CHALMERS
UNIVERSITY OF TECHNOLOGY

Goal-Oriented Process Plans in a Multiagent System for Plug & Produce

Downloaded from: https://research.chalmers.se, 2026-11-07 07:09 UTC

Citation for the original published paper (version of record):
Bennulf, M., Danielsson, F., Svensson, B. et al (2021). Goal-Oriented Process Plans in a Multiagent System for Plug & Produce. IEEE Transactions on Industrial Informatics, 17(4): 2411-2421. http://dx.doi.org/10.1109/TII.2020.2994032

N.B. When citing this work, cite the original published paper.

© 2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, or reuse of any copyrighted component of this work in other works.

(article starts on next page)

Goal-Oriented Process Plans in a Multiagent System for Plug & Produce

Mattias Bennulf, Fredrik Danielsson, Bo Svensson, and Bengt Lennartson, *Fellow, IEEE*

***Abstract*—This article presents a framework for Plug & Produce that makes it possible to use configurations rather than programming to adapt a manufacturing system for new resources and parts. This is solved by defining skills on resources, and goals for parts. To reach these goals, process plans are defined with a sequence of skills to be utilized without specifying specific resources. This makes it possible to separate the physical world from the process plans. When a process plan requires a skill, e.g., grip with a gripper resource, then that skill may require further skills, e.g., move with a robot resource. This creates a tree of connected resources that are not defined in the process plan. Physical and logical compatibility between resources in this tree is checked by comparing several parameters defined on the resources and the part. This article presents an algorithm together with a multiagent system framework that handles the search and matching required for selecting the correct resources.**

***Index Terms*—Multiagent, Plug & Produce, process plan, robotics.**

I. Introduction

SINCE late 1980s mass customization has become more common and now aims at reaching production costs close to dedicated manufacturing systems [1]. The life cycle for products is becoming shorter, making traditional approaches for automation ineffective. There is a need to develop new control strategies that can handle various changes without reprogramming, such as production fluctuations, the addition of resources, and the introduction of new products [2].

Conventional centralized approaches are dedicated to specific tasks, forcing personal to understand much of the code and logic, manually programmed in robots and programmable logic controllers (PLCs) when changes are made to manufacturing systems [3]. Instead, this article aims at spreading out the logic and parameter data on agents related to each resource and part in a manufacturing system. In this article, the parts are metal pieces to be processed, and the resources are one industrial robot surrounded by different process modules for machining and storage. Distributing the controller on multiple agents makes it possible to change the behaviour of a resource or part, without considering other resources or parts. For example, if introducing a completely new type of tool to the robot cell, our approach requires no downtime. The tool can be calibrated in a separate environment and data saved in its agent, before adding it to the manufacturing system. In a traditional approach, the robot cell commonly needs to be stopped and the robot code changed to achieve this reconfiguration.

Manufacturing system concepts have varied over time. Initially, functional workshops were used as a norm [4]. Functional workshop structures still exist today, due to their ability to handle low volume products with a very diverse range of products, but they are characterized by a low level of automation due to their complexity [5]. The complexity of such a system can become immense, and it is difficult to get an overview of its flow. Moreover, if shared by many products, it will generate complex and unpredictable flows, which are hard to balance. It is easier to focus on resource efficiency (overall equipment effectiveness) rather than flow efficiency in such a situation [6].

Reconfiguration and flexibility have been researched for several decades in automation [7]. Flexible manufacturing systems (FMSs) was developed in the 80's [8] and reconfigurable manufacturing systems (RMSs) in the 90's [8], [9]. They both aim at taking care of customization and short product life cycles. Even if FMS and reconfigurable concepts are examples of existing solutions for the automation of functional workshops, and the literature confirms the benefits of flexibility in automated manufacturing, the industrial experience still points out several shortcomings. FMS still have too high installation cost, due to rigid control solutions, and RMS are still not flexible enough to support fast reconfiguration, where machines are to be added and removed [7]. Manufacturing systems that handle quick connection and use of new devices are often regarded as Plug & Produce systems. This concept was first introduced in [10], where multiple resources could be added, containing a local controller.

This article addresses reconfigurability by defining a new multiagent system (MAS) framework for Plug & Produce. The framework is general and can be applied to many manufacturing systems, but the focus in this article is on local robotized flows in functional workshops.

The main idea is to be agile and able to create manufacturing systems when needed on short term notice. For Plug & Produce,

Manuscript received January 15, 2020; revised March 24, 2020; accepted April 21, 2020. Date of publication May 28, 2020; date of current version January 4, 2021. This work was supported by Miljö för Flexibel och Innovativ Automation under Project no. 20201192 funded by the Europeiska regionala utvecklingsfonden/VGR. Paper no. TII-20-0219. *(Corresponding author: Mattias Bennulf.)*

Mattias Bennulf, Fredrik Danielsson, and Bo Svensson are with the Department of Engineering Science, University West, 46132 Trollhättan, Sweden (e-mail: mattias.bennulf@hv.se; fredrik.danielsson@hv.se; bo.svensson@hv.se).

Bengt Lennartson is with the Department of Electrical Engineering, Chalmers University of Technology, 41296 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se).

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2020.2994032

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

the time to set it up should be measured in minutes rather than days or months as for older concepts. The setup time can be divided into two main parts, the hardware installation time (hardware time) and the time spent on programming and configuring the system (software time). The hardware time can be handled by using standardized connectors and standardly sized modules. This has been described in previous work, such as [11], [12], where it is defined as mechatronic compatibility. Modular hardware architectures have been implemented and tested in other works such as [13], [2]. Using standardized physical connectors and modules, it is easy to move resources around to form local setups on demand. However, for the software time , there is also a need for reconfiguration/reprogramming of the logical system to integrate the added components [12], [14].

The focus of this article is to formulate a solution that avoids the time it takes to program equipment for new tasks, i.e., decreasing time spent on software time. The key components are intelligent and collaborative resources. An intelligent and collaborative resource is not programmed in a traditional way, such as PLC and robot control code, where logic and data are mixed in one big integrated and dedicated solution. Instead, each resource is assigned an agent upon activation. An agent is a standardized software package, able to communicate and collaborate with other agents [15]. A software agent is unique in the world by its configuration, which is loaded when it is instantiated. The configuration mainly describes the physical properties and skills associated with a specific resource. In this way, each agent becomes a unique controller for a specific resource. When several smart resources are grouped together, they collaborate to form a local manufacturing system. Together they can offer more aggregated and advanced skills, depending on the resources involved.

In the same way, every part in the system, that should be processed, has a related agent representing its physical properties. The part agents have goals that they want to reach by using available skills of the resources. Multiple process plans can be defined in the system, describing how to reach a goal. These plans are written like recipes rather than programs. They describe how skills on resources should be used without specifying specific resources or routes through the manufacturing plant. Each skill on a resource has its own process plan for executing the skill. These plans might require additional skills on other resources, forming a tree of connected agents, collaborating to solve a part goal. This further simplifies the process plans for part goals, by hiding the chain of skills and resources needed for a specific step in that process plan.

The use of goals and configurations associated with parts simplifies the process of adding new products. The goals and configuration values are the only information needed to describe what to be done for a specific part. The process plan separates the skills of resources from the part goals. This makes it possible to have several potential solutions in the system that become available, depending on what resources currently are connected to the system.

The main contribution of this article is a new framework for developing MASs for Plug & Produce, where no programming is needed when new parts are introduced. Additionally, the time spent on programming resources is decreased drastically. This makes it possible to add new types of products and resources in terms of minutes rather than days in traditional approaches. It includes a novel approach for defining process plans that describe how to reach a specific part goal in a manufacturing system. A recursive search algorithm is developed that can form the tree of connected resources needed to run a given process plan, defined for a goal. Resources are checked for both physical and logical compatibility before added to the tree. The framework described in this article has been implemented and tested in our lab, based on an industrial scenario described in this article. The framework extends a previously developed MAS in [16] and [17].

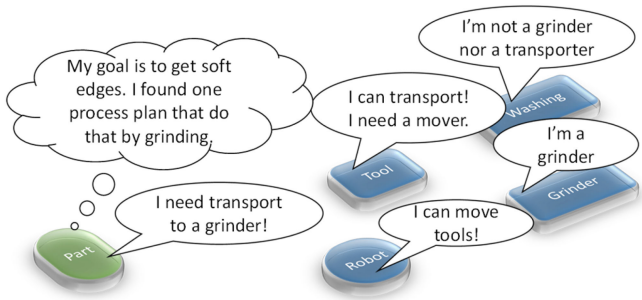

Fig. 1. Simplified example of the Plug & Produce concept.

The rest of this article is organized as follows. Section II introduces related work and compares it to this article. Section III introduces the Plug & Produce framework together with an algorithm for mapping goals to resources. Section IV presents an experiment where the proposed algorithm is tested using an industrial scenario. Section V gives the evaluation of the experiments conducted, and finally, Section VI concludes this article.

II. Background

MASs offer a distributed approach to specifying system behaviours, instead of writing programs with a list of low-level sequential instructions. An agent can be instructed what to do in terms of more high-level goals it must fulfil and communicates with other agents to find solutions for reaching those goals [18]. In Fig. 1, a simplified example is shown where a part has the goal to get soft edges. The part is equipped with a strategy to find one or more process plans for this and starts to communicate with the other agents to find a feasible solution.

Similarly to our article, Krothapalli and Deshmukh [19] present a multiagent manufacturing system where parts and resources are agents with communication capabilities. Parts have a primary objective to perform specific processing. Parts communicate with resources or other parts by broadcasting messages to all agents. Parts will be processed on any machine that can perform the required process.

However, the use of MASs in manufacturing systems is still uncommon today [20], [21]. To change this, there is a need

for simplification of configuration tools that enable system designers to configure the agents without understanding the complexity of the underlying MAS [20], [22]. Instead, the system designer should be separated from low-level communication and negotiation strategies of the agents. It is also clear that MASs have to be easy to integrate with already existing resources in a manufacturing system [23], [24].

A description of agents was published in 1995 by Wooldridge and Jennings [25]. They describe an agent as some hardware or software, operating without human intervention. Agents perceive the environment and react to it. Several agents can communicate with each other through agent-communication languages. They can also have goals that they want to reach in the world. MASs enable devices to adapt to new situations [26], which is of importance in a Plug & Produce system. Examples of physical agents could be autonomous robots [27], and software agents could be implementations of services in a system. However, the distinction between these two is not always clear, since robotic systems are hardware-based, while the robot controller usually is software-based. In this article, an agent is described as a piece of software, representing some object. The object can be physical, e.g., a part or a resource, or it can be a software function, e.g., transportation planning.

Standards for multiagent design and communication were defined already in 1997 [28] by the foundation for intelligent physical agents (FIPA). This is an IEEE organization that focuses on developing standards for MASs [29]. FIPA presents a collection of several specifications. Two important specifications defined by FIPA are the "agent management specification" [30] that describes general guidelines on how to design an MAS, and the "agent communication language" specification [31] that gives guidelines on how to design agent communication. Java agent development framework (JADE) is a library for Java used for agent implementation that follows several standards from FIPA [32].

However, the FIPA specification and the JADE library describe nothing about how to develop a framework for manufacturing systems where fast reconfiguration for new parts and resources is needed. This is the topic of which the Plug & Produce framework presented in this article is focused.

A. Related Work

This section presents several articles with related work and compares them with this article.

Schou and Madsen [14] describe a Plug & Produce framework for industrial robots. They divide devices like grippers and robots into different agents to form an MAS. The article presents a control framework that is supposed to handle quick and easy exchange of hardware modules. They aim at solving this by separating the high-level task control from the hardware.

Instead, the focus of this article is on distributing the controller on more agents. This means that the combination of agents to form, for instance, a robot with a gripper is performed in a completely distributed way, where the agents for the gripper and the robot agree on how to collaborate. This further simplifies the adding of new devices.

Järvenpää *et al.* [33] describe a system where combined capabilities/skills can be described by defining a list of capabilities required for the combined capability, e.g., by combining a robot with the capability *moving* and a gripper with capability *holding*, the combined capability *transportation* could become available in the system. Similarly, Antzoulatos *et al.* [34] present an MAS developed on the JADE platform that can match the capabilities/skills of resources with product specifications. They give resources skills like *move* with a robot and *grasp* with a gripper. They may also form complex capabilities combining these capabilities to create a *pick and place* capability. This is done by manually defining the required capabilities to be performed for the complex capability.

In this article, we use a different approach. For instance, a skill *transport* can be defined on the gripper. The gripper cannot perform the skill *transport* alone, so it has a requirement for an additional skill *move* that could exist on a robot. This connection by requirements for further skills will form a tree of connected agents that are working together, where the knowledge of requirements is entirely distributed.

Park *et al.* [35] present an agent communication framework for rapid reconfiguration of distributed manufacturing systems. They separate physical and logical reconfigurability and identify that both are required for a manufacturing system to be effectively reconfigured. A system has been developed that is able to perform automatic layout change detection in the manufacturing system, using infrared sensors between modules.

In addition, our work considers the physical and logical compatibility of resources when combining skills into complex skills.

Agents can communicate in order to get information about each other, or they can use a centrally stored knowledge base about the resources in the system, to avoid broadcasting. In [36], such a knowledge base is used for MAS planning of manufacturing sequences.

In this article, we have avoided a central knowledge base, and each agent instead builds up their own knowledge base.

Sutton *et al.* [37] describe hierarchical reinforcement learning with options. For example, an option that describes how to open a door consists of three components, a policy, a termination condition, and an initiation set. The policy describes the actions defined for reaching a final state, in this case: reaching, grasping and turning the doorknob. The terminating condition is the knowledge that the door has been opened and the initial state defines the requirement that the door should be present.

The options described in [37] have similarities to the process plans presented in this article since both describe a set of actions to be taken in order to reach a final state. Both approaches are used for planning the behaviour of an agent, moving around in a physical environment. However, it should be noted that our framework is applied and verified in an industrial manufacturing system and that the focus in this article is not planning. Instead, our focus is to decrease the time to add new parts and resources to a manufacturing system.

Vallèe *et al.* [38] show a MAS that uses ontologies for expressing concepts and properties of agents in the system. This is done to ensure a common understanding between agents

when communicating. This was done to avoid traditional agent approaches where reasoning about concepts is hardcoded into the agent's behaviors.

In this article, agents also share a common understanding of concepts, without any hard coding. This is part of the agent configuration and can be changed without reprogramming.

III. Plug & Produce framework

To achieve physical flexibility, the system described in this article uses self-aware and independent Plug & Produce process modules. Several modules can be grouped together to form an automated local manufacturing setup. To handle such a flexible system, a controller that adapts to available resources is necessary. To implement the controller, a distributed control strategy based on a MAS framework is adopted.

A MAS consists of many single agents that can interact with each other. Two types of agents are considered, part agents and resource agents. Part agents have goals and use process plans to reach them. A process plan translates design information (goals) into a sequence of operations (skills) needed to produce a part with the desired properties. To run a process plan on a part agent, several skills and variables are required to exist on resources in the agent network. In this article, this is noted as demands. Resources also have plans that define how their skills are executed. These plans might require additional skills to exist on other resources in the agent network. A search algorithm distributed on each agent is used to find this tree of connected interfaces between agents.

Agents always interact with each other through interfaces, meaning that a resource used by a part must have an interface that is compatible with one of the parts interfaces. This ensures that they are compatible both physically and logically. The demands introduced earlier should be seen as requirement specifications for what skills and variables an interface needs to have. The interface can, for instance, describe that a grinding wheel is compatible with a motor. If an interface with the keyword "ifTool" exists on a grinding wheel, then a motor to be connected to it also needs an interface with that keyword, to ensure physical compatibility. Interfaces also need to have compatible signals, i.e., variables. If the wheel requires to set the speed of the motor on variable RPM $=$ 500, then that variable must exist on the motors interface and be able to handle that speed. In this way, physical and logical compatibility is checked to match resources by searching interfaces.

A. Multiagent System

In the proposed Plug & Produce framework an agent a belongs to the set of agents A, i.e., $a \in A$. Two types of agents exist, parts p and resources r, see Fig. 2. A part p is included in the set of parts P, while a resource r is a member of the set of resources R. Hence, $p \in P$ and $r \in R$. The set of all agents A consist of all resources R and parts P, i.e., $A = R \dot{\cup} P$. Parts and resources have different agent strategies, where parts are trying to reach goals, while resource agents represent available physical or virtual resources.

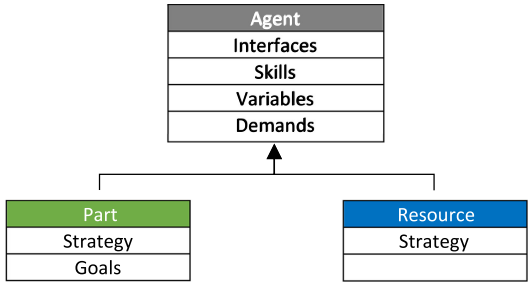

Fig. 2. Diagram with classes for part and resource agents, which have different strategies for part and resource.

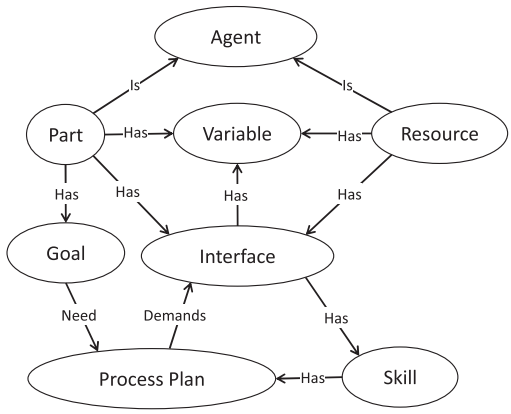

Fig. 3. Diagram, showing the agent configuration classes and their relations to each other, i.e., the agent ontology.

Goals can be described quantitatively by specifying parameters together with the goal, e.g., SoftEdges(RPM: $=$ 500), where the soft edges should be produced with a speed of 500, in the case that it is achieved by grinding. A resource agent has no goals, but can be used by part agents. In this way, a resource agent will facilitate the production of parts in the manufacturing setup by offering services, e.g., grinding, transportation, or path planning. The core idea is to be able to plug in all kinds of resources, needed to handle the on-going manufacturing. Resources not needed can be inactive or unplugged and stored for later use.

Each agent $a \in A$ has a configuration. The configuration is created manually and may apply to several agents, e.g., several parts of the same type. It is through the configuration that goals, skills, interfaces, variables, and demands are defined, without programming. Once the configuration is downloaded to an agent, it becomes unique for that specific agent instance. In Fig. 3, the agent configuration classes and their relations to each other are shown. This ontology is used by all agents in the system to share a common understanding of how data is constructed. Furthermore, when these classes are instantiated with configuration data, all agents must understand the naming of skills and variables. This requires all agents to follow some naming standards in order to communicate.

All agents have at least one associated interface, $if \in I$. An interface represents a point of interaction between two agents that are compatible both physically and logically. The interface

defines the compatibility between agents by defining its skills $s \in S$ and configuration variables $v \in V$. Hence, an interface if is defined by the tuple

$$if = \langle S_{if}, V_{if} \rangle$$

where $S_{if} \subseteq S$, and $V_{if} \subseteq V$. A variable v can, for instance, be a coordinate for a resource, a path for a robot, or a motor start signal. A skill on a resource r is defined by a name together with a process plan π_s for executing the skill, forming the tuple

$$s = \langle \text{name}, \pi_s \rangle .$$

A single skill s represents a service, presented through an agent interface that can be utilized by other agents on request. However, a skill is only available if certain demands are fulfilled. For instance, the skill *grip* on a robot gripper has a demand for a robot to be mounted on. The robot needs to have the skill *move* and certain variables available. In some cases, a demand includes several skills that must exist on the same resource instance. For example, if a part needs a gripper for transportation, the same gripper must have the skills *pick* and *place*. It would not make sense if these skills were on two different physical grippers.

Since available resource interfaces are unknown at the planning stage, they are defined as abstract interfaces $u \in U = \{u_1, u_2, \ldots, u_{n_u}\}$. When executing a process plan at runtime, mapping of the abstract interfaces in U to resource interfaces in I is carried out by an interface mapping algorithm, presented in Section III-C. The algorithm generates demands $d_u \in D = \{d_1, d_2, \ldots, d_{n_d}\}$ and

$$d_u = \{S_u, V_u\} .$$

Thus, a demand d_u defines skills and variables that an abstract interface u should be able to satisfy.

In addition to interfaces, parts also have an associated set of goals, $G_p \subseteq G$, where one goal $g \in G_p$. A goal represents a result that a part should achieve with available resources, e.g., to get soft edges. The part and resource agents can thus be described as tuples

$$p = \langle G_p, I_p, V_p \rangle$$
$$r = \langle I_r , \mathrm{V}_r \rangle .$$

Note that interfaces contain skills that have process plans. Hence, resource agents with skills carry their own process plans. To find a solution that solves a part goal, we need to map goals on parts with skills on resources. This is typically done by a process plan. As already has been mentioned, a process plan translates design information (goals) into a sequence of operations (skills) needed to produce a part with the desired properties. Process plans can be generated automatically or designed manually by a human. For industrial manufacturing, it is difficult for softwares to create a process plan that meets specific demands. This is knowledge that today is more suitable to be defined manually by humans [39], [33].

All process plans in the system are defined and included in the set $\Pi = \Pi_G \dot{\cup} \Pi_S$, where one plan is $\pi \in \Pi$. The process plans for part goals in Π_G are general and shared among all parts. Process plans for skills $\pi_s \in \Pi_S$ are instead defined for a specific skill s on a resource r, describing how that skill should be executed.

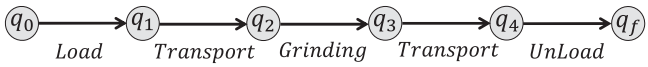

Fig. 4. Example of a process plan with five skills $\{\text{Load}, \text{Transport}, \text{Grinding}, \text{Transport}, \text{UnLoad}\}$ and six states, where the initial state is q_0 and the final state is q_f.

In this article, a process plan π defines a sequence of skills $(s_1, s_2, \ldots, s_{n_\pi})$, that should be executed in a specific order. Process plans for goals π_g only describe the solution for a single goal. However, there may be several ways to achieve that goal. This is managed through the fact that several process plans in the set Π_g may exist for the same goal g. For each goal $g \in G_p$, there must exist at least one process plan in Π. A process plan, π_g or π_s, can be formulated as a finite state automaton

$$\pi = \langle Q, S, \delta, q_0, Q_f \rangle$$

where Q is the set of states in the process plan, S is the set of all skills, $\delta : Q \times S \rightarrow Q$ is the transition function, $q_0 \in Q$ is the initial state, and $Q_f \subseteq Q$ is the set of acceptable final states. This means that a process plan may include possible alternative sequences of skills. In Fig. 4 an example is shown of a process plan π_g solving the goal $g = \text{SoftEdges}$, where the single final state q_f is the only element in Q_f.

B. Agent Strategies

As soon as a new part or resource is added to the manufacturing system, a corresponding agent a is instantiated representing that specific object. The idea behind the agent concept is that each object should be independent, self-aware, and autonomous.

Part agent strategy: A part agent p will start by trying to fulfill the first goal g in the set of personal goals G_p, by finding all process plans $\Pi_g \subseteq \Pi_G$ that describe how to reach that goal. When a goal is reached the agent continues with the next goal. When all goals in G_p have been achieved, the part agent p is deleted, and the corresponding part is considered as completed.

To select the most suitable process plan for a specific goal g, each plan $\pi_g \in \Pi_g$ is checked for availability by asking all resource agents in the agent network if they have any of the skills required in π_g and has a compatible interface for interaction. The compatibility between interfaces could for instance deal with the interaction between a gripper and a robot. A resource might need to ask other resources to assist in order to fulfil a desired skill. In Fig. 5, this is illustrated, where a part p has three goals in $G_p = \{g_1, g_2, g_3\}$. The first goal g_1 has two alternative process plans $\Pi_{g_1} = \{\pi^1_{g_1}, \pi^2_{g_1}\}$ that can solve g_1. Both these plans are checked for availability. However, in the figure, only plan $\pi^1_{g_1}$ is described. Plan $\pi^1_{g_1}$ has two demands, $d(s_1, v_5)$ for skill s_1 and $d(s_2, v_4, v_3)$ for skill s_2. This means that s_1 has to execute on an interface containing a v_5 variable and s_2 needs to execute on an interface that has a v_4 and a v_3 variable. The agent searches the network and finds the interfaces if_3 and if_4, respectively.

When each plan in Π_{g_1} is checked for availability (by running the algorithm described in this article), the one with the lowest

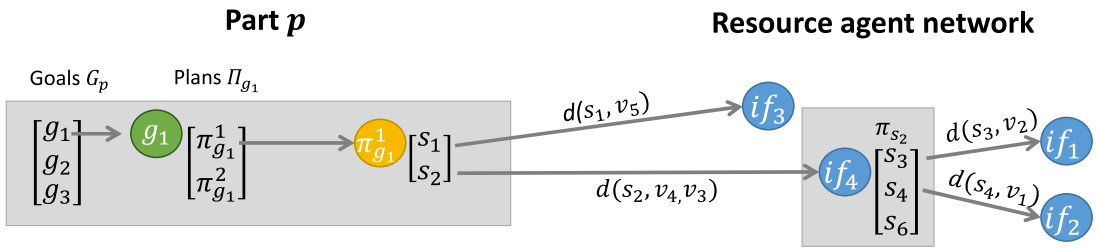

Fig. 5. Process plan that achieves the goal g_1 is checked for availability. The first plan $\pi^1_{g_1}$ requires skills that exist on another agent through the interfaces if_3 and if_4, where if_4 requires additional skills that exist on if_1 and if_2.

cost is selected, i.e., selected plan $\pi_{g_1} = \min(\Pi_{g_1})$. In this way, an agent can minimize the cost (execution time) by selecting the most effective process plan. After a plan is selected, each skill in the plan is executed on resource agents in the network. The following steps summarize the part agent strategy.

Part Agent Strategy:

Step 1: Find next goal g *in* G_p that is not yet reached.

Step 2: Find available process plans Π_g that fulfill g.

Step 3: Select the process plan π_g with the lowest cost.

Step 4: For each skill in the selected plan, execute it on a resource agent.

Resource agent strategy: In the same way as for the parts, each resource is handled by an agent that is instantiated when the resource is connected to the system. Each resource agent can execute associated skills. A skill can be executed on request from other agents and is only available if the demand d_u is fulfilled for that skill. Several agents that cooperate in this way can be viewed as an aggregated agent capable of more complex actions, as illustrated in Fig. 5. For each skill s on a resource r a process plan π_s is configured with knowledge about the execution of that skill. In contrast to the process plans in Π_G for part goals, a plan π_s for skill s only describes the use of a skill belonging to a specific agent type. For instance, for agents close to the hardware there might be a need for setting I/O values. In Fig. 5 it is shown that the plan π_{s_2} for executing the skill s_2 on interface if_4 requires additional skills s_3, s_4, s_6 and finds if_1 and if_2 for the demands $d(s_3, v_2)$ and $d(s_4, v_1)$. Skill s_6 is a local skill and is locally executed. Since all skills for process plan π_{s_2} exist, if_4 becomes available.

C. Interface Mapping Algorithm

In this article, a process plan needs to be connected to interfaces on resource agents in the network. This results in an executable process plan π^e. When running the interface mapping algorithm, process plans in Π are connected with resources in the network through interfaces, forming a set of executable plans Π^e. Hence, Π^e contains the executable versions of the plans in Π. For part goals, the single executable process plan π^e that can reach that goal with the lowest cost is selected. The cost is specified on each resource skill and can be of any type as long as it is expressed as an integer number. In the scenario presented in this article, the cost is the execution time for a process plan. The cost for one process plan includes the costs for all process plans needed to execute in the underlying tree of connected interfaces.

A general algorithm has been developed that is implemented in all agents (parts and resources), taking a plan π as input and determine if it is available or not. If the plan π is available, an executable process plan π^e is returned, describing what resources, interfaces, and variables to use for the skills in the plan.

The algorithm works by generating demands d_u for each abstract interface u in the process plan π. These demands are then broadcasted to resources in the agent network that reply if they fulfil the demand d_u or not. The algorithm can be divided into three main steps.

Step 1: Find all demands D in π. Individual demands d_u consist of required skills S_u and variables V_u

$$D = \{d_1, d_2, \ldots, d_{n_d}\}$$
$$d_u = \{S_u, V_u\}.$$

Step 2: Each identified demand d_u, in step 1, must be fulfilled by an interface on a resource. Hence, the algorithm requires abstract interfaces $U = \{u_1, u_2, \ldots, u_{n_u}\}$. For each demand d_u, search interfaces in I to check if they can perform all needed skills S_u with the required variables V_u. The agent a running this algorithm has a set of local interfaces I_a, where one local interface is defined as $if_a \in I_a$. For each interface $if \in I$ that meet the demand d_u, check if it is compatible with any of the local agent's interfaces $if_a \in I_a$. If they are compatible, store them locally as potential interfaces I^p

$$\text{for each abstract interface } u \text{ in } U \begin{cases} d_u = \{S_u, V_u\} \\ I^p_u = \{\text{ if} \in I \mid if \text{ fulfils } d_u \ \Lambda \text{ if compatible with if}_a \in I_a\} \end{cases}.$$

Step 3: From the potential interfaces I^p, choose the ones with the lowest cost and store as selected interfaces I^s. If π is feasible, then generate an executable process plan π^e containing the selected interfaces I^s together with the original plan π. Return this as the result of the algorithm

$$\text{for each element } u \text{ in } U: \ \{I^s = \min(I^p_u)$$
$$\pi^e = \langle \pi, I^s \rangle.$$

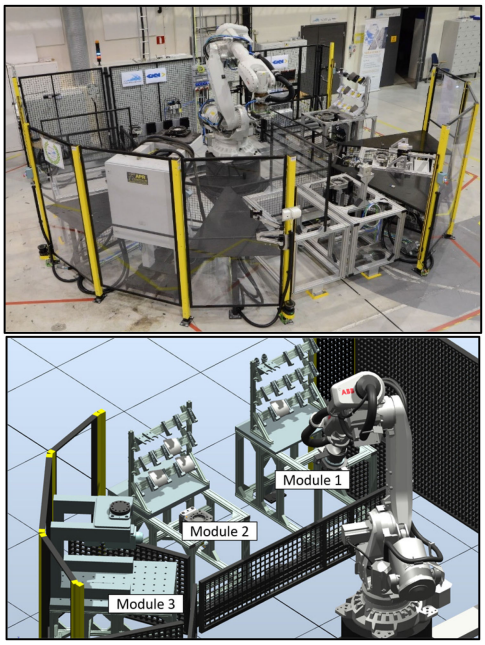

Fig. 6. Real industrial Plug & Produce demonstrator at University West and a simulation model. This demonstrator was used for testing the proposed Plug & Produce framework presented in this article. It was developed in close collaboration with GKN Aerospace, a company producing metal parts for the aeronautics industry.

IV. Experiments

In this section, the proposed framework for Plug & Produce is evaluated. An existing Plug & Produce demonstrator at University West, has been developed in close collaboration with industry, see Fig. 6. The simulation shown, contains three process modules. *Module 1* is an operator-assisted unload station for parts, *module 2* is an operator-assisted load station for parts, and *module 3* contains a motor that has an attached grinding tool. The demonstrator has ten slots (1–10) for process modules. To decrease the hardware time, identical connectors are used for all modules. Thus, it is possible to quickly connect modules on available slots, and with one single cable connect power, air and network. Each slot has a fixture with positioning pins that makes sure that modules are placed correctly. This avoids recalibration of positions, in order to reduce the software time.

The framework described above has been implemented and tested using this demonstrator. Indeed, it is possible to use the conventional agent framework JADE for implementing the agent communication needed for our algorithm. However, we have chosen the agent handling system (AHS) described in [17]. This AHS has been used in the implementation of our algorithm since it includes support for the OPC UA protocol, which is compatible with various industrial devices. OPC UA was developed by OPC foundation and is a platform-independent protocol for communication in industrial automation [40].

The goal for the Plug & Produce demonstrator in this work is to make soft edges on metal engine parts for the aeronautic sector. With the proposed Plug & Produce framework it should be easy to set up a local robot cell attached to an existing manufacturing flow. A local cell should be easy to set up when needed, e.g., to handle rush orders or variations in supply/demand. In the demonstrator, several process modules can be plugged in and out to quickly change the manufacturing setup.

TABLE I
Process Plan for a Goal g, Describing How to Make Soft Edges by Defining a Sequence of Skills S_u Using Variables V_u.

Process plan $\pi_g(RPM)$
Goal:
$g = SoftEdges$
Abstract Interfaces:
$U = \{a, b, c, d\}$
Sequence of Skills:
S_a.Load:
S_b.Transport: V_b.From=Part.GripLocation, V_b.To=V_c.StartPos
S_c.Grinding: V_c.Speed=RPM
S_b.MoveAlong: V_b.From=V_c.StartPos, V_b.To=Part.GrindPath
S_c.Grinding: V_c.Speed=0
S_b.Transport: V_b.From=Part.GripLocation, V_b.To=V_d.LeavePos
S_d.UnLoad:

Note that the Abstract Interfaces *a*, *b*, *c*, and *d* are Unmapped in the Process Plan and Will be Identified During Runtime by the search Algorithm.

A. A Scenario for Soft Edges

The robot cell considered contains the following resources: $R = \{\text{Robot, Gripper 1, Gripper 2, Motor, GrindingWheel, Load station, Unload station}\}$, see Fig. 7. In this scenario, the cost refers to the execution time of a process plan. One metal part p is introduced to the system and a corresponding agent is instantiated with the goal $g = \text{SoftEdges}$. Several process plans Π_g may be formulated for the specific goal g.

The process plan π_g in Table I solves the goal $g = \text{SoftEdges}$, using the abstract interfaces $U = \{a, b, c, d\}$. The plan describes the following sequence of skills.

1) The metal part p appears at the load station (in Fig. 7 referred to as Part).
2) The part is transported (using the skill Transport) to a grinding wheels StartPos using a gripper connected to the part on GripLocation.
3) The grinding wheel that is pre-mounted manually to a motor should start to rotate with the speed defined on RPM.
4) The robot moves the part against the grinding wheel based on the predesigned path GrindPath that is attached to the part agent.
5) The grinding wheel is stopped.
6) The part leaves the system by moving to the unload station position LeavePos.
7) The part is removed from the system and the agent is deleted.

B. Evaluating the Algorithm

This section describes each step in the algorithm, considering the process plan in Table I and the resources in Fig. 7. The

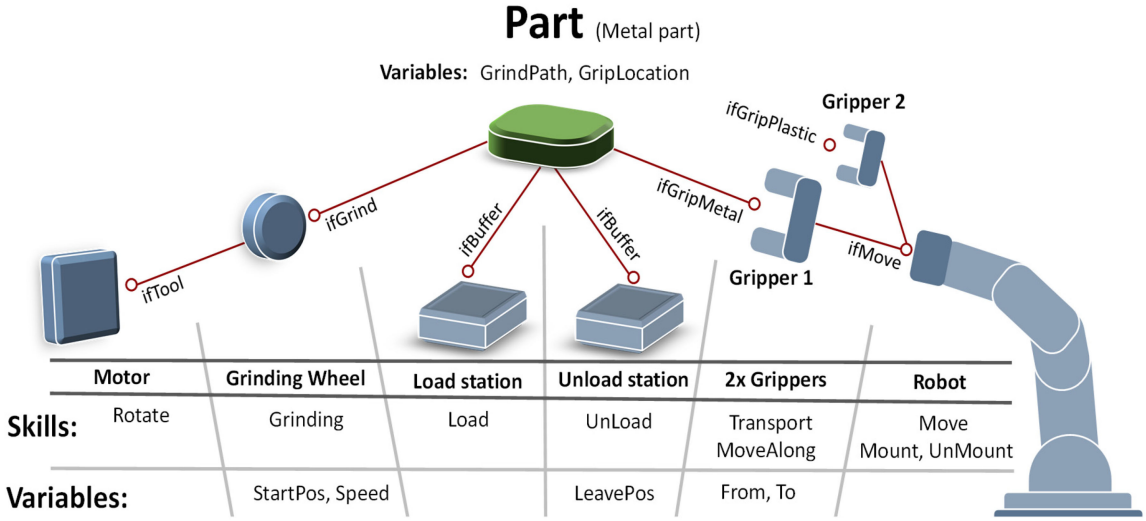

Fig. 7. Using interfaces to connect a part with resource skills. Available interface connections are illustrated by red lines with a circle marking the connection point.

algorithm in this example explains how the metal part agent will run this algorithm, i.e., this example describes the algorithm from the part perspective.

Step 1: The process plan π_g has four demands $d_u \in D$, based on the abstract interfaces $U = \{a, b, c, d\}$ in π_g. The demands consist of demanded skills S_u and variables V_u, forming the set D of all demands for U

$$S_a = \{\text{Load}\}$$
$$S_b = \{\text{Transport, MoveAlong}\}$$
$$S_c = \{\text{Grinding}\}$$
$$S_d = \{\text{Unload}\}$$
$$V_a = \{\ \}$$
$$V_b = \{\text{From, To}\}$$
$$V_c = \{\text{Speed, StartPos}\}$$
$$V_d = \{\text{LeavePos}\}$$
$$d_a = \{S_a, V_a\}$$
$$d_b = \{S_b, V_b\}$$
$$d_c = \{S_c, V_c\}$$
$$d_d = \{S_d, V_d\}$$
$$D = \{d_a, d_b, d_c, d_d\}\,.$$

Step 2: For each abstract interface $u \in U$, the related demand d_u is broadcasted to resource agents in the network that reply if they have an interface that meets the demand d_u. The local agent (in this case the part p) checks if any of the found interfaces are compatible with the local agents interfaces. If they are compatible, they are added to the potential interfaces I^p, in this case, representing a total of four interfaces on resources. Hence,

$$I^p_a = \{\text{if Buffer}\}$$
$$I^p_b = \{\text{if GripMetal}\}$$
$$I^p_c = \{\text{if Grind}\}$$
$$I^p_d = \{\text{if Buffer}\}$$
$$I^p = \{I^p_a, I^p_b, I^p_c, I^p_d\}\,.$$

For this specific case, there is only one element in each set I^p_u, however, more alternatives could be available if multiple compatible resources would be available for the same skill.

Step 3: From the potential interfaces I^p_u, select the interfaces with the lowest cost I^s, where

$$I^s = \{\min(I^p_a), \min(I^p_b), \min(I^p_c), \min(I^p_d)\}$$

and use the selected interfaces I^s to map the plan π_g to physical resources in the agent network. Save the mapped process plan as an executable process plan

$$\pi^e_g = \text{map}(\pi_g, I^s)\,.$$

The algorithm that was described turns process plans π_g into executable process plans π^e_g. Since several process plans $\pi_g \in \Pi_g$ for a goal g can exist, Π^e_g is formed, where $\pi^e_g \in \Pi^e_g$.

Since each element in Π_g is an alternative process plan that can reach the goal $g = \text{SoftEdges}$, the executable process plan $\pi^e_g \in \Pi^e_g$ with the lowest cost for reaching the goal is now chosen. This makes the plan ready to run since the abstract interfaces $U = (a, b, c, d)$ are now mapped to interfaces on physical resources in the agent network.

TABLE II
NUMBER OF ACTIVITIES THAT USES SOFTWARE TIME, COMPARED BETWEEN CASES 1, 2, 3, AND 4, WHEN USING THE PLUG & PRODUCE FRAMEWORK

Description	A1 (Goals)	A2 (Plans)	A3 (Interfaces)	A4 (Programming)
Case 1	1	1	11	4
Case 2	1	1	0	0
Case 3	0	1	1	1
Case 4	0	0	0	0

TABLE III
TIME COMPARED BETWEEN CASES 1, 2, 3, AND 4. THESE NUMBERS WERE FOUND DURING A CLOSE COLLABORATION WITH GKN AEROSPACE.

Description	A1 %	A2 %	A3 %	A4 %	Total %
Case 1	<1	1	57	41	100
Case 2	<1	1	0	0	2 *
Case 3	0	1	5	10	16 *
Case 4	0	0	0	0	0 *

*Note that Each Value in Cases 2, 3, and 4 are Percentages Out of the Total Time of Case 1.

V. EVALUATION

The main motivation for this article is to minimize software time spent on programming and configuration of Plug & Produce systems. From the presented Plug & Produce framework, four cases can be identified that highly affect the software time, cases 1–4. Four separate activities are observed that contribute to software time: ($A1$) preparing goals, ($A2$) creation of process plans, ($A3$) defining interfaces and ($A4$) programming, as given in Table II. The introduction of new parts relates to activity $A1$ and $A2$ while the preparation of resources relates to activities $A2$, $A3$, and $A4$. Activity $A4$, i.e., programming, regards to the time spent on adapting a resource to the Plug & Produce framework. In order to adapt a resource to our framework, some code must be written to make it compatible with the Plug & Produce framework.

Case 1—Creating a new robot cell: All new resources not previously prepared for the Plug & Produce framework have to be programmed ($A4$) and configured, i.e., creating interfaces with skills ($A3$) and plans for running those skills ($A2$). Hence, if all resources are new, there will be a considerable time spent adapting them to the Plug & Produce framework. However, this is a one-time effort. If new goals and plans are introduced, they will require time for creating goals ($A1$) and defining process plans ($A2$). The use of the proposed framework simplifies the programming (compared to traditional central control) since no dependencies or communication between resources have to be defined.

Case 2—Changing, modifying or replacing a part: In this case, the agent configuration must be updated to reflect this. It might also be necessary to change the physical configuration of the robot cell. For minor changes like adjusting how soft the soft edges should be or what paint to use when painting a part, only the goals ($A1$), plans ($A2$) and their related variables are modified. This case shows the main benefits of the presented Plug & Produce framework, since no programming or configuration has to be performed when changing goals or process plans. This can be compared with a traditional central control, where reprogramming commonly has to be performed.

Case 3—Adding a new resource: If a new resource is introduced, then it has to be adapted to the Plug & Produce framework by plans ($A2$), interfaces ($A3$) and programming ($A4$). The benefit of using the Plug & Produce framework is that the resource can be developed and tested offline without interrupting ongoing manufacturing. In the same way as case 1, the programming is simplified by letting the agent system manage all communication.

Case 4—Recycling of manufacturing systems: In an industry with needs for flexibility, a robot cell will not last forever. When rebuilding, moving, or recycling a robot cell, it is desirable to reuse the resources, corresponding programming and agent configurations. Reused resources can drastically decrease the deployment time. The Plug & Produce framework use a distributed approach for the controller of each agent. This makes it possible to configure one resource or part without considering any other objects in the system. Hence, an agent configuration can effortlessly be moved together with the resource to another robot cell. The agent configuration can be compared to a software driver for a USB device with plug and play functionality. Additionally, the code written inside the resources, like robot code and PLC code in the process modules, can be reused, since it has no dependencies with any other device in the system. Device code and agent configuration will only be modified if the resource should receive new functionalities, e.g., adding a new sensor or button.

Case comparison: In Table II, each activity that adds to the software time has been counted and sorted into the cases 1, 2, 3, and 4. These cases are taken from the presented scenario in Fig. 7 and assumes that the Plug & Produce framework is used. The scenario requires one goal, one plan, 11 interfaces and four resources. In the first case, all 11 Interfaces and four resources must be configured and programmed together with one goal and process plan defined. In the second case, a part is modified, needing a new goal and process plan to be defined. None of the interfaces needs to be changed, and no programming is required. In the third case, one new process module is configured and programmed, resulting in one interface and one program needed to be created, together with one process plan to be defined for solving its skills. In the fourth case, recycling is performed of one process module without using any software time. The number of activities needed for each case is given in Table II.

The time consumed on the various activities has been measured and confirmed during collaboration with GKN Aerospace. From this data, it was found that out of the total time consumed in case 1 (100%), a goal ($A1$) took less than 1%, a plan ($A2$) 1%, one interface ($A3$) ∼5%, and programming of one device ($A4$) ∼10%. This is shown in Table III.

The time (case 1) for the 11 interfaces was 57% and the time for programming 4 devices was 41%, as given in Table III. We can also see that case 2 uses only a total of 2% of the total time needed for cases 1 and 3 requires a total of 16% of the total time of case 1, while case 4 required no software time.

This result clearly shows that the presented Plug & Produce framework can decrease the software time compared to traditional approaches. In case 1, the programming is simplified by letting the agents solve all dependencies and communication between resources. Case 2 shows that the change of part goals and plans has a low influence on the software time. In case 3, a new process module was added similarly to case 1. In case 4, there was no software time needed. The reason is that resources can be integrated automatically if they were previously prepared for the Plug & Produce framework. The hardware time to physically install a module was, during the conducted experiments, found to be around one minute. This time is the same for any of the process modules, as long as they use the standardized hardware connectors mentioned above. This implies that if a framework that works as described in this article would be used as a standard, then a company could buy a resource that is delivered with configuration data much like a USB device for a computer that is delivered with a driver. Then there would be no time spent on the programming $(A4)$ or interface activities $(A3)$ of Table II, thus avoiding the activities with the highest software time when a new robot cell is created.

VI. Conclusion

In this article, a framework for Plug & Produce was formulated. This includes a new way of describing process plans with unmapped resources by formulating abstract interfaces. The mapping of resources to process plans was accomplished by generating demands for interfaces on other resources in the agent network. A mapping algorithm was described that can connect resources to form trees of collaborating agents. This algorithm runs on every agent in the system, making the search distributed. The algorithm was implemented and tested in a physical demonstrator, which verified that the proposed Plug & Produce framework works.

The main benefit of the proposed framework was that it makes it possible to add new types of products faster in terms of minutes rather than days in traditional approaches. It also encapsulates resources so that they have no dependencies between each other. This makes it much easier to develop resources and to move them between manufacturing systems, without adapting them to specific new scenarios.

References

[1] S. Hu *et al.*, "Assembly system design and operations for product variety," *CIRP Ann. Manuf. Technol.*, vol. 60, no. 2, pp. 715–733, 2011.

[2] M. Onori, N. Lohse, J. Barata and C. Hanisch, "The IDEAS project: Plug & produce at shop – floor level," *Assem. Automat.*, vol. 32, no. 2, pp. 124–134, 2012.

[3] Z. Pan, J. Polden, N. Larkin, S. V. Duin and J. Norrish, "Recent progress on programming methods for industrial robots," *Robot. Comput.-Integr. Manuf.*, vol. 28, no. 2, pp. 87–94, 2012.

[4] T. Blecker and G. Friedrich, *Mass Customization: Challenges and Solutions*. New York, NY,USA: Springer, 2006.

[5] M. R. Pedersen *et al.*, "Robot skills for manufacturing: From concept to industrial deployment," *Robot. Comput.-Integr. Manuf.*, vol. 37, pp. 282–291, 2016.

[6] G. Lanza, J. Stoll, N. Stricker, S. Peters, and C. Lorenz, "Measuring global production effectiveness," in *Proc. 46th CIRP Conf. Manuf. Syst.*, 2013, pp. 31–36.

[7] H. Elmaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *Int. J. Flexible Manuf. Syst*, vol. 17, no. 4, pp. 261–276, 2005.

[8] P. Coletti and T. Aichner, *Mass Customization: An Exploration of European Characteristics*. New York, NY, USA: Springer, 2011.

[9] Y. Koren *et al.*, "Reconfigurable manufacturing systems," *CIRP Ann.*, vol. 48, no. 2, pp. 527–540, Aug. 1999.

[10] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota, "Agile assembly system by plug and produce," *CIRP Ann. Manuf. Technol*, vol. 49, no. 1, pp. 1–4, 2000.

[11] L. Ribeiro, J. Barata, M. Onori and J. Hoos, "Industrial agents for the fast deployment of evolvable assembly systems," in *Industrial Agents*, Amsterdam, The Netherlands: Elsevier, 2015, pp. 301–322.

[12] A. Zoitl, G. Kainz and N. Keddis, "Production plan-driven flexible assembly automation architecture," in *Industrial Applications of Holonic and Multi-Agent Systems*, New York, NY, USA: Springer, 2013, pp. 49–58.

[13] M. Hvilshøj and S. Bøgh, ""Little helper" - An autonomous industrial mobile manipulator concept," *Int. J. Adv. Robot. Syst*, vol. 8, no. 2, pp. 1–11, 2011.

[14] C. Schou and O. Madsen, "A plug and produce framework for industrial collaborative robots," *Int. J. Adv. Robot. syst.*, vol. 14, no. 4, pp. 1–10, 2017.

[15] M. Skilton and F. Hovsepian, *The 4th Industrial Revolution - Responding to the Impact of Artificial Intelligence on Business*, Cham, Switzerland: Palgrave Macmillan, 2018.

[16] B. Svensson and F. Danielsson, "P-SOP – A multi-agent based control approach for flexible and robust manufacturing," *Robot. Comput. Integr. Manuf.*, vol. 36, pp. 109–118, 2015.

[17] M. Bennulf, F. Danielsson and B. Svensson, "Identification of resources and parts in a plug and produce system," in *Proc. 29th Int. Conf. Flexible Automat. Intell. Manuf.*, 2019, pp. 858–865.

[18] V. Mařík and J. Lažanský, "Industrial applications of agent technologies," *Control Eng. Pract.*, vol. 15, no. 11, pp. 1364–1380, Nov. 2007.

[19] N. K. C. Krothapalli and A. V. Deshmukh, "Design of negotiation protocols for multi-agent manufacturing systems," *Int. J. Prod. Res.*, vol. 37, no. 7, pp. 1601–1624, 1999.

[20] P. Leitão, V. Mařík and P. Vrba, "Past, present, and future of industrial agent applications," *IEEE Trans. Ind. Informat*, vol. 9, no. 4, pp. 2360–2372, Nov. 2013.

[21] P. Leitão and S. Karnouskos, "A survey on factors that impact industrial agent acceptance," in *Industrial Agents*, Amsterdam, The Netherlands: Elsevier, 2015, pp. 401–429.

[22] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Eng. Appl. Artif. intell.*, vol. 22, no. 7, pp. 979–991, Oct. 2009.

[23] W. Shen, Q. Hao, H. J. Yoon and D. H. Norrie, "Applications of agent-based systems in intelligent manufacturing: An updated review," *Adv. Eng. Informat.*, vol. 20, no. 4, pp. 415–431, Oct. 2006.

[24] S. Karnouskos and P. Leitão, "Key contributing factors to the acceptance of agents in industrial environments," *IEEE Trans. Ind. Informat*, vol. 13, no. 2, pp. 696–703, Apr. 2017.

[25] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995.

[26] C. Carabelea, O. Boissier and F. Ramparany, "Benefits and requirements of using multi-agent systems on smart devices," *in Proc. Euro-Par Parallel Process.*, 2003, pp. 1091–1098.

[27] L. Steels, "When are robots intelligent autonomous agents?," *Robot. Auton. Syst.*, vol. 15, no. 1/2, pp. 3–9, 1995.

[28] "FIPA 97 Part 1 Version 1.0: Agent management specification," Found. Intell. Phys. Agents, Geneva, Switzerland, 1997.

[29] S. Poslad, "Specifying protocols for multi-agent systems interaction," *ACM Trans. Auton. Adaptive Syst.*, vol. 2, no. 4, pp. 15–24, 2007.

[30] "FIPA agent management specification," Found. Intell. Phys. Agents, Geneva, Switzerland, 2002.

[31] "FIPA ACL message structure specification," Found. Intell. Phys. Agents, Geneva, Switzerland, 2002.

[32] F. Bellifemine, G. Caire and D. Greenwood, *Developing Multi-Agent Systems With JADE*, New York, NY, USA: Wiley, 2007.

[33] E. Järvenpää, M. Lanz and R. Tuokko, "Application of a capability-based adaptation methodology to a small-size production system," *Int. J. Manuf. Technol. Manage.*, vol. 30, no. 1/2, pp. 67–86, Apr. 2016.

[34] N. Antzoulatos, E. Castro, L. d. Silva, A. D. Rocha, S. Ratchev and J. Barata, "A multi-agent framework for capability-based reconfiguration of industrial assembly systems," *Int. J. Prod. Res.*, vol. 55, no. 10, pp. 2950–2960, 2017.
[35] J. W. Park, M. Shin and D. Y. Kim, "An extended agent communication framework for rapid reconfiguration of distributed manufacturing systems," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 3845–3855, 2019.
[36] S. Rehberger, L. Spreiter and B. Vogel-Heuser, "An agent-based approach for dependable planning of production sequences in automated production systems," *Automatisierungstechnik*, vol. 65, no. 11, pp. 766–778, 2017.
[37] R. S. Sutton, D. Precup and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artif. Intell.*, vol. 112, no. 1-2, pp. 181–211, 1999.
[38] M. Vallée, M. Merdan, W. Lepuschitz and G. Koppensteiner, "Decentralized reconfiguration of a flexible transportation system," *IEEE Trans. Ind. Informat.*, vol. 7, no. 3, pp. 505–516, Aug. 2011.
[39] D. Smale and S. Ratchev, "A capability model and taxonomy for multiple assembly system reconfigurations," *IFAC Symp. Inf. Control Problems Manuf.*, vol. 42, no. 4, pp. 1923–1928, Jun. 2009.
[40] W. Mahnke, S.-H. Leitner and M. Damm, *OPC Unified Architecture*, New York, NY, USA: Springer, 2009.

Mattias Bennulf was born in Levene, Vara, Sweden in 1992. He received the B.S. degree in computer engineering and M.S. degree in robotics and automation from University West, Trollhättan, Sweden, in 2014 and 2015, where he is working toward the Ph.D. degree in production technology with a focus on multiagent technology used in manufacturing systems.

Fredrik Danielsson was born at Orust, Sweden, in 1972. He received the Ph.D. degree in mechatronics from De Montfort University, Leicester, U.K., in 2002.

From 2003 to 2015, He was the Head of the Robot and Automation education at advanced level with University West. Since 2008, he has been the Head of the Flexible Automation Research Group at the Department of Engineering Science, University West. He has co-authored of more than 70 peer reviewed papers in international journals and conferences. His current main research interests include flexible automation, multi-agent control systems, virtual commissioning, AI and robot systems.

Bo Svensson was born in Mariestad, Sweden, in 1959. He received the M.S. degree in electrical engineering in 1984 and the Ph.D. degree in automation in 2012 from Chalmers University of Technology, Gothenburg, Sweden.

He was a Design Engineer with SAAB Space AB from 1984 to 1987. From 1987 to 1994, he was a System Engineer with SAAB Automobile AB. Since 1994, he has been a Senior Lecturer with the Department of Engineering Science, University West, Trollhättan, Sweden, with teaching and research. His current research interest include flexible industrial automation, plug and produce, multiagent system control, human-robot collaboration, safety, and simulation-based optimization.

Bengt Lennartson (Fellow, IEEE) was born in Gnosjö, Sweden, in 1956. He received the Ph.D. degree in automatic control from Chalmers University of Technology, Gothenburg, Sweden, in 1986.

Since 1999, he has been a Professor of the Chair of Automation, Department of Electrical Engineering. From 2004 to 2007, he was the Dean of Education with Chalmers University of Technology. Since 2005, he has been a Guest Professor with University West, Trollhättan, Sweden. He contributes to hybrid and discrete event systems for automation and sustainable production. He has co-authored of two books and more than 300 peer reviewed papers in international journals and conferences. His current research interest include AI planning and learning, discrete event and hybrid systems, and robust feedback control.

Prof. Lennartson was the General Chair of the 11th IEEE Conference on Automation Science and Engineering (2015), and the 9th International Workshop on Discrete Event Systems (2008) and an Associate Editor for *Automatica* (2002–2005) and IEEE TRANSACTION ON AUTOMATION SCIENCE AND ENGINEERING (2012–2015).