Model-based generation of manufacturing process plans through incremental topology formation

Omar Adalat, Muhammad Talal, Mohammed Adem Ali Cherif, and Daniele Scrimieri

Department of Computer Science, University of Bradford, Bradford BD7 1DP, UK
{o.j.adalat,m.talal,m.a.alicherif2,d.scrimieri}@bradford.ac.uk

Abstract. In advanced manufacturing systems, the production of complex and highly customisable products requires the preparation of many different product specifications and associated manufacturing process plans. The creation of these plans involves the search for the production resources (e.g. robots, machine tools, inspection devices) that are needed to implement the product specifications and how to orchestrate them. We present a model-based approach to the automatic generation of manufacturing process plans from the models of the target products and available resources. The modelling language is based on labelled transition systems, which are useful to represent sequences of operations that can be executed in parallel by multiple production resources. Some preliminary experimental results demonstrate the feasibility of the presented approach.

Keywords: process plan, controller, realisability, labelled transition system, robotics, manufacturing

1 Introduction

Modern manufacturing markets are characterised by high product diversity and short life cycles. In this landscape, manufacturing companies need to respond promptly to ever-changing customer requirements and adapt their production systems accordingly. The digital transformation started by the forth industrial revolution (Industry 4.0) is introducing new levels of adaptability for manufacturing systems, which are becoming more and more intelligent, autonomous and decentralised.

Technology in the area of *automatic computer-aided process planning* [1] plays a fundamental role in achieving the vision of "smart factories", whereby production facilities are organised with minimal human intervention. Given the design of a product, process planning consists of selecting the manufacturing processes (e.g. assembly, machining, additive manufacturing) and equipment, and defining the sequence of operations to implement that design. Despite the considerable research to automate process planning, there are still many barriers due to the diversity of the tasks involved and their multidisciplinary nature.

This paper addresses the problem of determining whether a given product can be manufactured with the available production resources and, if so, how to

orchestrate the available resources to manufacture the product. Both the 'recipe' to make a product and production resources are modelled in terms of production stages and operations using labelled transition systems. We present an algorithm to both check if a product can be made and, if that is the case, generate at the same time a process plan or high-level 'controller' that can execute the required operations on appropriate resources. Controllers are also represented with a labelled transition system consisting of composite states and transitions including the states and transitions of the individual resources.

Our approach differs from existing ones, such as [6], in that only the relevant states of the 'topology', i.e. the composition of all the resources' labelled transition systems, are explored for potentially generating a controller. This is particularly useful for reconfigurable or 'plug-and-produce' systems, in which production resources are added or removed dynamically, thereby altering the topology.

2 Background and related work

Flexible and reconfigurable manufacturing systems have been designed with the objective of facilitating system changes to adjust functionality and production capacity in response to market changes [8]. For example, the Evolvable Assembly Systems project investigated the architecture and overall philosophy of manufacturing systems with enhanced responsiveness to changes in products, processes and markets [4]. The research built on the PRIME project [3], which used agent technology to configure adaptive plug-and-produce manufacturing systems. Agent-based systems and 'holonic' control architectures for manufacturing systems have contributed significantly to the dissemination of Industry 4.0 technologies [7].

Capabilities of production systems are modelled in various research works, in which they are matched against production requirements as part of reconfiguration processes [2,10]. Although the modelling aspects (using, in particular, ontologies) and the architectures (in the form of, e.g., multi-agent systems) have been extensively analysed, the computational problems of matching capabilities and requirements, as well as generating process plans deserve further investigation. Algorithms for 'topology generation', 'realisability' of a product recipe and 'control' are presented in [6,9]. Our research develops a more integrated and efficient solution by synthesising a controller (if the recipe is realisable) during the topology generation process itself. Capability modelling could also support automated learning for adapting plug-and-produce systems. An experience-based learning technique is presented in [11].

3 Production resources

Production resources, or simply resources, are formalised as Labelled Transition Systems (LTSs). A Labelled Transition System is a quadruple $(S, \sum, s_0, \rightarrow)$, where S is a finite set of states, $\sum$ is a finite set of labels, $s_0 \in S$ represents the initial state, and $\rightarrow$ is a transition relation such that $\rightarrow \subseteq S \times \sum \times S$ (written

$s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$). The states represent the stages of the production process of the resource. The transitions represent particular operations that can occur. The list of operation types is as follows:

- Observable operations, which are the manufacturing capabilities or behaviours possible within each resource.
- Synchronisation (or transfer) operations, which are internal operations represented with in_n and out_n transitions ($n \in \mathbb{N}$). They model the transfer of a part between resources. If an out_n transition is executed in the current state of a resource, it is expected that an accompanying in_n transition is simultaneously executed from the current state of another resource.
- Idle operation, a special operation that represents the resource idling, denoted by the transition label 'nop'.

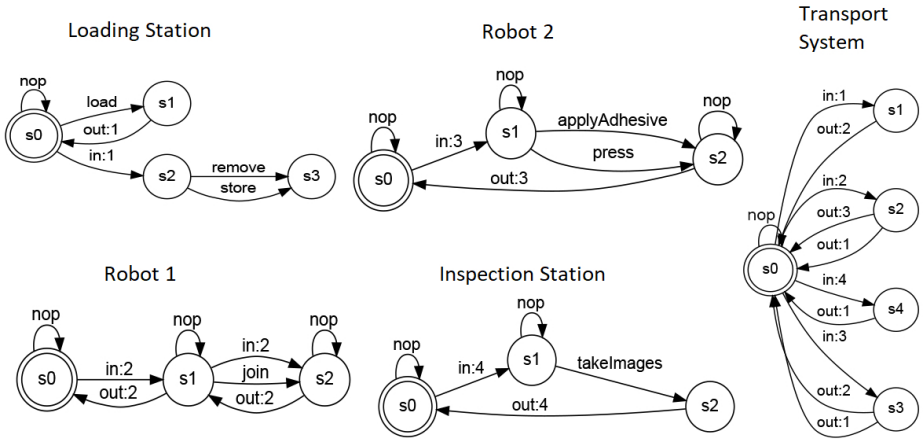


Fig. 1: Resources of a robotic production system modelled with LTSs. Double circles denote the initial states.

Figure 1 shows the LTSs of the robotic production system that will be used in the experimental evaluation (based on the system described in [6]).

4 Product recipe

A product recipe, or simply recipe, specifies the sequence of operations that must be carried out to manufacture a certain product. A recipe is also represented as a LTS. A recipe is independent of the production resources: no underlying assumptions about any of the resources are made. Each state represents a particular stage of the process flow and transitions represent composite operations. A composite operation is formed of the following:

– A guard operation, that is a test condition applied to input parts. The guard returns a success or failure flag depending on the outcome of the test.
– A variable number of sequential operations. Each sequential operation is also able to specify a variable number of input and output parts.

Without guard conditions, the product would be unconditionally formed, in spite of manufacturing defects that are expected to occasionally occur.

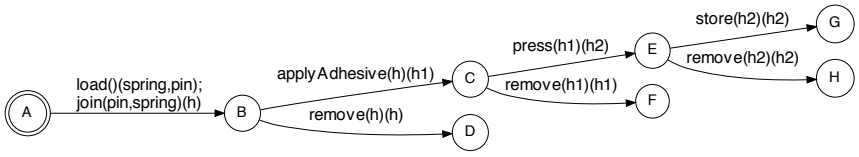


Fig. 2: The recipe for assembly a hinge formalised as a LTS.

Figure 2 displays an example of a recipe for assembling a hinge used in the cabin of a truck. Firstly, the hinge pin and springs are loaded and attached together. Then, an adhesive glue is applied to the hinge. Dependent on the guard conditions and their runtime output, the hinge is then either stored or removed.

5 Topology

A complete topology is defined as the cartesian product of the production resources' Labelled Transition Systems where synchronisation transitions are only present if the inverse transition is present within another resource in the current combined state. A combined state is a n-tuple where n is the number of resources and each tuple element represents the state of an individual resource. For example, the combined state $(s_1, s_2, s_3, s_4, s_5)$ indicates that the first resource is in s_1, the second resource is in s_2, etc. The maximum total number of topology states is bound by $\prod_{i=1}^{n} \lambda_i$, where λ_i denotes the number of states in resource i and n is the total the number of resources.

To generate the complete topology, all possible transitions are applied, starting from the combined initial state, until no further transitions can be applied from any of the individual resources. In order to visit all the other states, a depth-first search or breadth-first search may be used as the exploration mechanism. Since a particular state may be explored or reached by multiple different transitions, topology states that have already been visited must be tracked and flagged as such in order to prevent their duplicate processing.

5.1 Incremental formation

Forming the complete topology is extraneous when only a specific product is targeted. Moreover, due to the exponential nature, computing the complete

topology may become computationally infeasible when provided a large enough number of resources. As a result, it is possible to incrementally form the topology whilst generating a process controller, where only necessary and incremental expansions occur.

To form the incremental topology and controller that targets a specific recipe, the initial states of all resources are gathered as normal. Then, the topology states are expanded on an on-demand basis (as they are needed during controller synthesis), instead of preemptively applying all possible transitions. Consequently, this precludes the topology from containing states and transitions of operations that are not pertinent towards the target output.

Algorithm 1: ExpandState

```
Data: CombinedState, Topology τ, VisitedStates, Resources
Result: State
if CombinedState ∈ VisitedStates then
    return CombinedState
end
VisitedStates ← VisitedStates ∪ {CombinedState};
forall Resources do
    forall Transition ∈ CombinedState_Resource do
        if Transition is synchronisation then
            if FindInverse(Transition, Resources, CurrentResource) then
                δ ← InverseTransitionState with TransitionState;
                τ ← AddTransition(CombinedState, Transition, δ);
                τ ← AddTransition(InverseState, InverseTransition, δ)
            else
                continue;
            end
        else
            δ ← TransitionState;
            τ ← AddTransition(CombinedState, Transition, δ)
        end
    end
end
return CombinedState
```

Algorithm 1 is used to expand a topology state. ExpandState first performs a constant-time lookup to determine whether or not the given combined state has already been expanded and, if so, returns it. Otherwise, the state is expanded through iterating over all of the individual resources and applying all possible transitions, with due consideration to synchronisation transitions. Function *AddTransition* takes a triple (start state s, action a, end state s'), representing the transition $s \xrightarrow{a} s'$, and adds it to the topology. *FindInverse* searches all the other resources at their respective states (from the current combined state) for a matching synchronisation operation.

Since the incremental topology is a subset of the complete topology, a fully-expanded incremental topology is the same as the complete topology. A partial topology alleviates and minimises the state space explosion problem [12]. Otherwise, when the entire topology is present (or already largely expanded), controllers can be generated in effectively negligible time.

6 Controller synthesis

A process *controller* details the steps for each production resource to execute in order to manufacture the product. A controller is first created, in the interim, as a Labelled Transition System. The controller Labelled Transition System spans a subset of the entire topology, which includes only the states and transitions that are required to form the recipe. It is possible that a controller cannot be synthesised, in which case the recipe is deemed non-realisable. This may be due to either lacking the needed input parts from the recipe at some point throughout controller synthesis or the inability to perform the necessary operations because they are not present within any of the reachable resource states. If the recipe is realisable, once a controller Labelled Transition System is created, it can then be refined into low-level machine language code (such as G-code for CNC machines), or used within the production facility via a higher-level language such as the Business to Manufacturing Markup Language (B2MML).

A controller can be synthesised once a topology is available, or the topology can be expanded alongside control generation, as demonstrated within Section 5.1. Generating the controller uses a recursive depth-first strategy with backtracking. Ostensibly, the idea behind controller synthesis can be thought of as a simple traversal of the topology. However, other factors must be considered that distinguish the controller synthesis problem as a *constraint satisfaction problem*. If a part is not present within a resource but it is required by the recipe, the controller is no longer realisable. The parts are considered present if there is a one-to-one mapping of the name of the parts present within the resources and the name of the parts within the recipe. When the topology cannot directly transition and fulfil the target behaviour/operation, transfer operations between resources must be considered (iterating over all synchronisation operations, again in a depth-first search). Controller synthesis is also an optimisation problem, since it is possible that shorter routes during the traversal exist that minimise the number of synchronisation operations followed or resources used, hence resulting in a *best controller*.

A controller that realises the hinge recipe is shown in Figure 3. A controller is synthesised using Algorithm 2, in which four supplementary functions are used: $AllocateParts$, which ensures that parts are present at the required resources; $SynchroniseParts$, which synchronises the parts when transfer pathways are followed; and $AddParts$, which intuitively adds the output parts of an operation to the correct resource, and $AddTransition$. $AddTransition$ differs to Algorithm 1's definition as a check is performed against resources that do not transition to ensure they all contain the valid *nop* (idling) transition for their current states.

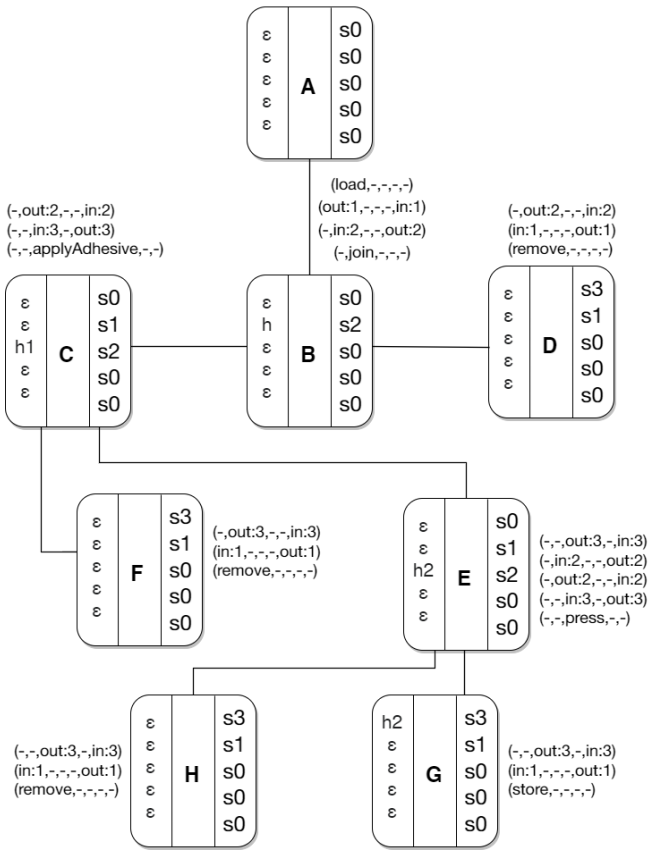


Fig. 3: Controller that realises the hinge recipe where parts, recipe states, and topology states are collated. Epsilon denotes that no parts are present within the resource and '-' represents idling.

Algorithm 2: BuildPlan

```
Data: Controller, Topology τ, TopologyState, Recipe, NextRecipeState,
      CompositeOperation, PlanTransitions, PlanParts
Result: whether or not the recipe is realisable and the associated Controller
found ← false;
operation, input, output ← next operation to process from
 CompositeOperation;
forall Transition ∈ TopologyState do
    if Transition is ¬synchronisation then
        if Transition is target operation then
            nopize ← AddTransition(τ, PlanTransitions, TopologyState,
             Transition, TransitionState) ;
            allocate ← AllocateParts(Transition, input, output) ;
            if allocate ∧ nopize then
                PlanParts ← AddParts(operation, transition);
                TopologyState ← Transition;
                found ← True;
                break;
            end
        end
    else
        Append Transition to SynchronizationTransitions;
    end
end
if ¬found then
    forall Transition ∈ SynchronizationTransitions do
        PlanParts ← SynchronizeParts(Transition, input);
        TopologyState ← SynchronizationTransition;
        nopize ← AddTransition(τ, PlanTransitions, TopologyState,
         SynchronizationTransition, TransitionState);
        if ¬nopize then
            continue;
        end
        found ← BuildPlan(Controller, τ, TransitionState,
         Recipe, NextRecipeState, CompositeOperation, PlanTransitions,
         PlanParts);
        if found then
            break;
        end
    end
    return found
end
if CompositeOperation has no further operations then
    Update Controller with current PlanTransitions;
    Clear PlanTransitions;
    forall Transition ∈ NextRecipeState do
        result ← BuildPlan(Controller, τ, TransitionState,
         Recipe, TransitionRecipeState, CompositeOperation, PlanTransitions,
         PlanParts);
        if ¬result then
            return False
        end
    end
    return True
else
    return BuildPlan(Controller, τ, TransitionState,
     Recipe, NextRecipeState, CompositeOperation, PlanTransitions,
     PlanParts);
```

7 Experimental evaluation

An experimental evaluation was conducted using the models of the resources of the production system in Figure 1 and the recipe in Figure 2. Dedicated benchmarks were written and run using the *Google Benchmark* C++ library. The machine used has 16 GB of RAM and an AMD Ryzen 7 5800H with 8 cores and 16 threads, having a base clock speed of 3.2GHz and maximum boost clock speed of 4.4GHz. The CPU times were averaged over a course of 10 iterations. The source code and models are available at https://github.com/nightly/pcs.

7.1 Topology generation

Figure 4 shows the time taken to generate a complete topology with an increasing number of resources. As can be seen, the time is exponential in the number of resources. However, our strategy to generate a topology incrementally can limit exponential growth. Figure 5 shows an outline of the incremental topology that is generated whilst synthesising a controller.

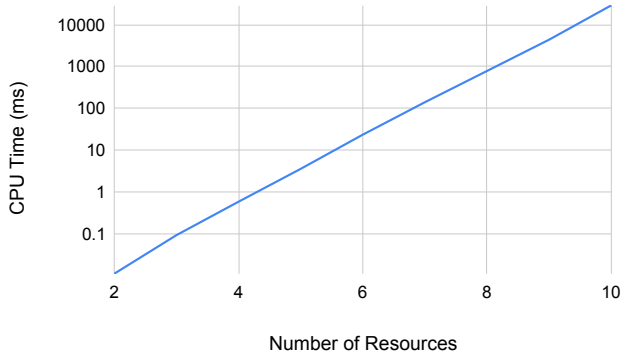
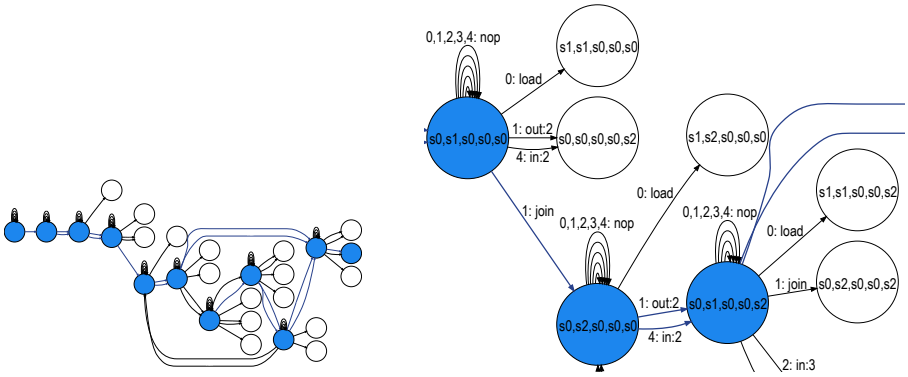


Fig. 4: CPU time (logarithmically scaled) to generate a complete topology.

7.2 Controller synthesis

The process of controller synthesis itself typically takes milliseconds, with the generation of the topology occupying the vast majority of time. Generating a controller whilst incrementally forming the topology is over 10 times faster using our example. The complete topology was formed of 102 states and 716 transitions, whereas the incremental topology was formed of 27 states and 87 transitions. The incremental topology has a 73.5% decrease in the number of states and a 87.8% decrease in the number of transitions.



(a) The full incremental topology (b) Portion of the incremental topology

Fig. 5: Incrementally generated topology. The states and transitions that compose the controller are highlighted in blue.

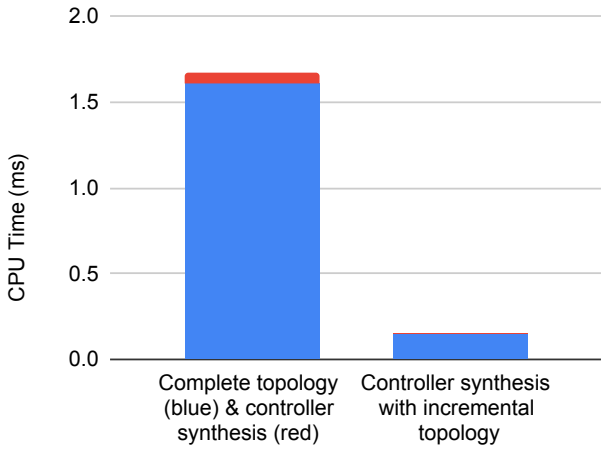


Fig. 6: CPU time for controller synthesis using the complete topology versus the incremental topology.

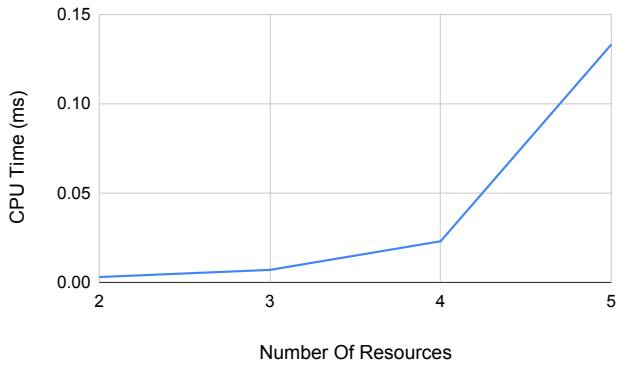


Fig. 7: CPU time to synthesise a controller with an incremental topology.

Figure 6 shows that synthesising a controller whilst generating the topology incrementally is substantially faster than full topology generation followed by controller synthesis. Figure 7 shows the time to generate a controller, using an incremental topology, for variants of the product recipe of Figure 2 requiring an increasing number of resources. As can be observed, the time retains the speed of milliseconds despite a considerably large state-space to search.

8 Conclusion and future work

In this paper, we have extended previous work by developing an efficient strategy for tackling the state space explosion problem via incremental topology generation. Considering that topology generation is the most computationally expensive step of the process, this translates to a large performance speedup and allows our synthesis approach to be viable and practical for larger-scale production facilities. Notably, incremental generation is also suitable to be used for dynamically reconfigurable production systems, as complete topology regeneration may become impractical with excessive wait times for simple changes.

Future work includes the possibility of synthesising a partial controller where missing parts and observable operations are signified during the realisation of the recipe, allowing these parts to be ordered or a given production resource to be reconfigured by switching out or ordering a new tool, for example. Another avenue of research is selecting the best possible controller (combinatorial optimisation), whereby the controller that minimises the total number of synchronisation operations is deemed optimal. A class of controllers that can be synthesised in polynomial time may be characterised by using languages with implicit complexity, such as [5]. Resource modelling can also be extended to introduce non-deterministic transition trajectories. Furthermore, observable operations involving multiple resources can be performed on the same part in parallel [9].

Acknowledgements

This work was supported by the SURE Research Projects Fund of the University of Bradford.

References

1. Mazin Al-Wswasi, Atanas Ivanov, and Harris Makatsoris. A survey on smart automated computer-aided process planning (ACAPP) techniques. *The International Journal of Advanced Manufacturing Technology*, 97(1):809–832, 2018.
2. Nikolas Antzoulatos, Elkin Castro, Lavindra de Silva, André Dionisio Rocha, Svetan Ratchev, and José Barata. A multi-agent framework for capability-based reconfiguration of industrial assembly systems. *International Journal of Production Research*, 55(10):2950–2960, 2017.
3. Nikolas Antzoulatos, Elkin Castro, Daniele Scrimieri, and Svetan Ratchev. A multi-agent architecture for plug and produce on an industrial assembly platform. *Production Engineering*, 8(6):773–781, 2014.
4. J.C. Chaplin, O.J. Bakker, L. de Silva, D. Sanderson, E. Kelly, B. Logan, and S.M. Ratchev. Evolvable assembly systems: A distributed architecture for intelligent manufacturing. *IFAC-PapersOnLine*, 48(3):2065–2070, 2015. 15th IFAC Symposium on Information Control Problems in Manufacturing.
5. Emanuele Covino, Giovanni Pani, and Daniele Scrimieri. Compile-time computation of polytime functions. *Journal of Universal Computer Science*, 13(4):468–478, 2007.
6. Lavindra de Silva, Paolo Felli, David Sanderson, Jack C. Chaplin, Brian Logan, and Svetan Ratchev. Synthesising process controllers from formal models of transformable assembly systems. *Robotics and Computer-Integrated Manufacturing*, 58:130–144, 2019.
7. William Derigent, Olivier Cardin, and Damien Trentesaux. Industry 4.0: contributions of holonic manufacturing control architectures and future challenges. *Journal of Intelligent Manufacturing*, 2020.
8. H. A. ElMaraghy. Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems*, 17:261–276, 2006.
9. Paolo Felli, Lavindra de Silva, Brian Logan, and Svetan Ratchev. Process plan controllers for non-deterministic manufacturing systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 1023–1030, 2017.
10. Eeva Järvenpää, Niko Siltala, Otto Hylli, and Minna Lanz. The development of an ontology for describing the capabilities of manufacturing resources. *Journal of Intelligent Manufacturing*, 30(2):959–978, 2019.
11. Daniele Scrimieri, Nikolas Antzoulatos, Elkin Castro, and Svetan M. Ratchev. Automated experience-based learning for plug and produce assembly systems. *International Journal of Production Research*, 55(13):3674–3685, 2017.
12. J. Zaytoon and B. Riera. Synthesis and implementation of logic controllers – a review. *Annual Reviews in Control*, 43:152–168, 2017.