Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 253 (2025) 1860–1869

Procedia Computer Science

www.elsevier.com/locate/procedia

6th International Conference on Industry 4.0 and Smart Manufacturing

Flexible assembly planning for self-organizing production cells

Christian Lehner[a,*], Oliver Kosak[a,*], Hella Ponsar[a,*], Wolfgang Reif[a]

[a]*Institute for Software & Systems Engineering, University of Augsburg, Universitätsstraße 2, 86159 Augsburg, Germany*

Abstract

In the era of Industry 4.0 and individualized mass customization, the demand for adaptable manufacturing systems is paramount. Therefore, assembly plans used in such systems must also allow a high degree of flexibility to adapt to changing requirements and conditions. At the same time, they must still guarantee the successful execution of the assembly process. In this paper, we present a novel approach to modeling assembly plans in the context of an ongoing, larger project in which we intend to implement flexible manufacturing in a real production cell. In our assembly plans, we specifically emphasize flexibility regarding the order of assembly steps, the allocation of appropriate teams of robots to execute these steps with applicable parts, and the ability to respond to failures in the system. We present a comprehensible modeling of flexible assembly plans, accompanied by the semantics of Petri nets, illustrated by example, and comparatively evaluated against other approaches.

Keywords: Flexible Manufacturing Systems; individualized mass customization; Industry 4.0; assembly planning; Petri nets

1. Introduction

In the evolving landscape of manufacturing, flexible manufacturing systems (FMS) stand as a response to the demands of the factory of the future. With the advent of Industry 4.0 and the desire for individualized mass customization, the necessity for adaptable and agile production methodologies is greater than ever. In this context, we want to build on our previous work [25] on self-organizing manufacturing systems, which are capable of autonomously adapting to changing requirements and conditions.

Project context. In particular, we want to show the usefulness of our paradigms in a production cell with industrial robots, to which tools like grippers and screwdrivers can be attached, and mobile robots that can transport parts and actively participate in the assembly process with an attached gripper. The cell can reconfigure itself by moving the robots around, and by changing the tools attached to the robots, allowing changes to the set of skills of each robot during operation. Usually, a team of robots is required to execute an assembly step. What distinguishes our

* Corresponding author.
E-mail address: lehner@isse.de

1877-0509
10.1016/j.procs.2025.01.248

approach from others [2, 20, 24] is the combination of our focus on producing variable products with the possibility of continuously placing orders for new products without significantly disrupting the operation of the cell.

To realize this vision, we have to address various challenges arising from the complexity of combining high flexibility with the needs of a practical and efficient production process. This paper introduces the modeling of flexible assembly plans as a crucial step towards achieving our goals. From this assembly plan, our system can autonomously decide during operation which steps to take next, which robots should execute them, and which parts should be used. We are implementing a component that precisely plans which services the robots have to execute in order to reach a state where the next step can be executed based on the current situation in the cell, which is unknown at design time. Finally, we also plan to implement the services that the robots have to execute in order to perform the assembly steps appearing in various example plans from our case study, which we will present in the following paragraph.

Case study. To illustrate and evaluate our approach to describing assembly plans, we build structures from elements of the *item MB Building Kit System* [11], mainly using aluminum profiles and angle brackets. Like other projects [20, 21], we use such structures because they allow for a high degree of flexibility in the assembly process using relatively few base parts. Consequently, product variations that share many parts and sub-assemblies can be easily defined, aligning well with our objectives.

Contribution. While the concept of assembly planning is not new, its relevance persists in the context of our project. A crucial prerequisite for flexible production is the use of flexible assembly plans that allow for partially ordered steps. Compared to specifying a fixed sequence of assembly steps [28] or generating such a sequence with automated planners before execution [22], we wish to provide a high degree of freedom for allowing self-organization in FMS.

In previous work [17], we have presented some approaches for generating parallel and interleaved execution plans for assembly jobs with multiple cooperating robots. In this work, we focus on a representation of assembly plans that allows the system to continuously make local optimization decisions about which robots should execute which assembly steps with which parts, just in time when the decision is needed. This is crucial for implementing a production cell where multiple teams of robots can work on fulfilling continually arriving orders for new products with many variations. By increasing the flexibility within our assembly plans, we specifically aim to achieve the following goals:

goal 1 Flexibility regarding the team of robots executing an assembly step: The assembly plan should only specify the skills required by the robots to execute a particular assembly step. The system can then find appropriate teams in different configurations of a cell that can reconfigure itself to adapt to current needs.

goal 2 Flexibility regarding the sequence and timing of assembly steps: The system should be able to decide on the fly in which order to execute the steps or if they can be executed in parallel. This allows the system to choose steps that can be executed efficiently by an available team of robots at any given time.

goal 3 Ability to respond to failures of robots or tools without major re-planning: This objective highlights the capacity to adjust production processes autonomously to ensure continuity and resilience. By decomposing the plan into small assembly steps assigned to individual teams, the system can reallocate tasks or reorder steps as needed to accommodate failures.

goal 4 Late allocation of parts or sub-assemblies that can be used for multiple assembly steps or products: This allows optimizing resource allocation, minimizing inventory, and enhancing efficiency by avoiding unnecessary predetermination of which among several identical parts will be used in each assembly step. This approach is particularly beneficial for products like the aluminum profile structures in our case study, where many parts can be used interchangeably in various steps. The system can thus often take any of multiple possible instances for the execution of an assembly step.

In this paper, we present an understandable model of adaptable assembly plans, integrating Petri net semantics, demonstrated through examples, and assessed in comparison to other proposals.

The remainder of the paper is as follows: In Section 2, we investigate related work concerning FMS as well as the current state of the art in assembly planning. The main contribution of this paper, our model of a flexible assembly plan, is presented in Section 3, where we will explain both the syntax of the assembly plans and the Petri net-based semantics. Section 4 will give some project context on how we envision the execution of our flexible assembly plans in an FMS. In Section 5, we discuss our approach in comparison to other methodologies, highlighting its advantages. Finally, we summarize our findings and provide a brief outlook concerning future work in Section 6.

2. Related work

Given the widespread demand for greater flexibility in manufacturing systems to meet the challenges of future factories, numerous approaches have been proposed. A review of many aspects related to intelligent manufacturing was given by Zhong et al. [29]. For this article, we will focus on the most relevant works that are closely related to our approach, the specific challenges we face in our project, and the use of assembly plans.

Some approaches for FMS still assume a fixed sequence of assembly steps [28]. Others use Assembly Sequence Planning (ASP) [1], which has the goal of generating an optimal assembly sequence. The input for ASP is often a liaison graph [5] or an And/Or graph [16]. Thomas et al. [26] pointed out that in many cases, constraints arising in actual assembly cells are not considered and introduced a method for ASP that also includes grasp planning with grippers for industrial robots. They still only considered planning for a concrete assembly with a fixed set of robots that execute various steps sequentially, falling short of our desired flexibility.

Nottensteiner et al. [20] presented a system for the automated production of a single product in a cell of robots. This involved not just an ASP implementation, but also mapping the assembly tasks to concrete skills from a skill library in order to execute the planned assembly sequences with real robots. An extension of their approach [22, 21] enabled the recognition of applicable assembly sequences on novel products based on machine learning.

For our self-organizing manufacturing cell, we need a more flexible approach than the classical ASP methods. Rather than planning the entire assembly process in advance, our approach allows for on-the-fly decisions about the next steps. This is necessary as new orders for products may arrive at any time, and the cell should be able to adapt to changes in the environment. This could invalidate previously determined optimal assembly sequences and lead to a re-planning of the entire process. Therefore, instead of a purely sequential order of assembly steps, we propose a more flexible approach that allows a partial ordering. This preserves the ability for the cell to dynamically decide between multiple possible next steps while the assembly of a product is in progress.

A prominent approach to implementing the factory of the future is that of Holonic Manufacturing Systems (HMS) [7]. Various architectures for these systems have been published, like PROSA [27], ADACOR, and its successor, ADACOR^2 [14]. In these architectures, a "product holon" represents the information about the products to be produced and their production methods. The suggested ADACOR semantics even considers a product made of independently assembled sub-products [14]. Still, these works focus on a general architecture for HMS instead of a particular realization of assembly plans dealing with the challenges we wish to address in our work. In recent years, Buckhorst et al. [2] developed a holonic approach to control Line-less Mobile Assembly Systems (LMAS).

Another approach is that of Evolvable Assembly Systems (EAS). EAS react to the changing demands on the products to be manufactured by reconfiguring a production line based on these needs [4]. A process for designing and controlling such systems, whose components inhibit self-x properties, was introduced by Sanderson et al. [24]. They define the production processes in "recipe files", which are labeled transition systems.

The differences between our approach and EAS [4, 24], LMAS [2, 3], and the work of the group around Nottensteiner et al. [20] and Rodríguez et al. [22, 21] are further elaborated in Section 5.

Nikolakis et al. [19] introduce an approach where a centralized node makes high-level decisions, while execution control is handled by function blocks wrapped in Docker containers deployed on edge devices. They use a task planner to break down the assembly process into these blocks before assigning them to resources for execution. This is in violation of our goal 2. Hu et al. [10] use deep reinforcement learning for scheduling processes and Petri nets to model both the processes and resources in an FMS. However, their approach comes with significant technological overhead and cannot easily be integrated into our system with its self-organizing teams of robots.

We utilize colored Petri nets (CPNs) to formalize the semantics of our assembly plans. Petri nets have been extensively used for assembly planning and control in manufacturing systems for several decades, as indicated by a survey by Rosell [23]. Among the works cited, some specifically address FMS, including Kasturia et al. [13]. Gradišar and Mušič [8] presented the use of timed Petri nets for analyzing a production system. More recently, Grobelna and Karatkevich [9] reviewed the role of Petri nets in modern manufacturing systems and the challenges faced in the context of developments like Industry 4.0. We leverage this established formalism to provide a semantic framework for assembly plans, without claiming novelty in its use. Unlike many other approaches, we do not focus on representing a concrete system with its robots and logistics using a particular Petri net. Instead, we introduce a general model of how an assembly plan can be used to guide the execution of the system.

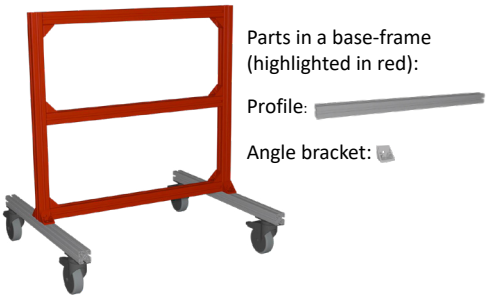

Fig. 1. Tool trolley assembled from item profiles.

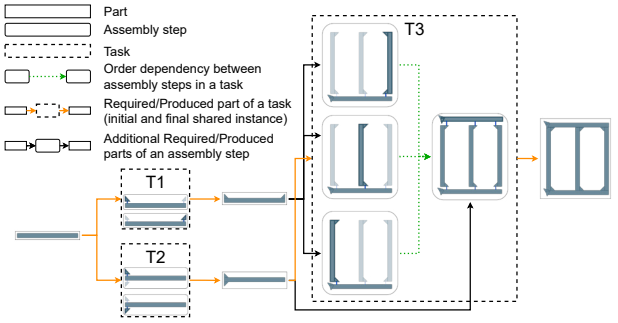

Fig. 2. Assembly plan for the base frame of the tool trolley.

3. Assembly plan representation

In previous work [25], we used self-organization to assign robots to certain assembly steps and to react to failures and new products, but the assembly plan was still predefined and fixed. In this work, we consider the assembly plan as the starting point for the assembly process. As an example within our case study (cf. Section 1), we will show an assembly plan of the "base frame" of a tool trolley we plan to manufacture, as highlighted in Fig. 1. An assembly plan for the base frame is illustrated in Fig. 2.

The **parts** can be either atomic parts, like the aluminum profiles in our case study, or assemblies of multiple sub-parts. More complex products might have many different atomic parts. Angle brackets and connecting parts like screws or slot nuts are considered bulk goods, which are assumed to be available for all robots in sufficient numbers. Thus, they are not relevant as individual objects when it comes to planning the next steps and are omitted from the assembly plan. Screws and slot nuts are omitted completely from the pictograms in Fig. 2. Assemblies contain multiple parts and connections between those parts, e.g., an aluminum profile and two angle brackets screwed to the profile.

The **assembly steps** are individual jobs that need to be executed in order to produce the product. Usually, a team of multiple robots is required to execute an assembly step. Each assembly step is characterized by its required and produced parts and the skills required by the robots forming the team executing the step. A simple screwing process might need two robots with a gripper to hold a part each and a robot with a screwdriver to screw them together. During the execution, the system chooses these robots based on the required skills and the current situation in the cell, as required by goal 1. It also decides which concrete part instances will be used, conforming to goal 4.

The **tasks** are collections of multiple assembly steps required to produce a certain part. This part can be a final product or a sub-assembly that can be used at multiple points in the same product or in different products. In Fig. 2, there are three tasks. While *T1* and *T2* produce different assemblies of an aluminum profile with two angle brackets, *T3* constructs the base frame of the tool trolley by connecting multiple copies of these sub-assemblies.

In the assembly produced by a task, a part must be defined to which other parts are iteratively connected. This part is the required part of the task and initially becomes its **shared instance**. Every step of the task then applies some changes to the shared instance, e.g., by connecting it to another part. After all steps have been completed, the shared instance is the part produced by the task. In addition to the shared instance, an assembly step may require additional parts, which it usually connects to the shared instance during its execution. The initial and final parts of a task are shown by the orange arrows in Fig. 2, while the black arrows indicate additional required parts of an assembly step.

As every assembly step in a task requires the shared instance of the task, only one of them can be executed at a time. However, unless explicitly specified otherwise, there is no fixed order for the steps in a task, and multiple executions of different tasks are allowed to happen in any order or in parallel, as we demanded in goal 2. The system can choose the most suitable next step based on the current situation in the cell. This can avoid unnecessary waiting times and allows reacting flexibly to changes or disruptions, as specified by goal 3.

Despite the demand for flexibility, it can be beneficial to maintain certain restrictions regarding the order of assembly steps, e.g., to avoid sequences that cannot actually be executed by the robots in the cell. This also reduces the search space when determining the next steps to execute. A process engineer can specify the sub-assemblies for constructing a product when determining the particular tasks in an assembly plan. They can also restrict the order of the assembly steps in a task to those deemed sensible, as indicated by the green dotted arrows in task *T3* in Fig. 2.

The step at the source of an arrow must be executed before the step at the target. It is therefore possible to define any partial order of the steps within a task, but the number of such restrictions should be kept to a minimum to preserve flexibility for the execution of the assembly. Steps with no order dependency between them can be executed in any order.

3.1. Petri net representation

We have developed a concept to translate the comprehensible assembly plan representation as shown in Fig. 2 into a representation using colored Petri nets (CPN) [12]. Utilizing Petri nets enables us to depict and analyze the semantics of our assembly plan using a widely recognized and established formalism. The Petri net allows us to recognize which steps can be executed next and which parts can be used for the execution.

In a Petri net, tokens represent objects or pieces of information and move between places (graphically shown as circles) via the firing of transitions (graphically shown as rectangles). In CPNs, places are associated with a type, and the tokens have a particular value ("color") of that type. The arcs connecting places and transitions define the tokens to be consumed and produced by a transition. An arc can be annotated with an expression restricting the values of these tokens. Variables in these expressions can be bound to any value, but multiple occurrences of the same variable in the incoming or outgoing arcs of a transition must be bound to the same value. In our models, the types of tokens are either tuples of boolean values or simple units without additional information. In the latter case, the places are not annotated, and the tokens are represented by black dots. The expression for the arcs connected to these places can be omitted, as an arc will by default consume or produce one such token.

Fig. 3 shows the Petri net corresponding to task *T1* from Fig. 2. The different visual styles of the places and the dotted boxes around each task serve only as a visual aid and don't have any semantic meaning for the CPN execution model. We want to remark that the Petri net is only used to define formal semantics for the assembly plan. The more intuitive representation in Fig. 2 is the one used by a process engineer and by users of the system.

For every distinct part that appears in an assembly plan, there is a white place with each token in it representing a particular instance of the part. For every ongoing execution of a task, there is a token representing its state. This token is a tuple of boolean values, where each position corresponds to one assembly step of the task and has the value 1 if the step has been executed and 0 otherwise. If no step is currently being executed, the token is in the place with solid diagonal lines, representing shared instances of task executions that are available for further assembly steps. During the execution of an assembly step, the token is in the place with the dashed lines for that step.

As Fig. 3 shows, every step has a transition firing at the start of its execution and one firing at its completion. The starting transition consumes a token from the place with the solid diagonal lines with the value 0 at the position corresponding to the step itself and puts it in the place with the dashed lines for the step. This token represents the shared instance of a task execution where the step has not yet been executed. The transition for the completion of the step consumes this token and places a token with the value 1 at this position in the place with solid diagonal lines.

In the absence of order restrictions, the tuple's other positions are represented by variables, allowing both 0 and 1 values. To model an order dependency where step S_1 must precede step S_2, the value at the position for S_1 must

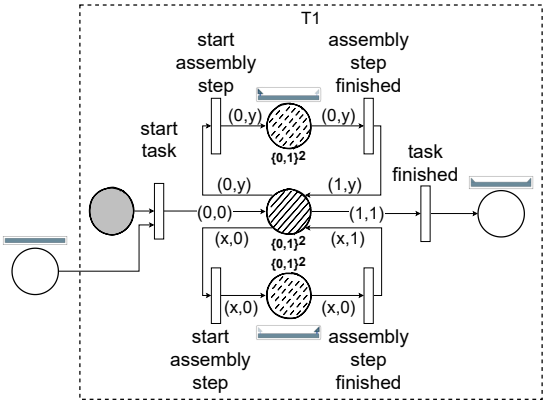

Fig. 3. Petri net representation of the task *T1*.

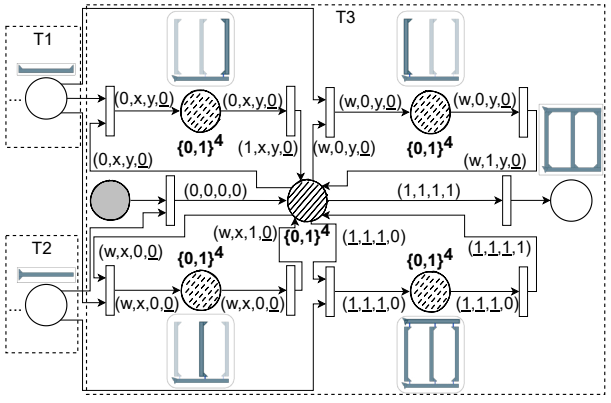

Fig. 4. Petri net representation of the task *T3*.

be set to 1 when the execution of S_2 starts. The value of the position for S_2 is fixed to 0 during the execution of S_1. Fig. 4 shows this by the example of task *T3* of Fig. 2, where the bottom right step must be executed after the others. The fixed values arising from this order dependency are underlined in the figure.

Each task has a gray place for tokens for initiated executions of the task that have not yet been started. This place is re-filled at runtime based on the orders arriving at the cell, as detailed in Section 4. When the task execution is started, a token is consumed from this place, and the initial shared instance is assigned.

The transitions for starting assembly steps and tasks consume a token from the white place for each of their required parts. If a token with all values set to 1 is in the shared instance place of a task, this triggers the transition for finishing the task. It consumes this token and puts a token representing the produced part in the white place for the finished parts of the task.

As seen in the Petri net models in Fig. 3 and Fig. 4, for the execution of a task or assembly step, any instance from the place corresponding to a required part type can be selected. This allows the system to choose the most appropriate instance based on the current situation in the cell. In Fig. 4, there is no difference between the part used as the initial shared instance for task *T3* and the part used for the assembly step shown at the bottom right. The decision as to which concrete part will be used in which of these roles is only made when the part is actually used in a concrete assembly step, as outlined in goal 4. This empowers the system to select optimal next steps and teams based on predefined criteria, e.g., how close the parts are to the robots that can form a team executing a subsequent step. This can lead to a reduction in the number of required transports and, ultimately, the time needed to produce a product.

We want to allow multiple products, each with their own assembly plan, to be assembled in the cell at the same time. Each active assembly plan is translated into a Petri net model. If the same part is used in multiple assembly plans, the place representing this part is shared between their Petri net models. Therefore, the assembly plans known to the cell are translated into a vast Petri net with shared places and entire tasks. This means that even the product a particular sub-assembly will be used for is not fixed when this sub-assembly is produced. The approach proves especially advantageous for modular product families, where the re-use of parts, sub-assemblies, and tasks is common.

In summary, the simple plan representation from Fig. 2 is useful for a human expert to define the assembly plan and for users of the system to understand it easily. Meanwhile, the Petri net representation is useful for defining semantics on how the autonomous execution of the assembly within the cell should be done.

3.2. Design of assembly plans

Typically, various assembly plans can be devised for a product. Crafting a good assembly plan for our system hinges on two main considerations. Firstly, every assembly step must be executable by a team of robots in the real cell. Secondly, as much flexibility as possible should be preserved for the cell to freely choose appropriate steps at assembly time, as outlined by our goal 1 and goal 2 presented in Section 1. Balancing these considerations involves trade-offs; increased flexibility demands a broader range of defined processes for the robots to execute. In addition to these main considerations, sometimes domain-specific constraints can also play a role in deciding which assembly plans for a product are best for enabling quick and problem-free assembly.

While these considerations show that not all possible assembly plans are equally good, there is still often no clear best plan, and multiple design decisions are possible. For instance, in the plan introduced in Fig. 2, task *T3* could be split into two tasks. In that case, the first task would only assemble the “E”-shaped sub-assembly with the three independent assembly steps, and the second one would then connect it to the final aluminum profile to complete the base frame. When producing variations of a tool trolley, the “E”-shaped sub-assembly is unlikely to be used in other situations than for producing the base frame. Therefore, this separation would not lead to additional flexibility. Also, the plan as shown in Fig. 2 allows us to demonstrate the feature of defining order dependencies between the assembly steps of a single task.

Currently, the decomposition of a product into sub-assemblies, and consequently, the structuring of its assembly plan into tasks and steps, remains a human expert’s prerogative. We believe that the presented formalisms can be adopted for various other production processes beyond the case study presented in this paper. The exact design of assembly plans is highly dependent on the specific product and the production environment, e.g., the available robots and the layout of the production cell. However, there are future research opportunities to automate the process of assembly plan production. This could be achieved by identifying already known sub-assemblies in CAD-designed products or leveraging machine learning for product decomposition.

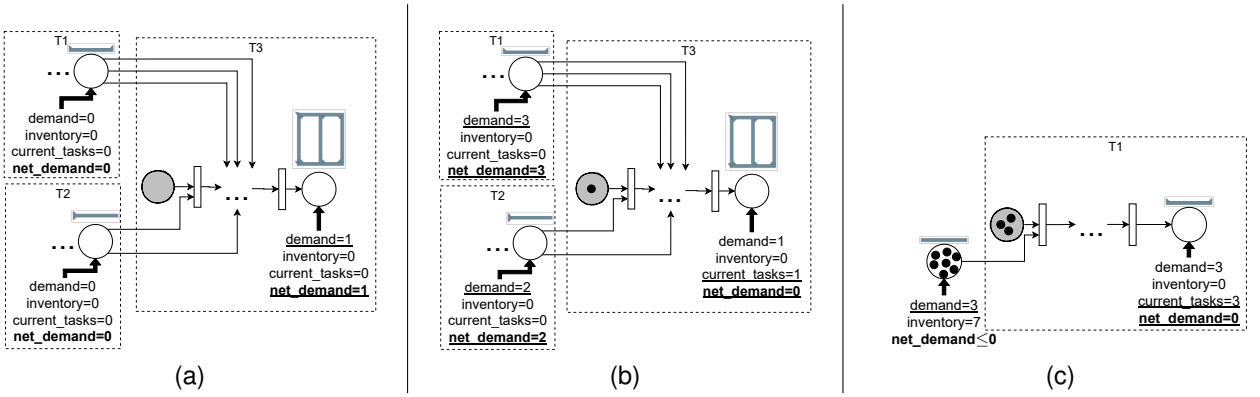

Fig. 5. Autonomous recursive starting of tasks after a demand for a base frame of the tool trolley. In (a), *T3* receives the demand for the base frame. In (b), *T3* is started (token in gray place) and demands for required parts are made. In (c), *T1* is started after *T3* made a demand for its produced part. The analogous step for *T2* is omitted for brevity. In management components, changes compared to the previous state are underlined.

4. Assembly plan execution

To elucidate the role of our assembly plans in autonomous product assembly within the FMS, we will outline the initiation of tasks and decision-making processes for step execution. Specifically, we want to describe in detail how the assembly plans allow finding potential next steps to execute by updating the assembly plan execution state based on the Petri net model. Subsequently, we will quickly introduce the other important components of our intended system in order to choose among these potential next steps, assign them to teams of robots, and execute them. Some of these components have already been implemented, while others are still under development. A detailed description of the system will be presented in the future.

We have successfully implemented the mechanism that updates the state of the assembly plan based on the Petri net and identifies potential next steps to execute. Our system autonomously initiates tasks in response to a demand for the parts to be produced. A demand can arise from an external production order or when another task requires the part for its execution. Fig. 5 depicts the autonomous recursive starting of tasks in our system using the Petri net model. In this example, at the time of receiving this demand, no relevant sub-assemblies are already produced, no tasks are running, and an adequate number of profiles have been provided to the cell.

Every part has a management component that keeps track of the demand for that part, the number of existing instances of the part (inventory), and the number of unfinished initiated tasks for producing new instances (current_tasks). In the Petri net (cf. Fig. 5), the inventory is the number of tokens in the white place for parts produced by the task. The current_tasks are the sum of the tokens in the gray place (no step was started yet), the place with the solid diagonal lines (execution of the task was started; currently no step is executed), and the places with dashed lines (a step is currently executed) for that task. The net_demand is the number of parts that must be produced to meet the demand and for which no tasks have started yet. It is calculated as net_demand = demand – inventory – current_tasks.

If a demand for a part is received, its management component updates the demand and recalculates the net_demand. It is an invariant of every management component that the net_demand must be zero or negative. If this is violated, the management component initiates new tasks until the net_demand is zero. In the Petri net, a token is placed in the gray place, as shown in Fig. 5b and 5c. If the net_demand does not exceed zero, the management component does not initiate a new task, as the parts already existing or in production can fulfill the demand. This may occur, e.g., if some sub-assemblies were pre-produced or if a previous order was cancelled.

When putting a token in the gray place, the management component also makes a demand for all parts required by the task. In the Petri net, those are the parts represented by white places with an arc to the task or any of its assembly steps. Fig. 5 shows how a demand to produce a product often triggers a chain of demands for all required sub-assemblies and parts. *Atomic parts*, like the aluminum profiles in our case study, must be supplied from outside. The management components for these parts still maintain demand and inventory. If the demand for an atomic part exceeds the inventory, the management component triggers a message alerting the cell operator of the material shortage.

A step is executable if the transition for starting it is enabled, i.e., there is a token in the shared instance place with appropriate values and a token in each white place for other required parts. An exception applies to the first step of a

task, as the shared instance is only assigned when the first step is chosen for execution. Therefore, if the transition to start a task is enabled, any potential first step, i.e., any step whose starting transition could fire if the transition starting the task fired immediately before, is a potential next step.

After identifying the potential next steps, the system must decide: (1) which of these potential next steps should actually be selected for execution; (2) which concrete part instances should be used in the chosen steps; and (3) which team of robots should execute the chosen steps.

We have specified a Constraint Satisfaction and Optimization Problem (CSOP) using MiniZinc [18] and use the Chuffed solver to find solutions. The input data for the problem are the current state of the cell and the potential next steps, including the parts they require and the skills the robots in the team executing the step must possess. Constraints ensure the chosen robots possess these skills, have the ability to form a team, and can be supplied with the chosen parts for the step execution, among others. The optimization criteria lead to choosing steps that can be executed as quickly as possible and prioritizing the most urgent steps. While we already see that the constraint solver approach works as we use it in our system, we still need to evaluate it in more detail when it comes to performance and experiment with possible improvements. In future publications, we will show a quantitative analysis comparing it with other potential approaches to solving the problem of choosing steps and allocating robot teams for them.

The robots in the cell offer a variety of services, some of which are already implemented while others are still under development. After selecting a step and the team to execute it, a planning component schedules the services to call for the robots in the team. After the robots have performed those services, the step execution is finished, and the team allocation component will again assign the next steps to be executed. At the time of writing this article, this planning component is actively being developed and will be presented in future work.

The Petri net transitions to start and finish a task or assembly step execution fire when the respective event happens. An order, started by a demand for a product, can be fulfilled if there is a token in the white place for that product. As with the assigning of the next steps, if multiple orders compete for a product, the system decides which order will be completed with which part instance. When a part instance is used for a task or assembly step or to fulfill an order, the token in the white place for that part is removed, and its management component decreases the demand by one.

This mechanism of starting and executing tasks ensures that all orders can be autonomously fulfilled if a sufficient number of atomic parts are supplied, a robot team can be found for every step, and there are no unexpected errors during the execution. Practical challenges arising in the cell, e.g., error recovery, dealing with limited capacities to store products, and deadlock avoidance, are not addressed in the Petri net model. They must be handled by other system components based on the specific requirements of the cell.

5. Discussion

In this section we wish to discuss the usefulness of our approach by comparing it with some other approaches from the literature. For each of these, we will explain why we chose it for comparison and how it differs from our approach. For this analysis, we will focus on the goals we have set for our approach in Section 1.

The approach of Evolvable Assembly Systems (EAS) [4, 24] is interesting for us because it also focuses on a system with self-x properties that is able to reconfigure itself based on its requirements. One strength of EAS is the ability of the system to change its own configuration based on the product to produce, so that appropriate processes for the product to find its way through the cell can be found. This conforms to our goal 1. They also consider that the system should be able to adapt to failures, conforming to goal 3. Towards goal 2, the use of labeled transition systems to model the processes for assembling a product proposed in [24] does allow for more flexibility in the order of assembly steps than a fixed sequence. On the other hand, the focus of the EAS concept is that the entire cell will change its configuration based on factors like the product to produce. Our work instead considers a cell that may be reconfigured based on particular needs but also allows many different product variants to be produced at the same time in an interleaved fashion or even in parallel by different teams. Furthermore, there is also no mention of a dynamic assignment of re-usable parts to different products, as our goal 4 requires.

LMAS [2, 3] is another approach that is interesting for us. It also focuses on a system that should be able to autonomously assemble multiple products, potentially in parallel, in a highly flexible and reconfigurable cell. In LMAS, a product is associated with a precedence graph of the assembly steps. Therefore, the definition of the product already contains some restrictions regarding the order of assembly steps while leaving some flexibility to the system. This

is quite similar to our work as it also defines fixed operations for the robots on the shopfloor to execute while still allowing the planning components integrated in the system to decide the exact schedules of the operations and their distribution to the various stations and their robots. Hence, it covers goal 1 and goal 2 quite well. Furthermore, goal 3 is partially covered, as LMAS does have mechanisms to recover from failures, but those might trigger a global re-planning of the already decided schedules. By only planning a few next steps at a time, our approach does not involve any costly and large re-plannings in case of failures or other disruptions. On the flip side, we might have to accept a less globally optimal schedule than LMAS or other approaches that plan larger parts of the process at once. Finally, goal 4 is not covered by LMAS as the parts have a fixed connection to the product they are part of from the beginning. This is a legitimate assumption if the individual products do not usually share parts. However, in domains where there are many parts and sub-assemblies that can be re-used in multiple products, we found it necessary to allow for a more flexible allocation of parts to the final products. This is done by only deciding which of multiple identical parts to use in an assembly step when it is actually needed. This requires a separation of the production of these re-used sub-assemblies from their actual usage in a product, motivating our concept of tasks whose produced components can then once again be flexibly used in later tasks.

We also analyzed the work of the group around Nottensteiner et al. [20] and Rodríguez et al. [22, 21]. Their example domain conforms to our case study, i.e., they also aim for the assembly of aluminum profiles connected by angle brackets. Yet, their focus is quite different from ours, as they only consider the assembly of a single product at a time in a cell in which all robots can work together. They do not cover goal 1 and goal 2 as they do not consider the possibility of multiple products being assembled in parallel by different teams. As a consequence, they also do not cover goal 4, as the idea of assigning a part to one of multiple possible assembly steps or products is not applicable in their case. Their approach is able to avoid many failure scenarios, such as collisions between parts or robots, during its assembly planning [22]. However, they do not address goal 3 as we have defined it, i.e., the ability to recover from unexpected failures of robots and tools during the production process.

In summary, our approach to representing assembly plans for FMS has several advantages over the existing approaches we found. Achieving the goals we have outlined in Section 1 means that no pre-planning of the entire execution, including all eventualities, is necessary. We could not determine any existing methodology that fulfills all these goals. Especially, goal 4 is rarely considered by others. Therefore, we believe that the approach we have presented is a valuable contribution towards achieving practicable self-organizing production cells where many variations of a product can be produced in parallel.

6. Conclusion and future work

The paper introduced a novel approach for representing assembly plans within a flexible manufacturing cell, addressing the growing demand for adaptable and agile production methodologies in modern manufacturing. We showed a comprehensible approach to modeling assembly plans based on clear semantics using Petri nets. With an assembly case study, we illustrated the various reusable modeling elements, thus highlighting the flexibility of the approach. By leveraging these elements, our approach enables the dynamic adaptation of production processes to varying requirements and conditions, thereby enhancing the efficiency and responsiveness of manufacturing operations. Through this modeling methodology, we can achieve the goals outlined in the introduction: (1) flexible assignment of teams of robots to assembly steps; (2) flexible decisions on the order and timing of steps; (3) error handling; and (4) allocation of parts at the latest possible stage.

In the future, we plan to closely examine how well our approach works when putting it into practice in a flexible manufacturing cell. For this, we can also build on our previous achievements for executing static assembly plans with our service architecture for distributed robot control [6]. We plan to finish, refine, and evaluate the various components of our system, which we briefly introduced in Section 4, and integrate the assembly plans with these components into one system architecture. As all components of our system are realized with the Robot Operating System (ROS2) [15], they can be relatively easily integrated with the services of the real robots in the cell.

By conducting rigorous testing and evaluation, we aim to elucidate the practical implications and potential benefits of our methods for enhancing manufacturing processes, allowing us to better understand their performance and effectiveness in real-world scenarios. In addition, we plan to conduct further studies to explore the applicability of our approach across various case studies, enabling comparative analysis with existing methodologies from the literature.

As we already mentioned in Section 3.2, a concrete future research direction is the automation of the assembly plan creation from a CAD model of the product to be assembled.

References

[1] Bahubalendruni, M.R., Biswal, B.B., 2016. A review on assembly sequence generation and its automation. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 230, 824–838.

[2] Buckhorst, A.F., Grahn, L., Schmitt, R.H., 2022. Decentralized holonic control system model for line-less mobile assembly systems. Robotics and Computer-Integrated Manufacturing 75, 102301.

[3] Buckhorst, A.F., Huettemann, G., Grahn, L., Schmitt, R.H., 2019. Assignment, sequencing and location planning in line-less mobile assembly systems, in: Tagungsband des 4. Kongresses Montage Handhabung Industrieroboter, Springer. pp. 227–238.

[4] Chaplin, J., Bakker, O., De Silva, L., Sanderson, D., Kelly, E., Logan, B., Ratchev, S., 2015. Evolvable assembly systems: a distributed architecture for intelligent manufacturing. IFAC-PapersOnLine 48, 2065–2070.

[5] De Fazio, T., Whitney, D., 1987. Simplified generation of all mechanical assembly sequences. IEEE Journal on Robotics and Automation 3, 640–658.

[6] Eymüller, C., Hanke, J., Poeppel, A., Wanninger, C., Reif, W., 2023. RealCaPP: Real-time capable Plug & Produce service architecture for distributed robot control, in: 2023 Seventh IEEE International Conference on Robotic Computing (IRC), pp. 352–355.

[7] Giret, A., Botti, V., 2009. Engineering holonic manufacturing systems. Computers in industry 60, 428–440.

[8] Gradišar, D., Mušič, G., 2007. Production-process modelling based on production-management data: a petri-net approach. International Journal of Computer Integrated Manufacturing 20, 794–810.

[9] Grobelna, I., Karatkevich, A., 2021. Challenges in application of Petri nets in manufacturing systems. Electronics 10, 2305.

[10] Hu, L., Liu, Z., Hu, W., Wang, Y., Tan, J., Wu, F., 2020. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. Journal of Manufacturing Systems 55, 1–14.

[11] item Industrietechnik GmbH, . item MB Building Kit System. https://de.item24.com/en, accessed on 2024-05-08.

[12] Jensen, K., 1998. An introduction to the practical use of coloured Petri nets. Lectures on Petri Nets II: Applications , 237–292.

[13] Kasturia, E., DiCesare, F., Desrochers, A., 1988. Real time control of multilevel manufacturing systems using colored petri nets, in: Proceedings. 1988 IEEE International Conference on Robotics and Automation, IEEE. pp. 1114–1119.

[14] Leitao, P., Colombo, A.W., Restivo, F., 2003. An approach to the formal specification of holonic control systems, in: International Conference on Industrial Applications of Holonic and Multi-Agent Systems, Springer. pp. 59–70.

[15] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W., 2022. Robot Operating System 2: Design, architecture, and uses in the wild. Science robotics 7, eabm6074.

[16] Homem de Mello, L., Sanderson, A., . AND/OR graph representation of assembly plans. IEEE Transactions on Robotics and Automation 6, 188–199.

[17] Nägele, L., Schierl, A., Hoffmann, A., Reif, W., 2019. Multi-robot cooperation for assembly: Automated planning and optimization, in: International Conference on Informatics in Control, Automation and Robotics, Springer. pp. 169–192.

[18] Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G., 2007. Minizinc: Towards a standard cp modelling language, in: International Conference on Principles and Practice of Constraint Programming, Springer. pp. 529–543.

[19] Nikolakis, N., Senington, R., Sipsas, K., Syberfeldt, A., Makris, S., 2020. On a containerized approach for the dynamic planning and control of a cyber-physical production system. Robotics and computer-integrated manufacturing 64, 101919.

[20] Nottensteiner, K., Bodenmueller, T., Kassecker, M., Roa, M.A., Stemmer, A., Stouraitis, T., Seidel, D., Thomas, U., 2016. A complete automated chain for flexible assembly using recognition, planning and sensor-based execution, in: Proceedings of ISR 2016: 47st International Symposium on Robotics, VDE. pp. 1–8.

[21] Rodríguez, I., Nottensteiner, K., Leidner, D., Durner, M., Stulp, F., Albu-Schäffer, A., 2020. Pattern recognition for knowledge transfer in robotic assembly sequence planning. IEEE Robotics and Automation Letters 5, 3666–3673.

[22] Rodrıguez, I., Nottensteiner, K., Leidner, D., Kaßecker, M., Stulp, F., Albu-Schäffer, A., 2019. Iteratively refined feasibility checks in robotic assembly sequence planning. IEEE Robotics and Automation Letters 4, 1416–1423.

[23] Rosell, J., 2004. Assembly and task planning using Petri nets: a survey. Proceedings of the institution of mechanical engineers, part B: journal of engineering manufacture 218, 987–994.

[24] Sanderson, D., Chaplin, J.C., Ratchev, S., 2019. A function-behaviour-structure design methodology for adaptive production systems. The International Journal of Advanced Manufacturing Technology 105, 3731–3742.

[25] Seebach, H., Nafz, F., Steghöfer, J.P., Reif, W., 2011. How to design and implement self-organising resource-flow systems. Organic Computing—A Paradigm Shift for Complex Systems , 145–161.

[26] Thomas, U., Stouraitis, T., Roa, M.A., 2015. Flexible assembly through integrated assembly sequence planning and grasp planning, in: 2015 IEEE International Conference on Automation Science and Engineering (CASE), IEEE. pp. 586–592.

[27] Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P., 1998. Reference architecture for holonic manufacturing systems: PROSA. Computers in industry 37, 255–274.

[28] Wang, S., Wan, J., Zhang, D., Li, D., Zhang, C., 2016. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. Computer networks 101, 158–168.

[29] Zhong, R.Y., Xu, X., Klotz, E., Newman, S.T., 2017. Intelligent manufacturing in the context of industry 4.0: a review. Engineering 3, 616–630.