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].

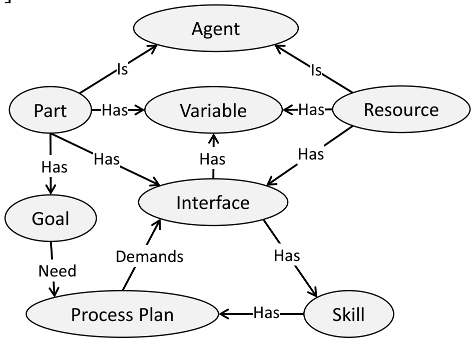


Figure. 1. Agent classes for defining an agent configuration.

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.

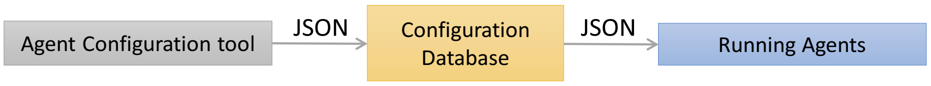


Figure. 2. This figure shows the typical information flow for committing updates from the configuration tool to the agents through the configuration database.

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.

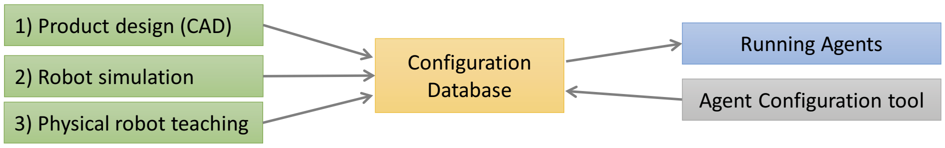


Figure. 3. This figure shows the agent configuration tool together with some additional supporting tools 1,2 and 3. All these four tools can be connected to the agent configuration database and automatically deployed to agents in the manufacturing system.

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.

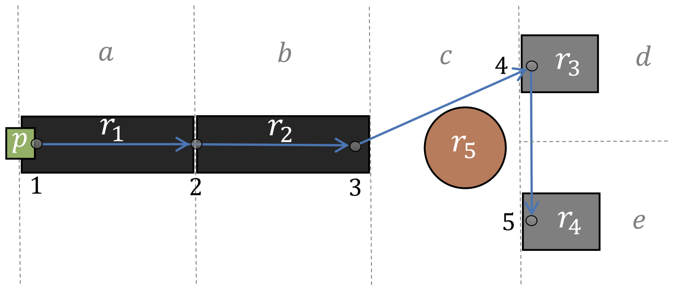


Figure. 4. Example of a part p located on a conveyor $r_1$with the goal $g = PaintBlue$, that is solved by moving to the paint station r_3, using resources $r_1.r_2$ and r_5.

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.

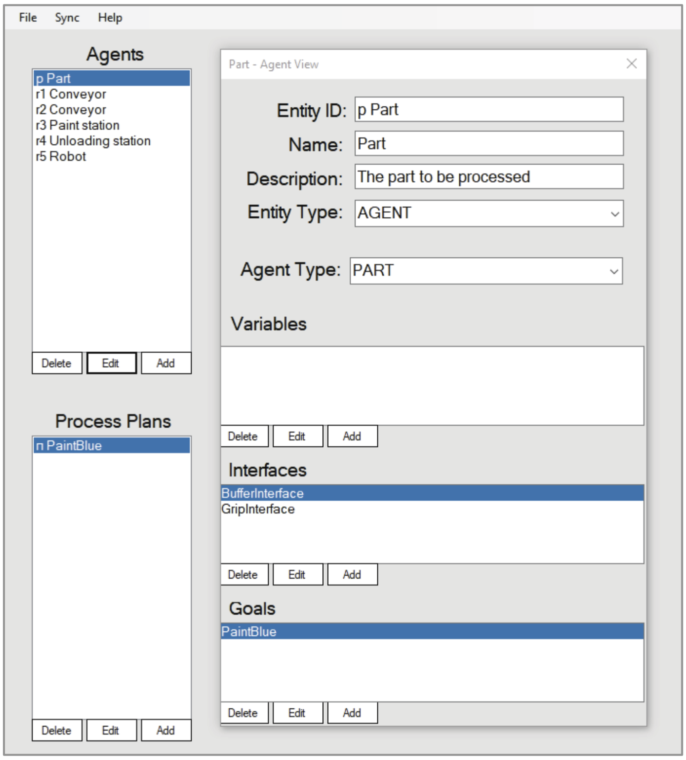
Multiple process plans may exist that can solve the same goal. In this paper, only one process plan for the goal g is considered. In Figure. 4, the letters a, b, c, d, e are used to define needed resources in the manufacturing system that are not known at the configuration phase. Since the physical resources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



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

Process Plans: π PaintBlue

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

Variables

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.