PhD Thesis
Production Technology
2023 No. 54

HÖGSKOLAN VÄST

A Control Framework for Industrial Plug & Produce

Mattias Bennulf

Tryck: Stema Specialtryck AB, Januari 2023

SVANENMÄRKET
Trycksak
3041 0234

PhD Thesis
Production Technology
2023 No. 54

HÖGSKOLAN VÄST

A Control Framework for Industrial Plug & Produce

Mattias Bennulf

University West
SE-46186 Trollhättan
Sweden
+46 52022 30 00
www.hv.se

ISBN 978-91-89325-42-5 (Printed version)
ISBN 978-91-89325-41-8 (Electronic version)

Acknowledgements

The work presented in this thesis was mainly carried out at University West in Trollhättan, Sweden. It was a part of the project: Miljo:FIA (Miljö för Flexibel och Innovativ Automation), project reference: 20201192, funded under: Europeiska regionala utvecklingsfonden/VGR. The work was also part of the project: PoPCoRN (Plug & Produce with Configurable Resources inspired by Natural language), funded by the K-K foundation, reference number: Dnr 20200036, Sweden.

This thesis is to a considerable extent based on my previously published licentiate thesis: *A User-Friendly Approach for Applying Multi-Agent Technology in Plug & Produce Systems,* that was covering the first half of my PhD studies.

I thank my supervisors Fredrik Danielsson, Bo Svensson and Bengt Lennartson for supporting me and guiding me through these years.

Mattias Bennulf

Trollhättan, January 2023

Populärvetenskaplig Sammanfattning

Titel: Ett ramverk för styrning av industriell Plug & Produce

Nyckelord: Flexibel tillverkning, Ontologi, Multiagentsystem, Automation, Planering

Kundanpassade produkter och korta produktionsserier blir alltmer populärt. Detta har lett till problem för dedikerade tillverkningssystem som är designade för massproduktion. Det krävs ofta långa produktionsserier för att det ska bli en rimlig investering att ställa om produktionen. Därför används människor för tillverkningsuppgifter som ofta ställs om. Denna avhandling fokuserar på konceptet Plug & Produce, som gör det enklare att flytta, lägga till och ta bort resurser från ett tillverkningssystem. Tanken är att resurser placeras i processmoduler som alla har samma fysiska gränssnitt för att kopplas in i tillverkningssystemet. Styrningen av tillverkningssystemet görs av ett multiagentsystem där varje detalj som ska produceras för produkter får en egen agent som representerar detaljen och agerar som styrningsmjukvara. Varje detaljs agent tar hand on sina egna tillverkningsmål genom att kommunicera med resursagenter i systemet som används för styrning av resurserna. I detta arbete, presenteras ett ramverk för Plug & Produce som består av ett konfigurerbart multiagentsystem, samt ett konfigurationsverktyg som kan användas för att definiera agenterna. Arbetet inkluderar metoder för att identifiera inkopplade resurser, kommunikation mellan agenter, schemaläggning som kan undvika konflikter mellan agenter, samt metoder för att automatiskt hitta vägar för transport genom tillverkningssystemet.

Abstract

Title: A Control Framework for Industrial Plug & Produce

Keywords: Flexible Manufacturing, Ontology, Multi-agent system, Automation, Planning

ISBN: 978-91-89325-42-5 (Printed)
978-91-89325-41-8 (Electronic)

Customized products and low-volume production are becoming more popular resulting in a problem for dedicated manufacturing systems that are designed for mass production. Adapting a system to new demands is expensive and requires many products to be produced before it becomes a reasonable investment. This has forced factories to use human workers for manufacturing tasks that often change. This thesis focuses on a concept called Plug & Produce, which makes it easier to move, add, and remove resources in manufacturing systems. This is done by containing resources in process modules that all have the same physical connectors. To handle the control of the manufacturing system a multi-agent system is considered where each part to be produced for products has a part agent software running that represents that part. Each part agent takes care of their own manufacturing goals by communicating with resource agents that control the resources in the system. In this thesis, a Plug & Produce framework is described that consists of a configurable multi-agent system, together with a configuration tool for defining agent behaviours. Methods for identifying the resource that has been connected to a Plug & Produce system are investigated. Communication between agents in Plug & Produce is investigated. Scheduling is described for the presented systems to avoid conflicts when running multiple agents. Also, a pathfinding method for Plug & Produce is presented, which automatically gathers the necessary information for finding paths to transport parts through the manufacturing system.

Acronyms

DM	**Dedicated Manufacturing** Static systems, designed for specific products
FMS	**Flexible Manufacturing Systems** A system that can react to changes
RMS	**Reconfigurable Manufacturing Systems** A system that can manage rapid changes in hardware and software
CNC	**Computer Numerical Control** Automated control of machines
HMI	**Human Machine Interfaces** An interface for interaction between a human and a machine
PLC	**Programmable Logic Controller** A programmable computer for controlling automation systems
C-MAS	**Configurable Multi-Agent System** A control framework for Plug & Produce
ST	**Structured Text** A programming language
AHS	**Agent Handling System** A system to manage agents in the manufacturing
OPC UA	**OPC Unified Architecture** A communication protocol
BDI	**Belief-Desire-Intention** A software model for intelligent agents
FIPA	**Foundation for Intelligent Physical Agents** Is an organization that promotes agent-based technology
AP	**Agent Platform** Provides the physical infrastructure to deploy agents in systems following FIPA standards
AMS	**Agent Management System** Keeps a record of agents and their identifiers in a system that follows FIPA standards
MTS	**Message Transport Service** As defined by FIPA this can be used for communication between agents

DF	**Directory Facilitator** FIPA has a Directory Facilitator that acts as a "Yellow Pages" service for agents. Agents in FIPA can be connected to this service
SPADE	**Smart Python Agent Development Environment** A multi-agent framework
Cougaar	**Cognitive Agent Architecture** A multi-agent framework
JADE	**Java Agent Development Framework** A multi-agent framework
KQML	**Knowledge Query Manipulation Language** A multi-agent framework
ACL	**Agent Communication Language** A standard language for agent communication defined by FIPA
CAL	**Communicative Act Library** Library with communicative acts defined by FIPA
RRT	**Rapidly exploring Random Tree** An algorithm for automatically generating collision-free paths
CAD	**Computer-Aided Design** The use of computer software for design
OMPL	**Open Motion Planning Library** A library with many algorithms for motion planning
ROS	**Robot Operating System** A set of software libraries and tools for robot applications
DHCP	**Dynamic Host Configuration Protocol** A protocol for automatically assigning IP addresses
IP address	**Internet Protocol address** A numerical label for connected devices in a network
ER model	**Entity-Relationship model** Model to describe interrelated things

Nomenclature

$a \in A$	One agent in the set of all agents
$p \in P$	One part agent in the set of all part agents
$r \in R$	One resource agent in the set of all resource agents
$A = R \cup P$	There are resource agents and part agents
$g \in G_p$	One goal in the set of goals for one part p
$v \in V_p$	One variable in the set of variables for one part p
$v \in V_r$	One variable in the set of variables for one resource r
$if \in IF_p$	One interface in the set of interfaces for one part p
$v \in V_g$	One variable in the set of variables for one goal g
$\pi_g \in \Pi_g$	One process plan in the set of process plans for one goal g
$\pi_g^e \in \Pi_g^e$	One executable process plan in the set of executable process plans for one goal g
$\pi_s^e \in \Pi_s^e$	One executable process plan in the set of executable process plans for one skill s
pre_g	Set of all goals that must be fulfilled before this goal g can be assigned
$s \in S$	One skill in the set of all skills for the whole system
π_s	A process plan for one skill s
$if \in IF$	One interface in the set of all interfaces
$s \in S_{if}$	One skill in the set of all skills on interface if
$v \in V_{if}$	One variable in the set of all variables on one interface if
$u \in U$	One abstract interface in the set of all abstract interfaces

d_u	One demand
G	All goals for all part agents in the system
q	A state in a process plan
G^{pa}	Set of goal sequences running in parallel
G^{sq}	A sequence of goals
g^h	A goal to be scheduled or that already is scheduled
s^h	A skill to be scheduled or that already is scheduled
$if^{dp} \in IF_r^{dp}$	One interface dependency in the set of interface dependencies on r
SC_r	Local schedule on r
n	A name, such as the name of a skill
st_s	A state that the interface having s should be in
if_s^{lo}	Local interface
if_s^{re}	Remote interface
S_s^h	Other skills needed to run s
γ	A local graph for one part
IF_γ	Set of interfaces used as nodes in one graph
$\tau \in \mathrm{T}_\gamma$	One transfer in the set of transfers between nodes in the graph γ
if_{pre}	Interface before transfer
if_{post}	Interface after transfer
c_s	Cost of running skill s
if_s	Address to the interface of skill s

List of Figures

Figure 1. A Plug & Produce system located in the research laboratory at University West in Trollhättan, Sweden. 8
Figure 2: Plug & Produce system considered in this thesis, where the colours represent the different types of modules. 9
Figure 3: An agent sensing and reacting to its environment. 10
Figure 4: When multiple agents connect in a network, they form a multi-agent system. 11
Figure 5. This figure shows that the computational time, due to planning in a manufacturing system is a function of computational complexity and online flexibility. The highest computation time is type 4 and the lowest is type 2, noted with Max and Min. 15
Figure 6: Top view of a Plug & Produce robot cell, containing one industrial robot in the centre, surrounded by ten process modules. The robot should move between the start position $p1$ and target position $p2$. 17
Figure 7: Dijkstra's algorithm, where the blue marked line is the shortest path between the start and target positions. 18
Figure 8: Industrial robots' reachability example for checking if they can reach the point $p1$ and $p2$ marked in the figure. 19
Figure 9. Class diagram showing the ontology used when configuring one agent. 21
Figure 10. Flowchart showing a simplified part agent strategy where the detailed steps are hidden. 24
Figure 11. Flowchart showing the strategy of a resource agent, that receives messages and manages them based on communicative acts. 25
Figure 12: Process plan with five skills, where the initial state is $q0$ and final state is qf 26
Figure 13: A plan πg is checked for availability in the agent network 27
Figure 14. Example of one part interacting with three resources. 28
Figure 15: Entity-Relationship model (ER model), showing a simplified view of the ontology used for configuring the agents. For further simplicity, the attributes are not included. 29
Figure 16: A combined model, showing a partial view of the internal ontology of three agents $p, r1,$ and $r2$. In this example, the part agent p is requesting a resource $r1$ to run a skill, and that resource is requesting a second resource $r2$ to run an additional skill. 30
Figure 17. A part agent, placed on a buffer, requests a transporter to move the part from the buffer. This requires the part to first ask the buffer for the world coordinate that the part is placed on. 31
Figure 18: Conceptual model for agent communication. 32

Figure 19: The data hub for synchronizing data between agents and their related objects in the world. 33
Figure 20: This figure shows the Agent Handling System connected to two devices on the right side that are controlled by the two agents on the left side. 34
Figure 21: Method for registering new devices added to the Plug & Produce system. 35
Figure 22. This illustrates an example of goal sequences Gsq running in parallel Gpa, with goals gh having skills sh to be scheduled. Each Gpa must wait for each arrow pointing to it to be done for Gpa to start executing any of its skills. 37
Figure 23: Scenario for transporting a part p from buffer $r1$ to a painting station $r2$, using gripper $r3$, and robot $r4$. 39
Figure 24: Variables for locations 1-4. 41
Figure 25: Interface connections are shown with lines for interfaces 1-7. 41
Figure 26: Example of two process modules: one with a part placed on top of it, and another with a robot that can interact with its neighbouring modules 42
Figure 27: Example, with part and gripper having compatible interfaces since signals and skills are matching. 43
Figure 28: Example, with a part and paint station having compatible interfaces. 43
Figure 29: Gripper and robot connecting through an interface. 43
Figure 30: Process plan πg for solving the goal *PaintBlue*. 44
Figure 31: A process plan $\pi s3$that is written specifically to execute the skill *Transport* for the gripper. 44
Figure 32: A plan πg is checked for availability in the agent network 45
Figure 33: Main view, showing five agents and one process plan. 46
Figure 34: Agent view for the part, showing variables, interfaces and goals 47
Figure 35: Interface view for the BufferInterface on the Paint station 48

List of Tables

Table 1. FIPA ACL message acts in FIPA 2002. 13
Table 2. Communicative acts in FIPA 2002... 14
Table 3. Configuration values for the scenario. 40
Table 4. Specialized communicative acts. 49

Appended Publications

The following publications are appended to this thesis:

Paper A. Goal-Oriented Process Plans in a Multiagent System for Plug & Produce
Published in scientific journal – IEEE Transactions on Industrial Informatics – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson, Bengt Lennartson

Author's contributions: Principal author and partially idea initiator. Implemented and tested proposed algorithms. Carried out experiments and analysed results.

Paper B. Identification of resources and parts in a Plug and Produce system using OPC UA
Presented at conference – 29th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM 2019, in Limerick, Ireland, June 2019 – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson

Author's contributions: Principal author and idea initiator. Implemented and tested proposed methods. Carried out experiments and analysed results.

Paper C. A conceptual model for multi-agent communication applied on a Plug & Produce system
Presented at conference – 53rd CIRP Conference on Manufacturing Systems 2020, CIRP CMS 2020, in Chicago, IL, U.S., July 2020 – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson

Author's contributions: Principal author and idea initiator. Implemented and tested proposed models.

Paper D. A Method for Configuring Agents in Plug & Produce Systems
Presented at conference – The 10th Swedish Production Symposium, SPS2022, in Skövde, Sweden, April 2022 – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson

Author's contributions: Principal author and idea initiator. Implemented and tested proposed systems. Carried out experiments and analysed results.

Paper E. Part Oriented Planning for Unpredictable Events in Plug & Produce
Submitted to a scientific journal – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson

Author's contributions: Principal author and idea initiator. Designed and evaluated proposed systems.

Paper F. Online Generation of Graphs used for Pathfinding in Plug & Produce Systems
Submitted to a scientific journal – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson

Author's contributions: Principal author and idea initiator. Designed and evaluated proposed systems.

Other publications by the author:

1. Safety System for Industrial Robots to Support Collaboration

   International Conference on Applied Human Factors and Ergonomics, AHFE, in Orlando, Florida, USA, July 2016 – Authors: Gunnar Bolmsjö, Mattias Bennulf, Xiaoxiao Zhang

2. Verification and deployment of automatically generated robot programs used in prefabrication of house walls

   CIRP Conference on Manufacturing Systems, CIRP CMS, in Stockholm, Sweden, June 2018 – Authors: Mattias Bennulf, Fredrik Danielsson, Bo Svensson

3. Automated Path Planning for Plug & Produce in a Cutting-tool Changing Application

   IEEE Conference on Emerging Technologies and Factory Automation, ETFA, in Zaragoza, Spain, September 2019 – Authors: Sudha Ramasamy, Xiaoxiao Zhang, Mattias Bennulf, Fredrik Danielsson

4. A Classification of Different Levels of Flexibility in an Automated Manufacturing System and Needed Competence

   Changeable, Agile, Reconfigurable and Virtual Production Conference, CARV2021, in Aalborg, Denmark, October/November 2021 – Authors: Anders Nilsson, Fredrik Danielsson, Mattias Bennulf, Bo Svensson

Table of Contents

Acknowledgements iii
Populärvetenskaplig Sammanfattning v
Abstract vi
Acronyms vii
Nomenclature ix
List of Figures xi
List of Tables xiii
Appended Publications xiv

1 Introduction 1
1.1 Background 1
1.2 Research Questions 4
1.3 Research Methodology 5
1.4 Contributions 6
1.5 Scope and Limitation 6

2 Preliminaries 7
2.1 Plug & Produce 7
2.1.1 Implementation 8
2.2 Multi-Agent Systems 9
2.2.1 Multi-Agent Frameworks 11
2.2.2 Agent Communication 12
2.3 Automated Planning and Scheduling 14
2.3.1 Computation Time 15
2.3.2 Path Planning 16
2.3.3 Pathfinding 17
2.3.1 Scheduling 19

3 Proposed Plug & Produce Framework 21
3.1 Agent Definition 21
3.2 Agent Types 23

3.2.1 Part Agent Strategy 23
3.2.2 Resource Agent Strategy 24
3.3 Agent Interfaces 25
3.4 Process Plans 26
3.4.1 Example 1 27
3.5 Configuring the System 28
3.5.1 Example 2 30
3.6 Translation of Locations 31
3.6.1 Example 3 31
3.7 Agent Communication 31
3.8 Agent Handling System 33
3.9 Configuration Tool 35
3.10 Agent Planning 36
3.10.1 Scheduling of Process Plans 36
3.10.2 Pathfinding for Part Transfer 38

4 Evaluation 39

4.1 Manufacturing Scenario 39
4.2 Agent Interfaces 41
4.3 Process Plans 44
4.4 Configuration Tool 46
4.5 Agent Communication 49

5 Conclusions 51

6 Summary of Appended Papers 53

7 References 55

Appended Publications

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

Paper B. Identification of resources and parts in a Plug and Produce system using OPC UA

Paper C. A conceptual model for multi-agent communication applied on a Plug & Produce system

Paper D. A Method for Configuring Agents in Plug & Produce Systems

Paper E. Part Oriented Planning for Unpredictable Events in Plug & Produce

Paper F. Online Generation of Graphs used for Pathfinding in Plug & Produce Systems

1 Introduction

This chapter introduces the thesis by giving the background information followed by the research question, methodology, contribution, and scope.

1.1 Background

When dedicated manufacturing (DM) was introduced to the industry more than one hundred years ago, it replaced manual workshops. To optimize product flows, products were standardized. This vastly reduced production costs and increased production volume. The drawback was that it became expensive to change the product design. This became a problem since new models of various products are constantly developed to compete with their competitors. This results in a continuously increasing demand for customized production and low-volume production. For example, some products are expected to have new models enter the market frequently. This puts high requirements on the factories that manufacture these products. Today it is difficult and expensive to adapt manufacturing systems to new product designs [1], [2]. Both the changeover time and the related cost for personnel involved in the change are high. This makes it difficult today to automate customized, and low-volume production. Because of that, many factories still use human workers since they are flexible. The result is that factories move to countries with lower wages. A trend that has emerged in research is to develop flexible systems that adapt to new product designs with minimal effort [3]. However, most flexible systems use a static customization framework, where customers can choose a combination of static options [4]. The product is customized and unique, but the options are not unique, only the combination. This makes it expensive to adapt the manufacturing to new models if they involve new options. The reason is that there is still a lack of flexible and reconfigurable manufacturing systems that can easily add new options for a customer if it involves significant product modifications [5].

Reconfigurability and flexibility have been the focus of research for many years to make manufacturing systems better at adapting fast to new product types [6]. Flexible Manufacturing Systems (FMS) was first introduced in the 1980s [7]. Later in the 1990s, Reconfigurable Manufacturing Systems (RMS) was introduced [8]. Both FMS and RMS aim at being able to handle products with short life cycles. Today, automation in industries typically includes several resources, such as Computer Numerical Control (CNC), industrial robots, sensors, transportation devices and Human Machine Interfaces (HMI). These are typically connected via a central controller such as a Programmable Logic Controller (PLC). Adjusting such systems to manage new product types commonly includes reprogramming and physical changes. Thus, each new product or modified product design, introduced to the system, generates costs related to reprogramming and physical changes. The physical flexibility can be solved by standardizing process modules that are connected to the system similarly to Plug & Play on computers. Ribeiro da Silva e al. [2] writes about a similar concept with a robot resource contained in a module that can be shared by moving it between stations. S. Hjorth et al. [9] describe a concept where modules are moved around and placed in locations that are marked areas defined on Plug & Produced enabled stations. Making resources integrated into the ongoing production requires the system to understand when and how to use the new process modules. This typically requires extensive reprogramming in PLCs that can take several months. The reason is that when making changes to these systems, personnel are forced to understand most of the code in the manufacturing system, due to strong dependencies between the logic controlling each resource and product part [10]. This makes it almost impossible today to automate customized production and low-volume production, even when the physical system is standardized and flexible [11]. Thus, it is important to develop new control strategies that can handle various products and resources with a low amount of reprogramming [12]. One measurement of this is to consider the software time needed for adapting a manufacturing system for new products and resources. The time for preparing a system for new products and resources can be divided into hardware installation time (hardware time) and the time spent on programming and configuring the system (software time). The hardware time can be solved by using standardly sized process modules and connectors, which are described in other work as increasing the mechatronic compatibility [13], [14]. Modular approaches for manufacturing systems have been implemented in other works such as [15]. To decrease the software time, it is important to look at limiting the time consumed on configuring the system and time for programming resources [16].

One approach for managing flexible systems is to divide the controller into agents representing parts and agents representing resources. The agents can be placed physically on the object it represents, but they can also be placed anywhere remotely, such as in a cloud service. Agents that communicate with each other form a multi-agent system. Such systems are not new, and an example is the one that Krothapalli et al. [17] presented in the year 1999. Using a distributed approach removes the need for a central controller such as a PLC used in traditional automation. The resource agents present services called skills, that are used to reach the manufacturing goals, defined for the parts. In this way, no single agent has a central role in the system. Hence, it is possible to design agents without knowing the code of other agents, only the interfaces need to be known, for example: what skills and variables another agent has. Manufacturing goals can also be defined to run in parallel to make manufacturing more efficient, as described by Nilsson et al. [18].

Today, examples of multi-agent systems running in the industry are not common [19], [20]. The main reason seems to be that there are no simple configuration tools for the manufacturing industry, which hides the complexity of the agent technology [19], [21]. According to Pulikottil et al. [22] there is a lack of scheduling methodologies, standard architectures, and frameworks that targets general manufacturing environments. A configuration tool for Plug & Produce should provide the functionality to prepare a manufacturing system for production by defining its behaviour on a high level. This will replace the traditional rewriting of programming code for each new situation that arises in the manufacturing system. Also, automating functions such as path planning, pathfinding, and conflict avoidance can further simplify the adaption of manufacturing systems to new product designs. Then the system would be able to act autonomously, taking new orders without human intervention. It is also important to design a Plug & Produce framework that can be used with already existing resources in the industry [23], [24]. This will make the transition to these new systems smoother.

In this thesis, various methods are presented for designing a framework for Plug & Produce that aims at limiting the software time. The overall approach has been to work with a concept called Configurable Multi-Agent System (C-MAS) for designing the framework. The presented Plug & Produce framework is used to design Plug & Produce systems and to adapt them for changes such as new product designs and changes in the demanded production volume. Each agent can be configured by using a configuration tool, to let it know what it represents. This includes specifying what goals, skills, interfaces, and variables it has. In this work, skills and process plans for reaching goals can be defined with structured text (ST) code, based on the standard IEC 61131-3. The agents also contain multiple functions for planning, which makes the coding in each agent and related object simpler since planning is completely standardized and reused. Resources' internal logic becomes isolated by the nature of multi-agent systems, making them loosely coupled and easy to add, remove and move. This clear separation between resources simplifies the work of designing them. In C-MAS, agents always interact through clearly defined interfaces that must be compatible to connect and collaborate. Once the interfaces are defined and local behaviours created the communication and collaboration among agents are automatically solved by algorithms in the Plug & Produce framework. By using the methods presented in this thesis, manufacturing system resources can be installed and removed in terms of minutes, rather than days or weeks in traditional approaches. Changes in product design will many times require no programming or changes to the manufacturing system. Instead, a new process plan is simply designed and deployed to the system.

1.2 Research Questions

To create the Plug & Produce framework described earlier in this introduction, three research questions have been formulated. The first research questions (RQ1) consider how multi-agent systems can be designed to quickly adapt to new products and resources introduced to the system. The second research question (RQ2) asks how agents can be designed to reduce manual reprogramming. The aim is that the system should instead be configured on a high level, where descriptions are simplified, thus easier to understand and faster to modify. The third research question (RQ3) looks at online dynamic planning and scheduling that will increase the systems' online flexibility, adaptable to new situations and handling unpredictable events. It also decreases the software development time, since these functions can be used for any configuration, thus reducing the need for implementing this for each agent.

The research questions are:

RQ1. How can a multi-agent system be designed, to decrease the software development time in a Plug & Produce system?

RQ2. When introducing new products and resources, how can functionality for agent collaboration and reasoning be reused to decrease reprogramming time?

RQ3. How can dynamic planning and scheduling in configurable multi-agent systems be designed for Plug & Produce, which can handle unpredictable events?

1.3 Research Methodology

In this section, the method for each research question is listed with explanations about each step needed to reach the contributions of this paper.

Research question 1 (RQ1) is solved by contribution C1 which was achieved by the following steps:

- Investigate current approaches for Plug & Produce.
- Explore existing multi-agent systems.
- Identify what is missing in those approaches.
- Give design suggestions.

Research question 2 (RQ2) is solved by contributions C2 and C3 that were achieved by the following steps:

- Design a configurable multi-agent system.
- Present a conceptual model showing how multi-agent system communication for Plug & Produce could be designed.
- Show how physical devices can be connected to corresponding agents.
- Build and evaluate the designed system.

Research question 3 (RQ3) is solved by contribution C4 which was achieved by the following steps:

- Explore existing planning approaches for multi-agent systems.
- Present methods showing how multi-agent system online dynamic planning and scheduling for Plug & Produce could be designed.

1.4 Contributions

The contributions are:

C1. Give design suggestions for Plug & Produce that can help to decrease the adaption time for preparing a system for new products and resources. This is presented in the appended Paper A.

C2. Develop and evaluate a reconfigurable Plug & Produce system based on multi-agent technology and show how physical devices are connected to the agents and their configuration. This is presented in Paper B and Paper D.

C3. Formulate a conceptual model that describes how configurable multi-agent systems for Plug & Produce can communicate. This is presented in Paper C.

C4. Design methods that describe online dynamic planning and scheduling in configurable multi-agent systems for Plug & Produce. This is presented in Paper E and Paper F.

1.5 Scope and Limitation

Requirements for hard real-time communication between agents are not investigated. It is instead assumed that scenarios requiring hard real-time communication are considered as a whole agent instead of dividing it into several resources.

The Plug & Produce framework presented in this thesis is general and can be used for many types of systems. However, the focus of this work is limited to manufacturing systems.

This thesis is organized as follows: In Section 2, the preliminaries are presented, including an introduction to Plug & Produce, Multi-Agent systems, and Automated Planning and Scheduling. Section 3 presents the proposed framework. Section 4 goes through the evaluations of the proposed framework. Section 5 gives the conclusions. Section 6 gives a summary of the appended papers.

2 Preliminaries

An introduction to the knowledge required for understanding the designed Plug & Produce framework is given in this chapter.

2.1 Plug & Produce

Plug & Produce was first introduced in [25] by Arai et al. It aims at dividing a manufacturing system into process modules that can be connected while production is continuing. The idea is to be able to reconfigure a manufacturing cell in minutes rather than days in traditional approaches. A common approach in research is to use standardly-sized process modules and standard connectors. This has been done previously in [13], [14], [15]. However, to reach actual Plug & Produce the system also needs to detect each connected module and integrate them into the ongoing production. This can be compared with a conventional computational cluster, where computer nodes are added simply by connecting them with power and ethernet. Software and settings are then automatically downloaded to each detected node, this includes installing the complete operating system on them. Thus, any standard computer connected to the network is converted to a computational node that starts to receive tasks and replies with the calculated results. A similar approach is required in Plug & Produce systems. To design such a flexible manufacturing system there is a huge requirement for defining standardized communication interfaces for each process module in the system.

The concept of Plug & Produce can also be compared to the concept of Plug and Play, where connected resources are matched with a driver stored in the host computer. Similarly, the proposed Plug & Produce framework detects process modules using an Agent Handling System (AHS) and selects a correct agent configuration from a centrally stored database. The configuration is chosen based on the information given by the connected module. This is similar to the approach of selecting a driver when connecting a plug and play device such as a USB keyboard to a computer. The main difference is that Plug & Play only connects the device, making it available to the system. In Plug & Produce, the device instead becomes integrated into the ongoing production. The Plug & Produce framework is in this thesis used to achieve this integration.

In a Plug & Produce system the processes modules and products are assumed to be changed over time, making it difficult to use a central static control approach. Thus, a multi-agent system approach has been used for the Plug & Produce framework presented in this thesis.

2.1.1 Implementation

A physical Plug & Produce system was considered when developing the framework presented in this thesis, see Figure 1. This system was also used for the experiments conducted in some of the appended papers. The system was built and located at the research laboratory at University West in Trollhättan, Sweden.

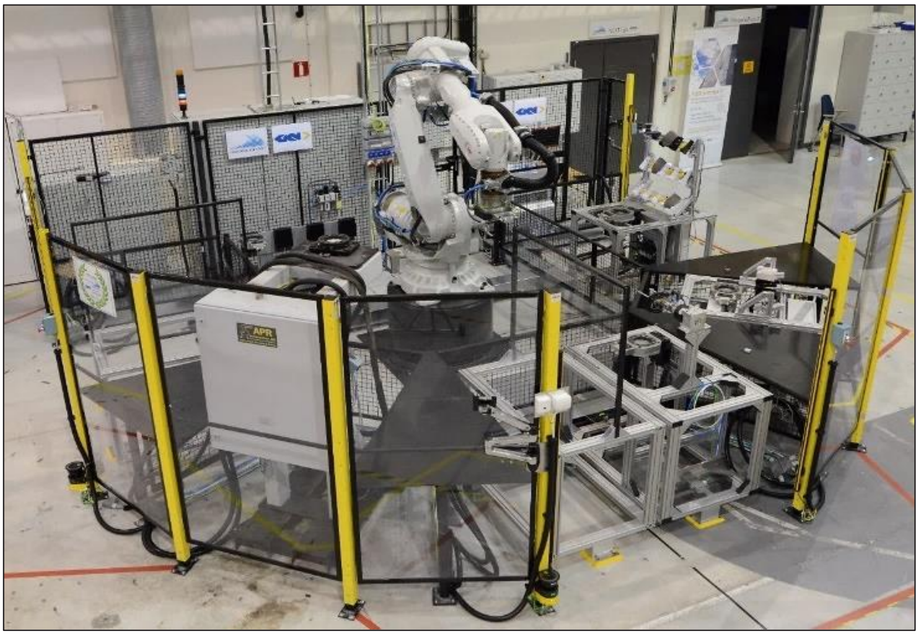


Figure 1. A Plug & Produce system located in the research laboratory at University West in Trollhättan, Sweden.

The manufacturing cell is divided into several process modules, that can be connected fast without affecting other resources. These modules are connected in a manufacturing cell with 10 different slots for connection, as shown in Figure 2. An industrial robot is placed in the centre of the cell, to be able to reach all process modules and assist with transportation. The colours in Figure 2 identify the different types of modules currently connected. Slots 1, 3, and 9 are type 1. Slots 2, 8, and 10 are type 2. Slots 4, and 6 are type 3. Slots 5, and 7 are type 4.

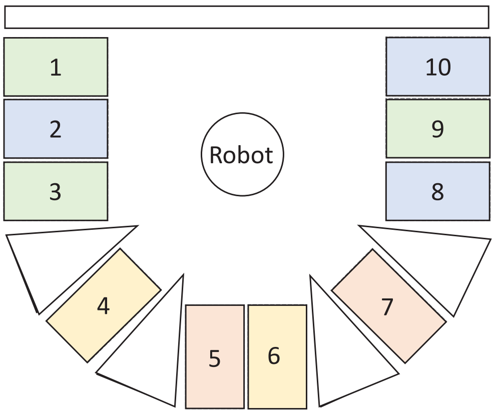


Figure 2: Plug & Produce system considered in this thesis, where the colours represent the different types of modules.

Each module is connected to the slots by standardized connectors including power air and ethernet. To handle safety, laser scanners are installed that can detect humans as they get close to the cell. It was designed to make the robot slow down when a human is close to the cell and stops if you they are inside. However, if the robot is working on the opposite side and cannot harm you, then it continues to work. In theory, this makes it possible to change a process module while the system is running.

Process modules can be given a Programmable Logic Controller (PLC), running an OPC UA (OPC Unified Architecture) server, that presents all sensor values and control signals to the network. This can be used by the cyber components (agents) to communicate with the process modules.

2.2 Multi-Agent Systems

Agents can be instructed on a high level instead of writing low-level programs for defining the behaviours of the system [26]. Sometimes, goals are defined for agents, that they want to reach. The agents then communicate with each other to reach those goals. Agents can be physical like a robot [27] or logical like services for path planning. Agents are commonly thought of as pieces of autonomous software [28]. Wooldridge et al. [29] described in 1995 that agency can be described with weak or strong notation. According to the weak notation agents have the following properties:

- Autonomy: agents handle their decisions without being directly controlled by other external programs or humans.
- Reactivity: agents react to the environment.

- Pro-activeness: agents take the initiatives to reach their goals.
- Social ability: agents can interact with each other using an agent communication language.

The strong notation of agency also includes cognitive behaviours with beliefs, desires, and intentions [30].

It is common to divide agents into two types: reactive agents and planning agents [31]. Reactive agents perceive their environment and take action to change it. This can be implemented by defining a finite-state machine [32]. Planning agents take more advanced decisions. For planning, it is possible to use the Belief-Desire-Intention (BDI) model first proposed in [33]. Beliefs are the knowledge that an agent has about the world. This knowledge is not necessarily true according to other agents; thus it's called beliefs. Desires are, for example, goals that the agent wants to reach. Intentions can, for example, be plans defining sequences of actions in the format of recipes rather than complex code.

A multi-agent system is a collection of multiple agents. Each agent is a program that runs independently and perceives its environment using its inputs and reacts to it through its outputs. This is shown in Figure 3. Agents commonly have their own goals, such as getting soft edges or changing colour to blue, implying that some machining and painting must be performed.

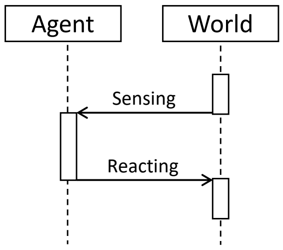


Figure 3: An agent sensing and reacting to its environment.

When several agents are connected, they form a multi-agent system as shown in Figure 4. In such a system, all agents collaborate using an agent communication language to reach manufacturing goals.

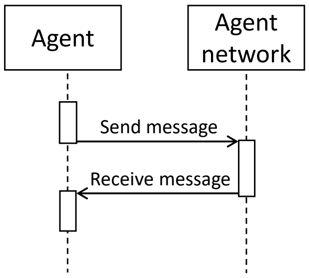


Figure 4: When multiple agents connect in a network, they form a multi-agent system.

When designing multi-agent systems for manufacturing, one approach is to create a part agent for each physical part to be produced in the system [34]. These part agents contain product design knowledge based on the customer specifications. Also, each resource in the system can have a related resource agent defined to control it. Then all these parts and resources communicate and plan the production.

In 1997 the Foundation for Intelligent Physical Agents (FIPA) presented an agent specification including the Agent Management Specification [35]. This was later updated to the current version in 2002 [36]. These specifications from FIPA describe how an agent network should be designed, including communication protocols and agent languages. It also includes specifications on how agents should be designed. FIPA is today one of the most used agent standards. It defines an Agent Platform (AP), where an Agent Management System (AMS), Message Transport Service (MTS) and a Directory Facilitator (DF) can be used [36]. The AMS registers agents, making them available for communication in the agent network. The MTS takes care of the message transportation between agents [37]. The DF is a "yellow pages" service where agents publish their skills. Agents can search the DF to find out which agent has the skills needed to reach a goal. According to FIPA, the DF is not mandatory, agents are permitted to contact each other directly.

In this thesis, the Plug & Produce framework was developed specifically for manufacturing systems. This includes an Agent Handling System (AHS), similar to the AMS but with some differences in the design.

2.2.1 Multi-Agent Frameworks

Agents can be implemented directly in any programming language. However, using an agent framework will drastically reduce the time to develop a new agent system. Agent frameworks commonly connect all agents through a communication channel and handle the publishing of agent skills, making them visible to other agents.

There exist many agent frameworks, such as the Smart Python Agent Development Environment (SPADE) [38], the Cognitive Agent Architecture (Cougaar) [39], the Magentix platform [40], and the Java Agent Development Framework (JADE) [41].

The most used agent framework that implements the FIPA standards is JADE [41]. This framework has built-in support to create containers in separate computers, where agents can be instantiated. These containers are connected with a communication channel, making it possible for agents on different computers to communicate without knowing the addresses of each other. There must exist one main container that hosts the AMS and DF. When instantiating a new agent, it must know the address of the main container. In JADE, the agents are mainly written in Java code by defining behaviours such as cyclic and one-shot behaviours. Since JADE is a general framework for agents, it is not adapted for manufacturing systems and lacks supporting tools for such scenarios. Thus, JADE still requires experienced designers and skilled programmers with high knowledge of the agent technology used.

The Plug & Produce framework presented in this thesis was instead developed specifically for manufacturing systems and includes supporting configuration tools, communication, and negotiation to decrease the amount of time spent on programming.

2.2.2 Agent Communication

Agents need to communicate with each other. This can be implemented through a standardised communication language. Two existing languages for agent communication are the Knowledge Query Manipulation Language (KQML) and the Agent Communication Language (ACL). KQML was developed in the early 1990s as part of the DARPA Knowledge Sharing Effort [42]. ACL was developed by the Foundation for Intelligent Physical Agents (FIPA), which is an IEEE organization, that develops standards for multi-agent systems [43]. In 1997 a collection of specifications was published named FIPA97 which includes the FIPA ACL. In 2002 an updated version was published including an updated ACL [44].

Both KQML and ACL are speech act based. Speech acts are expressions by individual agents that involve an action taking place. For example, if one agent asks another agent to perform something, then this is considered a speech act.

In Table 1, the ACL message structure for FIPA ACL is shown, consisting of 13 different parameters. As described in [45], the performative is the speech act name, the language is the language to express the content of the message, the ontology is the ontology name and gives meaning to the symbols of the expression, and the content is the actual message.

Table 1. FIPA ACL message acts in FIPA 2002.

Parameter	Description
performative	The type of communicative act (speech act)
sender	The sender of the message
receiver	The receiver of the message
reply-to	Receiver of replies
content	Message content
language	Language of the content
encoding	Encoding of the content
ontology	Used to understand the meaning of the message
protocol	Interaction protocol
conversation-id	Id of the conversation.
reply-with	Used for the In-reply-to message
in-reply-to	Replies with a reply-with content
reply-by	Deadline for reply

The FIPA standard from 2002 defines a Contract Net protocol [46], [47] as a definition of how two agents communicate. This protocol is also presented by Smith et al. [47] in 1980. It describes how an "initiator" can make a call to a "participant", asking it to give a proposal. The participant replies with a refusal or proposal. The initiator then rejects or accepts the proposal response. Finally, the participant informs the initiator of what has been achieved.

In FIPA 2002, speech acts are referred to as communicative acts and are presented in their Communicative Act Library (CAL). It includes 22 different communicative acts shown in Table 2. Note that these describe general communication without any specific considerations for manufacturing systems.

Table 2. Communicative acts in FIPA 2002.

Communicative act	Description
Accept proposal	Accept a submitted proposal
Agree	Agree to perform some action
Cancel	Cancel an action
Call For Proposal	Request proposals
Confirm	Confirms a proposition
Disconfirm	Disconfirm a proposition
Failure	Inform that action failed
Inform	Inform about a proposition being true
Inform If	Inform if a proposition is true
Inform Ref	Asks for the value of the expression
Not Understood	Did not understand the message
Propagate	Asks agents to forward this message
Propose	Send a proposal
Proxy	Ask the agent to act as a proxy
Query If	Ask the agent if the proposition is true
Query Ref	Ask for an object
Refuse	Refuse to perform the action
Reject Proposal	Rejecting a given proposal
Request	Request agent to perform an action
Request When	Request when the proposition is true
Request Whenever	Always run when the proposition is true
Subscribe	Let the other agent send updated data

Hence, the FIPA standards are general for any kind of agent system, giving no help for the specific problems in manufacturing systems.

2.3 Automated Planning and Scheduling

In multi-agent systems, planning is important. Three different types of planning activities are considered in this thesis: (1) Process planning, including scheduling of resources, (2) pathfinding algorithms that can find the shortest path through a manufacturing system, and (3) path planning that generates collision-free paths for industrial robots. These are all important tools for making a Plug & Produce system since they give the possibility to automate additional steps that would otherwise be performed manually. Process plans are written on a high level and defined manually by humans, but automatically deployed by selecting resources to be used and scheduling them. Pathfinding algorithms help to automatically find and optimize product paths through the system. Paths for robots would have to be developed and tested manually if not using automated path planning.

2.3.1 Computation Time

Something that also can take considerable time is the computations of the manufacturing system, especially for optimal solutions. The complexity and flexibility are affecting each other and a function can be defined that gives a total computational time value

$$Time = f(Complexity, Flexibility).$$

This value defines how much computation time is used for the manufacturing system due to planning for the agents in the system. In Figure 5, this is illustrated with five points that are given where systems of different types are pinpointed. Point number five is the type of system proposed throughout this thesis. The advantage of point five is that it has less computation time than point four while still maintaining high enough online flexibility and optimality for the scenarios considered in this thesis. Heuristic planning and scheduling are approaches used to achieve this. This thesis focuses on Plug & Produce to implement such a system as given by point five.

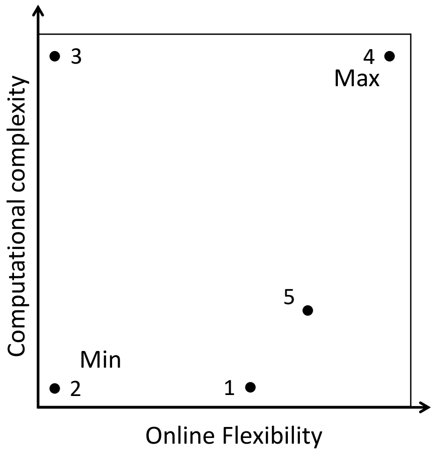


Figure 5. This figure shows that the computational time, due to planning in a manufacturing system is a function of computational complexity and online flexibility. The highest computation time is type 4 and the lowest is type 2, noted with Max and Min.

2.3.2 Path Planning

The Plug & Produce framework cannot solve everything by itself since it is only a platform for developing agent systems. All required data such as geometries, positions and process plans must be given to the agents. For example, an industrial robot, i.e. 6-axis arm based, need to have a collision-free path for moving in robot cells as shown in [48]. If a robot is supposed to be autonomous and without the need for manual reprogramming, it must be possible for the robot to find paths on demand automatically. Otherwise, these must be pre-programmed manually for each possible scenario. One approach is to use a path planner that can be used online when the manufacturing system is running. Example of suitable planers is RRT and RRT-connect [49] due to their ability to efficiently find solutions. An improved planner called RRT* has also been developed that tends to find shorter paths than RRT.

A rapidly exploring random tree (RRT) is used for finding a collision-free path through a space with obstacles, such as a robot cell [50]. It searches a space by building a random tree from the start point until it reaches its target point. RRT-connect is based on RRT and is instead using a bidirectional search, starting in both the start and target positions. It then generates random points until the two threes meet. Using these types of path planners, industrial robots in manufacturing systems can become completely autonomous. This will enable them to handle new types of products instantaneously. However, it puts a requirement on having a completely accurate simulation of the robot cell with computer-aided design (CAD) objects that are accurate enough. The planner algorithms need this simulation to make the paths collision-free. One famous library that has implemented RRT-Connect and many other planners is the Open Motion Planning Library (OMPL). This library is not containing any collision detection since it has no 3D representation of the world. One software that adds that layer is the MoveIt software which is a part of the Robot Operating System (ROS), that uses OMPL.

The problem with such simulation software is that their simulations typically are updated manually by humans. This includes designing CAD models in external software, importing them, and setting their positions in the simulation. It is acceptable that CAD models need to be designed by humans, to make sure that they are accurate enough. However, the positioning of the CAD models in the simulation should be based on the actual state of the online robot cell and that is difficult for humans to do in a Plug & Produce system where modules are moving around constantly, see Figure 6, where the robot should move between positions p_1 and p_2. This is just too costly for humans to perform in terms of working hours. Additionally, some dynamic obstacles such as operators moving inside the robot cell and tools forgotten inside can't be created manually in CAD software, due to the random nature of these objects. Thus, the cell could instead be scanned using a vision system to find these obstacles. These can be automatically added to the simulation by generating many small cubes to approximate the shape of the real obstacles. Hence, two approaches are needed to work concurrently for automatically updating the obstacles in the simulation: 1) Automatically placing existing CAD objects and 2) generating approximated shapes based on online sensor data.

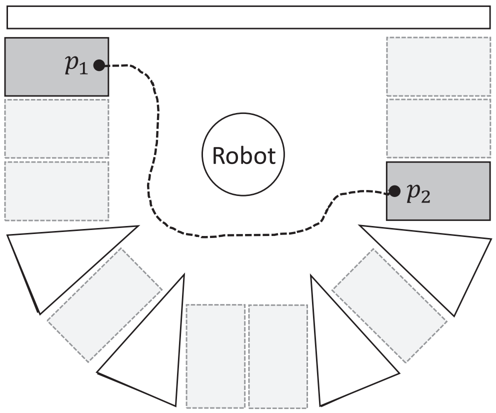


Figure 6: Top view of a Plug & Produce robot cell, containing one industrial robot in the centre, surrounded by ten process modules. The robot should move between the start position p_1 and target position p_2.

2.3.3 Pathfinding

Pathfinding refers to finding the shortest path from one node to another in a set of nodes. This is relevant when each part agent plans its transfer through the manufacturing system. Dijkstra defined an algorithm already in 1959 [51], where the shortest path between two nodes can be found. Since that time, extensions have been created such as the A* algorithm. In Figure 7, the shortest path is found between a start node A and a target node E. This shortest path is marked with blue colour and a thick line. Each line has an integer cost for travelling.

When multiple agents should find collision-free paths, conflicts could be introduced when running A* [52]. Such planning is often regarded as multi-agent pathfinding (MAPF) [53]. There is often also a focus on minimizing a cost function such as the length of the path of all agents or the total makespan. Li et al. explain that MAPF typically does not include the agents' physical shape when planning, thus collisions could exist between geometrical shapes if they are not considered when planning [54]. Sartoretti et al. [55] state that MAPF is relying on central planning and not scaling well beyond a few hundred agents since the paths are recomputed online when things change. Two classes of MAPF are described by Sharon et al. [56]: (1) decoupled approaches where paths are found individually for each agent relatively fast but with no guarantee for a globally optimal solution. and (2) coupled approaches that can reach optimal solutions but takes more time for calculations. One example of decoupled planning is the cooperative A*, described by Silver et al. [57].

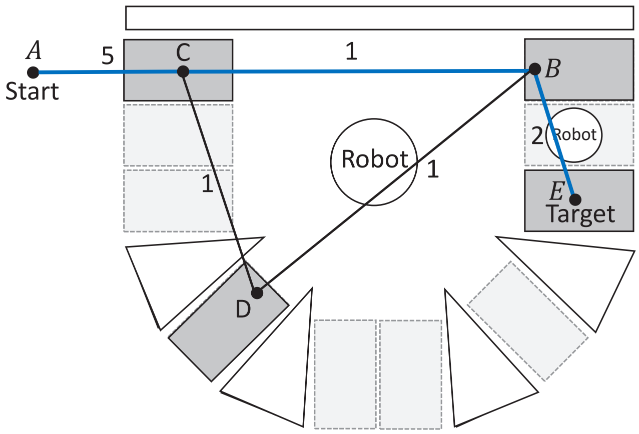


Figure 7: Dijkstra's algorithm, where the blue marked line is the shortest path between the start and target positions.

For an agent to use such an algorithm it requires knowledge about its environment. It needs to know each node, i.e., resource buffer and what the travelling costs are. This can be done by letting the agents ask other agents about their location and reachability. For transportation, an industrial robot might need to first run a short simulation before answering. In Figure 8, this is illustrated with two robots, each having its reachability areas, noted by A and B. However, since obstacles exist in the robot cell, zone B is not correct, and C becomes the actual zone after collision detection has been performed. This shows us that it is not possible to know in a Plug & Produce system if the robots can reach a certain point without doing some path planning and collision detection.

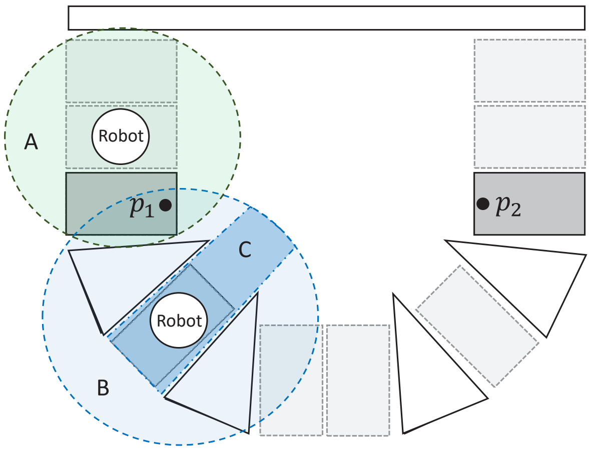


Figure 8: Industrial robots' reachability example for checking if they can reach the point p_1 and p_2 marked in the figure.

This is time-consuming and can take several seconds up to minutes to perform. Thus, it is desirable to store these simulations for later use if the same types of parts are going to ask the same questions again. Additionally, the agent can store the successful paths through the system, trying to use them for the next part that enters the system. In this way, the system will be slow at the beginning and speed up after some parts have gone through the system. It is also possible to do this in simulation first to let the agents find each other before deploying them to the online environment in the physical manufacturing system. Then the agents already will be trained before entering the online manufacturing system.

2.3.1 Scheduling

When multiple part agents share a set of resources in the system, scheduling is needed for avoiding conflicts. Things to consider include generating schedules, storing schedules, and optimization. Schedules can be generated by considering all part agents at the same time for reaching a global optimal schedule, or they can be generated locally for each part, with no guarantee of reaching a globally optimal solution. The software for generating schedules can be distributed to agents or it could be contained in a central software, that performs the calculations. The storage of the schedules can either be centrally located or they can be distributed among the resources to be scheduled so that each resource knows locally about its schedule. Then other agents can ask those resources for their schedules. When working with Plug & Produce the environment is dynamic since changes happen online. There are multiple approaches for dynamic scheduling and rescheduling [58]. One approach is to add new schedules to the end of the set of all already scheduled tasks, requiring no rescheduling [59]. If the system runs into a problem, it needs to rerun a sequence of skills. Then that is not necessarily possible if that sequence is in the middle of all scheduled tasks. According to Viera et al. solutions for job insertion include right shift, partial rescheduling, and regeneration [60].

3 Proposed Plug & Produce Framework

This chapter presents the Plug & Produce framework that has been developed in this work to reduce software time. It goes through the agent interfaces, process plans, and agent configurations. Then, the translation of locations is described to show how to manage local coordinates when moving process modules around in the system. Agent communication is described to show how the agents interact. Then the agent handling system is described to give a view of the agent life cycle management. Next, a configuration tool, that is used for defining the ontology is presented. Finally, the planning is described, and divided into scheduling and pathfinding.

3.1 Agent Definition

Agents can be seen as components of a cyber-physical system. Each agent is a cyber component running in cyberspace such as in a cloud service and is connected to a related physical or logical component in the real world.

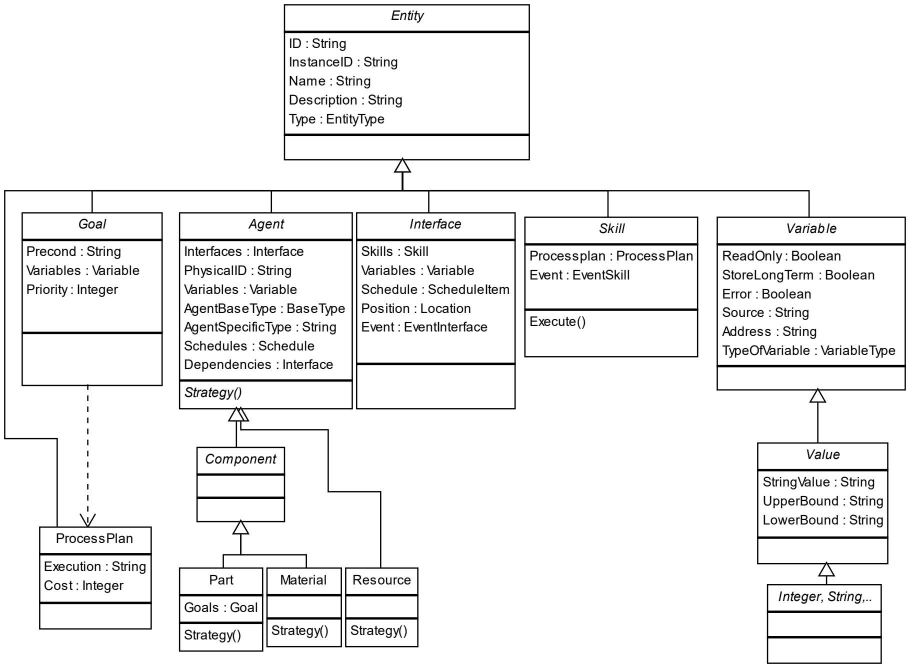


Figure 9. Class diagram showing the ontology used when configuring one agent.

In C-MAS, all agents use the same basic classes and are given their behaviours through a configuration called the global configuration. In Figure 9, a detailed description of this ontology is shown.

When an agent is instantiated, it becomes an individual object in the cloud and is also given a local configuration. The local configuration contains all the unique parameters that are not shared with other agents of the same type, i.e., that are using the same global configuration. Two types of agents exist, parts and resources. The difference is that parts have goals that they want to reach in the manufacturing system using available resources. To reach a goal, process plans are defined as recipes rather than programs. To run a process plan, several variables and skills are required to exist in the agent network. These requirements are in this thesis noted as demands. Agents interact by connecting through interfaces. An agent can have multiple interfaces that define its compatibility with other agents. Skills are always presented to other agents through interfaces. This is done to determine if their interfaces are compatible. Agents have variables that are accessible from all interfaces within the agent. Variables can also be local on individual interfaces. Skills describe functionalities that an agent can perform and are executed by running a process plan specifically written for that skill. Process plans might generate further demands for additional agents (including required skills and variables). This creates a chain of connected collaborating agents. An example of a chain is a gripper that transports a part where the gripper is connected to an industrial robot. These three physical components can be controlled by separate agents, connected in a chain of interfaces when interacting. Compatibility is checked by comparing the generated demands with the interfaces in the agent network, to find a compatible agent.

In the multi-agent system, a single agent a, belongs to the set of all agents A, i.e., $a \in A$. Typical agents in manufacturing systems represent physical components such as grippers, parts, buffers, and robots. The set of agents A contains all parts P and resources R, where $A = R \cup P$. A single part is noted as $p \in P$ and a single resource is noted as $r \in R$. The parts and resources can be described as:

$$p = \langle G_p, IF_p, V_p \rangle$$

$$r = \langle IF_r, \mathrm{V}_r \rangle$$

Where G_p is the set of goals for part p, IF_p is the set of interfaces if that the agent p has, and V_p is the set of all variables that p has. A single variable is noted as v and belongs to an interface or an agent. A variable can, for instance, be a path for grinding or a coordinate for picking up a part. Also, goals can have variables related to them for giving parameters for the goal.

Parts and resources have different behaviours called strategies. Parts are active with goals that they try to reach, while resources are more passive and wait for parts to give them requests for running skills. However, once a resource gets a job to perform, they will gather information about their surroundings and take the initiative to request further assistance from other resources. A goal for parts can, for instance, be to get soft edges. Goals can be described with attached arguments such as speed or size, making small changes to the consequence of reaching the goal. One goal is defined by the following tuple:

$$g =< n_g, V_g, pre_g >$$

where n_g is the name of the goal that should be matched with the name of a process plan π_g in Π_g, V_g is the set of all variables for the goal, and pre_g is the set of goals that must be fulfilled before this goal can be assigned. The set G_p contains all goals $g \in G_p$ for one part p. A part can have several goals, and each of them describes some manufacturing result that the part desires to reach. Variables can also be stored on an agent by storing them in V_p or V_r depending on the agent type. A single skill is noted as $s \in S$, where S contains all skills in the global configuration. A skill is defined as:

$$s = \langle n_s , \pi_s \rangle$$

where the n_s is the variable name and π_s is a specific process plan written for executing that specific skill.

3.2 Agent Types

Each agent type in the framework has its strategy that defines its behaviour. The defined types in this work are the part agents and the resource agents. However, other types can exist such as material agents that are similar to part agents but with no goals.

3.2.1 Part Agent Strategy

Part agents have goals and reach them by executing process plans for those goals. To make a process plans executable, multiple communications have to be done with other agents and skills scheduled before the plans can be executed. In Figure 10, a simplified overview of the part agent strategy is shown. The part agent first loads its configuration telling it that it is a part agent and loading the set of goals together with any other configured values. Next, all goals are scheduled, followed by going into a loop of starting skills on resources until all goals are reached. Finally, the part agent is shut down if no goals remain unfulfilled.

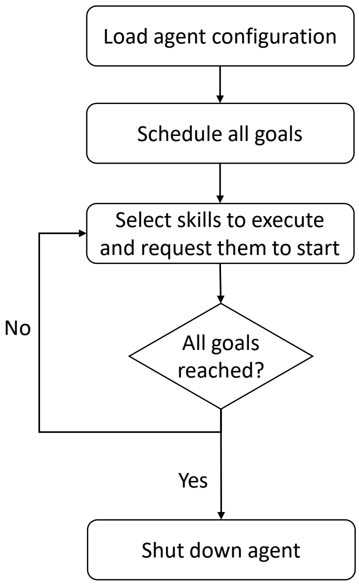


Figure 10. Flowchart showing a simplified part agent strategy where the detailed steps are hidden.

3.2.2 Resource Agent Strategy

Resources have skills that can be triggered to execute based on events. One event is that another agent requests the skill to execute. This requires that the skill has been correctly booked before the request. Another event is timers, which can be used to run a skill as a cyclic behaviour, meaning that it will execute at a certain frequency. Further, events based on variables are useful and can be based on user input. For example, when a button is pressed, a skill could be executed. In Figure 11, a flowchart is shown that describes the detailed behaviour of the resource strategy. The resource starts and loads its agent configuration, then it starts to receive messages and manages them based on the communicative acts used for each message. The function *setData()*, sets some data on the resource agent, *requestData()* returns some data from the resource agent, *runSkillNow()* executes a specific skill on the resource agent, *checkDemand()* checks if the resource agent fulfils a specified demand for the resource to have certain skills and variables, *bookInterface()* books an interface on the resource agent *unBookInterface()* unbooks an interface on the resource agent.

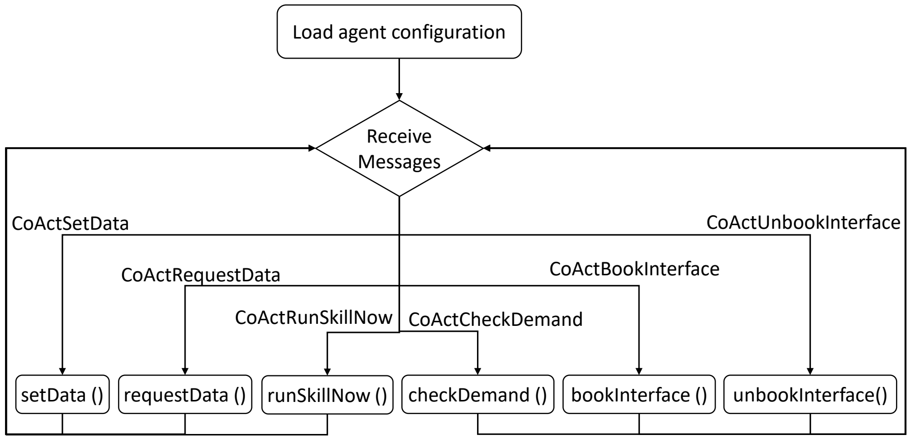


Figure 11. Flowchart showing the strategy of a resource agent, that receives messages and manages them based on communicative acts.

3.3 Agent Interfaces

Interfaces must match to connect. Agents have multiple interfaces, and interfaces have multiple skills. Each interface has defined inputs that describe needed variables to execute any of the presented skills on that interface. Some interfaces have variables that can be accessed by other agents by requesting their current value. One interface is noted as $if \in IF$, i.e., IF is the set of all interfaces in the global configuration. An interface is defined by the tuple:

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

where all skills s on that interface are placed in S_{if}, and all variables on the interface are placed in V_{if}.

Available resources are not known at the planning stage when process plans are created. Instead, the needed resources are defined by declaring an abstract interface, noted as:

$$u \in U = \{u_1, u_2, \dots, u_{n_u}\}$$

Each abstract interface u declared in a process plan is mapped to a real resource interface if at runtime before a process plan can be executed. This is done by an interface mapping algorithm, presented in Paper A. This algorithm generates demands $d_u = \{d_1, d_2, \dots, d_{n_d}\}$, where d_u is defined as:

$$d_u \; =< S_u, V_u >$$

Hence, the demand d_u defines the requirements that one agent has for another agent's interface. Each u represents an abstract interface. Note that both d_u and if defines skills and variables. Thus, if must have all skills and variables that are defined in d_u to be compatible. The agent that uses the process plan will look in the agent network to find one interface that can be used.

3.4 Process Plans

To reach a goal, a process plan is needed. A process plan defines at a high level how to reach a specific goal. This description is written like a recipe, rather than low-level code. A process plan translates one goal into a sequence of skills that can be executed on resources in the agent network. When a part agent is given a goal, it must find a suitable process plan that is available. Hence, it will go through all process plans in the global configuration until one is found where all demands are fulfilled, i.e., all needed resources are available with compatible interfaces. It is also possible to compare all process plans and select the one with the lowest cost, based on the currently installed resources as described in Paper A. Process plans can either be automatically generated or manually designed by a human. It is difficult, if not impossible today, for computer software to know how to design a successful process. This knowledge is today based on human experience [61], [62]. If a system would have to try repeatedly it might find a solution by using tools such as machine learning. However, this is today difficult to achieve on a high-level process plan and is more suitable for specific problems. A single process plan for reaching a goal is noted as:

$$\pi_g \in \Pi_g$$

and a process plan for executing a skill is noted as π_s. A process plan must be turned into an executable process plan $\pi_g^e \in \Pi_g^e$, or $\pi_s^e \in \Pi_s^e$. The set Π_G contains all process plans for reaching all goals for all parts that are stored in G. A single process plan π_g solves the goal g that can exist on any part p. On the other hand, the process plan π_s is only written for a specific skill s existing on a specific resource type that has that skill. A process plan defines a sequence of skills $(s_1, s_2, \ldots, s_{n_\pi})$. Figure 12 shows the states for a process plan with five skills 1-5, where the initial state is q_0 and the final state is q_f.

$$\begin{tikzcd} q_0 \arrow[r, "s_1"] & q_1 \arrow[r, "s_2"] & q_2 \arrow[r, "s_3"] & q_3 \arrow[r, "s_4"] & q_4 \arrow[r, "s_5"] & q_f \end{tikzcd}$$

Figure 12: Process plan with five skills, where the initial state is q_0 and final state is q_f.

3.4.1 Example 1

An example of a process plan π_g to reach goal g is: $u_1.s_1(), u_2.s_2()$. In π_g two skills are used: s_1 and s_2. The letters u_1 and u_2 in front of these skills are the abstract interfaces, used to show that these skills need to run on different resources. The letters are called abstract interfaces since they represent undefined interfaces needed on some resources in the manufacturing system. If the letter a would be used for both skills, that would mean that we are looking for an agent having an interface with both s_1 and s_2. Process plans exist for both reaching goals, and for executing skills. Thus, the connections between process plans create a tree of connected interfaces. An example of a tree of connected interfaces is shown in Figure 13. Here, the Part p has the goal g, solved by π_g. The process plan π_g generates certain demands d_{u_1} for an interface u_1 in the agent network and finds the matching interface if_b on r_1. Additionally, interface u_2 described in π_g generates the demand d_{u_2}. Thus, the part finds the resource r_2 with interface if_e. Resource r_1 has a process plan π_{s_1} with further demands d_{u_3} and finds r_3 on if_d.

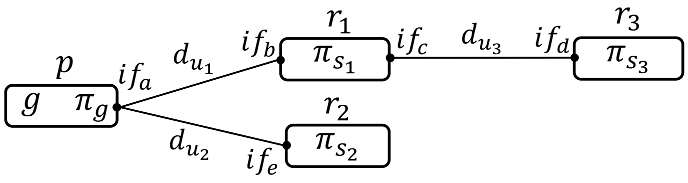


Figure 13: A plan π_g is checked for availability in the agent network.

The detailed interactions are shown in Figure 14, where each step is shown. The demand d_{u_1} is broadcasted to each resource in the current system, r_1, r_2, and r_3. Then r_2 and r_3 rejects the request since they do not fulfil the demand. The resource r_1 has the demanded skill but demands further skills from other resources and broadcasts that to the agent network in this example sent to r_2 and r_3. Resource r_3 accept the request and returns information about the interface if_d that was compatible. Then, r_1 can accept the demand from p by returning information about its compatible interface if_b. After that the second demand d_{u_2} is broadcasted and rejected by r_1 and r_3, but accepted by r_2 that returns information about the compatible interface if_e.

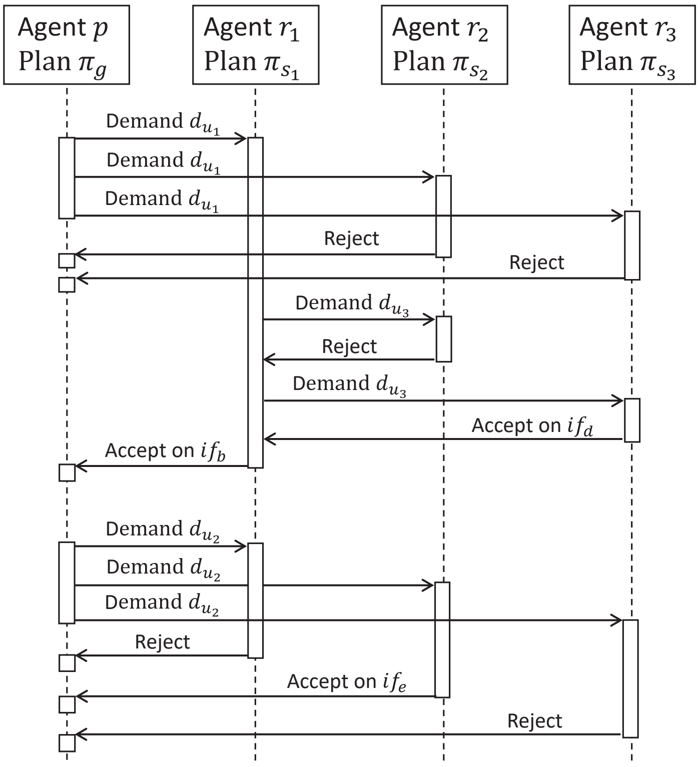


Figure 14. Example of one part interacting with three resources.

3.5 Configuring the System

As described earlier, all agents use the same basic code and are given their behaviours through a configuration called global configuration. When instantiated the agent is also given a local configuration that is based on the actual values of input data to the agent, such as sensor data from its physical component or information from other agents. All configurations for one agent (local and global) are arranged according to the configuration classes presented in Figure 15.

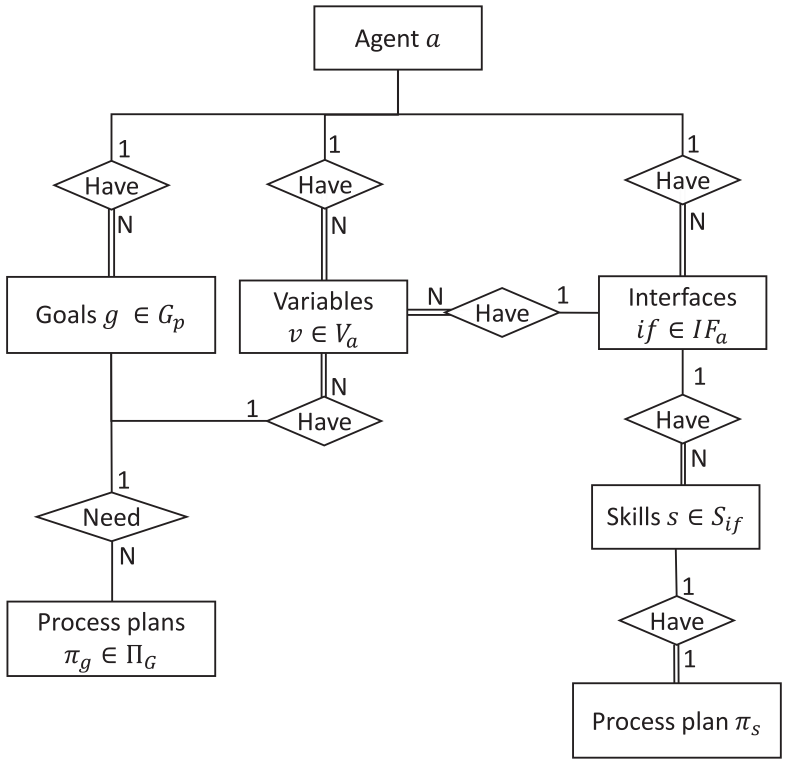


Figure 15: Entity-Relationship model (ER model), showing a simplified view of the ontology used for configuring the agents. For further simplicity, the attributes are not included.

The idea is to create a standardized agent code where only two types of agents exist, the part and the resource. This code never changes and is general for most manufacturing scenarios. Specific hardware codes will still exist, such as the controller for an industrial robot or a PLC for reading a sensor. The agents handle all communication among resources and parts of the system. It represents the hardware or in some cases software. To do this, the agent needs to know what services, i.e., skills it offers and what goals it has. Together with this, all variable data and interfaces must be defined. The configuration should contain enough information for the agent to know how it should behave. Global configurations are written offline in a user-friendly configuration tool. They are later uploaded to a configuration database, where all agents' global configurations are stored. Many users can log in and work simultaneously with the configurations. Further, configurations can automatically be deployed to an online manufacturing system. This is done by copying the developed configurations to another configuration database existing in the manufacturing system, used online. Next time a resource or part gets an agent instantiated, they will use the updated global configuration.

By separating the offline and online configuration databases, it is possible to test the new configurations together with a simulated manufacturing cell before deploying them to the online manufacturing system.

3.5.1 Example 2

In Figure 16, an example is shown of how one part agent p tries to reach a goal g. The goal is solved by process plan π_g that demands an interface with a specific skill, and finds s_1 on if_1. The skill s_1 executes its plan π_{s_1}, that demands a further skill and finds s_2 on if_2. Finally, skill s_2 executes its plan π_{s_2}. After this the chain is disconnected and the goal g reached.

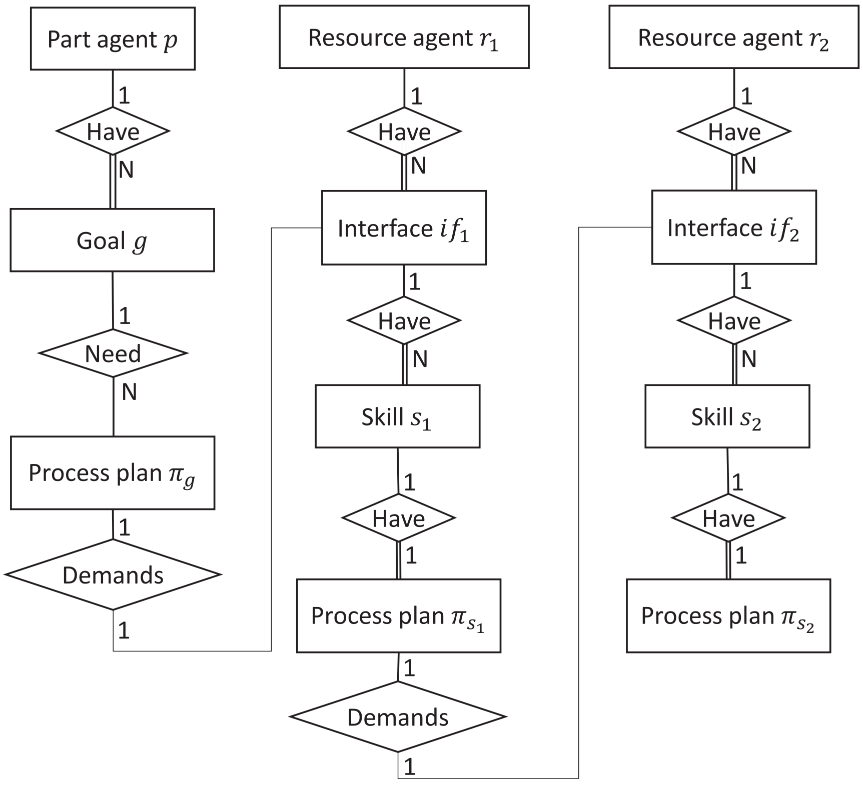


Figure 16: A combined model, showing a partial view of the internal ontology of three agents p, r_1, and r_2. In this example, the part agent p is requesting a resource r_1 to run a skill, and that resource is requesting a second resource r_2 to run an additional skill.

3.6 Translation of Locations

When variables are communicated between agents, they are first translated into world coordinates. If an agent wants to give its variable to another agent, it needs to find out where it is located in the world. Agents know that interfaces are connected while interacting with each other. They also need to have locations defined for those interfaces, specified in a local coordinate system for each agent. By connecting two interfaces, it is assumed that they always are at identical locations. However, it is not yet necessarily known where that is without further calculations. Agents can ask the other agent that they are attached to, to give their coordinates in world coordinates. If they are not having the world coordinate system, then this will continue as a chain of requests down to the agent that is in world coordinates, most commonly the items placed on the floor, such as a docking slot in the manufacturing cell. Each agent has this functionality to translate coordinates. The idea is that everything needs to be attached to some common physical component, like the manufacturing cell. This information can then be used by resources such as an industrial robot for transporting items around in the system.

3.6.1 Example 3

See Figure 17 for an example of a part asking the buffer that it is placed on for its world coordinate. This is then used to inform a transporter agent where to pick up the part.

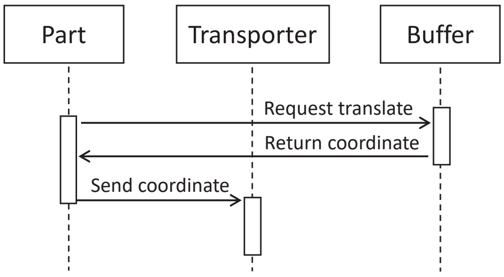


Figure 17. A part agent, placed on a buffer, requests a transporter to move the part from the buffer. This requires the part to first ask the buffer for the world coordinate that the part is placed on.

3.7 Agent Communication

Agents need to share a common language for communication, such as the FIPA ACL introduced earlier. FIPA describes a set of communicative acts (co-acts), such as Inform, Request and Agree. However, as described earlier, these are designed generally for all types of systems and lack specific support for manufacturing systems, such as booking resources and starting a process. Thus, it was found in Paper C that it is possible to define another layer with specialized co-acts designed for the specific type of scenario considered.

The naming of configuration values such as skills, variables, and interfaces must be standardized in the global configuration among the several agents. Two agents must share a common understanding of what these values are used for. For example, two variables Pick and Place both have the same data type, but different meanings. This must be understood by all agents using these variables. To manage this, it is desirable to have a configuration tool that guides the user and gives warnings in case of problems. If resources share common semantics for global configuration values, then it is possible to move a resource to another manufacturing cell or even to another company, assuming they are using the same semantic standard.

In Paper C a conceptual model was defined with four layers 1-4, see Figure 18. It describes different layers of communication that can exist in a multi-agent system. Layers one and two already exist in agent frameworks such as JADE, where layer two could be FIPA ACL. However, other agent communication languages could fit into this layer.

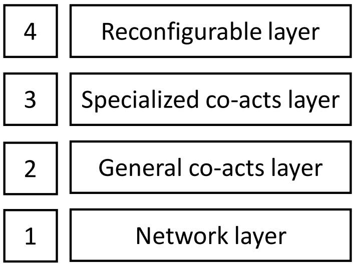


Figure 18: Conceptual model for agent communication.

The first layer (1) describes the basic communication protocol and the setup of the agent network. Layer two (2) describes the general communication functionalities between agents. Layer three (3) describes the special communication. In this thesis scenario, it is specialized for manufacturing systems. Layer four (4) is the reconfigurable layer where the global configurations are created. The idea is that most manual work is done in layer four where no programming is needed.

3.8 Agent Handling System

Agents are considered to be cyber-components while the physical resources or parts are the physical components. Some agents can also represent software, and other agents can exist without representing anything, acting only as a cyber component. Many times, agents need to be connected to the object it is controlling by a communication channel. This can be done using many protocols, but the implementation in this work was done using the OPC UA protocol over Ethernet. OPC UA is a platform-independent communication protocol that was developed by the OPC Foundation to be used in industrial automation [63]. In Figure 19, a data hub is shown, that communicates with the agents in the system on the left, and connects them with the object they represent on the right, using for example OPC UA. This means that industrial communication protocols can be added without changing the agent design. Agent 1 in Figure 19, is completely connected with object 1 and communication can be done in both directions. Agent 2, is connected to the data hub, but not to the object 2 that it represents. This can be the case when having objects that do not have any electronics on them, such as a part to be produced. In that case, the agent knows that the objects exist but only affects them indirectly by calling other agents on the left side to let them manipulate object 2. Agent 3 is a completely in the cyber-space, not having any object in the world or software to control. The use of this can be to add functionality that is only used for control of other agents.

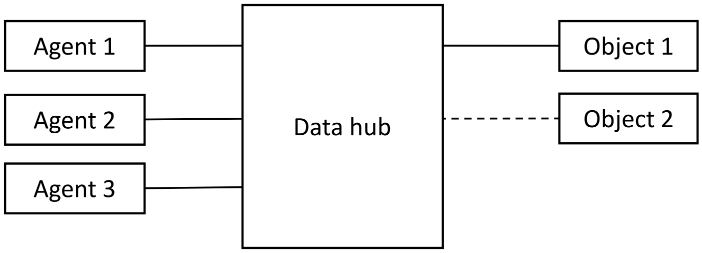


Figure 19: The data hub for synchronizing data between agents and their related objects in the world.

The data hub is part of the Agent Handling System (AHS), developed and presented in Paper B. The AHS is illustrated in Figure 20 and includes four parts: the Agent Creator, the DHCP server (Dynamic Host Configuration Protocol), the Agent Detector, and the Data HUB. DHCP stands for Dynamic Host Configuration Protocol. It is used to automatically detect devices in networks and to give them an Internet Protocol address (IP address).

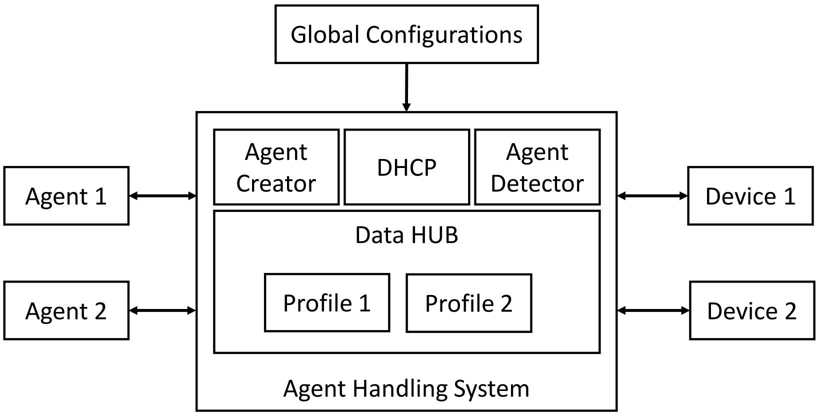


Figure 20: This figure shows the Agent Handling System connected to two devices on the right side that are controlled by the two agents on the left side.

In Figure 21, the behaviour of the AHS is described in more detail. It can be divided into six different steps:

1) *Detect new devices in the network:*
   A unique IP address is assigned to each new device detected in the network by the DHCP server. Each address is stored in the AHS.
2) *Establish a connection to the new devices:*
   Each IP address is used by the AHS to establish an OPC UA connection between the Data hub and the detected device
3) *Identify possible agents:*
   The Agent Detector searches the OPC UA server on each detected device, to gather information about what agent type it has.
4) *Get global configuration:*
   Based on the agent types detected in step 3, the AHS fetches the corresponding global configurations required for instantiating the needed agents.
5) *Get local configuration:*
   Each local configuration value is fetched by the AHS from the physical device and stored.

6) *Instantiate new agent program:*
   The Agent Creator instantiates a new agent using the selected global and local configurations for each detected device.

After these steps are performed, the data hub sets up the synchronization between the cyber and physical components as shown in Figure 21.

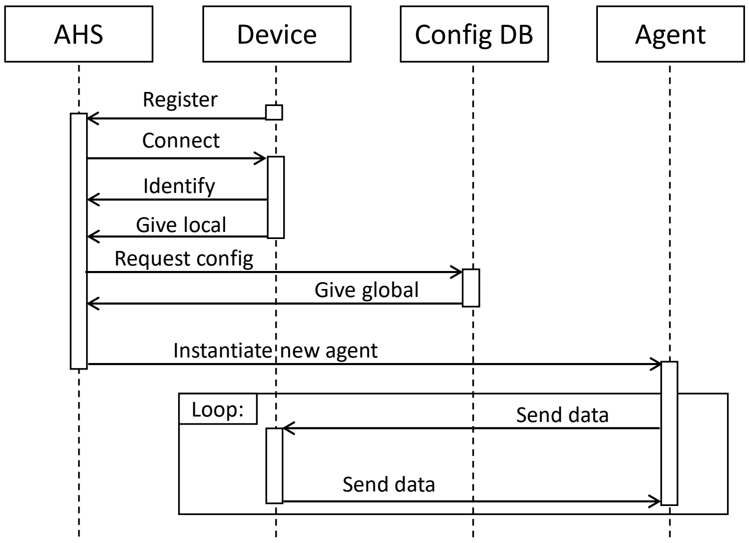


Figure 21: Method for registering new devices added to the Plug & Produce system.

3.9 Configuration Tool

In Paper D, a configuration tool was presented that is used to manually create global configurations. The configuration tool presents several different views to the user as defined in the following list:

Main view:

In the main view, agents can be added, removed and edited. Changes to the agents are done in the agent view. Additionally, the process plans can be added, removed and edited, by opening the process plan view.

Agent view:

In the agent view, we can set the agent name and choose if the agent is a resource or part. Variables can be added, removed and edited; this opens the variable view. Interfaces can be added, removed and edited; this opens the interface view. Goals can be added, removed and edited; this opens the goal view.

Interface view:

In the interface view, we can set the interface type. Variables can be added, removed and edited; this opens the variable view. Skills can be added, removed and edited; this opens the skill view.

Goal view:

The goal view is used to set the goal name.

Process plan view:

The process plan view has the functions to set a plan name, define a goal and write the actual process plan.

Skill view:

In the skill view, we can set the skill type and write the process plan for the skill.

Variable view:

In the variable view, we can set the variable name and its data.

3.10 Agent Planning

An agent needs to perform planning to adapt to new situations. This thesis has aimed to perform this online and with as few disturbances as possible to the manufacturing system. This means that they had to use some heuristic approaches to speed up the computations. This chapter presents the scheduling of process plans and an approach to pathfinding that coordinates multiple agents to find the shortest path with no conflicts with other agents' paths. The two concepts support each other since the scheduler presented is used by the pathfinding algorithm to solve conflicts, including the deadlock problem.

3.10.1 Scheduling of Process Plans

In Paper E, a new planning method is introduced, and an algorithm for scheduling is defined for our multi-agent system. The idea is that each part agent plans their production by planning all goals directly when added to the system. That means that each goal gets a process plan selected and scheduled. There is no central system where the schedule is stored, instead, each resource agent holds its schedules locally. When a part agent is looking for skills, the compatible agents will return their schedule to the part agent so that it can consider any conflicts. The part can decide by coordinating among its goals so that it can plan a way to go through the system. Then it communicates this plan to each resource so they can add it to their schedules. Since all goals are planned for at the beginning, the plan and schedule cannot be changed if something happens later that was not considered. Thus, all alternatives must be scheduled. This was solved for unpredictable events so that it can rerun a goal if it would fail. In Figure 22, two sets of parallel sequences G_a^{pa} and G_b^{pa} are shown. There are a total of four sequences of goals $G_{a_1}^{sq}, G_{a_2}^{sq}, G_{b_1}^{sq}, G_{b_2}^{sq}$. In those sequences there are a total of five goals $g_1^h, g_2^h, g_3^h, g_4^h, g_5^h$, containing the skills $s_1^h, s_2^h, s_3^h, s_4^h, s_5^h, s_6^h$. The goal g_3^h can rerun if the result from its skills is not acceptable. Each of the sets G^{pa} must wait for all arrows that are pointing to G^{pa} for it to start executing any of its skills. Meaning that the skill that the arrow is pointing from must be successfully executed with an acceptable result.

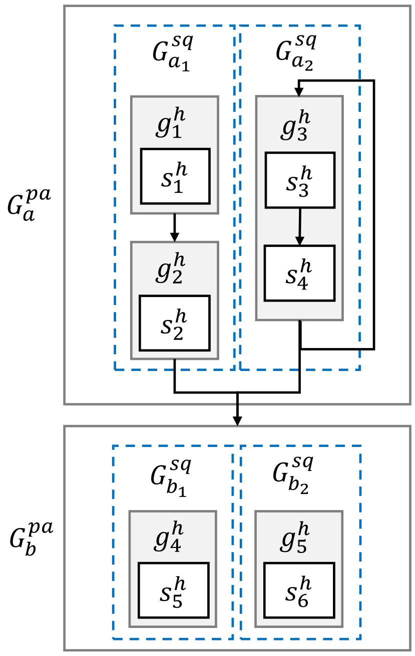


Figure 22. This illustrates an example of goal sequences G^{sq} running in parallel G^{pa}, with goals g^h having skills s^h to be scheduled. Each G^{pa} must wait for each arrow pointing to it to be done for G^{pa} to start executing any of its skills.

To make the schedule distributed the resource tuple was defined as the following

$$r = < n_r, IF_r, V_r, SC_r, IF_r^{dp} >$$

where, n_r is the name of the resource, IF_r is the set of interfaces, V_r is the set of variables, SC_r is the set of all schedules for each interface in IF_r, and IF_r^{dp} is the set of dependencies between interfaces. One dependency is defined as

$$if^{dp} =< expression, IF_{if}^{dp} >$$

Where if the $expression = true$ all interfaces in IF_{if}^{dp} will also be booked if if, is booked. This is useful when two interfaces are related and booking one might make the other unusable. Each interface $if \in IF_r$ has a local schedule SC_{if}, stored in SC_r. The schedule SC_{if} contains each scheduled skill

$$s^h =< n_s, n_u, g, st_s, if_s^{lo}, if_s^{re}, S_s^h >$$

Where n_s is the name of the skill s, n_u is the name of the abstract interface u that generates the demand for the skill s, st_s is the state that the interface having skill s should be in to run, if_s^{lo} is the interface used on the agent running the scheduler to connect to if_s^{re} that is the interface having the skill that is scheduled, and S_s^h is the set of other skills needed for executing s. For example, if a gripper tool should move a part, then that tool might need a robot to execute a skill to move the gripper tool.

3.10.2 Pathfinding for Part Transfer

In Paper F, pathfinding is defined for our multi-agent system. This is used to simplify the manually created process plans so that they do not need to include all transfer steps in a manufacturing system. It also increases flexibility, since the system can dynamically plan based on changes introduced when resources are moved, added or removed. Each part agent first broadcasts to all agents asking them for compatible buffers in the system. When it receives proposals for buffers, it stores them as nodes locally on a graph γ. Later a pathfinder is applied to this graph to find the shortest path through the system. By using a shared schedule, conflicts are avoided among the agents. Thus, each part agent p builds their local graphs γ, defined by:

$$\gamma =< IF_\gamma, \mathrm{T}_\gamma >$$

Where IF_γ is the set of interfaces if that can be used as nodes in the graph for holding the part, and T_γ is the set of transfers τ between those nodes. A transfer is defined by the following tuple:

$$\tau =< if_{pre}, if_{post}, n_s, c_s, if_s >$$

where if_{pre} is the interface that the part is attached to before transferring it, if_{post} is the interface that the part is attached to after transferring it, n_s is the name of the skill s that will transfer the part, c_s is the cost of executing the skill s , if_s is the address to the interface of the skill where s is defined.

4 Evaluation

The evaluation is done by using an industrial scenario and showing how the system would handle various aspects of that scenario.

4.1 Manufacturing Scenario

This scenario describes how a part p is transported from a buffer r_1 to a paint station r_2. This is done by using a gripper r_3 that is attached to a robot r_4, see Figure 23. Similar scenarios have been presented in the appended papers. However, note that the scenario presented in this section has slightly different configuration parameters. This scenario will be used throughout this chapter to make examples.

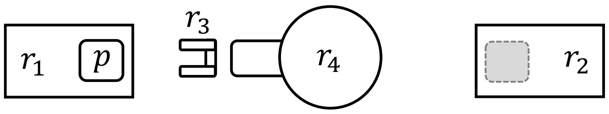


Figure 23: Scenario for transporting a part p from buffer r_1 to a painting station r_2, using gripper r_3, and robot r_4.

In Table 3, the configuration values for the scenario are shown, where the column: Agent refers to the agent that the configuration value belongs, the Interface is the interface to which the configuration value belongs, and the data type describes what format the data is in and the Name is the name of the configuration value. The part has one goal $g = PaintBlue$, that can be solved by the process plan π_g. The description "Input:" in the table is noting that a variable is not configurated with any value on the startup of the system. Instead, this is an input signal on an interface, that will get data from another agent.

Table 3. Configuration values for the scenario.

Agent	Name	Description	Data type	Interface
	π_g	Solves PaintBlue	Process Plan	
p	if_3	BufferInterface	Interface	
p	v_3	BaseLocation	Variable	if_3
p	if_4	GripInterface	Interface	
p	v_4	GripLocation	Variable	if_4
p	g	PaintBlue	Goal	
r_1	if_1	BufferInterface	Interface	
r_1	v_1	BufferLocation	Variable	if_1
r_1	s_1	Buffer	Skill	if_1
r_2	if_2	BufferInterface	Interface	
r_2	v_2	BufferLocation	Variable	if_2
r_2	s_2	Paint	Skill	if_2
r_3	if_5	GripInterface	Interface	
r_3	s_3	Transport	Skill	if_5
r_3	v_7	Input: PickAt	Variable	if_5
r_3	v_8	Input: PlaceAt	Variable	if_5
r_3	v_5	ToolData	Variable	if_6
r_3	if_6	ToolInterface	Interface	
r_3	v_6	BaseLocation	Variable	if_6
r_4	if_7	ToolInterface	Interface	
r_4	s_4	MoveTool	Skill	if_7
r_4	s_5	CloseTool	Skill	if_7
r_4	s_6	OpenTool	Skill	if_7
r_4	v_{10}	Input: To	Variable	if_7
r_4	v_{11}	Input: Tool	Variable	if_7

In Figure 24, the variables for locations 1-4 are shown, as defined in Table 3. These are needed for the agents to know where the interfaces are located. The variables are defined relative to the local agent's coordinate system and are translated into a common reference frame (world coordinates) when communicated from one agent to another.

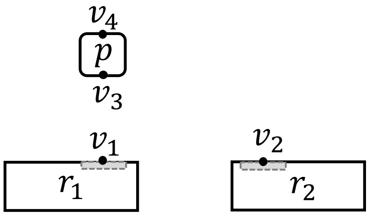


Figure 24: Variables for locations 1-4.

Figure 25 shows interfaces 1-7 defined in Table 3. The lines show how interfaces can be connected.

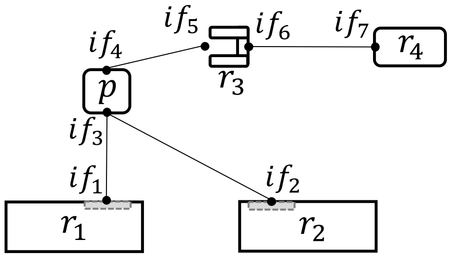


Figure 25: Interface connections are shown with lines for interfaces 1-7.

4.2 Agent Interfaces

When variables are communicated between agents they are first translated into world coordinates. If p wants to give its variable v_4 to r_3 it needs to find out where it is located in the world. Since p knows that if_3 and if_1 are connected (see Figure 25) it is known that v_3 and v_1 are located in the same world coordinate (see Figure 24). Thus, p asks r_1 to give its v_1 in world coordinates to p. Each agent has this functionality to translate coordinates. If r_1 is connected to further agents this will continue in a chain of connected agents translating down to a common reference frame. The idea is that everything needs to be attached to some common physical component, like the manufacturing cell. In this scenario r_4 and r_1 are attached to the same manufacturing cell and their world positions are defined as variables of the respective agents. When p gets the world position of v_3 it will have enough data to find the world position of v_4. This is then communicatied to r_3. Since the robot r_4 is attached to the same manufacturing cell it has a common reference shared with r_1 and r_2. To simplify the scenarios global configuration, the resource r_1, r_2 and r_4 are not attached to anything. Instead, they have their variables v_1, v_2 described in the robots coordinate system. However, let's go through how it would look if the modules were removable. In case the module and robot were removable they might be configured with interfaces as shown in Figure 26, where each of the ten dots represents an interface.

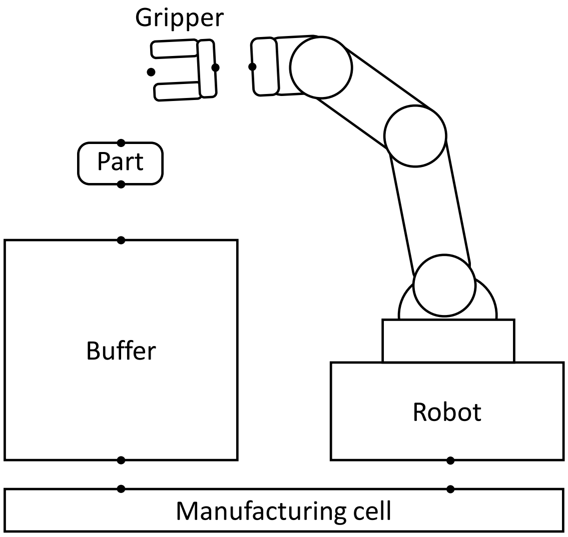


Figure 26: Example of two process modules: one with a part placed on top of it, and another with a robot that can interact with its neighbouring modules.

Here the process modules are connected to a manufacturing cell. The part can tell the robot where it is in world coordinates since it is attached to a buffer that is connected to the same manufacturing cell that the robot is connected to. The robot and the part can figure out where they are in the cells' coordinate system, i.e., world positions by asking the agents that they are attached to for a translation of coordinates. This works in a chain of attached agents so that they translate until they reach a common coordinate system, in this case, the manufacturing cells coordinate system.

Interfaces must match to connect. As described earlier, agents have multiple interfaces, and interfaces have multiple skills. Each interface has defined inputs that describe needed variables to execute any of the presented skills on that interface. The interfaces explained in this section are defined in Table 3. In Figure 27, the part and the gripper have compatible interfaces, since the outputs from the part match the inputs on the gripper, the required skill: Transport exists on the gripper interface and there were no required variables. Additionally, we can see that they are physically compatible since they both share the same interface type.

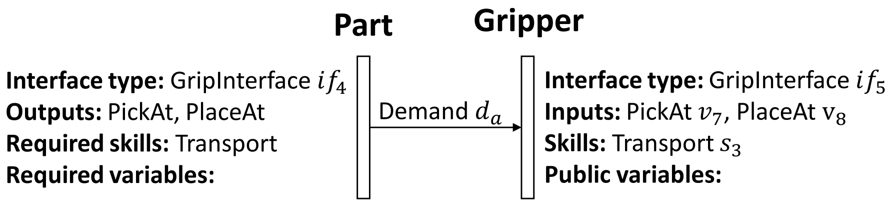

Part
Gripper
Interface type: GripInterface if_4
Outputs: PickAt, PlaceAt
Required skills: Transport
Required variables:
Demand d_a
Interface type: GripInterface if_5
Inputs: PickAt v_7, PlaceAt v_8
Skills: Transport s_3
Public variables:

Figure 27: Example, with part and gripper having compatible interfaces since signals and skills are matching.

Some interfaces have variables that can be accessed by other agents by requesting their current value. In Figure 28, the part checks its compatibility against a paint station. There are no required input or output signals, the skill paint is existing on the paint station and the BufferLocation that the part needs for transportation is publicly available on the paint station.

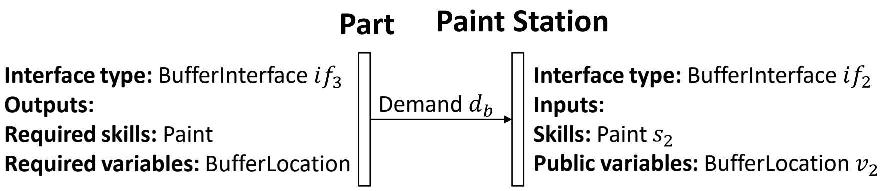

Part
Paint Station
Interface type: BufferInterface if_3
Outputs:
Required skills: Paint
Required variables: BufferLocation
Demand d_b
Interface type: BufferInterface if_2
Inputs:
Skills: Paint s_2
Public variables: BufferLocation v_2

Figure 28: Example, with a part and paint station having compatible interfaces.

In Figure 29, the interface connection between the gripper and the robot is shown. The gripper presents two output signals: To, which is the destination for travel and Tool, which is the tool data for the gripper. Three skills are required and found on the robot ToolInterface if_7.

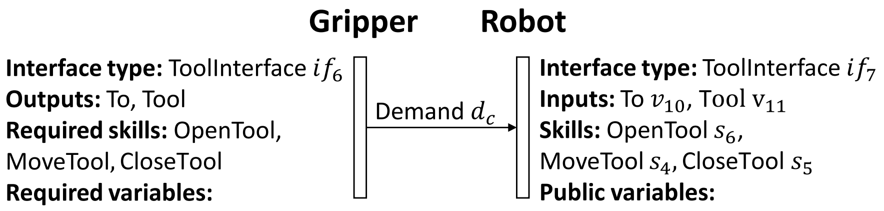

Gripper
Robot
Interface type: ToolInterface if_6
Outputs: To, Tool
Required skills: OpenTool, MoveTool, CloseTool
Required variables:
Demand d_c
Interface type: ToolInterface if_7
Inputs: To v_{10}, Tool v_{11}
Skills: OpenTool s_6, MoveTool s_4, CloseTool s_5
Public variables:

Figure 29: Gripper and robot connecting through an interface.

In the global configuration, there exist nothing called output signals, required skills, or required variables. These are generated automatically from the process plans like π_g or π_{s_3}. This is done to further decrease the number of manual configurations that have to be created. It would, of course, be possible to manually define these on each interface, but there is currently no need. Instead, interfaces are only configured in the global configuration as services that other interfaces connect to.

4.3 Process Plans

Figure 30 shows a process plan π_g that uses a robot gripper and a painting station.

```
{ Abstract interfaces }
Abstract a,b : interface;

{ Main procedure }
Begin
 a.Transport( PickAt := agent.GripLocation,
              PlaceAt := b.BufferLocation);
 b.Paint();
End.
```

Figure 30: Process plan π_g for solving the goal *PaintBlue*.

In π_g two skills are used: Transport and Paint. The letters a and b in front of these skills are used to show that these skills need to run on different resources. The letters are called abstract interfaces since they represent undefined interfaces needed on some resources in the manufacturing system. If the letter a would be used for both skills, that would mean that we are looking for an agent having an interface with both Transport and Paint skills, which is not the case in this scenario.

In Figure 31 the plan π_{s_3} for the gripper's skill Transport s_3 is shown. The gripper has further interaction with a robot as described in the skills process plan in Figure 31.

```
{ Abstract interfaces }
Abstract c : interface;

{ Main procedure }
Begin
 c.OpenTool();
 c.MoveTool(To := GripInterface.PickAt, Tool := agent.ToolData);
 c.CloseTool();
 c.MoveTool(To := GripInterface.PlaceAt, Tool := agent.ToolData);
 c.OpenTool();
End.
```

Figure 31: A process plan π_{s_3} that is written specifically to execute the skill *Transport* for the gripper.

The skills OpenTool, MoveTool, and CloseTool exist on the industrial robot in the same manufacturing cell. Here, the gripper carries its tool data, and forwards the input signals PickAt and PlaceAt positions to the robot, by assigning them to the variable: To.

The required skills and variables are declared directly in the process plan, such as b.BufferLocation in Figure 30, implies that resource b needs to have a BufferLocation variable presented publicly. This variable is then used to give a value to the output signal: PlaceAt. The same concept is used for the output signal PickAt, which takes a local variable GripLocation, that must exist on the part executing this process plan, otherwise, it is not compatible with this plan.

The connections between process plans create a tree of connected interfaces as shown in Figure 32. Here, the Part p has the goal g, solved by π_g. The process plan π_g generates certain demands d_a for an interface a in the agent network and finds the matching interface if_5 on the gripper. Additionally, interface b described in π_g generates the demand d_b. Thus, the part finds the painter resource with interface if_2. The gripper has a process plan π_{s_3} with further demands d_c and finds the robot on if_7.

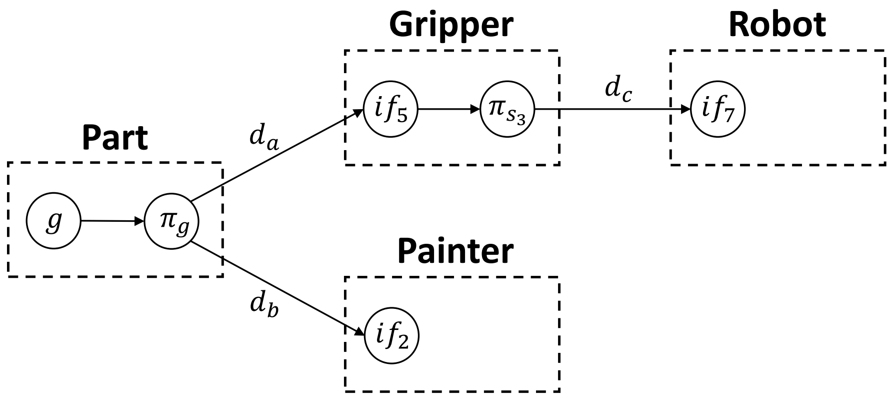
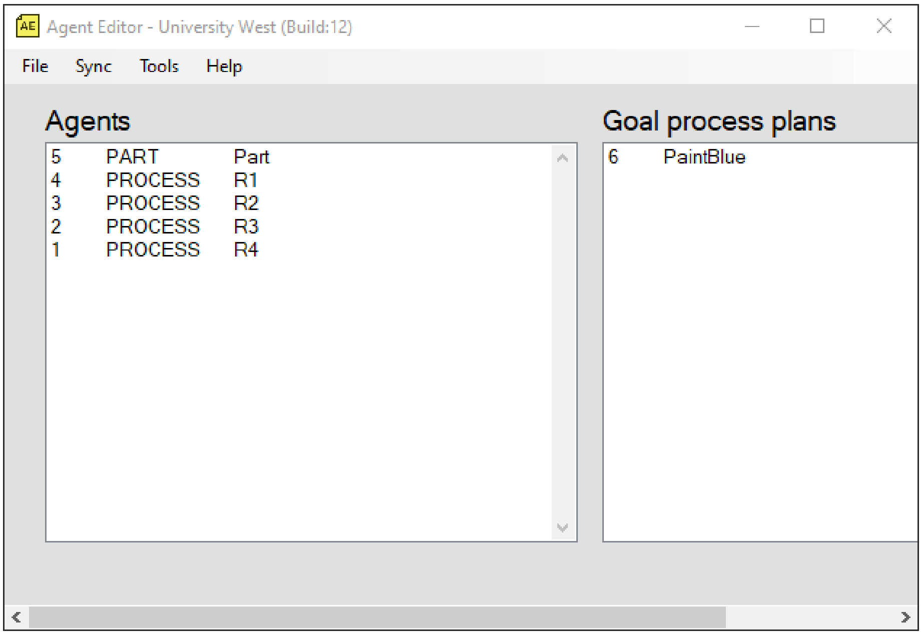


Figure 32: A plan π_g is checked for availability in the agent network.

4.4 Configuration Tool

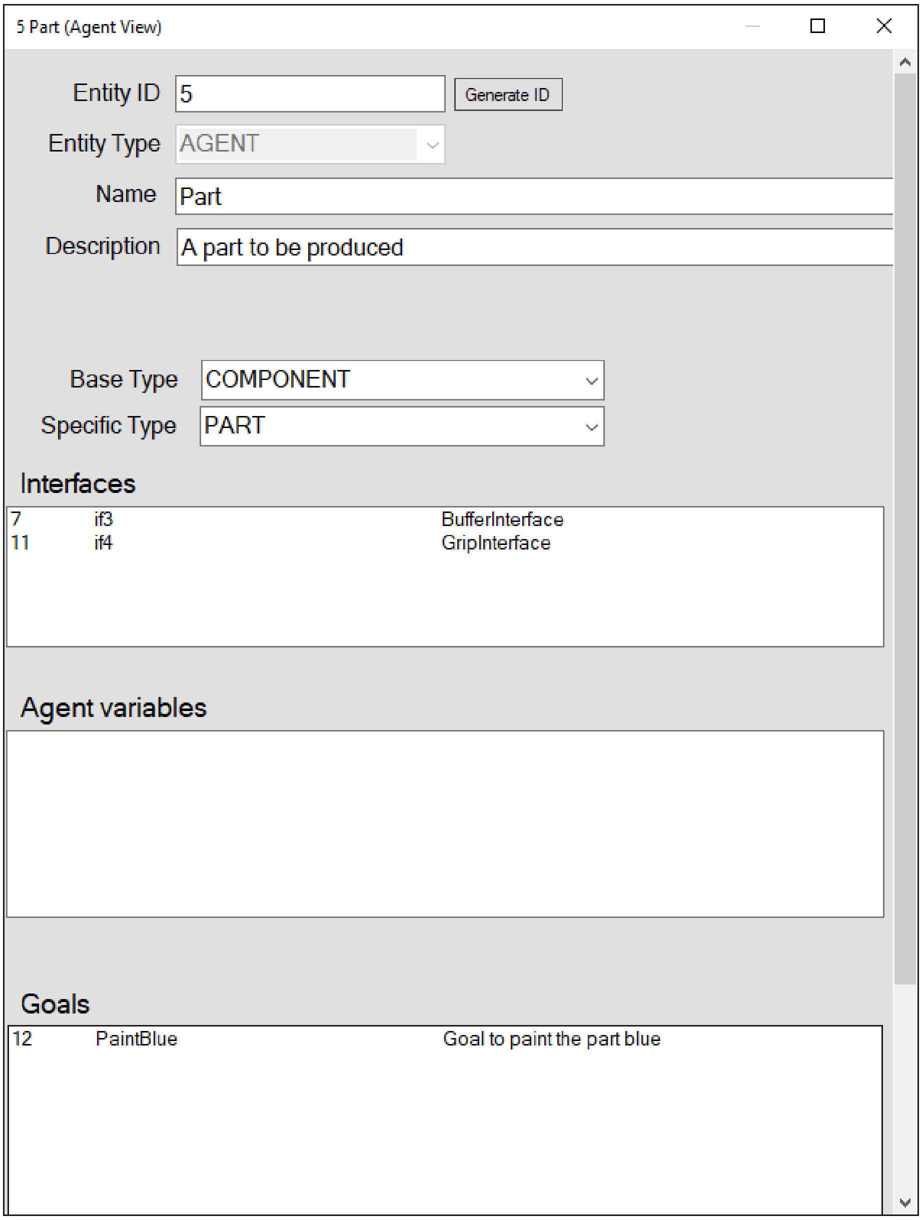
By defining the presented scenario in the designed configuration tool the main view looks as shown in Figure 33, where five agents are shown together with one process plan. In Figure 34 the agent view is shown for the part agent in the scenario, where two interfaces are shown together with one goal. Figure 35 shows the BufferInterface of the resource r_2, that has one location variable and one skill to paint.



5	PART	Part
4	PROCESS	R1
3	PROCESS	R2
2	PROCESS	R3
1	PROCESS	R4

6	PaintBlue

Figure 33: Main view, showing five agents and one process plan.


5 Part (Agent View)

Entity ID	5
Entity Type	AGENT
Name	Part
Description	A part to be produced
Base Type	COMPONENT
Specific Type	PART

Interfaces

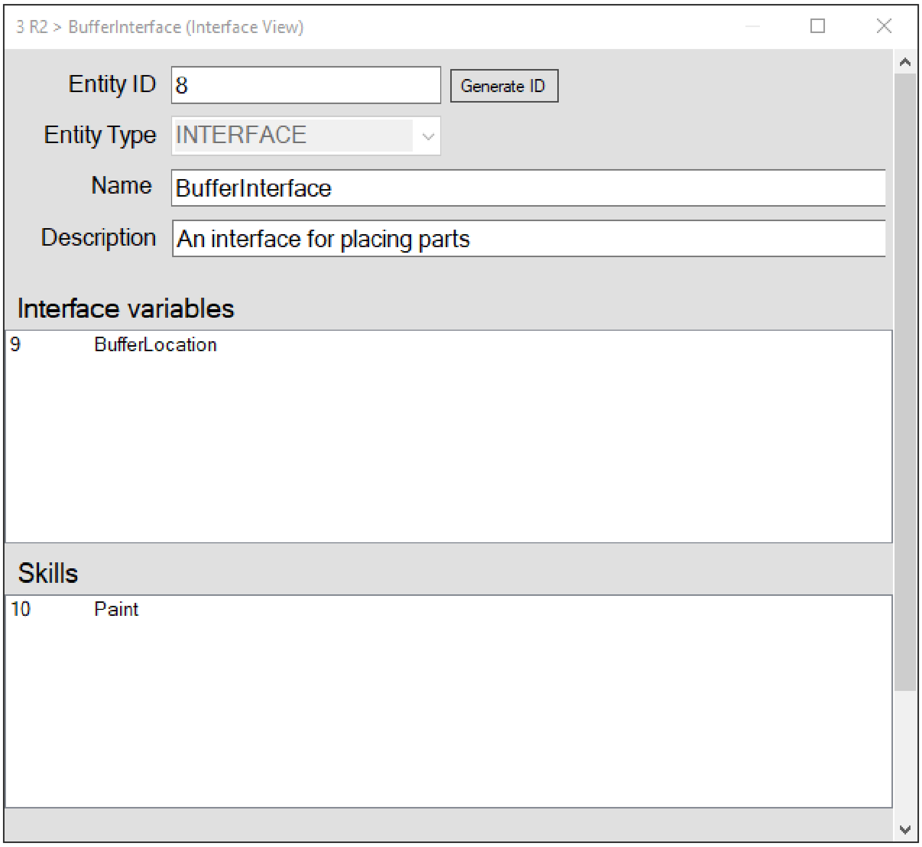
7	if3	BufferInterface
11	if4	GripInterface

Agent variables

Goals

12	PaintBlue	Goal to paint the part blue

Figure 34: Agent view for the part, showing variables, interfaces and goals.


3 R2 > BufferInterface (Interface View)

Entity ID	8
Entity Type	INTERFACE
Name	BufferInterface
Description	An interface for placing parts

Interface variables

9	BufferLocation

Skills

10	Paint

Figure 35: Interface view for the BufferInterface on the Paint station.

4.5 Agent Communication

In Paper C, the conceptual model was evaluated by using a manufacturing scenario that is similar to the scenario presented earlier in this thesis. However, there are some small differences. Paper C presents a list of each communicative step needed for its scenario. To make it easier to follow, these steps have been rewritten to use the configuration values presented in this thesis manufacturing scenario. The specialized communicative acts that were identified in Paper C are listed in Table 6.

Table 4. Specialized communicative acts.

Number	Description
1	Request information
2	Give information
3	Book/Unbook skill
4	Request skill to start
5	Attach/Detach

The following list describes each communication step, from the parts' perspectives. This is the communication needed for the manufacturing scenario presented earlier in this thesis, based on the findings in Paper C:

- (3) p tries to book r_2 if it has a skill $s_2 = Paint.$ The part p is compatible with the interfaces if_2 on r_2 and is therefore booked by $p.$
- (3) p tries to book r_3 if it has a skill $s_3 = Transport.$ The part p is compatible with the interface if_5 on r_3 and is therefore booked by $p.$
- (1) p is attached to r_1 with v_3 attached to $v_1.$ Thus, p asks r_1 to give the variable v_1. Resource r_1 translates v_1 to world coordinates before sending it to $p.$
- (1) To find the location for placing, p asks r_2 for the variable v_2. Resource r_2 translates v_2 to world coordinates before sending it to $p.$
- (2) p uses its grip location v_4 to calculate the pick and place location to move between r_1 and $r_2.$ Then these are sent to the gripper r_3, where they become the input signals v_7 and v_8.
- (4) p requests that r_3 runs the skill s_3
- (3) p unbooks the interface if_1 on r_1
- (5) p tells r_2 that p is attached to if_2
- (3) p unbooks the interface if_5 on r_3
- (4) p requests that r_2 runs the skill s_2

The evaluation shows that it is possible to limit the number of instructions that a system needs to be adapted for new scenarios, by using specialized communicative acts. The reason is that the specialized communicative acts hide the complexity of lower layers (1 and 2) in the conceptual model.

5 Conclusions

Existing approaches for Plug & Produce and multi-agent systems were investigated. It was identified that previous research has successfully created multi-agent systems that divide control logic into several resource modules that can be added quickly to a manufacturing system when needed. The agents are commonly developed by manual programming. However, they are still flexible once the code is written. The code is easier to understand than conventional resources code, due to the low number of dependencies between agents. This reduces the need to understand the complete system complexity when designing each resource. On the other hand, it was also identified that those systems were not commonly used in the industry. The work presented in this thesis has focused on closing that gap, bringing this technology closer to the industry. The main focus has been on decreasing the software development time that is consumed when manually preparing a system for new product designs (changeover time). This thesis shows a proposal for a Plug & Produce framework that addresses the software time by simplifying the steps to adapt a manufacturing system for new product designs. If new resources are needed to be added, removed or relocated, based on new production requirements, the system will handle that so that resources can be reused as much as possible. Also, the distributed concept makes it possible to develop resources without much knowledge about other resources.

Research question RQ1. How can a multi-agent system be designed, to decrease the software time in a Plug & Produce system?:

An agent ontology was created, defining the agent configuration classes and their relations to each other. These classes are evaluated by an implemented configuration tool and an implementation that runs on each agent. Agent strategies were designed that define a general agent behaviour, thus requiring only one single agent code to be developed. This agent code is reused for all agent instances and given its behaviours through configuration data. A Plug & Produce framework was designed and implemented as a configurable multi-agent system C-MAS.

Research question RQ2. When introducing new products and resources, how can functionality for agent collaboration and reasoning be reused to decrease reprogramming time?:

A configuration tool was developed, that gives the user form-based views for defining the complete manufacturing systems behaviour. Also, a method for deploying configurations was designed and tested, which also makes sure that devices are identified once connected. Communication between agents was standardized using a conceptual model, thus, removing the requirement for setting up the communication manually. The communication is instead based on configuration data, given through a configuration tool.

Research question RQ3. How can dynamic planning and scheduling in configurable multi-agent systems be designed for Plug & Produce, which can handle unpredictable events?:

To further decrease the changeover time, methods for planning and scheduling were designed and evaluated. This was done in two ways: by scheduling process plans automatically while avoiding conflicts between agents and by using pathfinding that automatically detects all locations where parts can be placed and generates a graph with all paths between those locations.

Future work:

In future work, the system can be extended with many supporting systems. This includes systems for learning that can be applied to the pathfinder, to enable learning what paths are the best to take. Zone management is something that could be of interest to work with and is strongly related to the scheduling algorithm described in this work. A zone needs to be scheduled to avoid conflicts. Many supporting tools, such as CAD designs could be used for defining products and automatically deploying that to the agent system. A restart of the system at failure, by automatically generating instructions for operators is something that can be developed. A more advanced and mature configuration tool for developing the agents is needed for the industry to accept the proposed system in this thesis. The configuration tool would benefit from including advanced debugging functionality, and templates for agent types to avoid beginning from an empty configuration for each new agent. The method of using views for configurations is useful, but throughout the evaluations, in this work, it has been noted that it quickly becomes difficult to overview the project. Thus, it would be a great improvement if this could be complimented with a way to visualise the interface connections between agents.

6 Summary of Appended Papers

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

The paper presents a framework for implementing the automation controller for Plug & Produce. It is a multi-agent system framework, where resources are assigned skills and parts are given goals. The framework solves the negotiation among agents to reach the given goals with available resources. This makes it possible to work with configurations rather than programming when making changes to the manufacturing system. Using the presented framework, it is possible to configure a robot gripper and related robot separately as individual agents and then to let them find each other by communicating and setting up collaboration automatically.

Paper B. Identification of resources and parts in a Plug and Produce system using OPC UA

In this paper, a method is presented and implemented, that solves automated detection, identification and configuration of added resources and parts in a Plug & Produce system. For each added physical device, a corresponding agent is instantiated based on the physical device type. A corresponding agent configuration is stored in a database, similar to a device driver in Plug & Play for computers. All agents are instantiated in a cloud service running outside the manufacturing system. The communication protocol OPC UA is used for communication between agents in the cloud and the physical device that they are controlling. This enables many industrial devices to be connected to the developed system.

Paper C. A conceptual model for multi-agent communication applied on a Plug & Produce system

The paper presents a new conceptual model for multi-agent communication applied in Plug & Produce. The conceptual model is an extension of the ideas of standardized communication presented by the organization FIPA. The model adds an abstraction layer where communicative acts designed specifically for manufacturing systems can be added. These communicative acts are then reused at a higher layer where the agent configurations are defined. This makes it possible to limit the number of choices an engineer must make.

Paper D. A Method for Configuring Agents in Plug & Produce Systems

A lack of user-friendly tools that hide the complexity of multi-agent technology in the underlying systems has been identified in previous research. Thus, this paper presents a theory and implementation of a configuration tool for multi-agent systems. The tool is developed as a Human Machine Interface (HMI) and aims at being user-friendly. A manufacturing scenario is presented and tested using the developed tool. This shows that it is possible to simplify the steps to adapt a multi-agent system for new manufacturing scenarios.

Paper E. Part Oriented Planning for Unpredictable Events in Plug & Produce

Planning that avoids conflicts among agents is described and tested with a manufacturing scenario. Planning is done individually by each part agent by scheduling each related process plan for the goals of that part. Resource agents have schedules that can be accessed by all other agents. This makes it possible for part agents to avoid adding items to the schedules that would introduce conflicts.

Paper F. Online Generation of Graphs used for Pathfinding in Plug & Produce Systems

Automated pathfinding is necessary for planning the agents' transfer when multiple transfer steps are needed. This paper explains how a graph automatically can be created by letting each agent find its surrounding agents through communication. This online graph creation is what makes the pathfinder suitable for Plug & Produce.

7 References

[1] Y. Zhang, H. Zhu, D. Tang, T. Zhou, and Y. Gui, “Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems,” *Robot Comput Integr Manuf*, vol. 78, p. 102412, Dec. 2022, doi: 10.1016/j.rcim.2022.102412.

[2] E. Ribeiro da Silva, C. Schou, S. Hjorth, F. Tryggvason, and M. S. Sørensen, “Plug &amp; Produce robot assistants as shared resources: A simulation approach,” *J Manuf Syst*, vol. 63, pp. 107–117, Apr. 2022, doi: 10.1016/j.jmsy.2022.03.004.

[3] S. J. Hu *et al.*, “Assembly system design and operations for product variety,” *CIRP Annals*, vol. 60, no. 2, pp. 715–733, 2011, doi: 10.1016/j.cirp.2011.05.004.

[4] S. Mayer, D. Plangger, F. Michahelles, and S. Rothfuss, “UberManufacturing,” in *Proceedings of the 6th International Conference on the Internet of Things*, Nov. 2016, pp. 111–119. doi: 10.1145/2991561.2991569.

[5] T. Blecker and N. Abdelkafi, “Mass Customization: State-of-the-Art and Challenges,” in *Mass Customization: Challenges and Solutions*, Boston: Kluwer Academic Publishers, pp. 1–25. doi: 10.1007/0-387-32224-8_1.

[6] H. A. ElMaraghy, “Flexible and reconfigurable manufacturing systems paradigms,” *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, Oct. 2005, doi: 10.1007/s10696-006-9028-7.

[7] P. Coletti and T. Aichner, “The Need for Personalisation,” in *Mass Customization*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–21. doi: 10.1007/978-3-642-18390-4_1.

[8] Y. Koren *et al.*, “Reconfigurable Manufacturing Systems,” *CIRP Annals*, vol. 48, no. 2, pp. 527–540, 1999, doi: 10.1016/S0007-8506(07)63232-6.

[9] S. Hjorth, C. Schou, E. Ribeiro da Silva, F. Tryggvason, M. Sparre Sørensen, and H. Forbech, “A Case Study of Plug and Produce Robot Assistants for Hybrid Manufacturing Workstations,” 2022, pp. 242–249. doi: 10.1007/978-3-030-90700-6_27.

[10] Z. Pan, J. Polden, N. Larkin, S. van Duin, and J. Norrish, “Recent progress on programming methods for industrial robots,” *Robot Comput Integr Manuf*, vol. 28, no. 2, pp. 87–94, Apr. 2012, doi: 10.1016/j.rcim.2011.08.004.

[11] M. R. Pedersen *et al.*, “Robot skills for manufacturing: From concept to industrial deployment,” *Robot Comput Integr Manuf*, vol. 37, pp. 282–291, Feb. 2016, doi: 10.1016/j.rcim.2015.04.002.

[12] M. Onori, N. Lohse, J. Barata, and C. Hanisch, “The IDEAS project: plug & produce at shop-floor level,” *Assembly Automation*, vol. 32, no. 2, pp. 124–134, Apr. 2012, doi: 10.1108/01445151211212280.

[13] L. Ribeiro, J. Barata, M. Onori, and J. Hoos, “Industrial Agents for the Fast Deployment of Evolvable Assembly Systems,” in *Industrial Agents*, Elsevier, 2015, pp. 301–322. doi: 10.1016/B978-0-12-800341-1.00017-6.

[14] A. Zoitl, G. Kainz, and N. Keddis, “Production Plan-Driven Flexible Assembly Automation Architecture,” in *Industrial Applications of Holonic and Multi-Agent Systems*, 2013, pp. 49–58. doi: 10.1007/978-3-642-40090-2_5.

[15] M. Hvilshøj and S. Bøgh, “‘Little Helper’ — An Autonomous Industrial Mobile Manipulator Concept,” *Int J Adv Robot Syst*, vol. 8, no. 2, pp. 80–90, Jun. 2011, doi: 10.5772/10579.

[16] 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, Jul. 2017, doi: 10.1177/1729881417717472.

[17] 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, May 1999, doi: 10.1080/002075499191157.

[18] A. Nilsson, F. Danielsson, and B. Svensson, “Customization and flexible manufacturing capacity using a graphical method applied on a configurable multi-agent system,” *Robot Comput Integr Manuf*, vol. 79, p. 102450, Feb. 2023, doi: 10.1016/j.rcim.2022.102450.

[19] P. Leitao, V. Marik, and P. Vrba, “Past, Present, and Future of Industrial Agent Applications,” *IEEE Trans Industr Inform*, vol. 9, no. 4, pp. 2360–2372, Nov. 2013, doi: 10.1109/TII.2012.2222034.

[20] P. Leitão and S. Karnouskos, “A Survey on Factors that Impact Industrial Agent Acceptance,” in *Industrial Agents*, Elsevier, 2015, pp. 401–429. doi: 10.1016/B978-0-12-800341-1.00022-X.

[21] 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, doi: 10.1016/j.engappai.2008.09.005.

[22] T. Pulikottil, L. A. Estrada-Jimenez, H. U. Rehman, J. Barata, S. Nikghadam-Hojjati, and L. Zarzycki, "Multi-agent based manufacturing: current trends and challenges," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2021, pp. 1–7. doi: 10.1109/ETFA45728.2021.9613555.

[23] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie, "Applications of agent-based systems in intelligent manufacturing: An updated review," *Advanced Engineering Informatics*, vol. 20, no. 4, pp. 415–431, Oct. 2006, doi: 10.1016/j.aei.2006.05.004.

[24] S. Karnouskos and P. Leitao, "Key Contributing Factors to the Acceptance of Agents in Industrial Environments," *IEEE Trans Industr Inform*, vol. 13, no. 2, pp. 696–703, Apr. 2017, doi: 10.1109/TII.2016.2607148.

[25] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota, "Agile Assembly System by 'Plug and Produce,'" *CIRP Annals*, vol. 49, no. 1, pp. 1–4, 2000, doi: 10.1016/S0007-8506(07)62883-2.

[26] V. Mařík and J. Lažanský, "Industrial applications of agent technologies," *Control Eng Pract*, vol. 15, no. 11, pp. 1364–1380, Nov. 2007, doi: 10.1016/j.conengprac.2006.10.001.

[27] L. Steels, "When are robots intelligent autonomous agents?," *Rob Auton Syst*, vol. 15, no. 1–2, pp. 3–9, Jul. 1995, doi: 10.1016/0921-8890(95)00011-4.

[28] M. Skilton and F. Hovsepian, *The 4th Industrial Revolution*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-62479-2.

[29] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *Knowl Eng Rev*, vol. 10, no. 2, pp. 115–152, Jun. 1995, doi: 10.1017/S0269888900008122.

[30] C. Bădică, Z. Budimac, H.-D. Burkhard, and M. Ivanovic, "Software agents: Languages, tools, platforms," *Computer Science and Information Systems*, vol. 8, no. 2, pp. 255–298, 2011, doi: 10.2298/CSIS110214013B.

[31] M. de Weerdt and B. Clement, "Introduction to planning in multiagent systems," *Multiagent and Grid Systems*, vol. 5, no. 4, pp. 345–355, Dec. 2009, doi: 10.3233/MGS-2009-0133.

[32] A. Shalyto, L. Naumov, and G. Korneev, "Methods of object-oriented reactive agents implementation on the basis of finite automata," in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems, 2005.*, pp. 460–465. doi: 10.1109/KIMAS.2005.1427125.

[33] Anand Srinivasa Rao and Michael P. Georgeff, "BDI Agents: From Theory to Practice," in *Proceedings of the First International Conference on Multiagent Systems*, 1995, pp. 312–319.

[34] I. Kovalenko, E. C. Balta, D. M. Tilbury, and K. Barton, "Cooperative Product Agents to Improve Manufacturing System Flexibility: A Model-Based Decision Framework," *IEEE Transactions on Automation Science and Engineering*, pp. 1–18, 2022, doi: 10.1109/TASE.2022.3156384.

[35] Foundation for intelligent physical agents, "FIPA 97 Part 1 Version 1.0: Agent Management Specification," *FIPA*. 1997.

[36] Foundation for intelligent physical agents, "FIPA Agent Management Specification," *FIPA*. Geneva, Switzerland, 2002.

[37] Foundation for intelligent physical agents, "FIPA Agent Message Transport Service Specification," *FIPA*. Geneva, Switzerland, 2002.

[38] E. Argente *et al.*, "Supporting Agent Organizations," in *Multi-Agent Systems and Applications V*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 236–245. doi: 10.1007/978-3-540-75254-7_24.

[39] S. Siracuse, R. Tomlinson, T. Wright, and J. Zinky, "Experience with Task/Allocation Coordination Primitive for Building SurvivableMulti-AgentSystems," in *2007 International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, Apr. 2007, pp. 40–45. doi: 10.1109/KIMAS.2007.369782.

[40] J. Alberola, J. Such, V. Botti, A. Espinosa, and A. García-Fornes, "A scalable multiagent platform for large systems," *Computer Science and Information Systems*, vol. 10, no. 1, pp. 51–77, 2013, doi: 10.2298/CSIS111029039A.

[41] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood, *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.

[42] Ramesh Patil *et al.*, "The DARPA Knowledge Sharing Effort: Progress Report," *Morgan Kaufman*, 1992.

[43] S. Poslad, "Specifying protocols for multi-agent systems interaction," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 2, no. 4, pp. 15:1-15:24, Nov. 2007, doi: 10.1145/1293731.1293735.

[44] Foundation for intelligent physical agents, "FIPA ACL Message Structure Specification," *FIPA*. Geneva, Switzerland, 2002.

[45] Mehdi Dastani, Amal El Fallah Seghrouchni, Alessandro Ricci, and Michael Winikoff, "Programming Multi-Agent Systems ," in *Fifth International Workshop, ProMAS 2007* , May 2007.

[46] Foundation for intelligent physical agents, "FIPA Contract Net Interaction Protocol Specification," *FIPA*. 2002.

[47] Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers*, vol. C–29, no. 12, pp. 1104–1113, Dec. 1980, doi: 10.1109/TC.1980.1675516.

[48] S. Ramasamy, X. Zhang, M. Bennulf, and F. Danielsson, "Automated Path Planning for Plug & Produce in a Cutting-tool Changing Application," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2019, vol. 2019-Septe, pp. 356–362. doi: 10.1109/ETFA.2019.8869398.

[49] M. Bennulf, B. Svensson, and F. Danielsson, "Verification and deployment of automatically generated robot programs used in prefabrication of house walls," *Procedia CIRP*, vol. 72, pp. 272–276, 2018, doi: 10.1016/j.procir.2018.03.025.

[50] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2000, vol. 2, pp. 995–1001. doi: 10.1109/ROBOT.2000.844730.

[51] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer Math (Heidelb)*, vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: 10.1007/BF01386390.

[52] T. Standley and R. Korf, "Complete Algorithms for Cooperative Pathfinding Problems," Barcelona, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-118.

[53] R. Stern *et al.*, "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks," *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, pp. 151–158, Sep. 2019, doi: 10.1609/socs.v10i1.18510.

[54] J. Li, P. Surynek, A. Felner, H. Ma, T. K. S. Kumar, and S. Koenig, "Multi-Agent Path Finding for Large Agents," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7627–7634, Jul. 2019, doi: 10.1609/aaai.v33i01.33017627.

[55] G. Sartoretti *et al.*, "PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning," *IEEE Robot Autom Lett*, vol. 4, no. 3, pp. 2378–2385, Jul. 2019, doi: 10.1109/LRA.2019.2903261.

[56] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif Intell*, vol. 195, pp. 470–495, Feb. 2013, doi: 10.1016/j.artint.2012.11.006.

[57] D. Silver, "Cooperative Pathfinding," in *Cooperative Pathfinding*, 2005, pp. 117–122. [Online]. Available: www.aaai.org

[58] D. Ouelhadj and S. Petrovic, "A survey of dynamic scheduling in manufacturing systems," *Journal of Scheduling*, vol. 12, no. 4, pp. 417–431, Aug. 2009, doi: 10.1007/s10951-008-0090-8.

[59] K. Z. Gao, P. N. Suganthan, M. F. Tasgetiren, Q. K. Pan, and Q. Q. Sun, "Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion," *Comput Ind Eng*, vol. 90, pp. 107–117, Dec. 2015, doi: 10.1016/j.cie.2015.09.005.

[60] G. E. Vieira, J. W. Herrmann, and E. Lin, "Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods," *Journal of Scheduling*, vol. 6, no. 1, pp. 39–62, 2003, doi: 10.1023/A:1022235519958.

[61] D. Smale and S. Ratchev, "A Capability Model and Taxonomy for Multiple Assembly System Reconfigurations," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 1923–1928, 2009, doi: 10.3182/20090603-3-RU-2001.0556.

[62] E. Järvenpää, M. Lanz, and R. Tuokko, "Application of a capability-based adaptation methodology to a small-size production system," *International Journal of Manufacturing Technology and Management*, vol. 30, no. 1/2, p. 67, 2016, doi: 10.1504/IJMTM.2016.075839.

[63] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.

Appended Papers

Paper A

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

A

Mattias Bennulf, Fredrik Danielsson, Bo Svensson, Bengt Lennartson

Published in
IEEE Transactions on Industrial Informatics

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

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

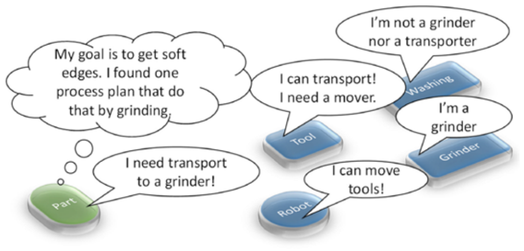

My goal is to get soft edges. I found one process plan that do that by grinding.
I need transport to a grinder!
Part
I can transport! I need a mover.
Tool
I'm not a grinder nor a transporter
Washing
I'm a grinder
Grinder
I can move tools!
Robot

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

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].

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.

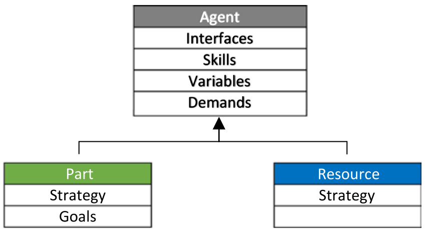


Fig. 2. Diagram with classes for part and resource agents, which have different strategies for part and resource.

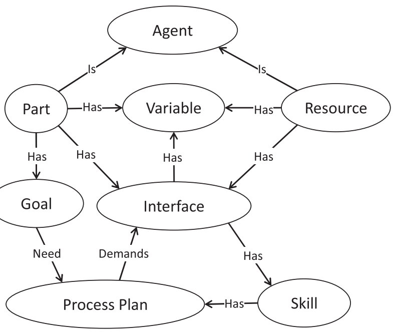


Fig. 3. Diagram, showing the agent configuration classes and their relations to each other, i.e., the agent ontology.

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.

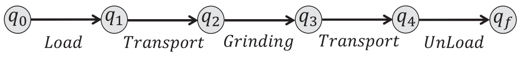


Fig. 4. Example of a process plan with five skills $\{\text{Load}, \text{Transport}, \text{Grinding}, \text{Transport}, \text{UnLoad}\}$ and six states, where the initial state is q_0 and the final state is q_f.

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, $\delta : Q \times S \to Q$ 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

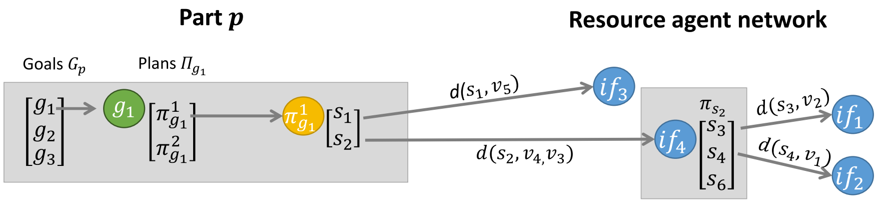


Fig. 5. Process plan that achieves the goal g_1 is checked for availability. The first plan $\pi^1_{g_1}$ requires skills that exist on another agent through the interfaces if_3 and if_4, where if_4 requires additional skills that exist on if_1 and if_2.

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

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

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

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

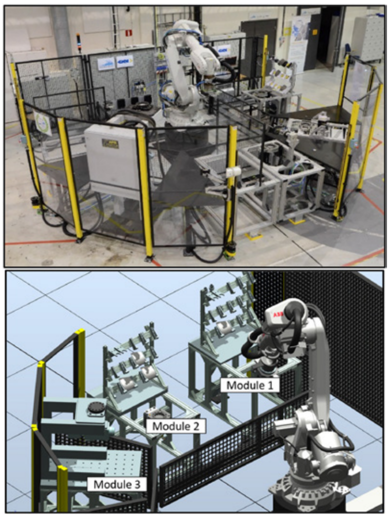


Fig. 6. Real industrial Plug & Produce demonstrator at University West and a simulation model. This demonstrator was used for testing the proposed Plug & Produce framework presented in this article. It was developed in close collaboration with GKN Aerospace, a company producing metal parts for the aeronautics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

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.

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.

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

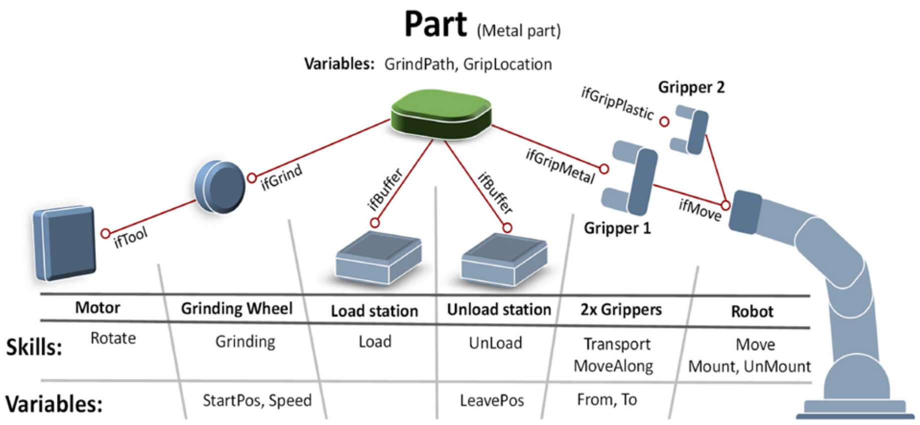


Fig. 7. Using interfaces to connect a part with resource skills. Available interface connections are illustrated by red lines with a circle marking the connection point.

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,

$$I_a^p = \{\text{if Buffer}\}$$
$$I_b^p = \{\text{if GripMetal}\}$$
$$I_c^p = \{\text{if Grind}\}$$
$$I_d^p = \{\text{if Buffer}\}$$
$$I^p = \{I_a^p, I_b^p, I_c^p, I_d^p\}\,.$$

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

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

$$I^s = \{\min(I_a^p), \min(I_b^p), \min(I_c^p), \min(I_d^p)\}$$

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

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

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

Since each element in Π_g is an alternative process plan that can reach the goal $g = \text{SoftEdges}$, the executable process plan $\pi_g^e \in \Pi_g^e$ 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 ($A3$) $\sim$5%, and programming of one device ($A4$) $\sim$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.

Paper B

Identification of resources and parts in a Plug and Produce system using OPC UA

Mattias Bennulf, Fredrik Danielsson, Bo Svensson

Presented at the Flexible Automation and Intelligent Manufacturing International Conference, FAIM, in Limerick, Ireland, June 2019

Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 38 (2019) 858–865

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2019), June 24-28, 2019, Limerick, Ireland.

Identification of resources and parts in a Plug and Produce system using OPC UA

Mattias Bennulf *, Fredrik Danielsson, Bo Svensson

University West, Department of Engineering Science, Trollhättan, Sweden

Abstract

This paper describes a method together with an implementation for automating the detection, identification and configuration of newly added resources and parts in a Plug and Produce system using OPC UA. In a Plug and Produce system, resources and parts are usually controlled by agents, forming a multi-agent system of collaborating resources. Hence, when a resource or part is connected to the system, a corresponding agent must be instantiated and associated with that specific device. In order to automate this, the system needs information about newly connected devices. This information could, for example, be positional data describing where the device is connected. Some devices like tools and parts to be processed have no own network connection, but still, they should get an agent with correct configuration instantiated. In this work, OPC UA is used for communication between devices and the corresponding agents. All agents and their communication are handled by an Agent Handling System, consisting of an OPC UA HUB together with functions for device detection and agent instantiation. The HUB is used for transferring data between devices and their agents in the network by OPC UA protocols. When a device is connected to the network, it is detected, and a connection is automatically created to the HUB that becomes configured for transmitting data between the device and its corresponding agent.

Keywords: OPC UA; Multi-agent; Industry 4.0; Smart Factory; Plug and Produce;

* Corresponding author. *E-mail address:* mattias.bennulf@hv.se

10.1016/j.promfg.2020.01.167

1. Introduction

Today, product lifecycles are decreasing (sometimes down to customized products) [1], resulting in difficulties for factories to maintain profitability, due to the cost associated with rapidly changing dedicated manufacturing equipment [2]. Instead, a trend now is to design automation systems that are reconfigurable for new products by decreasing the time it takes to add new resources to the production. A system that handles this automatically can be regarded as a Plug and Produce system and was firstly introduced in [3].

Adding a resource or a part, i.e., a device, to a system requires three main activities, 1) physically attaching the device to the system, 2) establish a communication to the device (or its representative) and 3) integrating the device in the production from a logical point of view. In this paper, activity 1 is handled by dividing resources into process modules (see Fig. 1 and Fig. 5), that can be connected to the system through standard connectors containing communication, air and power. The production cell at University West, referred to (see Fig. 1 and Fig. 5) and used for the implementation in this work has 10 standard connection slots where process modules can be placed. Further, an industrial robot is also a fixed part of the cell together with a safety system using laser scanners to protect the operators. Using standard modules for sharing hardware has been done before for production systems, e.g., [4], [5]. In other works, such as [6], [7] this type of design is regarded as increasing the mechatronic compatibility. The process module approach in Fig. 1 has been implemented and tested in the physical production cell at University West and has proved to solve the physical flexibility. However, to reach Plug and Produce, the modules also need to automatically be detected and integrated with the production logically. When adding new devices to a communication bus or a network, i.e., activity 2, the network configuration and setup is commonly done manually today and is to be considered as static. An example is that industrial automation devices commonly communicate with shared variables or memory areas and these must be mapped when a new device is connected. In this paper, a platform-independent communication protocol was preferred, rather than traditional vendor specific industrial fieldbuses to reach a general proposal. There exist many platform-independent protocols, however, in this work OPC UA is used due to its wide acceptance in the industry for automation. To handle the integration of Plug and Produce devices in production, i.e., activity 3, a multi-agent-based solution is preferred, where each resource and part has a unique agent representing them. Agents do not necessarily run on the hardware for the devices, it can run somewhere else in the production cell, e.g., on a server, a PLC or even in the cloud.

In this paper, two categories of agents are defined, resource agents and part agents. The agent logic is general and can be used to represent any device, i.e., the same software represents all parts and resources in the system. To prepare an agent for a specific device, an agent configuration is always needed. One agent configuration may be used for instantiating multiple agents, since several devices with identical type can be present in the system, e.g., several parts of the same type to be processed. However, each agent is unique through its instance. The agent configuration contains data describing the connected device physical and logical properties. This includes parameters such as definitions of item position and physical properties like locations for gripping an item or base for placing them on a table.

In this concept, all part agents have goals that they want to reach. A part agent searches the network for resources with the correct skills to assist in reaching those goals. When a new resource is connected, it will be included in the system and becomes visible to other resources and parts. For the process module in Fig. 1 (B) four agents can be identified: the *Cell* agent, the *Process module* agent, the *Part 1* agent and the *Part 2* agent. In this example, the process module is a station for loading/unloading parts. Agents can model their points of interaction with other agents by defining interfaces, shown as dots in the picture with a number for the interfaces local id on each agent. The interfaces, in this case, defines attachment between agents. For Fig. 1 (B), the interface connections are:

1) *Part 1* is attached on its interface 1 to the *Process module* interface 2,
2) *Part 2* is attached on its interface 1 to the *Process module* interface 3 and,
3) *Process module* is attached on its interface 1 to the *Cell* interface 9 (i.e., slot 9).

Resources and parts need positional data when requesting transportation by the robot in the production cell. Their agents can use defined interfaces for calculating its position in the production cell by knowing what it is attached to. It is essential that each agent has a correct description of what it is attached to, all the way down to the *Cell* agent which has an absolute position in the world. When a *Process module* is connected to the production cell, it needs to determine which slot it is placed in, i.e., which of the *Cell* interface 1-10 it is attached to. Similarly, *Part 1* and *Part 2* needs to detect its position on the *Process module*. However, the parts in this case have no network connection and must therefore rely on sensors on the *Process module*.

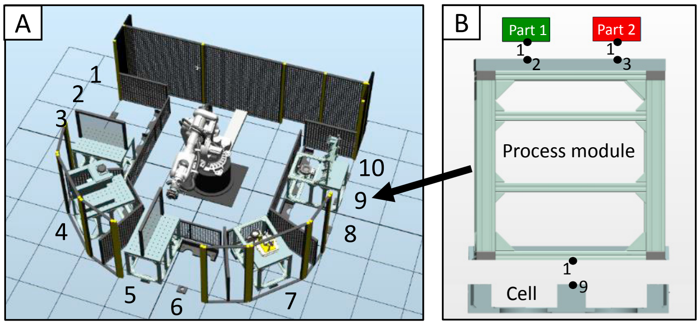


Fig. 1. Simulated production cell with process modules (A) and a close look at one process module that act as a load/unload station (B). An industrial robot with a tool changer is located in the middle of the cell in A.

This paper focuses on activity 2, i.e., a new method for discovering newly added devices in a Plug and Produce system and how to automatically configure the network to make those devices part of the OPC UA framework. A multi-agent system that handles activity 3 has been developed at University West and was used in the implementations described in this paper. This agent system extends a previously developed multi-agent system in [8]. When a device is attached to the system, a related agent should be instantiated and given a correct agent configuration based on what resource or part it is representing and then set up communication between the physical device and its agent. Some of the devices have no direct network connection but still should have a software agent instantiated for representing them.

2. Related work

Agents were described by Wooldridge et al. [9] in 1995. An agent is some software or hardware perceiving the environment and reacting on that. In this paper, parts have production goals to reach. A multi-agent system is a group of agents working together. Several researchers have described that the reason for not seeing multi-agent systems in production today is because the lack of systems and tools to reconfigure the system without understanding the agent-systems complexity [10], [11]. In this paper, this is regarded by focusing on simplifying the connection and setup of new resources and parts.

OPC UA is a platform-independent communication protocol for industrial automation, developed by OPC Foundation [12]. It enables the use of service-oriented architecture (SOA), that is a paradigm for designing software that is loosely coupled, decreasing the dependencies between the softwares in the network. Additionally, using OPC UA, it is possible to model data with object-oriented techniques, which makes communication more sophisticated. OPC UA supports client/server communication as well as publish/subscribe. In [13] automatic device discovery is investigated for OPC UA communication. They extend the OPC UA built-in Local Discovery Server (LDS) to make automatic detection of connected devices on the network. They describe that devices need the IP address to be either manually preconfigured or found by detecting the device on the network automatically. They focus on the automatic approach and use a DHCP server for giving the IP address to the connected device. Then another discovery server independently detects the connected device. Dynamic Host Configuration Protocol (DHCP) is a protocol that can be used to dynamically assign IP addresses to any device connected to a network.

For the work presented in this paper, their solution will not work by itself, since not all devices have a network connection, e.g., part or tool. Also, this paper aims at a system that could include other protocols than OPC UA for legacy devices not supporting OPC UA. This paper uses the DHCP server directly for detecting the device without the need for a discovery server. That makes it possible to detect legacy devices not running OPC UA and connecting to them using other protocols if implemented in the system.

3. Method for identification and agent instantiation

This chapter presents a method for automating the previously described activity 2, i.e., discovering added devices, automatic configuration of the network and instantiation of the corresponding agent representing the added devices. The method describes an Agent Handling System (AHS) that is controlling and interacting with: Agents, Devices on the network and the agent configuration database (Config-DB). A device, could, for example, be a process module that is connected to the network, carrying several agents without a network connection. The Config-DB is a database containing all needed agent configurations that are matching devices connected to the network.

The Agent Handling System method can be divided into six main steps and are shown in Fig. 2 and described in the following list:

1) *Detect new devices in the network:*
   Assign a unique IP address to each new device on the network and store the address in the AHS.
2) *Establish a connection to the new devices:*
   Use the new IP address found to establish an OPC UA connection between the new device and the OPC UA HUB that is a part of the AHS.
3) *Identify possible agents:*
   Use the established connection to identify all resources and parts on the newly attached device and save them in an array *A*.
4) *Get global configuration:*
   For each resource and part in *A* select correct agent configuration from the Config-DB and populate the OPC UA HUB with the new configuration values.
5) *Get local configuration:*
   For each new resource and part get local configurations from the new device and update the OPC UA HUB with local values.
6) *Instantiate new agent program:*
   Instantiate a new agent on an available CPU, using the OPC UA HUB with populated configuration and specific states created in the previous steps.

After these six steps are done, a loop starts in the OPC UA HUB that begins to send data between the physical devices and their corresponding agents, see Fig. 2.

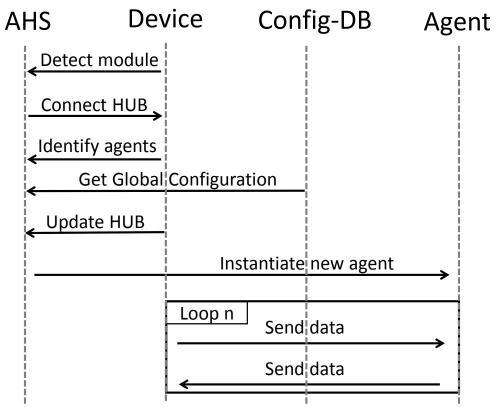


Fig. 2 Method for adding a network device to the Plug and Produce system by using an Agent Handling System.

4. Implementation

In the FIPA97 specification [14], an Agent Management System (AMS) is described as required for managing the agent's life cycles. Similarly, the Plug and Produce system in this paper has an Agent Handling System (AHS), containing an Agent Creator that takes care of agent instantiation.

4.1. Test scenario

The AHS consist of an OPC UA HUB, Agent Creator, DHCP server and an Agent Detector. In Fig. 3, an overview of the implemented Plug and Produce system is presented, with one process module (*Load Station*) carrying two devices (*Part* and *Load*) and another process module (*Motor Station*) carrying a device (*Motor*). The AHS connect the three devices, to the agents in the cloud. Using the strategy of having agents decoupled from the Agent Handling System and devices, implies that they can be placed anywhere in the network or even in a cloud service. This also increases the scalability of the agent concept, which is particularly useful when agents run extensive algorithms with a high computational load. A configuration database is also available containing all agents global configuration templates, related to device types presented on OPC UA Servers on the two process modules.

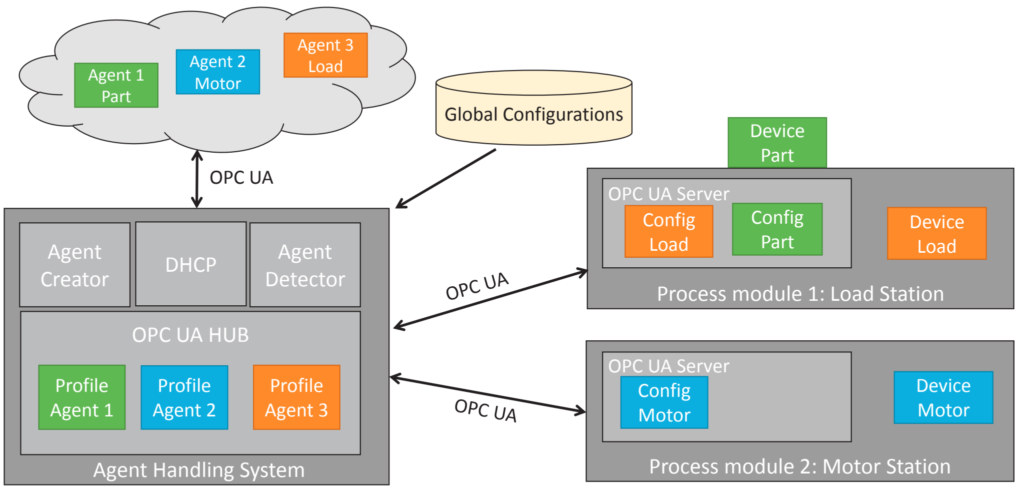


Fig. 3. Test scenario of the Agent Handling System and process modules.

The states of all agents in the production cell are mirrored to the OPC UA HUB in the AHS. Hence, the HUB contains all configuration data for agents and variables that could have changed value since agent instantiation. Each agent in the production cell has a profile in the HUB, created automatically when it is instantiated. The HUB has both OPC UA clients and servers available to be used for communication with the agents and devices, since agents and devices in the network may have either OPC UA server or client. Values in the HUB is synchronized between the agent and the physical world in real-time, making it necessary to map some variables used for communication between agents and their connected devices. Consider the example in Fig. 4 where the *Agent 1* for the *Part* needs a motor for a specific process. The *Agent 1* request the *Agent 2* for the *Motor* to run the skill *StartMotor*. The *Agent 2* for the *Motor* sets the variable $Motor = ON$, the physical motor device is mapped to that variable, so that it does indeed, start the motor. The *Motor* device also sends data to its agent, e.g. *Status: Running*.

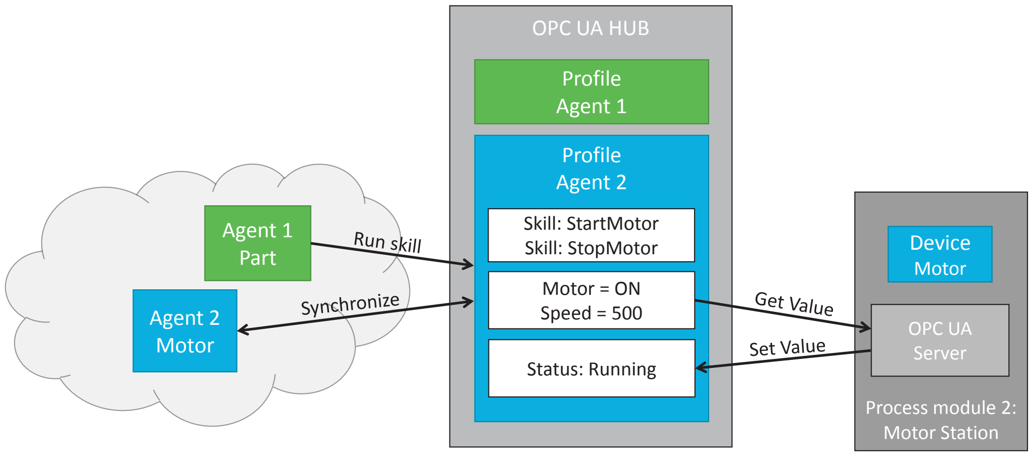


Fig. 4. Example of a motor syncing variables with its related agent.

4.2. Method validation

The method has been implemented and validated in the physical production cell at University West. The implementation details of each step in the described method are explained based on the scenario in Fig. 3, that is corresponding to the real setup in the production cell. For physically attaching the device to the system, standard connectors for the physical connection was used, containing communication, air and power, see Fig. 5.

Step 1) *Detect new devices in the network:* Both process modules 1 and 2 in the production cell, are equipped with a PLC running an OPC UA server. When the PLC is connected to the network using Ethernet, it is detected by the DHCP server and assigned an IP address. The DHCP server is designed to notify the Agent Detector (in the AHS) that a new device is connected. The IP address of the connected device together with the connection slot number is stored for later use in the AHS, to determine at what position it was connected. Hence, the AHS now know that the device is attached to the *Cell* agent.

Step 2) *Establish a connection to the new device:* The AHS establish an OPC UA connection between the process module and the central OPC UA HUB, using the IP addresses assigned by the DHCP.

Step 3) *Identify possible agents:* The OPC UA server on each process module presents information about each device its representing. For process module 1 in Fig. 3, it presents two devices, the *Part* and *Load*. For process module 2 there is only one device, the *Motor*. The *Part* on the *Load Station* have no electronics or network connection, so it needs to rely on the *Load Station* to detect it. The PLC on the *Load Station* is programmed to detect the *Part* when attached to it and present its local configurations on the modules OPC UA server. For each device presented on the modules OPC UA server a related agent should be instantiated by the AHS. To do that, the Agent Detector in the AHS searches each module discovered in step 1, to determine if that module needs any agents to be instantiated. Some devices could be connected for other purposes in the network and they should in this step be filtered out.

One way for the *Load Station* to detect the *Part* is to attach a QR code or RFID tag to the device, encoded with information that the module can read. Another approach is to predefine holders for loading specific part types to the system, then the PLC uses a sensor with Boolean values and is preconfigured with information to put into the OPC UA server on the module. In both approaches, the PLC adds the position of the *Part* to its OPC UA server. Hence, the AHS can get the data about where the *Part* was attached to the process module, (in case that there are several slots for placing parts). Hence, if connecting a process module carrying parts already attached to it, then those parts will be recognized by the readers and agents instantiated. In this way, parts can be removed and added to the process module even when it’s not connected to the production cell.

Step 4) *Get global configuration:* The agent creator now has all the information it needs and selects the correct agent configuration from the database of configurations and populates the OPC UA HUB with the new configuration values.

Step 5) *Get local configuration:* Update the OPC UA HUB with the local values like position fetched from the modules earlier. Additional values could be fetched in this step such as specific part identification number, for tracking device through the production or any other configuration values that deviate from the global configuration in the database.

Step 6) *Instantiate new agent program:* Configurations uploaded into the HUB needs to have one or more agent running somewhere in the network since the HUB only moves data and has no agent logic. The AHS has functions for searching the network to find possible servers that can be used for instantiating and hosting agents. Configurations in the HUB without any related agent instance is assigned a CPU in the network by the AHS. The configuration is used to create the new agent instance on the selected server. Finally, the agents on the servers start to sync data from and to the OPC_UA HUB. The HUB has a function that continuously goes through each agent’s profile and synchronizes it with the devices OPC UA servers.

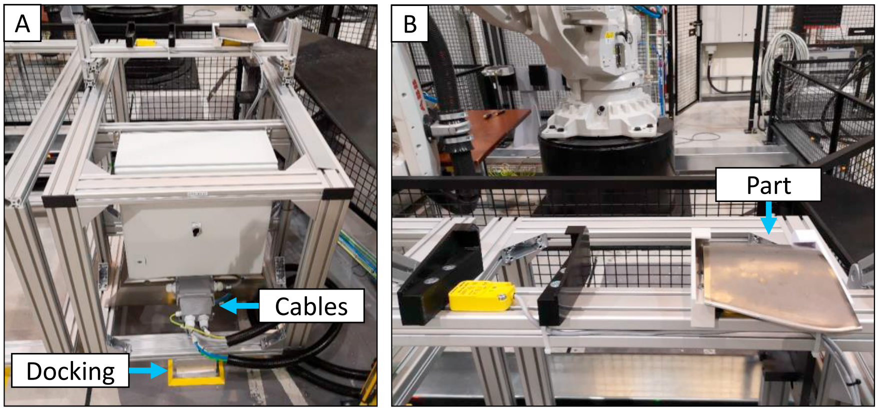


Fig. 5. Image A shows a process module (*Load Station*) from GKN Aerospace in the production cell at University West. The module has physical docking to the floor and cables containing communication, air and power. Image B shows a closer view of the two loading slots for parts.

The six steps described in the method was implemented and tested in the physical production cell. The *Part* in Fig. 5 (B) could be detected by the process module and published on its OPC UA server. The AHS instantiated the newly added devices and connected them to the OPC UA HUB.

5. Conclusion

In this paper, a new method is described that automate the detection, identification and configuration of newly added devices, i.e., resources and parts, in a Plug and Produce system. This was done by developing an Agent Handling System, that can control and interact with agents, devices and the global configuration database. Each device connected to the network in the system has an OPC UA server. Devices without a network connection, such as parts and tools can be detected by letting another device, connected to the network (such as process modules) present the types of devices that are attached to it. The Agent Handling System has the ability to detect newly added devices in the network and searches their OPC UA servers to identify presented devices to choose which agents to be instantiated, based on the device types presented. The instantiated agents are then automatically connected to the newly attached devices. The method was implemented and tested in the physical production cell at University West, focusing on a scenario with two process modules. The implementation had successful results, showing that the detection, identification and configuration are possible to automate using the developed Agent Handling System, which will decrease the time it takes to connect a new device in a Plug and Produce system. The production cell at University West has several laser scanners to protect the operator from robot movements. This made it necessary to stop the robot whenever a part or module was added. In future work it would be of interest to consider the safety systems in the production cell, to enable the human operator to add parts without interruption the ongoing production.

Acknowledgments

We thank GKN Aerospace for supporting and allowing us to work with their process modules that was used for this paper.

References

[1] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing,* vol. 37, pp. 282-291, 2015.

[2] Z. Pan, J. Polden, N. Larkin, S. V. Duin and J. Norrish, "Recent progress on programming methods for industrial robots," *Robotics and Computer-Integrated Manufacturing,* vol. 28, no. 2, pp. 87-94, 2012.

[3] T.Araia, Y.Aiyama, Y.Maeda, M.Sugia and J.Otaa, "Agile Assembly System by "Plug and Produce"," *CIRP Ann Manuf Technol,* vol. 49, no. 1, pp. 1-4, 2000.

[4] M. Hvilshøj and S. Bøgh, ""Little Helper" - An Autonomous Industrial Mobile Manipulator Concept," *Int J Adv Robotic Syst,* vol. 8, no. 2, pp. 1-11, 2011.

[5] M. Onori, N. Lohse, J. Barata and C. Hanisch, "The IDEAS project: plug & produce," *Assembly Automation,* vol. 32, no. 2, pp. 124-134, 2012.

[6] L. Ribeiro, J. Barata, M. Onori and J. Hoos, "Industrial agents for the fast deployment of evolvable assembly systems," in *Industrial Agents*, Elsevier, 2015, pp. 301-322.

[7] A. Zoitl, G. Kainz and N. Keddis, "Production Plan-Driven Flexible Assembly Automation Architecture," in *Industrial Applications*, Springer, 2013, pp. 49-58.

[8] B. Svensson och F. Danielsson, "P-SOP – A multi-agent based control approach for flexible and robust manufacturing," *Robotics and Computer-Integrated Manufacturing,* vol. 36, pp. 109-118, 2015.

[9] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review,* vol. 10, no. 2, pp. 115-152, 1995.

[10] P. Leitao, 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.

[11] P. Leitao, "Agent-based distributed manufacturing control : A state-of-the-art survey," *Engineering applications of artificial intelligence,* vol. 22, no. 7, pp. 979-991, Oct. 2009.

[12] W. Mahnke, S.-H. Leitner and M. Damm, OPC unified architecture, Springer Science & Business Media, 2009.

[13] S. Profanter, K. Dorofeev, A. Zoitl and A. Knoll, "OPC UA for Plug & Produce: Automatic Device Discovery using LDS-ME," in *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol, Cyprus, 2017.

[14] *FIPA 97 Part 1 Version 1.0: Agent Management Specification,* Foundation for intelligent physical agents, 1997.

Paper C

A conceptual model for multi-agent communication applied on a Plug & Produce system

Mattias Bennulf, Fredrik Danielsson, Bo Svensson

C

Presented at the CIRP Conference on Manufacturing Systems, CIRP CMS, in Chicago, IL, U.S., July 2020

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 93 (2020) 347–352

www.elsevier.com/locate/procedia

53rd CIRP Conference on Manufacturing Systems

A conceptual model for multi-agent communication applied on a Plug & Produce system

Mattias Bennulf*, Fredrik Danielsson, Bo Svensson

Production Systems, University West, Gustava Melins gata 2, Trollhättan, 46132, Sweden

* Corresponding author. *E-mail address:* mattias.bennulf@hv.se

Abstract

Today, multi-agent systems are still uncommon in the industry because they require more time to be implemented than traditional manufacturing systems. In this paper, a conceptual model and guidelines are defined for communication and negotiation between agents for Plug & Produce systems. Standards for agent communication exists today, such as the FIPA collection of specifications. However, FIPA is a broad and general standard for any kind of system and leaves a lot of room for interpretation. This paper presents a new conceptual model and guidelines on how to simplify the implementation phase by limiting the choices an engineer must make when implementing a multi-agent system for a manufacturing system.

Keywords: Multi-agent; Cyber-physical; Industry 4.0; Smart Factory; Plug & Produce;

1. Introduction

The demand for customized products and low volume production is increasing [1]. Due to the costs associated with changing dedicated manufacturing equipment, it is difficult to adapt to these new trends [2]. An alternative solution is reconfigurable systems that can decrease the time and cost it takes to add new resources or product types to ongoing manufacturing. These typically have limitations since they focus on hardware rather than software. To further decrease the cost for adapting the manufacturing, the software has to be taken into consideration by designing systems such as Plug & Produce, that was introduced in [3].

One approach for Plug & Produce is to create a cyber-physical system using a multi-agent system where each part and resource in the system is controlled by a piece of autonomous software called an agent. Together these agents collaborate and solve manufacturing goals. However, multi-agent systems are still uncommon in the industry due to the lack of standards and guidelines for implementing those systems. The result is that multi-agent systems demand more time and cost in research and development than traditional manufacturing systems. This makes multi-agent systems an expensive solution to implement in the industry today.

To make multi-agent systems available for companies to use, there have to be clear guidelines on how to implement such systems for manufacturing. Guidelines need to include a complete description of how to design a multi-agent-based manufacturing system. In this paper, we address this by proposing a conceptual model for communication and negotiation between agents for Plug & Produce systems. Currently, there exist standards for agent communication, such as the FIPA 97 [4], that was later updated in 2002 to the current FIPA specifications [5]. These standards were developed by the Foundation for Intelligent Physical Agents (FIPA) [9]. However, this standard is general, aiming for all kinds of agent systems. This leaves the engineer with too many options and decisions to make.

When building the core software for agents there are many things to consider. There is communication, negotiation, optimization, booking and much more. It can quickly become extremely complex to develop these systems. Guidelines

10.1016/j.procir.2020.04.004

designed specifically for communication between agents in Plug & Produce systems could help to simplify this by limiting the choices one has to make to design such a system. The guidelines need to include detailed information on how the communication between agents is supposed to be implemented. This paper presents a conceptual model and guidelines for agent communication together with a physical implementation in our labs. The conceptual model was developed, separating multi-agent communication in four different layers. The implementation generates further suggestions on how the agent environment, such as the physical flexibility and network layout should be designed.

2. Background

A multi-agent system consists of multiple agents. Agents were described by Wooldridge et al. [6] in 1995. They are pieces of software that run independently. They perceive the world, using inputs and reacts to it using its outputs. They can also have their own goals to solve. This is somewhat different from a traditional computer function, where a certain response is always expected. An agent can say no to a request or suggest another solution. This is what makes them autonomous, see Fig. 1.

Agent
Sensing
Reacting

Fig. 1. An agent sensing the environment and reacting based on an internal decision.

Agents can be used for implementing a cyber-physical system since they are a virtual representation of the physical objects. They help to clearly separate the cyber components from the physical world. In a multi-agent system, several agents collaborate by communicating and negotiating to reach manufacturing goals, see Fig. 2.

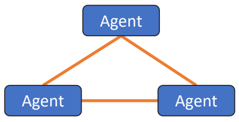


Fig. 2. A network of multiple agents that together makes a multi-agent system.

Cyber-physical systems are about combining cyber components with physical components. Cyber components are regarded as the computation and software components, while the physical components are the plant and process. Cyber components and physical components might be connected through communication networks.

To be able to communicate, agents all need to speak a common agent language. Agent communication standards exist, the FIPA specification is today widely used in research [5]. A multi-agent system can be used to create a Plug & Produce system with high flexibility and reconfigurability. Many researchers have worked with developing various types of agent systems. However, the use of these systems is still extremely uncommon in the manufacturing industry. Other researchers have identified that the main reason is the lack of a standardized abstraction layer that hides the agent complexity from the developer [7], [8]. To implement an agent system can be more time-consuming than to implement a traditional control system. A standardized abstraction layer, reuse of agent code and simplified industrial adapted configuration tools are ways to overcome the implementation cost.

FIPA or the Foundation for Intelligent Physical Agents is an IEEE organization developing standards for multi-agent systems [9]. FIPA defines a collection of specifications, including an Agent Communication Language (FIPA ACL) describing how communication is performed between agents in a multi-agent system [10] together with a Communicative Acts Library (FIPA CAL). FIPA also specifies an Agent Management Specification, describing general guidelines for designing a multi-agent system [5].

2.1. FIPA ACL

The FIPA ACL specification describes the message structure of agent communication [10]. An ACL message usually contains the message parameters: *sender*, *receiver*, *content* and *performative*. FIPA also describes many more message parameters, not listed in this paper. The *performative* describes the communicative act that the message is related to.

2.2. FIPA CAL

FIPA CAL introduces the concept of communicative acts between agents. Communicative acts are used to categorize different types of communication [11]. FIPA defines 22 different communicative acts shown in Table 1. These are not customized for manufacturing systems, making it time-consuming to implement new Plug & Produce systems using this library.

Table 1. Communicative acts in FIPA 2002.

Communicative act	Description
Accept proposal	Accept a submitted proposal
Agree	Agree to perform some action
Cancel	Cancel an action
Call For Proposal	Request proposals
Confirm	Confirms a proposition
Disconfirm	Disconfirm a proposition
Failure	Inform that an action failed
Inform	Inform about a proposition being true
Inform If	Inform if a proposition is true
Inform Ref	Asks for value of expression
Not Understood	Did not understand message
Propagate	Asks agents to forward this message
Propose	Send a proposal
Proxy	Ask agent to act as proxy
Query If	Ask agent if proposition is true
Query Ref	Ask for an object
Refuse	Refuse to perform action
Reject Proposal	Rejecting a given proposal
Request	Request agent to perform action
Request When	Request when proposition is true
Request Whenever	Always run when proposition is true
Subscribe	Let other agent send updated data

2.3. Agent Management

In FIPA Agent Management Specification [5] an Agent Platform (AP) is introduced. This platform provides an infrastructure for deploying agents. The platform consists of the computational hardware, agents and FIPA components. The components are the Directory Facilitator (DF), Agent Management System (AMS) and the Message Transport Service (MTS).

Director Facilitator: This component is acting as a "yellow pages" service where each agent can list their skills, in order to help other agents to find them.

Agent Management System: The AMS gives the agents a unique identification number when registered with the AMS. There can only exist one AMS in a single Agent Platform (AP).

Message Transport Service: This is the communication channel used for all agents to communicate with each other [12].

3. Agent communication

Today it is far too complex and expensive to design a multi-agent system in a profitable way for a manufacturing system, due to the lack of standards and tools that are easy enough to use. To overcome part of this, it is possible to let the software agents be based on one single agent class [13]. The agent class contains all methods for negotiation among other agents and strategies to solve the personal goals of the agents. This agent class can be instantiated as an object for each product and resource in the system.

3.1. Re-configuration

All agents in the system should be configured rather than reprogrammed to avoid costly implementation tasks. Resources have skills while parts have goals. All skills in the system are $S = \{s_1, s_2, \dots, s_{n_S}\}$, where $s \in S$. A resources r has its own subset of skills $S_r \subseteq S$. All goals in the system are $G = \{g_1, g_2, \dots, g_{n_g}\}$, where $g \in G$. A part p has its own subset of goals $G_p \subseteq G$.

The agent source code is never changed and can be considered static. Hence, the behaviour of an agent depends highly on its configuration. When configuring an instantiated agent, variables are also added with parameters. For a part p with the goal $g = PaintPartRed$, it would need a variable with the positions where a gripper can pick it up together with geometry data such as its weight and size. Then p could use these variables to find a resource in the system that can perform $s_1 = Transporting$ to a station with the skill $s_2 = Painting$. In this way, the system never needs to be reprogrammed but instead reconfigured for new scenarios.

Interfaces: When two agents collaborate, they need a shared understanding of points of interaction, e.g., when a robot tool is to be attached to a robot, they both need to know if they are compatible. This can be solved by defining interfaces for interaction, i.e., a compatible interface has to exist between two agents in order to carry out a specific skill that is going to be executed. Interfaces are also part of the agent configuration.

Process plans: For the industry, it is usually not desirable to have unpredictable systems, but rather extremely reliable systems. Because of this, it is not suitable to let the agents figure everything out by themselves. Instead, process plans need to be defined by human experts and given to the agents. Process plans should be abstract representations without too many details. Several process plans can be created for solving the same goal. Process plans should be written as recipes, rather than a program.

3.2. Semantics

Agents need to have a common language for communication. In FIPA ACL there exist many guidelines on how to set up communication and how to send data between agents. As described earlier, this includes a set of pre-coded communicative acts (co-acts), e.g., *Inform*, *Request* and *Agree* [14]. These types of co-acts are made to be general for any agent system and not specifically for manufacturing systems. What is not described in this specification is the specialized co-acts for manufacturing systems, such as booking a resource, starting a process or requesting an agent to translate coordinates. In this paper, the specialized co-acts are separated from the general co-acts making it easier to change the specialized co-acts.

Additionally, agents need to share a common naming standard about interfaces, skills and variables names. These semantics must be defined in each agent so that they have a common understanding when communicating. Semantics should be configured rather than programmed into the agents so that they can be changed and adapted to specific processes without re-programming. Researchers have previously suggested that agent-based solutions should be created as "black boxes" with simple configuration tools hiding the complexity of the agents from the user [8].

Variables such as $PickLocation = (x, y, z)$ describe a location. The knowledge about what this variable is representing can be regarded as semantics that is configured into the agent system. It is not enough to know that this is a location variable since there could be a location both for picking and another one for placing, e.g., $PlaceLocation = (x, y, z)$ in the same agent. This approach requires the user to maintain a strict and rigid standard for naming the variables. Additionally, a skill $transport$ describes functionality that needs to be understood by each agent using that skill. To support the user to preserve a strict consistent configuration, tools must be designed that handle the semantics. A promising approach is to have a database describing the semantics with all the required name definitions and descriptions in a human-readable way. Then when using the configuration tools users will get suggestions and the possibility to see the description for a given signal or skill. If creating a global standard for several companies, then agent configurations would be compatible with all those companies systems. This would make it possible to move a resource, e.g., a robot from one company to another without reconfiguring anything.

An example of semantics that is part of the agent language is if an agent a_1 asks agent a_2 if it has the skill $s = transport$. Then a message would be sent written in an agent language from a_1 to a_2 containing something like "Have skill: $transport$?". This sentence consisting of three keywords: "have", "skill" and "? " has to be understood in the same way on both agents and the semantics for these must be understood in both agents. The semantics becomes a part of the agent language while the content, e.g., the skill name is not part of the language, but rather something the agents talk about using the language. Both systems need to understand this syntax to be able to know the meaning of "have", what a skill is and the understanding of what a question is. This is especially of importance in agent systems where agents use a language more similar to humans languages.

3.3. Network design

It can be debated whether the actual software for the agents should run close to the physical device, e.g., in a robot controller or if it should be running in a cloud service. The benefit of distributing software agents on multiple hardware is that it decreases the risk of single-point failure. However, it is common in multi agent-systems to still have a central point of connection, e.g., "yellow pages" for looking up other agents.

For Plug & Produce systems it would be desirable to have a standardized connection between the cyber and physical components, such as OPC UA developed by OPC Foundation [15]. Together with this an automatic detection of connected physical components and the corresponding agent in cyberspace is needed. In this paper, OPC UA was used for connecting all resource agents with their physical resources. This approach has previously been implemented and verified in [13].

3.4. Communication layers

Four layers of communication can be identified. The first two layers identified already exists in previous implementations, while layers three and four are new and part of the contribution of this paper, see Fig. 3.

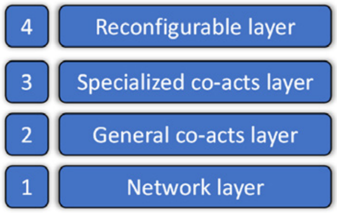


Fig. 3. The conceptual model with four layers, describing multi-agent communication.

The first layer (1) is a basic network protocol used to set up a channel for communication between agents, e.g., OPC UA or Data Distribution Service (DDS). The second layer (2) is the general communicative acts layer that is working for any type of agent system. Here, FIPA has developed a de facto standard describing how this could be implemented. The third layer (3) is application dependent. In this paper, it adds specialized co-acts for manufacturing systems. These are static and hidden from the other layers. It only needs to be updated if an entirely new concept is introduced that the agents need to communicate about, e.g., if an assembly application was not considered when designing the agent code, it could be necessary to add a new specialized co-act where and agent a_1 can inform an agent a_2 that they are now merged. However, most of the time any changes to a manufacturing system only affect layer four (4), the reconfigurable layer, e.g., when a new resource or new product is introduced the only task is to reconfigure the system on layer four. The specialized co-acts are implemented as static skills existing on each agent instance in the system.

4. Implementation

This chapter evaluates the conceptual model described and introduced in this paper with a given scenario. The scenario will be tested with an implementation based on the presented concepts in this paper.

4.1. Manufacturing scenario

This section describes a scenario where a part p is transported from a buffer r_1 to a paint station r_2, using a gripper r_3 that is attached to a robot r_4, see Fig. 4.

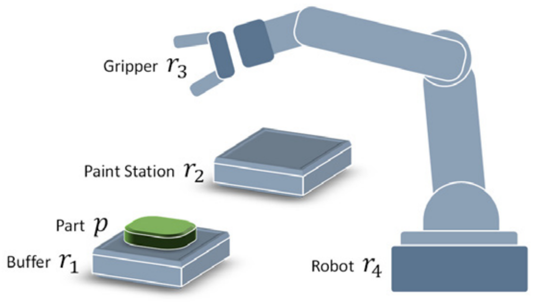


Fig. 4. A manufacturing scenario, transporting a part p to a painting station r_2 from buffer r_1.

In Table 2, all configuration values needed for this scenario are shown.

Table 2. Configuration values in reconfigurable layer (4).

Agent	Description	Data type	Name
p	Grip location	Location	v_2
p	Base location	Location	v_3
p	Paint location	Location	v_4
p	Paint part red	Goal	g
p	Buffer interface	Interface	if_6
p	Grip interface	Interface	if_7
r_1	Buffer location	Location	v_5
r_1	Buffer interface	Interface	if_3
r_2	Buffer location	Location	v_1
r_2	Buffer interface	Interface	if_1
r_2	Paint skill	Skill	s_2
r_3	Grip interface	Interface	if_2
r_3	Tool interface	Interface	if_4
r_3	Transport skill	Skill	s_1
r_4	Tool interface	Interface	if_5

The part p has one goal $g = PaintPartRed$ and three location variables v_2, v_3 and v_4. In Fig. 5, these variables are shown. The variable v_2 is a "grip location" where it is suitable to lift the part using a robot tool. v_3 describes a point underneath p that connects with the resource it is placed on. v_4 is a coordinate system used for painting.

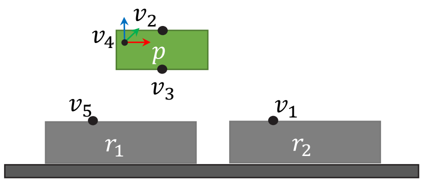


Fig. 5. Location variables for agents.

Location variables are defined locally on each agent for resources or parts. They must be translated into world coordinates before shared with other agents. In Fig. 5, the part p is placed on the resource r_1 and requests transport to r_2. The variable v_2 for gripping has to be translated before communicated to the robot gripper with transport skill s_1. This can be done by using the base location v_3. All agents have built in functionality to translate coordinates before communicating them to other agents. This assumes that the buffer r_1, paint station r_2 and robot r_4 all have been calibrated relative to some point defined in the world coordinate system to be able to calculate where the part p is located. Thus, the buffer locations v_5 and v_1 has to be described in the same reference system as the robot utilizes for transportation. However, note that the locations on p must be relative to where it is placed. Further, the interfaces $\{if_1, if_2, \dots, if_7\}$ is used to know that the part can physically be attached to the surface of the resources, i.e., they are mechanically compatible, see Fig. 6. The interfaces are discussed more in detail in [13].

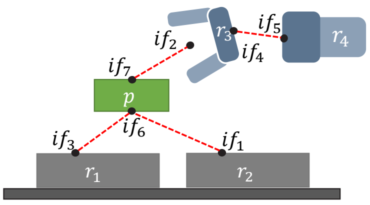


Fig. 6. Interfaces, defining mechanical compatibility between agents.

In this scenario, the part p can communicate with r_1, r_2 and r_3. The resource r_3 communicates further with r_4. This communication layout is shown in Fig. 7.

$$\begin{tikzcd}
 & r_1 & \\
p \arrow[ur, leftrightarrow] \arrow[r, leftrightarrow] \arrow[dr, leftrightarrow] & r_3 \arrow[r, leftrightarrow] & r_4 \\
 & r_2 &
\end{tikzcd}$$

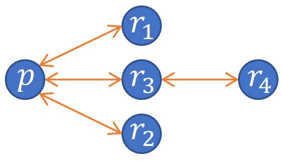


Fig. 7. Agent communication layout.

A process plan π is defined for solving the goal $g = PaintPartRed$. The plan describes two steps 1) transport part to paint station 2) paint part. The physical resources for transportation and paining are unknown when defining π. Hence the system must find matching resources before executing the process plan π. After the system has found these resources, the following two steps will be generated:

1. p is transported by r_3 from r_1 to r_2.
2. p is painted by r_2.

The robot is not part of the process plan since it is not known by the part p. Instead, it is the resource r_3 that collaborates with r_4.

4.2. Experiment

A multi-agent system was developed to evaluate the proposed conceptual model. The system has the functionality to add new specialized co-acts in layer 3. By testing multiple manufacturing scenarios it was found that all of them could be described with the following specialized co-acts: 1) Request information, 2) Give information, 3) Book/Unbook skill, 4) Request skill to start and 5) Attach/Detach. All coordinates shared between agents are translated to world coordinates before communicated to other agents.

It is important to understand that these co-acts work for many scenarios but are limited to a specific type of system. In this case, they apply to some types of manufacturing systems including the one in the scenario described in this paper. In Table 3, each specialized co-act for our scenario is listed.

Table 3. Specialized co-acts layer (3).

Number	Specialized co-act
1	Request information
2	Give information
3	Book/Unbook skill
4	Request skill to start
5	Attach/Detach

The following list describes each communication step in the scenario using the previously defined specialized co-acts for agent communication. This list is limited to the perspective of part p. Hence, the robot is not included. All variables in the following list are described in detail in Table 2. Each step below is noted with (x), where x is the related co-act number presented in Table 3:

- (3) p tries to book r_2 if it has a skill $s_2 = Paint$. The part p is compatible with the interfaces if_1 on r_2 and is therefore booked by p.
- (3) p tries to book r_3 if it has a skill $s_1 = Transport$. The part p is compatible with the interface if_2 on r_3 and is therefore booked by p.
- (1) p is attached to r_1 with v_3 attached to v_5. Thus, p asks r_1 to give the variable v_5. Resource r_1 translates v_5 to world coordinates before sending it to p.
- (1) To find the location for placing, p asks r_2 for the

variable v_1. Resource r_2 translates v_1 to world coordinates before sending it to p.
- (2) p uses its grip location v_2 to calculate the pick and place location to move between r_1 and r_2. Then these are sent to the gripper r_3.
- (4) p requests that r_3 runs the skill s_1
- (3) p unbooks the interface if_3 on r_1
- (5) p tells r_2 that p is attached to r_2
- (3) p unbooks the interface if_2 on r_3
- (2) p gives the variable data v_4 to r_2.
- (4) p requests that r_2 runs the skill s_2

5. Conclusion

In this paper, a conceptual model was presented that divides agent communication into four layers. The network layer (1), General co-acts layer (2), Specialized co-acts layer (3) and a Reconfigurable layer (4). This helps to separate the logic for general communication (layer 2) from the specialized communication in this paper a manufacturing system (layer 3). The reconfigurable layer (4) makes it possible to change a manufacturing system without reprogramming. A multi-agent system was developed and used to evaluate the conceptual model.

Two types of semantics were identified. The static semantics (a) were hardcoded into the specialized co-acts, layer three in the conceptual model. The reconfigurable semantics (b), i.e., interfaces, skills, and variables were configured in the reconfigurable layer (layer four). By only reconfiguring the agents, it is possible to change how they use the variables shared with other agents. Hence, we change the semantics in the system without reprogramming or changing anything in layer three (specialized co-acts layer). Based on this conclusion, the best practice is to place semantics that often changes their meaning in layer four (reconfigurable layer).

Further, the implementation shows that it is possible to limit the number of choices one must make to implement a multi-agent system for Plug & Produce by customizing layer three in our model for a specific type of application. In this paper, the scenario was customized for manufacturing systems. Hence, one implementation of layer three can be used for many scenarios with different configurations in layer four. This means that it is in many cases possible to work only in layer four when implementing a multi-agent system for a new scenario. This helps the industry by hiding the complexity of agent design and agent communication. Thus, making it possible to avoid reprogramming and its related educational requirements on personnel. Further, the flexibility of using a Plug & Produce system increases the adaption speed for adding new products and resources.

Acknowledgements

Miljö för Flexibel och Innovativ Automation, Project reference: 20201192, Funded under: Europeiska regionala utvecklingsfonden/VGR

References

[1] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing,* vol. 37, pp. 282-291, 2015.

[2] Z. Pan, J. Polden, N. Larkin, S. V. Duin and J. Norrish, "Recent progress on programming methods for industrial robots," *Robotics and Computer-Integrated Manufacturing,* vol. 28, no. 2, pp. 87-94, 2012.

[3] T.Araia, Y.Aiyama, Y.Maeda, M.Sugia and J.Otaa, "Agile Assembly System by "Plug and Produce"," *CIRP Ann Manuf Technol,* vol. 49, no. 1, pp. 1-4, 2000.

[4] *FIPA 97 Part 1 Version 1.0: Agent Management Specification,* Foundation for intelligent physical agents, 1997.

[5] FIPA, "FIPA Agent Management Specification," in *FIPA 2002*, Geneva, Switzerland, 2002.

[6] S. Poslad, "Specifying Protocols for Multi-Agent Systems Interaction," *ACM Transactions on Autonomous and Adaptive Systems,* vol. 2, no. 4, pp. 15:1-15:24, 2007.

[7] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review,* vol. 10, no. 2, pp. 115-152, 1995.

[8] P. Leitao, "Agent-based distributed manufacturing control : A state-of-the-art survey," *Engineering applications of artificial intelligence,* vol. 22, no. 7, pp. 979-991, Oct. 2009.

[9] P. Leitao, 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.

[10] FIPA, "FIPA ACL Message Structure Specification," in *FIPA 2002*, Geneva, Switzerland, 2002.

[11] FIPA, "FIPA Communicative Act Library Specification," in *FIPA 2002*, Geneva, Switzerland, 2002.

[12] FIPA, "FIPA Agent Message Transport Service Specification," in *FIPA 2002*, Geneva, Switzerland, 2002.

[13] M. Bennulf, F. Danielsson and B. Svensson, "Identification of resources and parts in a Plug and Produce system using OPC UA," in *FAIM 2019*, Limerick, Ireland, 2019.

[14] F. Bellifemine, G. Caire and D. Greenwood, Developing multi-agent systems with JADE, John Wiley & Sons, Ltd, 2007.

[15] W. Mahnke, S.-H. Leitner and M. Damm, OPC unified architecture, Springer Science & Business Media, 2009.

Paper D

A Method for Configuring Agents in Plug & Produce Systems

Mattias Bennulf, Fredrik Danielsson, Bo Svensson

D

Presented at the Swedish Production Symposium, SPS, in Skövde, Sweden, April 2022

SPS2022
A.H.C. Ng et al. (Eds.)

doi:10.3233/ATDE220133

A Method for Configuring Agents in Plug & Produce Systems

Mattias Bennulf [a,1], Fredrik Danielsson [a] and Bo Svensson [a]
[a] *Department of Production Systems, University West, Trollhättan, Sweden*

Abstract. Multi-agent technology, used for implementing Plug & Produce systems have many proposed benefits for fast adaption of manufacturing systems. However, still today multi-agent technology is not ready for the industry, due to the lack of mature supporting tools and guidelines. The result is that today, multi-agent systems are more complicated and time-consuming to use than traditional approaches. This hides their true benefits. In this paper, a new method for configuring agents is presented that includes automated deployment to manufacturing systems and by its flexible design opens the possibility to connect many other supporting tools when needed. A configuration tool is also designed that works with the proposed method by connecting to an agent configuration database. The overall aim of the method is to simplify the steps taken for adapting a manufacturing system for new parts and resources.

Keywords. Plug & Produce, Configuration, Multi-Agent System, Deployment, Industry 4.0

1. Introduction

For many years, the demand for low volume production and customization has increased [1]. It is costly to rebuild a manufacturing system for each new type of part to be produced, forcing many manufacturing processes to still be performed manually. The reason that it is difficult to change automated manufacturing systems is that they usually have rigid control solutions that are difficult to change, due to the lack of abstraction of logic and encapsulation of code. Programming of a resource such as a robot takes up a huge amount of time [2], limiting the possibilities to quickly add new product designs to the system. Further, in traditional systems with central control, there typically exists strong dependencies between resources. For example, it is not always easy to change the code of one industrial robot without changing the code also in surrounding resources such as a PLC or another robot. This makes it difficult to write the local code for one resource without considering the specific manufacturing system where that resource is to be used. This is a problem when implementing Plug & Produce systems, where resources should be easy to connect/disconnect and even have the possibility to be moved between different manufacturing areas or even plants.

Plug & Produce is a concept where resources are connected using standardized hardware connectors and are automatically included in the manufacturing [3]. One approach to implement the controller in a Plug & Produce system is to utilize multi-agent technology that was described by Wooldridge et.al. [4]. Multi-agent systems simplify the

[1] Corresponding Author. mattias.bennulf@hv.se

design of manufacturing system controllers by encapsulating each resource logic in the system. This is done by creating a separate agent (controller) for each resource and part in the system. Each agent can communicate with each other to reach manufacturing goals through a standardised communication interface. Hence, the agent for each resource is independent of logic and signals of other resources and can act independently through negotiation. This makes it possible to adapt to new types of parts and resources in minutes rather than days or months as in conventional systems with hierarchical central control. The reason that it is faster to adopt is that a resource such as an industrial robot only needs to know its own skills, e.g., how to perform transportation of a robot tool. While the tool, such as a robot gripper only need to know how to transport parts. Dependencies between the robot and the tool, in this case, is only based on demands of skills and are not hardcoded in the control logic itself. In this way, logic is completely isolated into carefully defined resources with different skills and parts with manufacturing goals. However, still today such a flexible system is not used in the industry due to a lack of mature tools and platforms [5], [6]. Prototypes has been developed but they are not used in large-scale production [7].

In existing multi-agent frameworks such as the Java Agent Development Framework (JADE), each agent still has individual tailormade control code, that is manually written. The idea is that the programmer writes the agent control code in such a way that the agents automatically communicate and negotiate to make the dependencies between them limited since they can find each other without for example mapping specific signals to static addresses. This makes it easier to add new resources since they do not have strong dependencies on surrounding resources. However, the control code still has to be written manually to create or change the local behaviour of an agent.

To avoid programming of agents, one approach presented in [8] proposes to write one single agent control code and then reuse the exact same agent code for all agents. The unique and local behaviour of each agent is given through configurations. In contrast to frameworks such as JADE, the one presented in [8] have predefined strategies for negotiating and communicating among agents, making it easier to introduce new agents. Our paper is a continuation of this agent framework presented in [8]. Hence, agents in this paper are instantiated based on one single agent control code and are given individual configuration data. The approach that one single agent code is developed and never changed drastically limits the need for a deeper understanding of the internal control logic since communication and negotiation are standardized and handled automatically by the agents. However, this requires a standardized configuration ontology as well as a special-purpose configuration tool that can be used to define the necessary configuration data for each individual agent.

In this paper, such a tool for configuring agents is proposed and evaluated. This paper also investigates how the configuration tool can include functionality for automated deployment to physical manufacturing systems. Further, connections with other supporting tools, such as extracting data from robotic simulations and product designs are investigated.

2. Configurable Agents

As described earlier, one approach to implement a Plug & Produce system is to create a multi-agent system, that consists of multiple agents that are communicating with each other. This means that physical objects, such as resources and parts have a corresponding agent (controller). If data should be synchronized between an agent and its corresponding physical object, they should be connected. The separation of cyber and physical components is identical to the concept of designing a Cyber-Physical System such as the one presented in [9].

Using agents limits the direct communication between physical objects and instead forces all resources to communicate through their agents using standardized communication interfaces. This completely removes the need for defining specific network details, such as addresses for communication between resources. Standards for implementing agents exist today. Some of these standards are developed by the Foundation for Intelligent Physical Agents (FIPA), which is an IEEE organization. They specify an agent communication language, specified in the specification: FIPA Agent Communication Language (ACL) [10]. Further, the FIPA Agent Management Specification [11] describes the general guidelines on how to design an agent system. To develop an agent system that follows FIPA it is common to use JADE. This is an agent framework that implements the agent standards from FIPA. However, JADE and FIPA are only considering agents in general, thus no information or supporting tools for the manufacturing industry are included today. Configuration tools are not considered in JADE, since the approach is mainly to design agents by manual coding in the JAVA language.

There currently is a lack of standardized agent configuration formats since many continue to write agent code manually. However, some initiatives exist, such as the one presented in [8] that specifies that an agent's configuration should consist of variables, goals, skills, process plans, and interfaces. Further, the agent itself is defined and specified as a part or a resource. This is shown in Figure. 1, which is based on the ontology presented in [8].

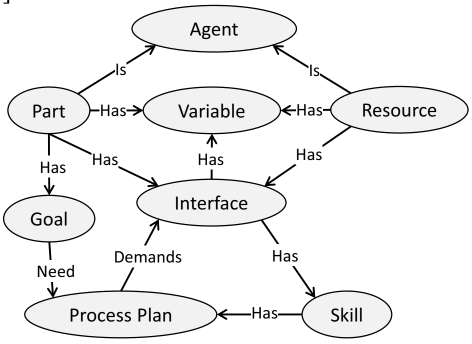


Figure. 1. Agent classes for defining an agent configuration.

In our method for configuring agents, these classes in Figure. 1 are used but extended with more details such as using pre-conditions on goals. Each of the agent classes is regarded as an entity that has a unique id, name, description, and type. The detailed description of the configuration format that will be considered in the rest of this paper is presented in the following sections:

Variable: A variable can be any data that is needed such as pick and place positions, tool specifications such as weight, or the speed of a motor. It could also be more advanced data such as a specific path for grinding with a robot. In that case, the path is not configured for the robot but instead configured as a path locally defined on the part agent.

Goal: Parts have goals that define what should happen to the part in the manufacturing system. A goal can for example be to drill a hole with a specific diameter or to grind a part to get soft edges. Goals are described with pre-conditions and can run in parallel if the pre-conditions allow this.

Skill: Resources in the system has skills that describe a specific capability. This can for example be a skill to transport a part, paint a part with colour, or store a part in a buffer. Skills are always defined on interfaces in order to organize them and ensure that hardware and software compatibility is maintained between connected agents. Implying that a skill can only be utilized by agents that have compatible interfaces.

Process Plan: A process plan defines how to solve a specific goal or to execute a skill. It is described as a recipe with demanded resource skills without referring to specific resources. The syntax is similar to a high-level programming language such as Structured Text (ST). However, the idea here is to not include advanced logic but to keep a strong abstraction from the details of the control logic. The process plans for goals typically only have a sequence of skills defined, while the process plans for running a skill includes skills as well as some local abilities such as setting a local variable to true.

Interface: An interface is a point of interaction between two agents and is used for agents to find each other. The interfaces are searchable in the agent network and can be used for collaboration. An interface could, for instance, act as the boundary between a robot gripper and a robot. Both the gripper and the robot need interfaces that are compatible in order to be connected. In that case, the robot presents on its interface, a skill with the functionality to transport the gripper. In this way, the gripper would never request to be attached to a resource that cannot transport or that is not physically compatible with the gripper.

There are many approaches to format such configuration data explained above. General data formats exist such as XML, JSON and AutomationML. Extensible Markup Language (XML) is a data format that supports objects and lists. Similarly, JavaScript Object Notation (JSON) can be used to structure the same data in a slightly different way. AutomationML is a standardized storage format for manufacturing systems, that was presented at the Hannover fair in 2008 [12]. It is based on XML and was designed to be used with engineering tools for manufacturing systems. AutomationML uses the object-oriented paradigm in order to describe plant components. It can store the plant topology, geometry, kinematic, behaviour description and references/relations. For this paper, JSON has been considered due to its multiple existing libraries for implementation with several programming languages and platforms. JSON is a format that is also currently implemented in various industrial devices, such as the “Robot Web Services” for ABB industrial robots. It also tends to be more lightweight as a format than XML, due to less overhead in its data structure. Because of this, it is a bit more difficult to read by humans than XML. However, this should not be a problem since the aim of this paper is to use a configuration tool instead of editing the data directly. Specifically, we have considered JSON RPC, which is a remote procedure call protocol for sending JSON data between different components in the system. Note that, since the configuration classes presented earlier are used for the configurations the JSON format must be combined with these configuration classes to work properly.

To work directly in text-based formats is time-consuming, complex and exposed to syntax mistakes when manually creating and handling the configuration parameters. General tools for editing these formats exist, such as the “AutomationML Editor”, Microsoft’s “XML Notepad” and Altova’s “XMLSpy”, which can edit both XML and JSON. The problem with these tools is that they do not consider the agent configuration classes presented earlier in this paper. It is however possible to include them in such tools as a class tree. But still, these tools do not give enough support to the user since they are not specifically made for use with the agent configurations. Instead, a configuration tool should give the user support and simplify the understanding of the agent's local configurations and their possible interactions and compatibility with each other. The configuration tool also needs to help the user to avoid syntax errors by verifying entered data directly when configuring. This is limited if not impossible to do in a general editing tool for these data formats. Further, in the considered agent system, all configurations are stored in one single database that the configuration tool should edit directly. Otherwise, it would be required to manually copy configuration files from the supporting tools to the agents when instantiated. However, in the existing editors, the synchronization with such an agent configuration database is not implemented. The reason for choosing a central database to store the configurations is to enable the integration of other software such as simulation tools which support the extraction of data that otherwise has to be manually copied between software. It also simplifies the deployment of configurations to agents and enables the possibility to work simultaneously on editing the agent configurations.

Thus, this paper aims to develop a configuration tool that is specifically made for the agent configuration format presented earlier in this chapter. All configuration data defined in the developed configuration tool is then placed in a JSON structure. This can then be communicated over JSON RPC to other devices. The approach is to use a central database, in our case an SQL database, that contains all configuration data needed for the complete manufacturing system. The configuration tool connects to this central database over the JSON RPC and downloads the latest configuration as JSON objects and uploads the changes made in the configuration tool. This will enable multiple users to work with the configurations simultaneously.

3. Proposed Configuration Method

The configuration classes shown in Figure. 1 needs to be implemented in all the agents, the configuration tool and in the data storage format for agent configurations. Only then is it possible to share a common knowledge about the meaning, i.e. ontology, of the configuration data. When a resource or part is added to the manufacturing system, an agent is instantiated, running in a cloud service. As described earlier, there is sometimes data to be controlled by the agent, e.g., sensor input or start signals to a motor. In that case, a network connection is established automatically, based on the configuration, between the agent in the cloud and the hardware in the added resource or part. A similar concept of agents running in a cloud service is presented in [13], where the agents are instantiated based on configurations in a configuration database. The correct configuration is chosen based on the agent type presented by the added resource or part.

3.1. Configuration tool design

The configuration tool presented in this paper is based on a standalone graphical HMI to assist the configuration. Using the proposed configuration tool, it is possible to focus on one single resource or part at a time. The work order for the configuration tool presented in this paper is that resource agents are configured first with their related interfaces and skills. Then the part agents and process plans are defined. Thus, it should be possible to make a list of available skills and variables on the resources and to drag and drop those onto the process plans and part configurations. This removes the need for the user to remember all variables and skill names of the resources. Descriptions can also be given on configuration parameters, describing with a user-friendly message what they do. For example, such a message could describe what a certain skill on a resource can achieve if executed. This is a great way to connect multiple users, working with configuration parameters for the same manufacturing system. The configuration tool also needs to assist the user with warnings when for example a resource with specific skills is missing in the configuration database. The proposed configuration tool presents several views to the user. Each of these views can also edit its related entity details, i.e., its id, name, description, and type. The views are created based on the classes described in Figure. 1. Thus, the views are *Main view*, *Agent view*, *Interface view*, *Goal view*, *Process plan view*, *Skill view*, and *Variable view.* Each of these views could for example describe an individual window in the configuration tool.

3.2. Database for storage

A database was chosen for storing the agent configurations. This enables multiple users to edit the data simultaneously. It also makes it possible to automate the deployment of agent configurations directly to a real manufacturing system. It is in some cases even possible to deploy updates while the manufacturing system is online. Imagine a new part with a new agent configuration. It is then possible to deploy that agent configuration directly to the configuration database used in the manufacturing system. When the new part is entering the manufacturing system it gets a corresponding agent associated with it, using the new configuration added.

The JSON format is used for communicating with the database, see Figure. 2. This means that a software is developed to convert between JSON and a database format, such as SQL queries in the case of using an SQL database. The main aim of not using for example SQL queries directly from the configuration tool is to avoid knowing anything about the database format or type. This makes it possible to completely change the database structure as long as the software attached to the database can convert it to JSON objects. On the configuration tool, there is also a software that can convert JSON to objects in the configuration tool. This can, for instance, be a JAVA object, in the case of using that programming language. This makes it easier to extend the agent configuration format in the future if needed.

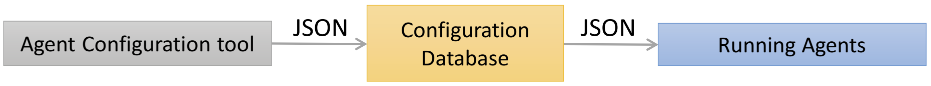


Figure. 2. This figure shows the typical information flow for committing updates from the configuration tool to the agents through the configuration database.

When an agent is instantiated using the standardized code for all agents, then it downloads an agent configuration from the database based on some information about what part or resource it should represent. All agents start by requesting their configuration from the database by a JSON RPC call. The configuration is then transferred to the agent as a JSON structure. Hence, the reason for using the database is to automate the deployment of new configurations and to share the configurations directly with other users of the configuration tool, enabling the possibility for collaborating in projects to develop an entire manufacturing systems configuration. We have now only considered a single database, meaning that we are changing the manufacturing system directly when changes are applied. Thus, in the future, it could be useful to make development copies of the entire system that could be used and tested in simulations before deploying to the online manufacturing system.

3.3. Connecting to other supporting tools

Many parameters that should be entered into a configuration tool are defined or calculated in other software such as 3D CAD tools or robotic simulation tools. This makes it necessary to develop a bridge between those tools and the agent configuration database. Such an approach is presented in Figure. 3, where the arrows show the information flow for updates made in the configurations. However, it should be noted that information can be accessed from the database by all software. This is needed when editing an already existing agent configuration, nonetheless, updates are always committed in the direction of the arrows.

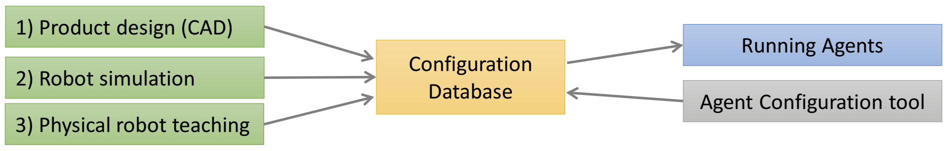


Figure. 3. This figure shows the agent configuration tool together with some additional supporting tools 1,2 and 3. All these four tools can be connected to the agent configuration database and automatically deployed to agents in the manufacturing system.

Product design 1): As an example, for a part designed in a CAD tool, it would be useful if all specifications such as the definition of a hole with a specific diameter would automatically be translated into a manufacturing goal for the part agent. This requires the CAD tool to be extended with an add-on feature that can identify such goals and synchronize them to the configuration database. In the configuration database there exist multiple process plans designed for solving specific types of manufacturing goals. Thus, the addon feature should fetch the goal names for these process plans as a list and present them in the CAD tool. The user will then manually add goals from that list directly on the part design. When the user adds such a goal, then it should also be visually reflected on the part design, where the user then would see a hole. This will give the user an experience similar to that of working in any other traditional CAD project. The main difference from using a regular CAD tool will be the limitations to only using predefined goals. However, this can in some cases completely remove all manual steps for translating and preparing the part design for manufacturing. Only when a completely new part is designed, details must be defined manually in the agent configuration tool. This is not always needed if smaller part changes are made in a CAD tool such as adding a goal.

Robot simulation 2): Similarly, other parameters such as pick and place locations on a table or a location in a buffer station are usually defined on either a physical robot or in a robotics simulation tool. One example of such a tool is RobotStudio, which is a robot simulation software from ABB where robot programs can be developed and tested offline [14]. However, when using the proposed concept of agents, then the robot should not have such a standard robot program and act as a central controller. Instead, the control is a shared task between each configurable agent in the system that the positions can be used on. For example, a part has local target positions that describe a suitable location for gripping it with a robot gripper or placing it on a buffer. These positions are related to the parts local coordinate system. The functions for translating between coordinate systems are included in all agents (resources and parts) and used when they communicate. A software should be developed that can identify which agent each target position should be attached with in the configuration database.

Robot teaching 3): It is also possible to create a software to extract positions directly from a physical robot, by teaching points and selecting which agent they belong to. This would require a user-friendly HMI used by the operator that manually moves the robot and stores target positions to the configuration database.

4. Evaluation

This section introduces a manufacturing scenario where a part should get painted and then leave the manufacturing system. All configuration parameters needed for this scenario are defined in this chapter. The proposed configuration tool is implemented in the C# language as a form application. To evaluate the implemented configuration tool, all parameters from the described scenario are entered into it.

The scenario, presented in this section, is used to evaluate the configuration tool. It includes one part p with a goal $g = PaintBlue$. The part is placed at the input position $InLocation$ on the conveyor r_1 and is moved to the paint station r_3 by conveyors r_1, r_2 and the robot r_5, see Figure. 4. After the part has been painted with the correct colour, it should be transported by the robot r_5 to the unloading station r_4. Each conveyor has two position variables, describing where the part can be located: $InLocation$ and $OutLocation$. Thus, r_1 has: $InLocation = 1$ and $OutLocation = 2$ while r_2 has: $InLocation = 2$ and $OutLocation = 3$. These correspond to the positions 1,2,3 in Figure. 4.

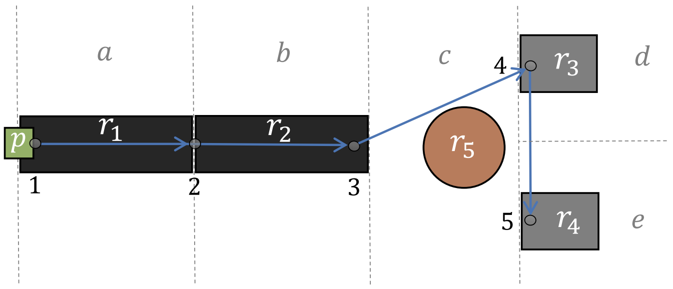


Figure. 4. Example of a part p located on a conveyor r_1 with the goal $g = PaintBlue$, that is solved by moving to the paint station r_3, using resources $r_1 . r_2$ and r_5.

The letters a, b, c, d and e, describes locations where a resource is expected to exist. This notation is used since resources can be replaced and the possibility exists that multiple alternative resources may exist in the same location. For example, there could exist parallel conveyors in location b. The part agent searches these locations for agents that are available and selects one of them. More details about this are described later in this paper, where the process plans are defined. All the required parameters for the manufacturing scenario are presented in Table 1.

Table 1. Configuration data for the presented scenario, sorted by agents.

Id	Parameter Name	Agent	Data type
π_g	PaintBlue		Process Plan
if_1	BufferInterface	p	Interface
if_2	GripInterface	p	Interface
g	PaintBlue	p	Goal
if_3	BufferInterface	r_1	Interface
v_1	InLocation	r_1	Variable
v_2	OutLocation	r_1	Variable
s_1	Load	r_1	Skill
s_2	Transport	r_1	Skill
v_3	Input: From	r_1	Variable
v_4	Input: To	r_1	Variable
if_4	BufferInterface	r_2	Interface
v_5	InLocation	r_2	Variable
v_6	OutLocation	r_2	Variable
s_3	Transport	r_2	Skill
v_7	Input: From	r_2	Variable
v_8	Input: To	r_2	Variable
if_5	BufferInterface	r_3	Interface
s_4	Paint	r_3	Skill
v_9	BufferLocation	r_3	Variable
if_6	BufferInterface	r_4	Interface
s_5	Unload	r_4	Skill
v_{10}	BufferLocation	r_4	Variable
if_7	GripInterface	r_5	Interface
s_6	Transport	r_5	Skill
v_{11}	Input: From	r_5	Variable
v_{12}	Input: To	r_5	Variable

The data types are the corresponding configuration classes from Figure. 1. The name is the agent classes entity name and the id is the entity id. The process plan on the first row in Table 1 that is noted as π_g is not directly related to any specific agent. It is related to solving the goal g, that may exist on multiple parts. A process plan is later selected automatically that can reach the goal for the part. Hence, there can be multiple plans that reach the same goal. Some variable descriptions in Table 1 are noted with “Input:” meaning that these variables have no defined value in the configuration but act as input signal holders that get values at runtime.

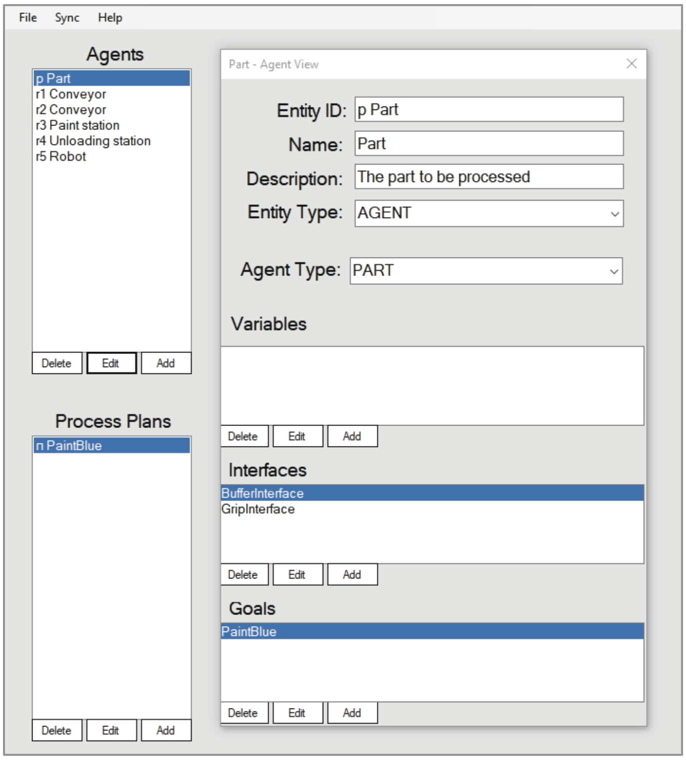
Multiple process plans may exist that can solve the same goal. In this paper, only one process plan for the goal g is considered. In Figure. 4, the letters a, b, c, d, e are used to define needed resources in the manufacturing system that are not known at the configuration phase. Since the physical resources are not known when the process plan is defined, they are searched for and found when the system is running.

For reaching the goal $g = PaintBlue$, the following process plan π_g is defined:

1. a.Load
2. a.Transport: a.From = a.InLocation
   a.To = a.OutLocation
3. b.Transport: b.From = b. InLocation
   b.To = b. OutLocation
4. c.Transport: c.From = b. OutLocation
   c.To = d. BufferLocation
5. d.Paint
6. c.Transport: c.From = d. BufferLocation
   c.To = e. BufferLocation
7. e.Unload

4.1. Implementation of the configuration tool

The configuration tool was implemented to evaluate the proposed design. All configuration parameters are added into the implemented configuration tool to evaluate it. In Figure. 5, the agent view of the configuration tool is shown for the part agent. We can see that it has a *BufferInterface*, *GripInterface* and one goal *PaintBlue*. Each configuration parameter presented in this view can be modified in detail on separate views.



p Part
r1 Conveyor
r2 Conveyor
r3 Paint station
r4 Unloading station
r5 Robot

n PaintBlue

Entity ID: p Part
Name: Part
Description: The part to be processed
Entity Type: AGENT
Agent Type: PART

Interfaces: BufferInterface, GripInterface

Goals: PaintBlue

Figure. 5. The configuration tool, with the main view shown in the background and the agent view of part p shown in the front.

All views use a window design, similar to the one presented in Figure. 5. Note that these views, together provide all the necessary functionality for creating the agent configurations needed for the scenario, presented earlier in this paper.

5. Conclusion

In this paper, a method for configuring agents was presented that enables users to adapt a manufacturing system to various scenarios with new parts and resources. The considered manufacturing system is based on multi-agent technology, where the configurations describe the agents for each resource and part. This makes it possible to focus on one device at a time, removing the need to understand other controllers in the manufacturing system. This is not the case in traditional systems where the users need to understand most of the other controllers to add new resources or parts. The developed method for configuring agents include a central agent configuration database. A configuration tool was also developed where all agent configurations can be managed. The configuration tool is connected to the database which makes the configurations easy to deploy since agents can fetch their configurations automatically when instantiated. It also enables multiple users to collaborate with the same configuration data. The configuration tool is designed with multiple views that are specifically designed for configuring agents, based on their configuration classes. This resulted in a lightweight tool that avoids any unnecessary steps or functionalities. The developed configuration tool was evaluated by configuring all necessary parameters for a manufacturing scenario. The presented method also prepares the multi-agent system for adding many supporting tools, for instance: product design tools for defining goals, and 3D simulation tools for defining target positions such as buffer locations and gripping points.

6. Acknowledgements

This paper was written as part of the PoPCoRN project, funded by the K-K foundation and the Miljö för Flexibel och Innovativ Automation, Project reference: 20201192, Funded under: Europeiska regionala utvecklingsfonden/VGR.

References

[1] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger and O. Madsen, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing,* vol. 37, pp. 282-291, 2015.

[2] Z. Pan, J. Polden, N. Larkin, S. V. Duin and J. Norrish, "Recent progress on programming methods for industrial robots," *Robotics and Computer-Integrated Manufacturing,* vol. 28, no. 2, pp. 87-94, 2012.

[3] T.Araia, Y.Aiyama, Y.Maeda, M.Sugia and J.Otaa, "Agile Assembly System by “Plug and Produce“," *CIRP Ann Manuf Technol,* vol. 49, no. 1, pp. 1-4, 2000.

[4] M. Wooldridge and N. R. Jennings, "Intelligent agents: theory and practice," *The Knowledge Engineering Review,* vol. 10, no. 2, pp. 115-152, 1995.

[5] P. Leitao, 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.

[6] P. Leitao, "Agent-based distributed manufacturing control : A state-of-the-art survey," *Engineering applications of artificial intelligence,* vol. 22, no. 7, pp. 979-991, Oct. 2009.

[7] S. Karnouskos, P. Leitao, L. Ribeiro and A. W. Colombo, "Industrial Agents as a Key Enabler for Realizing Industrial Cyber-Physical Systems: Multiagent Systems Entering Industry 4.0," *IEEE Industrial Electronics Magazine,* vol. 14, no. 3, pp. 18-32, 2020.

[8] M. Bennulf, F. Danielsson, B. Svensson and B. Lennartson, "Goal-Oriented Process Plans in a Multiagent System for Plug & Produce," *IEEE Transactions on Industrial Informatics,* vol. 17, no. 4, pp. 2411-2421, 2021.

[9] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, Orlando, FL, USA, 2008.

[10] FIPA, "FIPA ACL Message Structure Specification," in *FIPA 2002*, Geneva, Switzerland, 2002.

[11] FIPA, "FIPA Agent Management Specification," in *FIPA 2002*, Geneva, Switzerland, 2002.

[12] R. Drath, A. Lüder, J. Peschke and L. Hundt, "AutomationML - the glue for seamless automation engineering," in *Emerging Technologies and Factory Automation (ETFA)*, Hamburg, Germany, 2008.

[13] M. Bennulf, F. Danielsson and B. Svensson, "Identification of resources and parts in a Plug and Produce system," in *FAIM*, Limerick, Ireland, 2019.

[14] P. Abreu, M. R. Barbosa and A. M. Lopes, "Virtual experiment for teaching robot programming," in *International Conference on Remote Engineering and Virtual Instrumentation*, Porto, Portugal, 2014.

Tidigare avhandlingar – Produktionsteknik

PEIGANG LI Cold Lap Formation in Gas Metal Arc Welding of Steel An Experimental Study of Micro-lack of Fusion Defects, 2013:2.

NICHOLAS CURRY Design of Thermal Barrier Coatings, 2014:3.

JEROEN DE BACKER Feedback Control of Robotic Friction Stir Welding, 2014:4.

MOHIT KUMAR GUPTA Design of Thermal Barrier Coatings A modelling approach, 2014:5.

PER LINDSTRÖM Improved CWM Platform for Modelling Welding Procedures and their Effects on Structural Behavior, 2015:6.

ERIK ÅSTRAND A Framework for Optimised Welding of Fatigue Loaded Structures Applied to Gas Metal Arc Welding of Fillet Welds, 2016:7.

EMILE GLORIEUX Multi-Robot Motion Planning Optimisation for Handling Sheet Metal Parts, 2017:10.

EBRAHIM HARATI Improving fatigue properties of welded high strength steels, 2017:11.

ANDREAS SEGERSTARK Laser Metal Deposition using Alloy 718 Powder Influence of Process Parameters on Material Characteristics, 2017:12.

ANA ESTHER BONILLA HERNÁNDES On Cutting Tool Resource Management, 2018:16.

ASHWIN DEVOTTA Improved finite element modelingfor chip morphology prediction inmachining of C45E steel, 2020:34.

TAHIRA RAZA Process Understanding and Weldability of Laser-Powder Bed Fusion Manufactured Alloy 718, 2020:35

PARIA KARIMI Electron Beam-Powder Bed Fusion of Alloy 718. Effect of Process Parameters on Microstructure Evolution, 2020:37.

JONAS HOLMBERG High volumetric machining strategies of superalloy gas turbine components. Comparing conventional and non-conventional machining methods for efficient manufacturing, 2020:40.

SNEHA GOEL Thermal post-treatment of Alloy 718 produced by electron beam melting, 2020:41.

ARUN RAMANATHAN BALACHANDRAMURTHI Towards understanding the fatigue behaviour of Alloy 718 manufactured by Powder Bed Fusion processes, 2020:42

CHAMARA KUMARA Microstructure Modelling of Additive Manufacturing of Alloy 718, 2020:43.

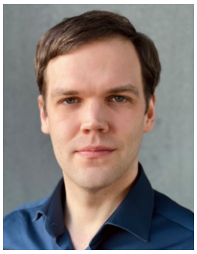
OLUTAYO ADEGOKE Processability of Laser Powder Bed Fusion of Alloy 247LC - Influence of process parameters on microstructure and defects, 2021:45

ANDERS JOHANSSON Challenging the traditional manufacturing objectives: Designing manufacturing systems for both product manufacturing and value production, 2022:48

SEYYED MOHAMMAD ALI NOORI RAHIM ABADI Laser metal fusion and deposition using wire feedstock: Process modelling and CFD simulation 2022:52

A Control Framework for Industrial Plug & Produce

Plug & Produce can be used for automation when a manufacturing system needs to adapt fast to changes such as new product designs or adjusted production volume. The demand for customized products and low-volume production is constantly increasing. The industry has for many years used dedicated manufacturing systems that are difficult and expensive to adapt to changes. The result of this is that factories are forced to use human workers for certain tasks that demand high flexibility. Resources in Plug & Produce are easy to add, move and remove, taking only minutes rather than days in dedicated systems, making the system flexible to changes. This thesis presents a framework that can be used as a control system for Plug & Produce. The framework is based on the distributed approach called multi-agent systems, where each resource and product part has a controller that communicates with each other to reach manufacturing goals. The idea is that the system automatically adapts itself to manage changes. This decreases the time spent manually preparing the system.



Mattias Bennulf
Mattias received a B.S. degree in computer engineering from University West, Sweden in 2014, and an M.S. degree in robotics and automation from University West, Sweden in 2015. His research interests include artificial intelligence, multi-agent systems, robotics, and automation.

978-91-89325-42-5 (Printed version)
978-91-89325-41-8 (Electronic version)