described in Section 5.3.d.

Similar to Chapter III, the optimization problem in Eq. 5.7 is a local optimization problem for the system. The solution to Eq. 5.7 depends on the individual goals of the product agent and the local environment model built through the exploration methodology described in Chapter IV. The z3 solver [16] is used by the DA-PA to solve this optimization problem, as described in Section 5.4.a.

5.3.c Coordination

The DA-PA has to coordinate its optimal path with other agents if there are any constraint violations found during its path planning phase, $\#(\Pi_\Gamma(\phi(s^*))) > 0$. The DA-PA finds the set of corresponding agents (PA or RA) for each constraint that has a non-zero slack valuation (violated constraint): $Agents = \{Ag(\mathcal{B}_v) \mid \mathcal{B}_v \in \mathcal{S}[val(C), val(\Gamma)], val(\gamma_{v,i}) > 0, val(\gamma_{v,i}) \in Val(\Gamma_v)\}$, where Γ_v are the slack variables associated in this constraint. Note that this mapping is not one-to-one as a single agent in the set *Agents* can be associated with multiple violated constraints.

The DA-PA sends a *coordination request* to each of these agents in *Agents*. For every state x_i^* in the state sequence of the optimal path $\Pi_X(\phi(s^*))$, the DA-PA finds the corresponding delay transition t_{d,τ_j} such that $(x_i^* \xrightarrow{t_{d,\tau_j}} x_i^*) \in \phi(s^*)$. The *coordination request* contains all of the states in the optimal path and the corresponding delay transitions. Thus, the *coordination request* represents new scheduling constraints to be considered by each agent, $ag_v \in Agents$. The contacted agent, ag_v, should be able to understand, reason about, and respond to the request from the DA-PA. Examples of how other agents understand, reason, and respond to this request is provided later in this section.

The DA-PA waits for responses to the *coordination request* for a specified amount of time, t_{coord}. If all of the contacted agents authorize their constraint violations, the DA-PA sends a message confirming the new scheduling constraints that correspond to the delay transitions in its optimal path to the other agents, $ag_v \in Agents$. Once the confirmation message is sent out, the DA-PA will start to schedule the path with the RAs, as described in Section 5.3.d.

However, a constraint violation can be denied, i.e. not authorized, by another agent in the system. Examples of why another agent would deny a constraint violation request are discussed later in the section. If the constraint violation is not authorized, the DA-PA may still identify feasible paths that can fulfill its goals. Thus, if a constraint violation is denied (i.e. non-authorized), the DA-PA will update the environment model accordingly to avoid violating the said constraint. Non-authorized constraints are hardened by setting the hardness coefficient in the constraint to 0 (see Definitions V.4 and V.5). Then, the DA-PA goes through each of the steps in the cooperation framework in Figure 5.2 with the new hardened constraints.

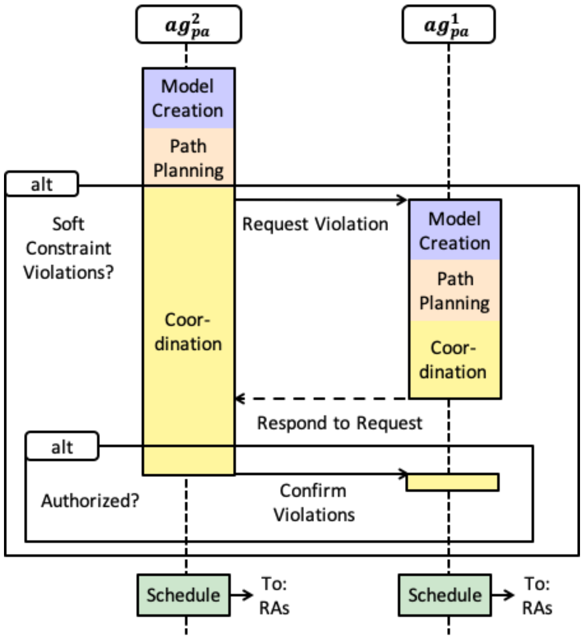


Figure 5.3: A sequence diagram for negotiation between two direct, actively cooperating product agents, ag_{pa}^1 and ag_{pa}^2.

5.3.c.1 Coordination with a product agent

The DA-PA requests constraint violations from other PAs in the system by requesting all of the delay transitions in its optimal path. To make a decision whether to authorize the constraint violation, the contacted PA, $ag_{pa} \in Agents$, must reason about the effect the constraint violation will have on its planned sequence of actions (i.e. path). If the decision making of ag_{pa} is rule-based, then ag_{pa} must have an appropriate rule that allows it to search for alternate paths in the system. Similarly, if ag_{pa} is a model-based PA, the the new scheduling constraints should be encoded in the environment model of the PA. If an alternate path is not found, then the rule-based or the model-based PA rejects the request from the DA-PA.

Since a DA-PA is classified as a model-based PA, the decision making framework proposed in this work can be used to find alternate paths in the system. The sequence diagram in Figure 5.3 shows what happens when one DA-PA, ag_{pa}^2, sends a violation request to another DA-PA, ag_{pa}^1. In this example, ag_{pa}^1 receives a request with new scheduling constraints from ag_{pa}^2 and progresses through the model creation, path planning, and coordination phases in Figure 5.2.

The requested violated constraints are added as hard constraints to the decision making model of ag_{pa}^1 during the model creation phase. Then, during the path planning phase, ag_{pa}^1 solves the optimization problem shown in Eq. 5.7. To prevent ag_{pa}^1 from contacting other agents in the system, ag_{pa}^1 can set ζ to 0 when finding the optimal path in Eq. 5.7. The coordination phase of ag_{pa}^1 consists of a reply to DA-PA1 to authorize or deny constraints based on whether or not a path was found in the path planning phase. Future work will look into allowing ag_{pa}^1 to request constraint violation from other RAs and PAs in the system by allowing positive values of ζ in Constraint (5.7d) of ag_{pa}^1.

5.3.c.2 Coordination with resource agents

The DA-PA may request constraint violations from the RAs in the system. In practice, the RAs often have a desired, optimal time to complete various resource actions based on their own resource models and collected production data. However, resources can often accomplish tasks faster if prompted. While this might be sub-optimal to the individual RA, it might help PAs in the system to accomplish their goals. For example, if a DA-PA requires a faster-than-usual transfer between two locations in the system, a material handling robot can accomplish this task at the cost of higher energy expenditure.

A soft constraint that is associated with an RA represents the RA's desired, optimal time required to finish the associated resource actions. However, when necessary and if beneficial to the DA-PA in the system, the constraint can be violated during a DA-PAs cooperation step. Therefore, once queried, a RA must decide whether to authorize or deny a violation request as the request would result in sub-optimal behaviour for the RA (e.g. much higher energy expenditure). While RA decision-making is not within the scope of this work, the RA must weigh the benefits and disadvantages of violating a constraint with respect to its associated resource. Then, the RA makes decisions whether to allow the DA-PA to violate a constraint and communicates this decision with the DA-PA.

5.3.d Scheduling

If a DA-PA did not find a path during the path planning phase, it will have to exit the system via the exit plan. Therefore, the DA-PA calls the exit plan provided during its initialization to exit the system. If the DA-PA finds a path s^* and all of the constraint violations are authorized by other agents, the DA-PA starts to schedule the transitions with the RAs in the system.

For every state x_i^* in the state sequence of the solution sequence $\Pi_X(\phi(s^*))$, the DA-PA finds the corresponding delay transition t_{d,τ_j} such that $(x_i^* \xrightarrow{t_{d,\tau_j}} x_i^*) \in \phi(s^*)$. Then, the DA-PA sends a scheduling request to the agents associated with the state, $x_i^* \in Ag(x_i^*)$. The request is a scheduling constraint based on the delay time τ_j. By performing the scheduling request for all the delay transitions in s^* the DA-PA requests to schedule time constraints on all the RAs in the solution path.

To transition between states x_i^* and x_{i+1}^*, the DA-PA sends a request to the agent, $Ag(e)$ where $e \in E$ is the corresponding edge in $t_{s,e} \in s^*$, i.e. $(x_i^* \xrightarrow{t_{s,e}} x_{i+1}^*) \in \phi(s^*)$. This discrete transition request is made at a desired time, which corresponds to the end of a current delay transition. The DA-PA continues to request the discrete transitions until it completes all of the events in the path s^*. To improve the robustness of the DA-PA, the optimization Eq. (5.7) is run after each discrete transition is taken by the agent. If the optimal path s^* changes between subsequent solutions of Eq. (5.7), the DA-PA reschedules the time constraint schedules with the RAs corresponding accordingly.

Since the DA-PA plans for a single manufacturing process at a time, the path s^*

ends at a marked state $x_i^* \in X_m$ that signifies the completion of a manufacturing process. After the path is finished, the DA-PA finds any unfinished manufacturing processes as the associated physical part goes through the system and updates $\mathcal{A}_{DM}$ to solve Eq. (5.7) for the next process.

5.4 Case Study

This section presents how direct, active cooperation can be used to improve the performance of manufacturing systems using simulations of the manufacturing environment. The case study showcases how the cooperation framework is used for the system introduced in Section 5.2.c.

5.4.a Simulation setup

The Repast Simphony agent-based simulation platform [91] was used to test and analyze the behaviour of a DA-PA in a manufacturing environment. This agent-based simulation software is discrete-time based, as the simulation is updated every time step (also known as every tick).

The developed manufacturing simulation contains the high-level discrete-event dynamics for material handling robots and machines used for manufacturing. The proposed cooperation framework was added to the model-based product agents and resource agents obtained from the simulation provided in [47]. A custom PTA modelling class was developed to encode the environment model and decision making model. The communication framework native to Repast Simphony was used to encode messages and enable the cooperation between agents.

A Cost Optimal Reachability Analysis (CORA) solver was developed to solve the optimization problem in Eq. 5.7 [6]. CORA has been previously used to find cost-optimal paths for PTA models. Two software tools that have been used to solve CORA for PTAs are UPPAAL CORA [12,13] and satisfiability modulo theories

(SMT) solvers [15, 36]. UPPAAL CORA is developed by using a branch-and-bound algorithm detailed in [13] and implemented in the UPPAAL CORA software [12]. However, there has been no development of tools for closed-loop extensions of CORA and existing CORA algorithms do not admit constraint violations. Thus, to solve the optimization problem with soft constraints, a custom implementation [16] of the z3 SMT solver was used to encode soft constraints and find paths in the DA-PA's decision making model [6]. z3 leverages an optimization modulo theories (OMT) solver. OMT is a branch of satisfiability modulo theories (SMT) with cost functions. OMT solvers find feasible Boolean and algebraic variables that satisfy all of the given constraints, that are optimal with respect to an algebraic cost function, and that readily admit constraint softening [16]. For this implementation, the dynamics and constraints of the decision making model were transformed into first-order logic expressions [6]. The solver output was the optimal path for the DA-PA and a set of constraint violations used in the cooperation framework. JavaScript Object Notation (JSON) was as an interface between the Repast Simphony simulation and the z3 SMT solver.

5.4.b Case study: small manufacturing system

The case study considers the manufacturing system described in Section 5.2.c. The system contains two robots (R1 and R2), two machines (M1 and M2), and two buffers (B1 and B2). New parts entering the system start at buffer 1 (B1) and, once the part enters the system, a new DA-PA is initialized. During initialization, the DA-PA is provided the process plan and deadlines, the exit plan, the performance weights, and the priority for the physical properties in the process plan.

In this example, the process plan contains one physical property: complete process 1 (P1). The exit plan is an agent that can take the part associated with the DA-PA out of the system. For this example, the DA-PAs attempts to minimize both the time and energy expenditure in the system with $\alpha_t = 1$ and $\alpha_e = 0.5$, respectively. To

Table 5.3: Time limits for resource actions in the simulation

Resource	**Program**	**Time (ticks)**
Robot 1	Move to Buffer 2	15
Robot 1	Move to Buffer 2 (Fast)	5
Robot 2	Move to Machine 1	17
Robot 2	Move to Machine 2	17
Machine 1	Process 1	100
Machine 2	Process 1	150

test the DA-PA cooperation framework, various priority values and deadlines were provided to the DA-PAs during initialization, as detailed in the following examples.

The time limits for the resource actions in the simulation are shown in Table 5.3. The time limits in Table 5.3 are used to build the constraints in the decision making model shown in Table 5.1. The time for optimization and coordination is 1 tick, i.e. $t_{opt} = 1$ and $t_{coord} = 1$. Note that for robot 1, the energy expenditure is much greater if the soft constraint is violated, i.e. if robot 1 has to move the part to B2 faster than 15 ticks, Thus, $c_{nm} = 10$ kWh and $c_{sft} = 100$ kwH are the costs for staying at state x_2, $\mathcal{K}_{nm}^{nrg}(x_2) = c_{nm} \cdot val(c_l^{x_2})$ and $\mathcal{K}_{sft}^{nrg}(x_2) = c_{sft} \cdot val(\gamma_l^{x_2})$. To conform with the simulation environment, we require the part to stay in every state for at least one tick. Therefore, an additional constraint, $val(c_l^x) >= 1$, is added for each state in the decision making model, $x \in X$.

5.4.b.1 Example 1 - Non-cooperative PAs

This example illustrates the scenario when two DA-PAs do not need to cooperate with each other to complete their individual goals and satisfy the system objectives. In this example, a part comes into the system with no deadline provided for P1. A DA-PA, ag_{pa}^1, is created with an importance value of 2. The DA-builds the decision making model shown in Figure 5.1 and calculates its optimal path using Eq. 5.7. The solver described in the previous section is used to find a cost-optimal path in the system. Then, after solving the optimization problem, ag_{pa}^1 schedules the path to complete P1 at M1.

Unexpectedly, a second part comes into the system 10 ticks after the first part. The second DA-PA, ag_{pa}^2, does not have deadlines and has an importance value of 2. The optimal path for ag_{pa}^2 does not have any constraint violations. Thus, ag_{pa}^2 schedules resource actions to take the associated part to the slower machine, M2, to complete its process plan. The flow times for ag_{pa}^1 and ag_{pa}^2 to complete their process plans are 141 and 197 ticks, respectively.

5.4.b.2 Example 2 - PA-PA cooperation with order prioritization

For this example, the manufacturer prioritizes the completion of the part associated with ag_{pa}^2 over any other parts in the system. Both ag_{pa}^1 and ag_{pa}^2 are set-up in the same way as Example 1. However, ag_{pa}^2 is given an importance value of 1, making it higher priority than ag_{pa}^1.

As in example 1, ag_{pa}^1 initially chooses to go to M1 to complete the process plan. As the part associated with ag_{pa}^1 is moving to B2 (state x_2 in Figure 5.1), the more important part enters the system and ag_{pa}^2 is initialized. By following the proposed cooperation framework, ag_{pa}^2 requests ag_{pa}^1 to reschedule its plan. ag_{pa}^1 authorizes the request and finds an optimal path that to take its associated part to M2 to complete P1. In this scenario, the flow times for the ag_{pa}^1 and ag_{pa}^2 are 221 and 145 ticks, respectively. Compared to the first example, the higher priority DA-PA is able to complete its process plan significantly faster. Note that the total flow time of both parts in the system is longer. However, since neither ag_{pa}^1 or ag_{pa}^2 had any deadlines in the process plan, all of the goals of both agents were accomplished.

5.4.b.3 Example 3 - DA-PA, DA-PA cooperation to meet a deadline

Both ag_{pa}^1 and ag_{pa}^2 are set-up in the same way as Example 1. Unlike example 2, the priority of both parts is the same for this example. However, ag_{pa}^2 is given a deadline of 180 ticks to complete P1.

As in examples 1 and 2, ag_{pa}^1 initially chooses to go to M1 to complete the process plan. However, when the part associated with ag_{pa}^2 comes into the system, ag_{pa}^2 requests the utilization of M1 from ag_{pa}^1. Similar to example 2, ag_{pa}^1 is able to find a new path that satisfies all of its goals that takes the associated path to M2. In this scenario, the flow times for ag_{pa}^1 and ag_{pa}^2 are 221 and 145 ticks, respectively.

5.4.b.4 Example 4 - DA-PA, RA cooperation to meet a deadline

Only one DA-PA is considered for this example. In this example, a part comes into the system and DA-PA is initialized with a deadline of 135 ticks. To accomplish this task, ag_{pa}^1 cooperates with the R1 RA and requests to move to B2 faster by violating R1's soft constraint. Since this will ensure that ag_{pa}^1 meets its deadline, the RA confirms this violation at the expense of a higher energy expenditure. In this scenario, the flow time for ag_{pa}^1 is 135 ticks.

5.5 Conclusions

A model-based decision making framework to enable direct, actively cooperative product agents is introduced. For this framework, a new environment model that leverages the priced timed automaton modeling formalism is proposed. This model explicitly represents the scheduling constraints in the system and separates the negotiable constraints (soft constraints) from the non-negotiable constraints (hard constraints). To identify which constraints conflict the product agent's goals, an optimization problem on the model is developed. A strategy for product agent coordination and negotiation with other agents in the system is also presented. This strategy resolves the conflicts in the system, allowing the agents to effectively control and coordinate the components on the factory floor. This framework is tested in a simulated manufacturing environment, showing how direct, active cooperation can improve the flexibility and performance of the manufacturing system.

CHAPTER VI

Implementations

Manufacturing system testbeds provide a place where new ideas can be developed and tested without the need to disrupt the production or distribution capabilities of an industrial plant [46]. A multi-agent controller with the model-based product agent was developed and tested in several manufacturing system testbeds. This chapter presents a description of multi-agent control implementations in three manufacturing testbeds: the Fischertechnik testbed at the University of Michigan [131], the myJoghurt Demonstrator at the Technical University of Munich [2, 121], and the System-level Manufacturing and Automation Research Testbed at the University of Michigan [46]. These implementations can be used as a blueprint for future implementations of the model-based product agent framework.

6.1 Fischertechnik

A small table-top testbed made by Fischertechnik was used to initially test the multi-agent control strategy [131]. This portable testbed is a small scale representation of a manufacturing facility, allowing us to test and study the multi-agent control strategy in a physical setting. The Fischertechnik testbed consists of conveyors, infrared (IR) sensors, limit switches, and machining stations. These components emulate functions of real-world manufacturing systems. The layout of the testbed is

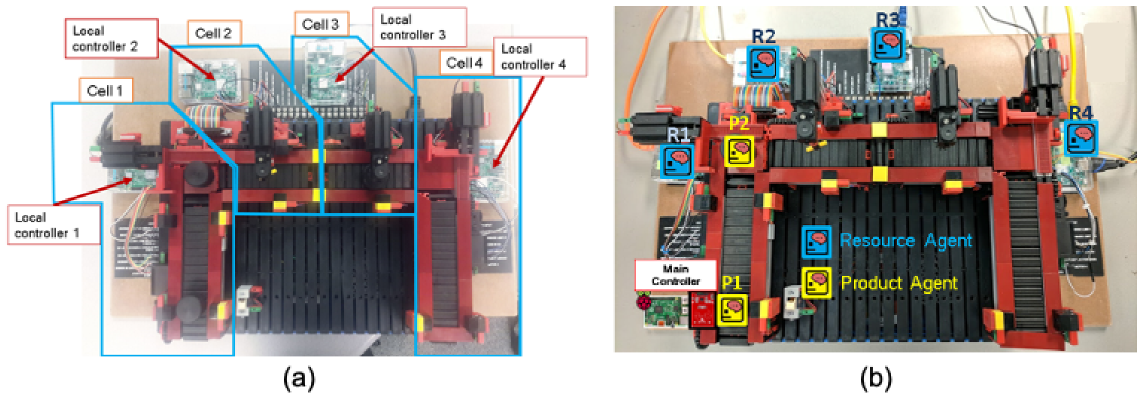


Figure 6.1: An overview of the Fischertechnik testbed. (a) shows the arrangement of the four cells and their logic controllers and (b) displays the agents on the Fischertechnik testbed and the location of the main controller.

shown in Figure 6.1.

The physical testbed is sectioned into four cells, as shown in Figure 6.1(a). Each cell contains a conveyor line and IR sensors to capture the location of the parts in the system. Cells 2 and 4 have machining stations with a mechanical spinner above the conveyor. The cell controllers can actuate this mechanical spinner, representing a manufacturing operation in the system. Each cell is controlled by a programmable logic controller (PLC) emulated by a Raspberry Pi (RPi) microcontroller [25]. Control code is uploaded to these micro-controllers, which enact the low-level control (e.g. motors) for the machines. The controllers are connected over an Internet Protocol (IP) network with static IP addresses.

6.1.a Agent control and communication architecture

The Java Agent Development Framework (JADE) [3], an agent development environment, is used to create the agents and establish the agent communication. The developed multi-agent architecture consists of four resource agents (RAs) and a number of product agents (PAs). The four RAs are high-level controllers for the cells and can send actuation commands to move the conveyor or start the manufactur-

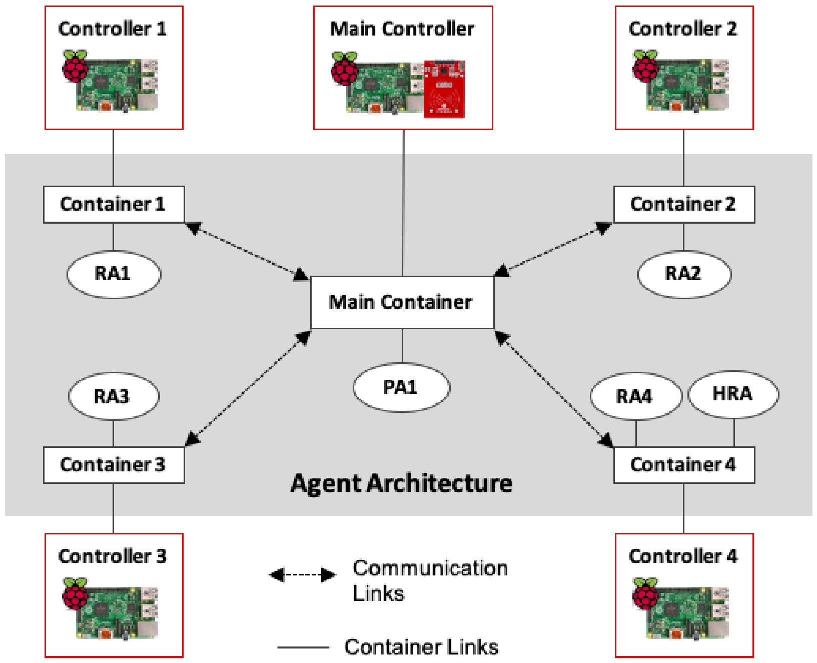


Figure 6.2: The setup of the Java Agent Development Framework (JADE) on the Fischertechnik testbed.

ing operation. Model-based PAs are created when new parts enter the system. The architecture is distributed over the five Raspberry Pis (RPis) in the system.

Four of the RPi's stores a cell RA and the low-level control logic for that cell. The agents and the low-level logic are hosted in JADE containers, which are native to the JADE environment. The other RPi is the main container in the system, facilitating the communication in the system and initializing new PAs in the system. PAs move between the RPis as the associated part enters the system and is transferred between the cells on the testbed. The agent platform is shown in Figure 6.2. The communication between agents in the system is accomplished using the FIPA (Foundation for Intelligent Physical Agents) ACL Message protocol in the JADE environment. The ACL Messages contain custom, serialized Java objects that are decoded by each of the agents in the system.

The agent architecture is shown in Figure 6.3. The agent layer establishes the high-level control of the testbed. At this control level, RAs and PAs communicate

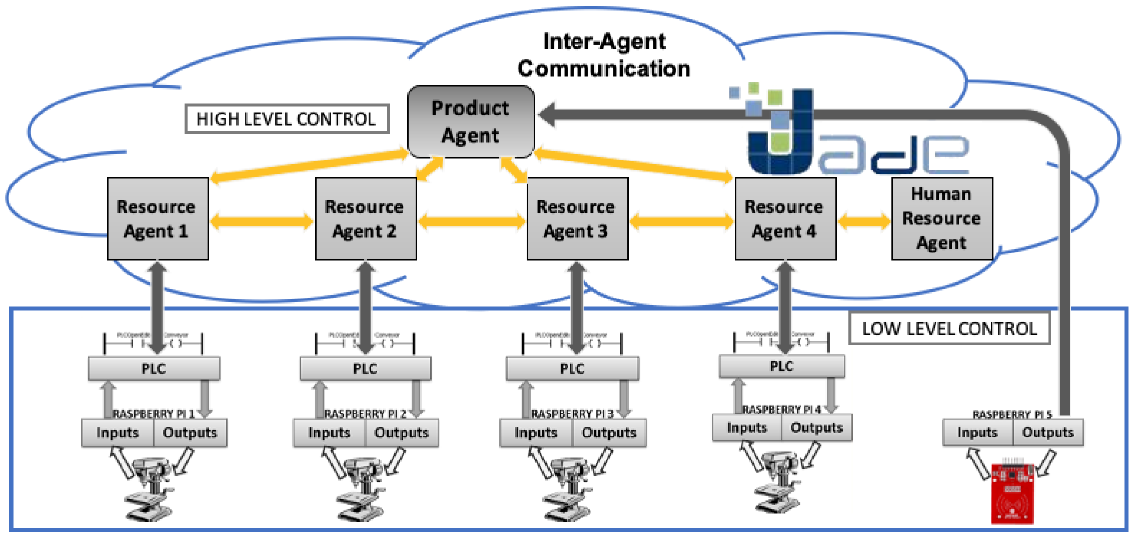


Figure 6.3: The multi-agent control architecture for the Fischertechnik testbed.

with each other, shown by the yellow arrows, and make decisions. These decisions are communicated to the PLC in each RPi, as shown by the dark grey arrows. Note that there is one dark grey arrow pointing to the PA in the system, which signifies that a PA is initialized using information obtained by that RPi.

The low-level control is established by PLC signals sent through input/output pins from the RPis to the cells on the testbed. In the low-level control, ladder logic in the corresponding RPi has direct access to the physical machines through GPIO (general-purpose input/output) pins.

In addition to the cell RAs, a human resource agent (HRA) is included. The HRA is a representation of an operator on the shop floor and is discussed in detail in [131]. The HRA interacts with the other agents to add flexibility to the system.

There are many paths of communication between the agents in the developed multi-agent controller. PA to RA communication involves requests of services from the PA to the RA. RA to RA communication is needed for the cooperation of machines to send and receive parts from and to their respective cells. PA to PA communication is used to implement priority of products - higher priority products can tell lower priority ones to recirculate. The HRA interacts with other agents to perform material handling or maintenance requests.

6.1.b Fischertechnik case studies and insights

The testbed is used to validate the behavior of the model-based product agent presented in Chapter III and the exploration methodology described in Chapter IV. The model-based product agent is provided a process plan that requires the associated physical part to be machined at either cell 1 or cell 2 for a specified amount of time. Following this machining operation, the associated part should move to the end of the conveyor line in cell 4.

A part is placed at the start of the line in cell 1 (e.g. P1 in Figure 6.3b). Once the cell 1 RA informs the associated PA that the part is in the system, the PA uses the exploration technique described in Chapter IV to build the model of the manufacturing environment. Using this model, the PA is able to find a sequence of actions to take the part from its current location (cell 1) to the machine specified in its process plan. The PA requests these operations and waits for feedback from the RAs in the system. Once an RA informs the PA that the requested machining operation is finished, the PA explores the system to find how to move to the end of the conveyor line in cell 4. The PA then requests this sequence of actions from the RAs in the system. Note that if an RA is ever shut down during this process, the PAs and RAs call the HRA [131] to assist with "fixing" the shut down in the system, showcasing the flexibility of the multi-agent architecture.

This simple manufacturing system shows that the model-based product agent is able to accomplish a process plan and request the appropriate actions from the RAs in the system. However, this type of testbed does not accurately represent the scale and complexity of manufacturing systems and existing manufacturing system control architectures. Therefore, to better simulate an industrial environment, the model-based product agent was testbed in two other testbeds: the myJoghurt demonstrator and the System-level Manufacturing and Automation Research Testbed.

6.2 myJoghurt Demonstrator

The concept of resource agent task negotiation described in Chapter IV was tested in the myJoghurt CPPS (Cyber Physical Production System) demonstrator at the Institute of Automation and Information Systems, Technical University of Munich [2, 121]. myJoghurt is a production facility used to test new manufacturing system control technologies for the production of personalized bottles of yogurt based on various customer orders [37]. The demonstrator consists of a bottle storage facility, a logistic system, and two filling stations with different colored pellets (Fig 6.4). Once a customer places an order for a personalized yogurt production, a bottle is taken from storage and placed into the logistic system. Then it is moved, by sets of conveyors, to desired locations around the demonstrator. The desired locations for each bottle depend on the customized process plan created using the individual customer order. Thus, individual bottles have different requirements based on their process plans, current locations, deadlines, etc. In addition, the capabilities of the logistic system are always changing due to varying routes of the bottles, conveyor maintenance schedule, or alterations to the sensors and actuators in the system creating a dynamic manufacturing environment.

6.2.a Agent control and communication architecture

The implementation of the multi-agent control architecture for the myJoghurt demonstrator is shown in Figure 6.4. The logistic component of the demonstrator contains 22 conveyor lines controlled via stepper motors. Switches route bottles at conveyor junctions. Light sensors indicate the presence of a bottle at each switch and

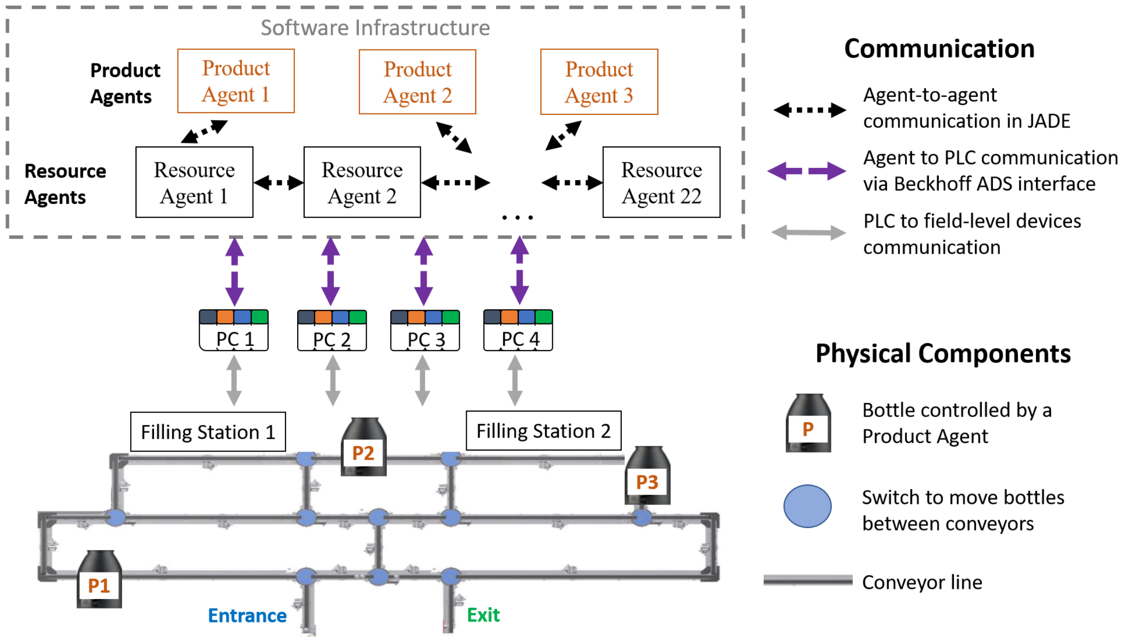


Figure 6.4: The setup and infrastructure for the myJoghurt system at the Institute of Automation and Information Systems, Technical University of Munich.

at the beginning and end of each conveyor line. The conveyor motors, switches, and sensors are connected to four embedded PCs (a Beckhoff CX2040 and three Beckhoff CX9020) via EtherCAT. The embedded PCs run Beckhoff TwinCAT3 to control the conveyors, set the direction of the switches, and obtain the presence of bottles at the light switches, among other functions.

Similarly to the Fischertechnik testbed, the multi-agent control architecture is implemented using the JAVA Agent DEvelopment Framework [3] on a Dell Intel Core i7 PC. Eighteen RAs are connected to the four embedded PCs using the Beckhoff Automation Device Specifications (ADS) interface [1] over EtherCAT. The FIPA ACL Message protocol with custom, serialized Java objects is used for communication between agents. The RAs are initialized with individual capabilities models encoded using the JUNG library [94].

The PAs control the behavior of the bottles in the system through communication requests with the RAs. Each PA is provided a process plan with locations around

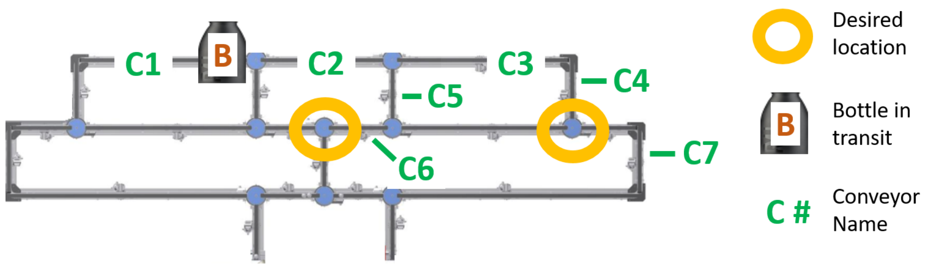


Figure 6.5: Set-up for an example scenario demonstrating the PA exploration

the system and/or the addition of pellets in the first or second filling station. Using the process plan, the PAs attempt to explore the system, make plans, and request actions from the RAs.

The model-based PA described in Chapter III explores the system using the negotiation strategy described in Chapter IV. Based on the returned bids, the PA can decide whether it needs to further explore the system. If the PA finds multiple paths to a desired state, it will plan to take the shortest time path available. After the PA plans a sequence of actions to take, it executes these action through queries to the respective RAs. The time-out times for the exploration and planning were 150 ms and 50 ms.

6.2.b Case study descriptions

Two case studies were performed in the myJoghurt demonstrator. The first case study looked at the performance of a single PA in the system, while the second case study focused on the behavior of the system with multiple PAs.

Case study 1: One product agent in the system

Figures 6.5 and 6.6 show the setup and the communication for one example scenario in the system. In this scenario, the RAs are initially provided with individual capabilities. For example, RA2 is provided a capabilities model that contains the following states: “C2 Start”, “C2 End”, “C2 to C3”, and “C2 to C5”. Each of these states is linked to a presence sensor in the system to signify when a bottle arrives

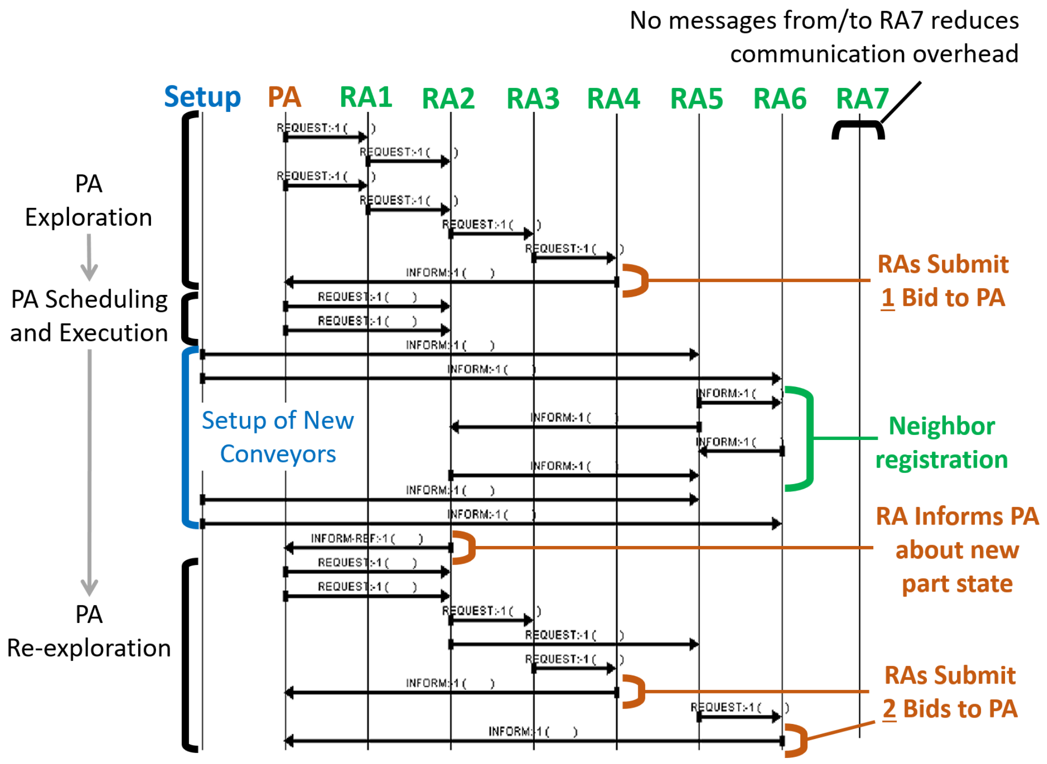


Figure 6.6: An annotated screenshot of the communication when a new RA enters the system. The annotations highlight how a PA re-explores the system to learn about this unexpected change.

at the state. In addition, the events of the capabilities model represent transitions between two states. For example, an event called "Run C2" is a transition between the "C2 Start" and "C2 End" states. These events are linked to PLC tags in the system. Hence, "Run C2' is linked to a tag that starts conveyor 2 and takes a bottle to the end of the line. Finally, the neighboring RAs and the corresponding states are provided to each RA. For example, "C2 to C3' is a neighboring state between RA2 and RA3.

An example of the behavior of a PA in the system is shown in Figure 6.6. In this scenario, the bottle is located on conveyor 1 (C1). Based on its process plan, the PA must guide the bottle to one of its two desired locations. First, the PA sends out bid requests with a small allowable time limit. Since the RAs cannot form a team to take the bottle from its current state to a state with a desired location in the allowable time, the PA increases the time limit for its second exploration request. With the new time limit, the team of RA2-RA3-RA4 sends the PA a bid. The PA uses Djikstra's algorithm to plan and schedule an event string to take the bottle from its current state to a desired state in the shortest amount of time [47]. Finally, once the bottle arrives at new states, the RAs inform the PA by monitoring the individual presence sensors.

The dynamic response of the system is also shown in the example. Specifically, C5 and C6 are placed into the system as the bottle is traveling on C1. The RAs associated with those resources enter the network of RAs through communication with their neighbors. Thus, when the PA re-explores the system, the RAs submit two bids that can take the bottle to a desired state. This example shows the flexibility of using the dynamic network of resource agents to quickly adapt to changes in the manufacturing system.

Case study 2: Multiple product agents in the system

To test the performance of multiple PAs in the system, 20 bottles were sent to the demonstrator. New bottles entered the system every 190 seconds, replicating a realistic production plant. Simulating a random customer order, seven PAs were provided a process plan that required pellets from Filling Stations 1 and 2. The process plans of the other PAs required only Filling Station 1 pellets. The goal of each bottle was to reach the exit of the demonstrator with the correct pellet distribution.

By utilizing the proposed negotiation procedure, the PAs representing the bottles were able to explore, plan, and request actions from the RAs representing the conveyors. Overall, 18 out of 20 bottles successfully navigated the system. 2 of the bottles became stuck due to a mechanical defect in the system. If stuck and unable to finish their process plan, the PAs requested a human operator to remove the associated physical part from the system.

Experiments showed that the RAs required 38 ± 24 (average, standard deviation) milliseconds to perform task negotiation and send back a bid to the PA using JADE's communication infrastructure. The size of submitted bids ranged from 1 to 17 events. The maximum amount of time for a bid to come back to the PA was 125 ms. The exploration time variability was mostly due to message transfer speed in the JADE environment. Once the bids were obtained, the PAs scheduled actions with the RAs. The average time to schedule an action with an RA was 3.2 ms. Finally, the PAs had to request actions from the RAs. On average, it took 19 ± 17 ms, with a maximum of 124 ms, between the time that the PA requested an action to the time when the RAs finished setting the appropriate tag in a Beckhoff PC using the ADS interface. The variability in execution time depended on both the speed of message transfer in the JADE environment and the speed of using the ADS interface.

6.2.c Insights from myJoghurt case studies

The feasibility of using this exploration technique in a real manufacturing system was shown in this implementation. Exploration was integrated with a controller that uses a common standard for industrial system control (IEC 61131-3) [118]. RAs and PAs made timely decisions to control bottles in the system. PA exploration took at most 125 ms with multiple agent interactions. This time may scale up as the number of agents and the amount of communication in the system increases.

The benefits described in Section 4.3.b are showcased in this implementation. The bottle can be placed in any location with a sensor and will start to explore the system, plan, and execute actions to reach a desired state. The discussed dynamic network of RAs adapts instantly to changes in the manufacturing system. These changes include resource addition, removal or maintenance, or changes in resources capabilities. Note that this exploration technique does not require a single global model or communication with all of the system RAs, as required by alternate exploration techniques. For example, in Figure 6.6, RA7 does not have to divulge information about its capabilities to the PA, preventing excess communication. Additionally, each RA updates its own capabilities model, reducing the need to store and update a global system model during PA exploration.

Some of the PA-RA communication parameters have to be tuned for the specific application of an agent-based controller. These parameters include the number of physical properties and the size of the time limit in the PA's bid request, the amount of time the PA needs to wait before it begins to plan and execute actions, and how often the PA must re-explore the system. These parameters will dictate PA robustness and responsiveness and are adapted to the manufacturing system. The selection of these parameters will be based on the speed of the agent communication infrastructure, the fidelity of each resource capabilities model, and the expected number and types of disturbances.

6.3 System-level Manufacturing and Automation Research Testbed

The University of Michigan has developed the System-level Manufacturing and Automation Research Testbed (SMART) [46]. An overview of the testbed is shown in Figure 6.7. First built within the University of Michigan's Engineering Research Center for Reconfigurable Manufacturing Systems in the early 2000s [85], the testbed has been used for a variety of manufacturing research projects including: the development and validation of a framework for logic control of a manufacturing system [28], testing a novel anomaly detection method [4], and implementing a Factory Health Monitoring system [109], among others [35, 70, 73]. Recently, in partnership with Rockwell Automation, the testbed has been upgraded and equipped with the latest control system technologies.

SMART has four computer numerical control (CNC) machine tools, two conveyors,

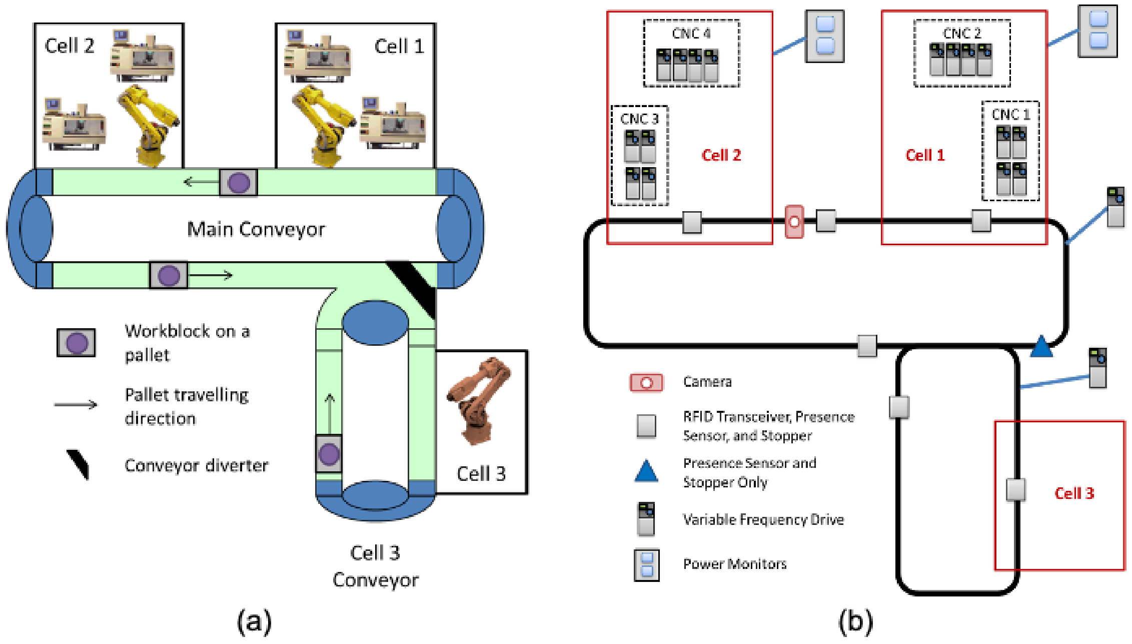
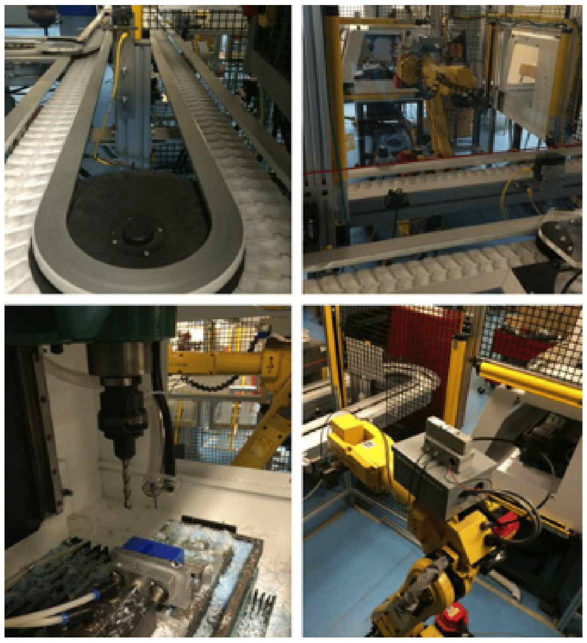
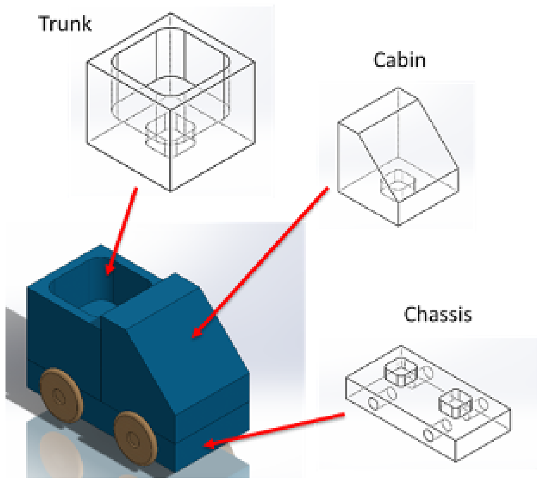


Figure 6.7: An overview of the System-level Manufacturing and Automation Research Testbed. (a) shows the setup for the testbed and (b) provides an overview of some of the sensors integrated into the system.

one gantry, and two industrial robots with an integrated industrial control system (provided by Rockwell Automation) connected through Ethernet/IP with industrial network switches. There are RFID sensors on the testbed, which identify parts in the system and provide their location to the central controller. Pneumatic stops are used to halt the parts at certain pick-up/drop-off locations. There are inspection cameras on the conveyor system as well as some of the CNCs. An industrial human-machine interface (HMI) provides necessary interfaces to control various components in the system. Additionally, each CNC has a dedicated HMI for operators to interface with each machine. Some pictures of the machines, robots, and conveyors in SMART are shown in Figure 6.8(a).

Currently, each CNC contains programs that can perform a number of machining operations to create three types of parts that can be assembled into a toy car shown in Figure 6.8(b). By changing the parameters in the CNC programs and utilizing


(a)



(b)

Figure 6.8: (a) Various views of the components (clockwise from the top left): the main conveyor line, Cell 1, the CNC and the robot from Cell 1, and the robot from Cell 2. (b) An example of a product manufactured using SMART.

other CNC tools, these parts can be altered to create parts with various desired dimensions. The conveyor lines and robots are able to handle a variety of parts on the pallets and with the interchangeable grippers. The parts can be rerouted and the speed of the robot handling, conveyor lines, and CNC processes can be altered to change the system throughput.

A computer connected to the main programmable logic controller (PLC) in the system is used for programming the control logic (with Rockwell's Studio 5000 software) and implementing Industrial Internet of Things (IIoT) applications that send/receive data from cloud-based storage and computation resources. There are various power monitoring sensors on the CNCs that are used to develop data analytics and predictive maintenance solutions.

6.3.a Agent control and communication architecture

Similar to the Fischertechnik testbed and the myJoghurt demonstrator, a multi-agent controller was implemented using the JADE framework for this testbed. Agents used the FIPA ACL Message protocol with custom, serialized Java objects for communication. Three RA agents were created for the system: a conveyor agent and two cell agents. Multiple customized parts were associated by the PA that guided the parts through the testbed. All of the PAs and RAs were developed using the methodology described in the myJoghurt case study.

The main goal of this implementation was to test the connection between the multi-agent controller and the Rockwell PLC. An OPC (Open Platform Communications) client for Rockwell's RSLinx software packaged was configured to make the PLC tags available for external client communicating with the PLC. PLC tags (i.e. variables that point to memory locations) were manually identified for each agent in the system. A custom interface using the JeasyOPC java libraries [24] was used to establish the connection and pass relevant data between the Rockwell PLC and the multi-agent controller in JADE. The conveyor agent was able to read/write the speed of the conveyor via the variable frequency drives, read/write data to the pneumatic stops, and read the data from the RFID (Radio-frequency identification) transceivers. The RFID transceivers were able to identify specific parts using RFID tags [79]. The two cell agents were able to read/write data to the robots and CNC machines.

6.3.b SMART case study and insights

The agent decision making and communication was tested in SMART. We created an RFID tag that contained the processes for one of the three parts shown in Figure 6.8(b) and attached this RFID tag to a wax workblock. Then, we placed the workblock onto one of the pallets on the conveyor line. The multi-agent architecture automatically created a PA and provided the process plan from the RFID tag. When the pallet containing the part arrived at one of the RFID transceivers around the system, the corresponding RA updated the PA with the latest state of the associated physical part. Similar to previous case studies, the PA was able to take the part to the appropriate resource and request the machining operation required by the process plan. Note that the presence of RFID tags allowed the PA to be tracked through the system. Future work includes writing information to the RFID tags to allow the PA to store information on the physical part.

The time latency of communication between the agents and the OPC layer was tested. The average time that it took an RA to send one request to the Rockwell PLC and obtain the appropriate data was 187 ms over 30 trials. The average time that it took an RA to write the data to the PLC was 2700 ms over 12 trials. Note that writing values took a significantly longer time. While the PAs and RAs were able to communicate efficiently and move the parts around the system, there were noticeable occurrences when the part stayed at a single position longer than necessary. This idling occurred because RAs were in the process of reading and writing the data to the testbed. Therefore, both the read and write times should be incorporated in the models and the control design of the PA to improve its decision making.

6.4 Lessons Learned and Insights

There were numerous lessons learned and insights gained during the implementations of the multi-agent controllers for the three testbeds. This section overviews three of the most important insights and lessons learned to effectively utilize a multi-agent control strategy for manufacturing systems. Specifically, agents used for multi-agent control of manufacturing systems should: (1) have an accurate representation of the states, events, and constraints in the system; (2) efficiently capture system disturbances; and (3) account for communication delays between the multi-agent architecture and the sensors and actuators on the shop floor. The rest of this section expands on each of these topics in more detail.

The model-based PAs obtain information about the system through the exploration methodology described in Chapter IV. This exploration methodology is based on the fusion of the capabilities models from various RAs. Therefore, for the implementations, each RA needed to have an accurate representations of the capabilities of its associated resource (see Chapter IV) and these representations had to be consistent for all of the RAs in the system. For the three implementations, we estimated the capabilities of each resource, i.e., the part states, resource actions, average time for each action, etc. If this estimation was inaccurate (e.g., the estimated time is significantly lower than the actual time for resource actions), then the decisions made by the model-based PA would not match the system capabilities and, thus, the PA would have to exit the system. Therefore, the RA capability models were created after running a number of system identification tests for each resource in this system. While this approach worked for the manufacturing testbed implementations described in this section, future work will include the development of an automated approach to estimate resource agent capabilities during initialization.

Additionally, the agents in the system have to accurately capture system disturbances in the environment model to ensure the effectiveness of the multi-agent control strategy. The agents in the implementations described in this chapter captured simple disturbances such as resource failures, new resources or new resource capabilities, and simple changes to customer orders. These disturbances were captured through various smart sensors on the shop floor (e.g., capturing when a machine is off or when a new conveyor line is added) and human-agent interfaces (e.g., an interface to change the customer specifications in the process plan). Various improvements, such as algorithms to capture machine degradation or the integration of new smart sensors in the system, can further improve the performance of the multi-agent control strategy.

When making decisions, agents need to take into account the communication delays between the multi-agent architecture, the logic controller, and the sensors and actuators on the plant floor. For example, the model-based PA needs to incorporate this information when making exploration, planning, and execution decisions. These communication times need to be dynamically updated due to the varying communication overhead, e.g., due to new resources added to the system, new agents in the system, or changes to the low-level control architecture. Similar to the resource capabilities, these communication delays were obtained through a number of trials in the system and updated. Further work will be necessary to ensure accurate estimation of the communication delays and accelerate the communication speed for the controllers in the system.

CHAPTER VII

Conclusions and Future Directions

Flexibility and adaptability are two important characteristics of manufacturing systems. The importance of these two attributes was recently demonstrated in the ongoing COVID-19 pandemic [41]. Manufacturers faced an unprecedented demand for medical equipment, such as face masks and ventilators, to help with the battle against the disease. Several companies put in significant engineering effort to reconfigure their entire existing manufacturing systems to produce new equipment [97,110]. However, this reconfiguration process took several weeks as new technology had to be integrated into manufacturing systems and new production schedules had to be developed. System-level control of the various parts, machines, and robots on the shop floor was was one of the challenges faced by manufacturers during this crisis.

One strategy that can be used to address these flexiblity and adaptability challenges faced by manufacturers is multi-agent control. In this strategy, a number of agents use data from the system, information from other agents, and a set of individual goals to drive the behavior of this system. This dissertation has focused on improving the intelligence of one of the most important agents in this control strategy – the product agent (PA). As described in Chapters I and II, the development of new models, methods, and algorithms can improve the performance of existing PAs and, thus, the multi-agent control strategy. Specifically, existing PAs primarily use rule-based reasoning to obtain the capabilities of the manufacturing system, schedule resource operations, and request actions from resource agents (RAs) in the system. This rule-based approach reduces the flexibility and adaptability of the PA and, in turn, of the multi-agent control strategy. Recently, model-based PAs have been proposed to improve the flexibility and adaptability of PAs. However, there are still a number of challenges that remain in the design, development, and implementation of the model-based PAs.

This dissertation improves the intelligence and capabilities of PAs by describing the models, interfaces, and communication required to make intelligent decisions in a dynamic manufacturing environment. The proposed product agent is able to efficiently explore the manufacturing environment, build a discrete event model to capture the dynamics and constraints of the system, and cooperate with other agents to achieve its goals. A multi-agent architecture with the proposed model-based PA was developed, tested, and analyzed using a simulated manufacturing environment and three manufacturing testbeds with various physical components. The results showcased in these experiments display the potential of the model-based product agent to develop more flexible manufacturing systems that can respond to a unexpected disturbances in the system caused by machine failures, new product orders, or, potentially, even a global pandemic.

7.1 Contributions

The core contributions of this dissertation are stated below. The first three contributions are also illustrated in Figure 7.1.

1) A model-based architecture for the product agents:

The design and testing of an architecture for a model-based product agent is described. As shown in Figure 7.1, the proposed product agent contains a Knowledge Base, Decision Maker, and Communication Manager. These components are used

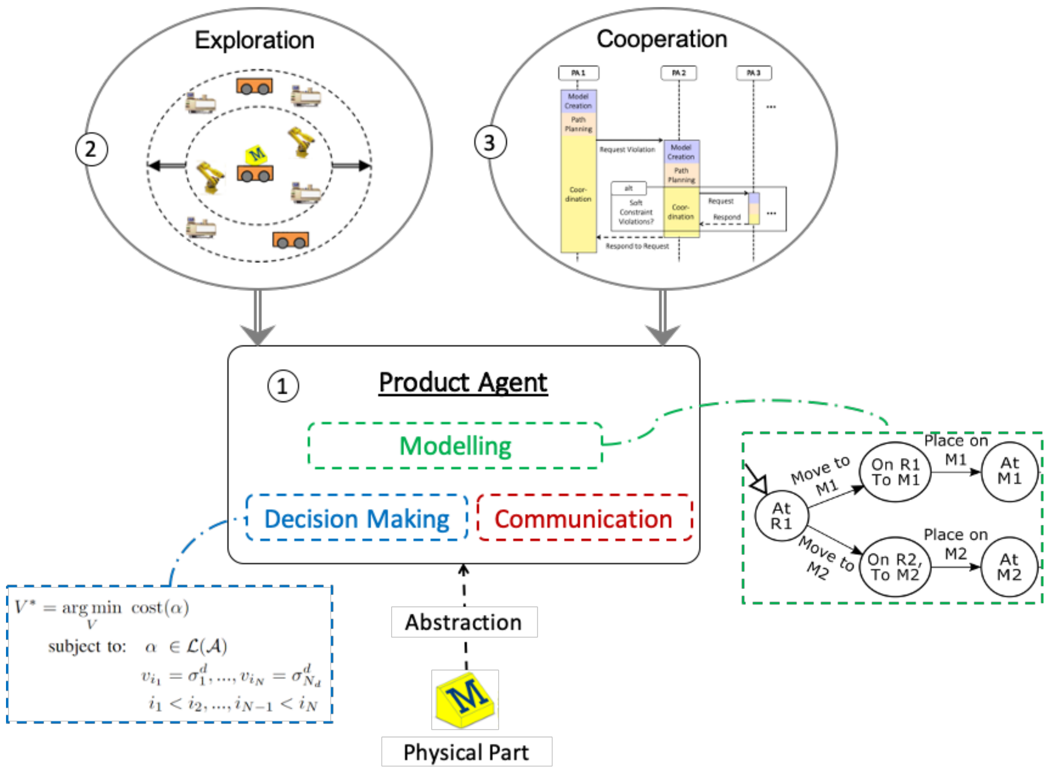


Figure 7.1: Three of the core contributions of the dissertation are (1) a model-based product agent architecture, (2) a methodology for product agent exploration, (3) a framework for direct, active cooperation in product agents.

by the product agent to explore the capabilities of surrounding resources, formulate plans based on its knowledge, and request actions to be taken by various resources based on the production goals of its associated physical part. This product agent architecture is 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.

2) An exploration methodology for efficient product agent model creation in a dynamic manufacturing environment:

A novel approach for PA exploration based on the propagation of bids and bid requests over a dynamic network of resource agents is presented. Using the proposed methodology, resource agents form teams to provide product agents with a comprehensive view of the surrounding environment. In addition to enabling RA task negotiation, this approach limits the amount of information available to agents, improving security and reducing communication overhead of agent systems. This exploration was tested and analyzed in three manufacturing testbeds.

3) A framework to enable direct and active cooperation for the product agent:

A model-based decision making framework to enable direct, actively cooperative product agents is introduced. For this framework, a new environment model that leverages the priced timed automaton modeling formalism captures the soft scheduling constraints from other agents in the system. Then, by solving an optimization problem, the direct, actively cooperating product agent identifies which of the soft scheduling constraints are conflicting with its individual goals. Finally, the product agent coordinates and negotiates with other agents in the system to resolve these conflicts and schedule its desired sequence of resource actions in the system. This framework is tested in a simulated manufacturing environment, showing how direct, active cooperation can improve the flexibility and performance of the manufacturing system.

4) Integration of the model-based product agent with industrial system controllers:

To showcase the potential of the models, methods, and algorithms developed in this dissertation, a multi-agent architecture with the model-based product agent was integrated into three manufacturing testbeds: the Fischertechnik testbed at the University of Michigan, the myJoghurt Demonstrator at the Technical University of Munich, and the System-level Manufacturing and Automation Research Testbed at the University of Michigan. The multi-agent controller was able to effectively produce customized parts in all three systems due to the decisions made by the model-based product agent. The communication times between the various agents and between the multi-agent architecture and lower-level system controllers were also analyzed, showing that this control strategy can be used in realistic, physical manufacturing systems.

7.2 Limitations and Future Work

This dissertation has explored several promising areas of research in the development of intelligent product agents for multi-agent control strategies for manufacturing. However, there are still a lot of remaining limitations, questions, and challenges in this field. While the model-based product agent was tested using simulations and real physical testbeds, there are still a number of challenges to scaling this model-based product agent to large, complex manufacturing systems. For example, there is a need to account for continuous manufacturing in the developed multi-agent controller by integrating other types of agents in the multi-agent architecture (e.g., a process agent). Another limitation to this work is the lack of a formal analysis to determine the difference in performance between the multi-agent control architecture and a traditional, centralized approach. While there are a number of other limitations and open questions for the model-based PA, three of the potential research directions that can leverage the work in this disseration are described in this section.

7.2.a Cooperative learning for product agents

The development of learning and self-adaptation is a popular research area for smart manufacturing systems [53, 61, 82]. However, the concept of learning has not been applied to intelligent PAs. While the manufacturing system may change dynamically or a new product may be requested by the customer, the PA can use some of the prior knowledge about the behavior of the manufacturing system to learn and improve its performance. By aggregating information about RA capabilities and previous decisions made by other PAs, the PA can make better decisions regarding its future plans.

An initial framework has been proposed in [92] that allows the automatic initialization of product and resource agents in manufacturing system. This framework takes in a customer order and information about the capabilities of the manufacturing system from a human to automatically create and initialize the process plan for a PA. The framework works offline to perform this matching and does not take into consideration the current state of the manufacturing system. An extension to this framework is to push this framework online, allowing it to synthesize and make decisions based on real-time data from the agents in the manufacturing system. This extension would enable the sharing of information between newly initialized PAs, PAs with parts in the system, and PAs that have completed their process plans. PAs would use this information to learn and improve their decision making capabilities over time.

7.2.b Developing complementary intelligence for resource agents

Due to the inherent nature of the multi-agent control strategy, the development of PA intelligence is linked to the behavior and performance of other agents, more specifically the RAs, in the system. Therefore, there is a significant amount of work to be accomplished for the development of intelligent RAs. To ensure that model-based PAs can effectively interact with RAs in the system, the knowledge base, decision making, and communication capabilities of the RAs needs to be developed and standardized. In addition, in this dissertation, it was assumed that RAs can accurately capture the capabilities of their associated resources, update their own knowledge of the environment through data received from the system, and efficiently respond to PA queries. However, none of these assumptions are trivial and the technology to enable these capabilities for RAs needs to be developed and tested. Finally, the cooperation capabilities of the RAs can be improved. For example, the methodology of RA teams in Chapter IV can be extended to improve RA responses to disturbances in the system. The models, methods, and algorithms proposed in this dissertation can be leveraged to improve the decision making of RAs. For example, the PTA-based environment model, the optimization formulation, and the solver developed for direct, active cooperation can be similarly leveraged by the RAs to improve their cooperation capabilities.

7.2.c Integration of intelligent product agents with a centralized control architecture

The pairing of the multi-agent control strategy with more traditional, centralized approaches is an interesting topic with a number of open research questions [21]. While traditional approaches are able to find optimal solutions to problems in system-level control, they lack the flexibility and adaptability that is inherent in multi-agent control strategies. Therefore, several frameworks have used a switching strategy to merge the two approaches [21]. In these frameworks, the centralized controller is used when production is going as planned and the multi-agent controller is turned on when there is a disturbance in the system. However, this hybrid approach does not fully utilize the full potential of both approaches.

Another approach to coupling both types of control strategies is to divide the responsibilities for each controller. In this type of approach, a centralized controller provides some control authority to agents in the system, allowing them to freely make their own decisions [84]. In the case of the product agent, a centralized controller would allow PAs to make decisions regarding which resource actions to schedule and request, but supervise the PAs to ensure that the parts are on track to complete their production requirements. A centralized controller that can interface with the model-based product agent developed in this dissertation would need to be developed to create this coupled control strategy.

The model-based PA developed in this dissertation should also be integrated with existing standards and requirements for Industry 4.0 systems. For example, the models and behavior of the PA should align with recent developments in the area of digital twins for manufacturing [86]. One potential standard that can be leveraged to standardize the model-based PA is the Asset Administration Shells (AAS) [117,124]. AAS is a framework that leverages Digital Twin concepts to ensure interoperability and exchange of information between components in Industry 4.0 systems. The model-based PA developed in this dissertation should be adapted and extended to meet standards and requirements of future manufacturing systems, such as the AAS framework or the digital twin framework proposed in [86].

7.3 Outlook and Impact

The work presented in this dissertation will enable small-batch manufacturing and more personalized production. Once the model-based PA is provided with a set of production requirements, it makes autonomous and intelligent decisions. Manufacturers will be able to complete small orders without the need to reconfigure or reschedule operations in the manufacturing system and, thus, will be more willing to accept a wider variety of customers and personalized orders.

Additionally, the integration of the multi-agent control strategy will allow more manufacturing technology to be incorporated into the shop floor. As shown in this dissertation, the model-based PA recognizes and communicates with a new resource and resource agent if both are added to the manufacturing system. Therefore, the multi-agent control strategy enables easier integration of new machines, robots, and other technology into the manufacturing system.

Finally, as a part travels between manufacturers, distributors, and customers, the associated PA can move with the part and store relevant information. The PA will be able to track the current state of the part, the completed manufacturing processes, and the part quality, among a number of other features. The PA will communicate this information with other agents, people, and companies, enabling a more connected manufacturing supply chain and improving the manufacturing process for manufacturers and customers alike.

BIBLIOGRAPHY

[1] ADS Automation Devices Specification

[2] Industrie 4.0. URL `http://i40d.ais.mw.tum.de/`

[3] Jade Site — Java Agent DEvelopment Framework. URL `http://jade.tilab.com/`

[4] Allen, L.V., Tilbury, D.M.: Anomaly detection using model generation for event-based systems without a preexisting formal model. IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans **42**(3), 654–668 (2012). DOI 10.1109/TSMCA.2011.2170418

[5] Ansola, P.G., García, A., de las Morenas, J., de las Morenas, J.: 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. Springer International Publishing, Cham (2015). DOI 10.1007/978-3-319-15159-5_9

[6] Balta, E.C., Kovalenko, I., Spiegel, I.A., Tilbury, D., Barton, K.: Application of Model Predictive Control of Priced Timed Automata Encoded with First-Order Logic. 2019 IEEE Transactions on Control Systems Technology (Submitted) (2019)

[7] Balta, E.C., Lin, Y., Barton, K., Tilbury, D.M., Mao, Z.M.: Production as a Service: A Digital Manufacturing Framework for Optimizing Utilization. IEEE Transactions on Automation Science and Engineering pp. 1–11 (2018). DOI 10.1109/TASE.2018.2842690

[8] Barbosa, J., Leitao, P.: Simulation of multi-agent manufacturing systems using Agent-Based Modelling platforms. 2011 9th IEEE International Conference on Industrial Informatics pp. 477–482 (2011). DOI 10.1109/INDIN.2011.6034926

[9] Barbosa, J., Leitão, P., Adam, E., Trentesaux, D.: Structural Self-organized Holonic Multi-Agent Manufacturing Systems. Industrial Applications of Holonic and Multi-Agent Systems SE - 6 **8062**, 59–70 (2013). DOI 10.1007/978-3-642-40090-2_6

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

[11] Barton, K., Maturana, F., Tilbury, D.: Closing the Loop in IoT-enabled Manufacturing Systems: Challenges and Opportunities. In: ACC, pp. 5503–5509. IEEE (2018)

[12] Behrmann, G.: UPPAAL CORA (2014). URL `http://people.cs.aau.dk/{~}adavid/cora/index.html`

[13] Behrmann, G., Larsen, K.G., Rasmussen, J.I.: Priced timed automata: Algorithms and applications. In: International Symposium on Formal Methods for Components and Objects, pp. 162–182. Springer (2004)

[14] Bemporad, A., Morari, M.: Control of systems integrating logic, dynamics, and constraints. Automatica **35**(3), 407–427 (1999). DOI 10.1016/S0005-1098(98)00178-2

[15] Bhave, D., Krishna, S.N., Trivedi, A.: On Nonlinear Prices in Timed Automata. In: Electronic Proceedings in Theoretical Computer Science, vol. 232, p. 65 (2016)

[16] Bjorner, N., Phan, A.D.: νz-maximal satisfaction with z3. In: 6th International Symposium on Symbolic Computation in Software Science, vol. 30, pp. 1—-9 (2014). DOI 10.29007/jmxj

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

[18] Brettel, M., Friederichsen, N., Keller, M., Rosenberg, M.: 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 (2014). DOI 10.1016/j.procir.2015.02.213

[19] Brettel, M., Klein, M., Friederichsen, N.: The Relevance of Manufacturing Flexibility in the Context of Industrie 4.0. Procedia CIRP **41**, 105–110 (2016). DOI 10.1016/j.procir.2015.12.047

[20] Cao, Y., Yu, W., Ren, W., Chen, G.: An overview of recent progress in the study of distributed multi-agent coordination. IEEE Transactions on Industrial informatics **9**(1), 427–438 (2012)

[21] Cardin, O., Trentesaux, D., Thomas, A., Castagna, P., Berger, T., El-Haouzi, H.B.: Coupling predictive scheduling and reactive control in manufacturing hybrid control architectures: state of the art and future challenges. Journal of Intelligent Manufacturing **28**(7), 1503–1517 (2017)

[22] Cassandras, C.G., Lafortune, S.: Introduction to discrete event systems, 2nd edn. Springer Science & Business Media (2009)

[23] De Las Morenas, J., Garcia-Higuera, A., Garcia-Ansola, P.: Shop Floor Control: A Physical Agents Approach for PLC-Controlled Systems. IEEE Transactions on Industrial Informatics **13**(5), 2417–2427 (2017). DOI 10.1109/TII.2017.2720696

[24] Diaconescu, E., Spirleanu, C.: Communication solution for industrial control applications with multi-agents using opc servers. In: 2012 International Conference on Applied and Theoretical Electricity (ICATE), pp. 1–6. IEEE (2012)

[25] Dias, J., Barbosa, J., Leitão, P.: Deployment of industrial agents in heterogeneous automation environments. In: 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), pp. 1330–1335. IEEE (2015)

[26] Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik **1**(1), 269–271 (1959)

[27] Du, J., Sugumaran, V., Gao, B.: Rfid and multi-agent based architecture for information sharing in prefabricated component supply chain. IEEE Access **5**, 4132–4139 (2017). DOI 10.1109/ACCESS.2017.2665778

[28] Endsley, E.W., Almeida, E.E., Tilbury, D.M.: Modular finite state machines: Development and application to reconfigurable manufacturing cell controller generation. Control Engineering Practice **14**(10), 1127–1142 (2006). DOI 10.1016/j.conengprac.2006.02.001

[29] Farid, A.M., Ribeiro, L.: An Axiomatic Design of a Multiagent Reconfigurable Mechatronic System Architecture. IEEE Transactions on Industrial Informatics **11**(5), 1142–1155 (2015). DOI 10.1109/TII.2015.2470528

[30] Fax, J.A., Murray, R.M.: Information flow and cooperative control of vehicle formations. IEEE transactions on automatic control **49**(9), 1465–1476 (2004)

[31] Ferrer, B.R., Ahmad, B., Lobov, A., Vera, D.A., Lastra, J.L.M., Harrison, R.: An approach for knowledge-driven product, process and resource mappings for assembly automation. IEEE International Conference on Automation Science and Engineering **2015-Octob**, 1104–1109 (2015). DOI 10.1109/CoASE.2015.7294245

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

[33] Gibson, I., Rosen, D.W., Stucker, B., et al.: Additive manufacturing technologies, vol. 17. Springer (2014)

[34] Hadeli, Valckenaers, P., Kollingbaum, M., Van Brussel, H., Hadeli, K., Valckenaers, P., Kollingbaum, M., Van Brussel, H.: Multi-agent coordination and control using stigmergy. Computers in Industry **53**(1), 75–96 (2004). DOI 10.1016/S0166-3615(03)00123-4

[35] Harrison, W.S., Tilbury, D.M., Yuan, C.: From hardware-in-the-loop to hybrid process simulation: An ontology for the implementation phase of a manufacturing system. IEEE Transactions on Automation Science and Engineering **9**(1), 96–109 (2012)

[36] Hekmatnejad, M., Pedrielli, G., Fainekos, G.: Task Scheduling with Nonlinear Costs using SMT Solvers. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE), vol. 2019-Augus, pp. 183–188. IEEE, Vancouver, Canada (2019). DOI 10.1109/COASE.2019.8843048

[37] Hoffmann, M.: Adaptive and Scalable Information Modeling to Enable Autonomous Decision Making for Real-Time Interoperable Factories. Dissertation, RWTH Aachen University, Aachen (2017). DOI 10.18154/RWTH-2017-07373. URL `https://publications.rwth-aachen.de/record/697974`

[38] Howden, N., Rönnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents-summary of an agent infrastructure. Management p. 6 (2001). DOI 10.1.1.133.8934

[39] Hsieh, F.: Dynamic configuration and collaborative scheduling in supply chains based on scalable multi-agent architecture. Journal of Industrial Engineering International **15**, 249–269 (2019)

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

[41] Ivanov, D., Dolgui, A.: Viability of intertwined supply networks: extending the supply chain resilience angles towards survivability. a position paper motivated by covid-19 outbreak. International Journal of Production Research **58**(10), 2904–2915 (2020)

[42] Kantamneni, A., Brown, L.E., Parker, G., Weaver, W.W.: Survey of multi-agent systems for microgrid control. Engineering applications of artificial intelligence **45**, 192–203 (2015)

[43] Kovalenko, I., Balta, E.C., Tilbury, D., Barton, K.: Direct, Active Cooperation for Product Agents in Manufacturing Systems. In-preparation (2020)

[44] Kovalenko, I., Barton, K., Tilbury, D.: Design and Implementation of an Intelligent Product Agent Architecture in Manufacturing Systems. In: IEEE Emerging Technology and Factory Automation (ETFA), pp. 1–8 (2017). DOI 10.1109/ETFA.2017.8247652

[45] Kovalenko, I., Ryashentseva, D., Vogel-Heuser, B., Tilbury, D., Barton, K.: Dynamic Resource Task Negotiation to Enable Product Agent Exploration in Multi-Agent Manufacturing Systems. Robotics and Automation Letters **4**(3), 2854–2861 (2019). DOI 10.1109/lra.2019.2921947

[46] Kovalenko, I., Saez, M., Barton, K., Tilbury, D.: SMART: A System-Level Manufacturing and Automation Research Testbed. Smart and Sustainable Manufacturing Systems **1**(1), 232–261 (2017). DOI 10.1520/ssms20170006

[47] Kovalenko, I., Tilbury, D., Barton, K.: The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems. Control Engineering Practice **86**(March), 105–117 (2019). DOI 10.1016/j.conengprac.2019.03.009

[48] Kovalenko, I., Tilbury, D., Barton, K.: Priced Timed Automata Models for Control of Intelligent Product Agents in Manufacturing Systems. In: 15th Workshop on Discrete Event Systems (WODES 2020) (2020)

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

[50] Krothapalli, N.K.C.C., Deshmukh, A.V.V.: Design of negotiation protocols for multi-agent manufacturing systems. International Journal of Production Research **37**(7), 1601–1624 (1999). DOI 10.1080/002075499191157

[51] Kusiak, A.: Smart manufacturing. International Journal of Production Research **56**(1-2), 508–517 (2018). DOI 10.1080/00207543.2017.1351644

[52] Larsen, K., Behrmann, G., Brinksma, E., Fehnker, A., Hune, T., Pettersson, P., Romijn, J.: As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In: International Conference on Computer Aided Verification, vol. 2102, pp. 493–505 (2001). DOI 10.1007/3-540-44585-4_47

[53] Lee, J., Bagheri, B., Jin, C.: Introduction to cyber manufacturing. Manufacturing Letters **8**, 11–15 (2016). DOI 10.1016/j.mfglet.2016.05.002

[54] Lee, J., Bagheri, B., Kao, H.A.: A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. Manufacturing Letters **3**, 18–23 (2015). DOI 10.1016/j.mfglet.2014.12.001

[55] Legat, C., Schütz, D., Vogel-Heuser, B.: Automatic generation of field control strategies for supporting (re-)engineering of manufacturing systems. Journal of Intelligent Manufacturing **25**(5), 1101–1111 (2014). DOI 10.1007/s10845-013-0744-z

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

[57] Leitao, P.: Multi-agent systems in industry: Current trends & future challenges. In: Beyond Artificial Intelligence, pp. 197–201. Springer (2013)

[58] Leitao, P., Colombo, A.W., Restivo, F.: An Approach to the Formal Specification of Holonic Control Systems. In: First International Conference on Industrial Applications of Holonic and Multi-Agent Systems, pp. 59–70 (2003). DOI 10.1007/978-3-540-45185-3_6

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

[60] Leitão, P., Karnouskos, S., Ribeiro, L., Lee, J., Strasser, T., Colombo, A.W.: Smart Agents in Industrial Cyber-Physical Systems. Proceedings of the IEEE **104**(5), 1086–1101 (2016). DOI 10.1109/JPROC.2016.2521931

[61] Leitão, P., Maík, V., Vrba, P.: Past, present, and future of industrial agent applications. IEEE Transactions on Industrial Informatics **9**(4), 2360–2372 (2013). DOI 10.1109/TII.2012.2222034

[62] Leitão, P., Restivo, F.: ADACOR: A holonic architecture for agile and adaptive manufacturing control. Computers in Industry **57**(2), 121–130 (2006). DOI 10.1016/j.compind.2005.05.005

[63] Leitão, P., Rodrigues, N., Barbosa, J., Turrin, C., Pagani, A.: Intelligent products: The grace experience. Control Engineering Practice **42**, 95–105 (2015). DOI 10.1016/j.conengprac.2015.05.001

[64] Leitner, J.: Multi-robot cooperation in space: A survey. Proceedings - 2009 Advanced Technologies for Enhanced Quality of Life, AT-EQUAL 2009 pp. 144–151 (2009). DOI 10.1109/AT-EQUAL.2009.37

[65] Lepuschitz, W., Zoitl, A., Vallee, M., Merdan, M.: 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 (2011). DOI 10.1109/TSMCC.2010.2059012

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

[67] Lopez, F., Shao, Y., Mao, Z.M., Moyne, J., Barton, K., Tilbury, D.: A software-defined framework for the integrated management of smart manufacturing systems. Manufacturing Letters **15**, 18–21 (2018)

[68] Lu, Y.: Industry 4.0: A survey on technologies, applications and open research issues. Journal of Industrial Information Integration **6**, 1–10 (2017). DOI 10.1016/j.jii.2017.04.005

[69] Lu, Y., Riddick, F., Ivezic, N.: 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. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-51133-7_90

[70] Lucas, M.R., Tilbury, D.M.: Methods of measuring the size and complexity of PLC programs in different logic control design methodologies. International Journal of Advanced Manufacturing Technology **26**(5-6), 436–447 (2005)

[71] Lüder, A., Klostermeyer, A., Peschke, J., Bratoukhine, A., Sauter, T.: Distributed automation: PABADIS vs. HMS. IEEE Transactions on Industrial Informatics **1**(1), 31–38 (2005). DOI 10.1109/INDIN.2003.1300283

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

[73] Luntz, J., Moyne, J., Tilbury, D.: On-Line Control Reconfiguration at the Machine and Cell Levels: Case Studies from the Reconfigurable Factory Testbed. In: IEEE Conference on Emerging Technologies and Factory Automation, vol. 1, pp. 641–648 (2005)

[74] Macal, C.M., North, M.J.: Introduction to Agent-based Modeling and Simulation (2006)

[75] Manyika, J., Sinclair, J., Dobbs, R., Strube, G., Rassey, L., Mischke, J., Remes, J., Roxburgh, C., George, K., O'Halloran, D., Ramaswamy, S.: Manufacturing the future: the next era of global growth and innovation. Tech. Rep. November, McKinsey Global Institute, New York (2012). URL `https://web.archive.org/web/20200129031406/https://www.mckinsey.com/business-functions/operations/our-insights/the-future-of-manufacturing`

[76] Marchetta, M.G., Mayer, F., Forradellas, R.Q.: A reference framework following a proactive approach for Product Lifecycle Management. Computers in Industry **62**(7), 672–683 (2011). DOI 10.1016/j.compind.2011.04.004

[77] McArthur, S.D., Davidson, E.M., Catterson, V.M., Dimeas, A.L., Hatziargyriou, N.D., Ponci, F., Funabashi, T.: Multi-agent systems for power engineering applications - part I: Concepts, approaches, and technical challenges. IEEE Transactions on Power systems **22**(4), 1743–1752 (2007)

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

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

[80] Meyer, G.G., Hans Wortmann, J.C., Szirbik, N.B.: Production monitoring and control with intelligent products. International Journal of Production Research **49**(5), 1303–1317 (2011). DOI 10.1080/00207543.2010.518742

[81] Mishra, N., Singh, A., Kumari, S., Govindan, K., Ali, S.I.: Cloud-based multi-agent architecture for effective planning and scheduling of distributed manufacturing. International Journal of Production Research **54**(23), 7115–7128 (2016). DOI 10.1080/00207543.2016.1165359

[82] Monostori, L.: AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. IFAC Proceedings Volumes (IFAC-PapersOnline) **15**(1), 119–130 (2002). DOI 10.1016/S0952-1976(03)00078-2

[83] Monostori, L., Váncza, J., Kumara, S.R.T.: Agent-based systems for manufacturing. CIRP Annals - Manufacturing Technology **55**(2), 697–720 (2006). DOI 10.1016/j.cirp.2006.10.004

[84] Moyne, J., Balta, E., Kovalenko, I., Qamsane, Y., Barton, K.: The digital twin in the manufacturing ecosystem of the future. In: 1st International Workshop on Smart Manufacturing Modeling and Analysis (2019)

[85] Moyne, J., Korsakas, J., Tilbury, D.M.: Reconfigurable factory testbed (RFT): A distributed testbed for reconfigurable manufacturing systems. In: Proceedings of the Japan-USA Symposium on Flexible Automation (2004)

[86] Moyne, J., Qamsane, Y., Balta, E.C., Kovalenko, I., Faris, J., Barton, K., Tilbury, D.M.: A requirements driven digital twin framework: Specification and opportunities. IEEE Access **8**, 107,781–107,801 (2020)

[87] Müller, J.P., Fischer, K.: Application impact of multi-agent systems and technologies: A survey. Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks **9783642544**, 27–53 (2014). DOI 10.1007/978-3-642-54432-3_3

[88] Myers, C.J., Meng, T.Y.: Synthesis of timed asynchronous circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **1**(2), 106–119 (1993)

[89] National Association of Manufacturers: Facts About Manufacturing — NAM. URL `https://web.archive.org/web/20200129030653/https://www.nam.org/facts-about-manufacturing/`

[90] Nilsson, F., Darley, V.: 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 (2006). DOI 10.1108/01443570610710588

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

[92] Ocker, F., Kovalenko, I., Barton, K., Tilbury, D., Vogel-Heuser, B.: A Framework for Automatic Initialization of Multi-Agent Production Systems Using Semantic Web Technologies. IEEE Robotics and Automation Letters **4**(4), 1–1 (2019). DOI 10.1109/lra.2019.2931825

[93] Olfati-Saber, R., Fax, J.A., Murray, R.M.: Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE **95**(1), 215–233 (2007)

[94] O'Madadhain, J., Fisher, D., White, S., Boey, Y.: The JUNG (Java Universal Network/Graph) Framework. Tech. rep., UCI-ICS (2003)

[95] Ota, J.: Multi-agent robot systems as distributed autonomous systems. Advanced engineering informatics **20**(1), 59–70 (2006)

[96] Panait, L., Luke, S.: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems **3**(11), 387–434 (2005). DOI 10.1007/s10458-005-2631-2. URL `http://link.springer.com/10.1007/s10458-005-2631-2`

[97] Park, C.Y., Kim, K., Roth, S.: Global shortage of personal protective equipment amid covid-19: supply chains, bottlenecks, and policy implications (2020)

[98] Pepyne, D.L., Cassandras, C.G.: Control of hybrid systems in manufacturing. Proceedings of the IEEE **88**(7), 1108–1122 (2000). DOI 10.1109/5.871312

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

[100] Polycarpou, M.M., Yang, Y., Passino, K.M., Shamma, J.S., Polycarpou, M.M., Yang, Y., Passino, K.M.: Cooperative control of distributed multi-agent systems. IEEE Control Systems Magazine **21**(June 2001), 1–27 (2001). DOI 10.1002/9780470724200

[101] Raman, V., Donzé, A., Maasoumy, M., Murray, R.M., Sangiovanni-Vincentelli, A., Seshia, S.A.: Model predictive control with signal temporal logic specifications. In: 53rd IEEE Conference on Decision and Control, pp. 81–87. IEEE (2014)

[102] Rehberger, S.: Combining Product- and Resource-Related Reasoning for Agent-Based Production Automation. Ph.D. thesis, Technische Universität München, München (2020)

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

[104] Rehberger, S., Spreiter, L., Vogel-Heuser, B.: An agent-based approach for dependable planning of production sequences in automated production systems. At-Automatisierungstechnik **65**(11), 766–778 (2017). DOI 10.1515/auto-2017-0040

[105] Ren, W., Beard, R.W., Atkins, E.M.: A survey of consensus problems in multi-agent coordination. In: Proceedings of the 2005, American Control Conference, 2005., pp. 1859–1864. IEEE (2005)

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

[107] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice Hall Press, Upper Saddle River, NJ, USA (2009)

[108] Sallez, Y., Berger, T., Trentesaux, D.: A stigmergic approach for dynamic routing of active products in FMS. Computers in Industry **60**(3), 204–216 (2009). DOI 10.1016/j.compind.2008.12.002

[109] Schroeder, K., Moyne, J., Tilbury, D.M.: A factory health monitor: System identification, process monitoring, and control. 4th IEEE Conference on Automation Science and Engineering pp. 16–22 (2008)

[110] Sean O'Kane: How GM and Ford switched out pickup trucks for breathing machines - The Verge (2020). URL `https://www.theverge.com/2020/4/15/21222219/general-motors-ventec-ventilators-ford-tesla-coronavirus-covid-19`

[111] Shen, W.: Distributed Manufacturing Scheduling Using Intelligent Agents. IEEE Intelligent Systems **17**(1), 88–94 (2002). DOI 10.1109/5254.988492

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

[113] Shen, W., Wang, L., Hao, Q.: Agent-based distributed manufacturing process planning and scheduling: A state-of-the-art survey. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews **36**(4), 563–577 (2006). DOI 10.1109/TSMCC.2006.874022

[114] Sisinni, E., Saifullah, A., Han, S., Jennehag, U., Gidlund, M.: Industrial internet of things: Challenges, opportunities, and directions. IEEE Transactions on Industrial Informatics **14**(11), 4724–4734 (2018). DOI 10.1109/TII.2018.2852491

[115] Stefan Aßmann: Potentiale von Industrie 4.0 bei Bosch. Tech. rep., Universität Stuttgart (2014)

[116] Tang, H., Li, D., Wang, S., Dong, Z.: CASOA: An Architecture for Agent-Based Manufacturing System in the Context of Industry 4.0. IEEE Access **6**, 12,746–12,754 (2017). DOI 10.1109/ACCESS.2017.2758160

[117] Tantik, E., Anderl, R.: Integrated data model and structure for the asset administration shell in industrie 4.0. Procedia Cirp **60**, 86–91 (2017)

[118] Tiegelkamp, M., John, K.H.: IEC 61131-3: Programming industrial automation systems, vol. 14. Springer (1995)

[119] Tilbury, D.M.: Cyber-Physical Manufacturing Systems. Annual Review of Control, Robotics, and Autonomous Systems **2**(1), null (2019). DOI 10.1146/annurev-control-053018-023652

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

[121] Vogel-Heuser, B., Diedrich, C., Pantförder, D., Göhner, P.: Coupling heterogeneous production systems by a multi-agent based cyber-physical production system. Proceedings - 2014 12th IEEE International Conference on Industrial Informatics, INDIN 2014 pp. 713–719 (2014). DOI 10.1109/INDIN.2014.6945601

[122] Vogel-Heuser, B., Lee, J., Leitão, P.: Agents enabling cyber-physical production systems. At-Automatisierungstechnik **63**(10), 777–789 (2015). DOI 10.1515/auto-2014-1153

[123] Vrba, P., Tichý, P., Maík, V., Hall, K.H., Staron, R.J., Maturana, F.P., Kadera, P.: 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 (2010). DOI 10.1109/TSMCC.2010.2055852

[124] Wagner, C., Grothoff, J., Epple, U., Drath, R., Malakuti, S., Grüner, S., Hoffmeister, M., Zimermann, P.: The role of the industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In: 2017

22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–8. IEEE (2017)

[125] Wang, S., Wan, J., Zhang, D., Li, D., Zhang, C.: 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 (2016). DOI 10.1016/j.comnet.2015.12.017

[126] Wannagat, A.: Entwicklung und Evaluation agentenorientierter Automatisierungssysteme zur Erhöhung der Flexibilität und Zuverlässigkeit von Produktionsanlagen. Ph.D. thesis, Technische Universität München, München (2010)

[127] Wong, T.N., Leung, C.W., Mak, K.L., Fung, R.Y.: Dynamic shopfloor scheduling in multi-agent manufacturing systems. Expert Systems with Applications **31**(3), 486–494 (2006). DOI 10.1016/j.eswa.2005.09.073

[128] Wooldridge, M.J.: An Introduction to MultiAgent Systems, 2nd edn. John Wiley & Sons, West Sussex UK (2009)

[129] Yao, X., Zhou, J., Lin, Y., Li, Y., Yu, H., Liu, Y.: Smart manufacturing based on cyber-physical systems and beyond. Journal of Intelligent Manufacturing pp. 1–13 (2017). DOI 10.1007/s10845-017-1384-5

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

[131] Zheng, G., Kovalenko, I., Barton, K., Tilbury, D.: Integrating Human Operators into Agent-based Manufacturing Systems: A Table-top Demonstration. Procedia manufacturing **17**, 326–333 (2018). DOI 10.1016/j.promfg.2018.10.053