The International Journal of Advanced Manufacturing Technology
https://doi.org/10.1007/s00170-026-19253-5

ORIGINAL ARTICLE

Dynamic machine-level planning for Plug & Produce manufacturing

Anders Nilsson[1] · **Fredrik Danielsson**[1] · **Bengt Lennartson**[1]

Received: 24 April 2026 / Accepted: 28 September 2026
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2026

Abstract
Manufacturing industries face persistent challenges due to a shortage of skilled labor and growing demand for customized products, while established automation methods are not cost-effective for high-mix, low-volume manufacturing. An intelligent automation concept capable of daily reconfiguration to accommodate new manufacturing conditions is required. Manufacturers request digital configuration over detailed programming of operational sequences and process data. Sub-operations, such as material transport and resource switching, are preferably automatically planned. This paper introduces an ontology to facilitate the digital configuration of agent-defined transfer operations for materials, parts, and resources. Automatic planning has previously been applied to higher-level path planning, such as the flow of materials and products through buffers and workstations. This study extends these algorithms by incorporating recursive planning for strategic routing and real-time reconfiguration of parts, materials, and resources at the machine level. The automatic planners are incorporated and modeled in a configurable multi-agent execution system. The planning problem is decomposed into simpler subproblems via a recursive, automated problem-generation process, while the number of edges is reduced by a proposed transport-edge mechanism. Validations are conducted at a Plug & Produce robot station equipped with tools, toolholders, parts, and associated materials. Planning Domain Definition Language (PDDL) planners are compared with Satisfiability Modulo Theory (SMT) solvers, where the latter performed best for the automatic planning of robotic tool-change operations and material transfer in a Plug & Produce automated manufacturing context.

Keywords Plug & Produce · Multi-agent systems · Manufacturing · Automated AI planning

1 Introduction

The industry seeks human-centric automation solutions that enable existing production staff to adapt the manufacturing system daily to new scenarios. In response to these demands, novel collaborative robots offer flexible solutions through hand-guided learning [1], graphical block-based configuration [2], and human–robot collaborative implementations [3]. These advantages enable a broader range of end users to deploy a robot effortlessly when needed. Similar concepts for generic automation are still absent. Traditional control systems, built on centralized architectures and statically programmed functions, struggle to meet the modern industrial demands. Largely due to limitations in current PLC technologies and the rigid structure of IEC 61131–3 programming [4]. Concepts such as Reconfigurable Manufacturing Systems (RMS), industrial-adapted versions of Digital Twins (DT), Internet of Things (IoT), Cyber-Physical Systems (CPS), and Artificial Intelligence (AI) have contributed to promising solutions that are slowly reaching industry [5]. For full implementation, reconfigurability needs to be more efficient and user-friendly. One way in that direction is to use graphical configuration tools and automated planners combined with multi-agent techniques. Automatic planning has been widely explored for the path planning of robot trajectory [6], routing of parts through manufacturing stations and line reconfiguration [7], resource scheduling [8], job shop scheduling [9], and automatic generation

✉ Anders Nilsson
anders.nilsson@hv.se

Fredrik Danielsson
fredrik.danielsson@hv.se

Bengt Lennartson
bengt.lennartson@hv.se

[1] Department of Industrial Automation, University West, Engineering Science, Gustava Melins Gata 2, 461 32 Trollhättan, Sweden

Published online: 09 October 2026

of process plans for assembly based on data from Computer-Aided Design (CAD) models [10].

A gap is identified concerning the dynamic planning of transfer tasks on the machine level, apart from the value-adding manufacturing processes, which leads to the following two research questions:

> RQ1: What ontology and algorithms support efficient and dynamic generation and execution of planning problems for material, part, and resource transfers in a Plug & Produce manufacturing system?
> RQ2: Which planning approach, Satisfiability Modulo Theories (SMT) or Planning Domain Definition Language (PDDL), is most efficient for automatic planning of machine-level transfer operations within a dynamic Plug & Produce manufacturing system?

This paper presents novel methods for strategic routing and dynamic reconfiguration of parts, materials, and resources at the machine level. A recursive approach to generating planning problems that break complex problems into smaller, faster-to-solve subproblems is proposed. A transport edge of three nodes is introduced to reduce the planner's search space compared to the ordinary edges of two adjacent nodes. Short solution times are critical because planning occurs at runtime. The proposed system is designed for Plug & Produce, enabling the seamless introduction of new parts and resources.

Two widely used automatic planning concepts will be compared and evaluated for their effectiveness: (1) PDDL is a problem description language for a wide range of automatic planners and is the first choice for planning these kinds of problems [11]. (2) SMT solvers address satisfiability problems and are widely used for formal verification, constraint satisfaction, and model checking [12]. Although SMT solvers were initially developed for automatic planning, their problem descriptions are more challenging to formulate than PDDL problem descriptions. The two planning approaches and various solving algorithms will be tested and evaluated in a robotized kitting application equipped with tools, tool holders, parts, and associated materials, in which the SMT solvers outperformed the PDDL-based planners. Kitting involves assembling required materials into ready-to-use kits to facilitate subsequent assembly.

The rest of the paper is organized as follows. Section 2 presents the background of agent-controlled Plug & Produce and the configurable Multi-Agent System C-MAS. Section 3 presents methodologies addressing RQ1. Section 4 provides an industrial example to validate the efficiency of various PDDL planners and SMT solvers addressing RQ2. Section 5 discusses the results, general benefits, and applications. Section 6 concludes the paper.

2 Background and related work

Reconfigurable Manufacturing Systems (RMSs) propose modular resources enabling rapid and cost-effective reconfiguration of the manufacturing system to fit the actual production [13]. RMS addresses integrality, convertibility, diagnosability, customization, and modularity [14]. They are designed to be modular and adaptable, but they still rely heavily on a fixed relationship between resources and products. Plug & Produce automatic manufacturing systems, introduced in the year 2000 by Arai et al. [15]. Pluggable modular resources operate autonomously, possess the ability to produce anything, and can be reconfigured on the fly. Plug & Produce reflects a broader shift toward decentralized, intelligent manufacturing to achieve dynamic manufacturing [16]. Plug & Produce systems are typically managed by an agent-based control architecture [17]. Agents are autonomous software entities that represent physical and logical objects, and a set of agents in a system forms a Multi-Agent System (MAS). Agents have goals to achieve and use their capabilities, often referred to as skills, to accomplish them. Agents interact and negotiate with one another to find solutions concurrently. A common design architecture for agent-based manufacturing concepts divides agents into part and resource agents [18]. A part agent represents a specific product part, has manufacturing goals and process plans, and negotiates with resource agents to determine how to achieve those goals. AI technologies, such as automatic planning, are suitable for integration into agents to increase their intelligence while maintaining the Plug & Produce flexibility [19]. AI techniques such as Large Language Models (LLMs) have lately been enhanced to support multi-agent manufacturing systems by enabling agents to communicate in natural language and interpret human instructions for decision-making [20]. Agents enhanced with LLMs can also replace predefined, fixed heuristic rules typically used in agent negotiations to reason dynamically about shopfloor conditions [21]. A hybrid framework combining agentic AI and multi-agent systems for smart manufacturing, where LLM-based agents handle high-level reasoning, planning, and coordination, while smaller, specialized agents use role-based and small language models to perform efficient domain-specific tasks [22]. This approach enables adaptive, transparent, and scalable prescriptive maintenance, demonstrating improved performance, robustness, and explainability in industrial applications. The AI planning algorithms in this paper are implemented in the Configurable Multi-Agent System (C-MAS) [23], a digitally configurable, multi-agent framework for Plug & Produce. C-MAS is designed to control workstations at the machine level, where quick responses are crucial. The ontology is generic and configurable for a variety of implementations, divided into part, material, and

resource agents. The part agents handle the manufacturing processes as a sequence of goals that the part must meet to be finalized. The sequence of goals is graphically represented in a sequence-of-goals chart. Goal parameters can be extracted directly from CAD models of the parts and stored together with the specific goal [24]. Material agents represent materials intended for assembly into the part and provide information about their location and properties. Resource agents are associated with resources, including devices and human operators, and present their skills to other agents through interfaces. Parallel goals can run concurrently in C-MAS when concurrent resources are available [23]. If not, the goals are treated sequentially, which helps balance manufacturing capacity by simply plugging resources in and out. Skills and parameters are advertised and reachable through interfaces. CMAS utilizes abstract interfaces to avoid strongly coupled connections between agents. Abstract interfaces are linked to genuine interfaces through agent negotiations, granting access to a skill only after the negotiation process is finalized. Abstract interfaces enable operators or process planners in-house within the manufacturing company to configure the sequence of operations and process parameters for part agents.

Ontologies are essential for shared understanding, interoperability, and modeling agents' behavior [25]. The ontology of C-MAS was initially influenced by the Instantly Deployable Evolvable Assembly Systems (IDEAS), an agent-based Plug & Produce execution system [26]. In IDEAS, agents are classified into transport, resource, coalition, and part agents. Resource agents offer fundamental skills for specific pluggable modules. The coalition agent negotiates and coordinates the execution of complex skills among various resource agents [27]. The part agents manage the negotiation of a sequence of operations to produce the desired part, such as C-MAS part agents. The transport agent establishes an interface that other agents can utilize to implement transport solutions. C-MAS does not employ coalition agents. Instead, all agent negotiations are conducted through agent interfaces that represent both logical and physical requirements and capabilities. The absence of coalition agents in C-MAS simplifies the architecture and reduces negotiation time, which is important in machine-close operations but may limit coordination in complex tasks. Specialized agent types are unnecessary in C-MAS because all skills are programmable and can be used for various specializations. IDEAS was continued by the Open dynamic Manufacturing Operating System for smart plug-and-produce automation components (OpenMOS) [28]. OpenMOS is configurable through a complex ontology and semantic model that describes most aspects of operating a Plug & Produce manufacturing execution system. This is possible for systems operating at a relatively high level, which orchestrate underlying PLCs and robot controllers that handle all detailed functionality [29]. Such detailed control implies that the underlying systems contain part-specific data, thereby restricting both product and resource flexibility. C-MAS communicates with external systems, such as robot controllers, through device primitives, which are generic and granular functions independent of a specific application. Furthermore, C-MAS supports programmable skills that enable direct control of any resource without external control systems. Another bottleneck for IDEA and OpenMOS is that, as the number of agents increases, negotiation processes can become prohibitively time-consuming. To address that issue, the negotiation process in C-MAS is short, consisting primarily of interface and variable matching. C-MAS is intended for implementation on a limited manufacturing workstation with a few active parts, materials, and resources. IDEAS and OpenMOS did not support automatic planning, which is now implemented in C-MAS.

Ontologies provide a shared, structured understanding of resources and processes, enabling seamless communication and interoperability [30]. They will also facilitate the automated generation of process plans and decision-making [31]. The number of proposals for ontologies supporting cognitive understanding and action has increased substantially for applications in autonomous service robots [32]. These methods are interesting but still too unreliable for industrial implementation, where predictable behavior is prioritized for safety and quality reasons. This paper presents an agent-structured ontology for C-MAS that supports automatic planning. For comparison, both PDDL-based planners and SMT solvers for planning problems were implemented, tested, and validated. The PDDL automatic planners inherit the Stanford Research Institute Problem Solver STRIPS, developed in 1971 by Fikes and Nilsson [33]. STRIPS was initially an automatic planner that later became a description language implemented in PDDL for first-order logical combinations of grounded predicates [34]. PDDL and related planners are continually developed, not least through the annual International Planning Competition (IPC) [35]. In contrast, the SMT solver's approach is to solve problems by reasoning over logical constraints to find satisfying solutions. Satisfiability (SAT) solvers specialize in Boolean problems using Davis–Putnam–Logemann–Loveland (DPLL) backtracking search algorithms [36]. Later versions incorporate branching heuristics, lookahead functionality, and conflict-driven clause learning to effectively reduce the search space [37]. Heuristic techniques combine classical AI planning with modern SAT-solving by encoding goal reachability as logical formulas that capture the constraints of a planning task. The solver searches for a sequence of states that satisfies these formulas, effectively generating a plan. Planning-specific heuristics have significantly

improved solver efficiency, sometimes even outperforming traditional state-based heuristic planners such as those in the HSP family [38]. This concept has been refined in several steps, most recently through a more effective search strategy that automatically splits action schemas during the grounding phase [39]. Satisfiability Modulo Theory (SMT) extends SAT by considering satisfiability over combinations of first-order theories of integers, rationals, arrays, and bit-vectors. SMT combines efficient theory-specific decision procedures with the DPLL framework, including first-order decision procedures, solver combinations, and partial quantifier handling, making SMT suitable for planning problems [40]. A loop of Bounded Model Checking (BMC) is used to generate a sequence of temporal goals from the initial to the goal state within a limited number of steps, the bound [41]. A foundational automata-theoretical framework with temporally extended goals facilitates constraint design. By translating temporal goals into automata and reducing the planning problem to check whether automata are non-empty, this approach facilitates the synthesis of sequential plans in a formally grounded and computationally efficient manner [42]. Valuable when the constraints in the C-MAS concept are dynamically auto-generated for Plug & Produce applications. Optimal solutions are prioritized, and the use of synchronized automata reduces the number of proposed actions [43]. This paper uses synchronized automata to model transfer operations, such as handing off entities from a buffer to a transporter.

3 The proposed method

Both PDDL planners and SMT solvers were implemented in C-MAS for comparison. Only one planner or solver can be used at a time, selectable in a configuration menu. The planner and solver problems are automatically generated from the agents' configuration and the actual Plug & Produce setup and situation. The agents can then execute the plan. This chapter will describe the theories, ontologies, and implementation.

To plan, for example, a tool-change operation, the automatic planner must be able to generate partially ordered plans that can be postponed for later use [44]. For handling PDDL descriptions, PDDL4J [45] was selected, as it includes both a PDDL parser and a variety of planning algorithms. Planners supported by PDDL4J that meet this criterion are the General Space Planner (GSP) [46], Heuristic Search Planner (HSP) [47], and Fast Forward (FF) [48]. Because SMT (Satisfiability Modulo Theories) solvers do not natively interpret PDDL, a generic Application Programming Interface (API) was implemented in C-MAS using the JavaSMT library [49]. SMTlib [50] is an international initiative that supports SMT for research and development by integrating several SMT solvers, including Z3 version 4.15.2, Princess version 2023, SMTInterpol version 2.5, and MathSAT5 version 5.6.10. All three PDDL planners and four SMT solvers were implemented in C-MAS version 2.11 for comparison and could be selected from the C-MAS configuration menu, one at a time.

3.1 Ontology for automatic problem generation and the execution of the generated plan

This section introduces an ontology that supports the automatic generation of transfer planning problems based on the C-MAS digital configuration and actual Plug & Produce setup. A configuration that allows manufacturing companies to configure their processes for parts without expressing all the transfer operations needed in detail. Each transfer operation is divided into three distinct operations: pick, transport, and place. The pick operation involves retrieving an item from a specific source interface location of an agent. The transport operation moves the item from the source interface location to a designated interface location via a transport interface. Finally, the place operation entails positioning the item at an agent's target interface location. The structure of the agents, goal, and interface classes is visible in (Fig. 1).

The agents, goals, and actions that match skills on interfaces are the backbone of C-MAS. The *AgentAction* class contains the process plans for actions to be executed using the skills provided by resource agents' interfaces. The material and part agents are both inherited from the common component class, where only the part agents have a sequence of goals. The *Goal* class inherits from the abstract class InterfaceBase, enabling each goal to directly connect to a skill exposed by an agent's interface. Further, values and structures of values are stored in instances of the abstract class *Variable* (Fig. 2).

Variables are primarily used to define and store process values instantiated from Value classes for Boolean, Integer, and Real. Variables can be grouped into arrays with the *VariableArray* class or structured as complex data types with the *VariableObject* class. *VariableArray*, *VariableObject*, and all *Value* classes inherit from the abstract *Variable* class, ensuring a consistent and extensible design. Process plans can be configured graphically by the manufacturing company using the C-MAS built-in sequence of goals chart [23], which is stored in an instance of the class *ProcessPlanGoals*. Process plans can also be written in the PLC programming language Structure Text (ST) by the class *StructurePlanST* through *AgentAction*. The common class *ProcessPlan* is controlled by the abstract *WorkFlow* and *Requirement* classes, which are introduced in this paper to regulate C-MAS behavior, primarily when automatic planners are used.

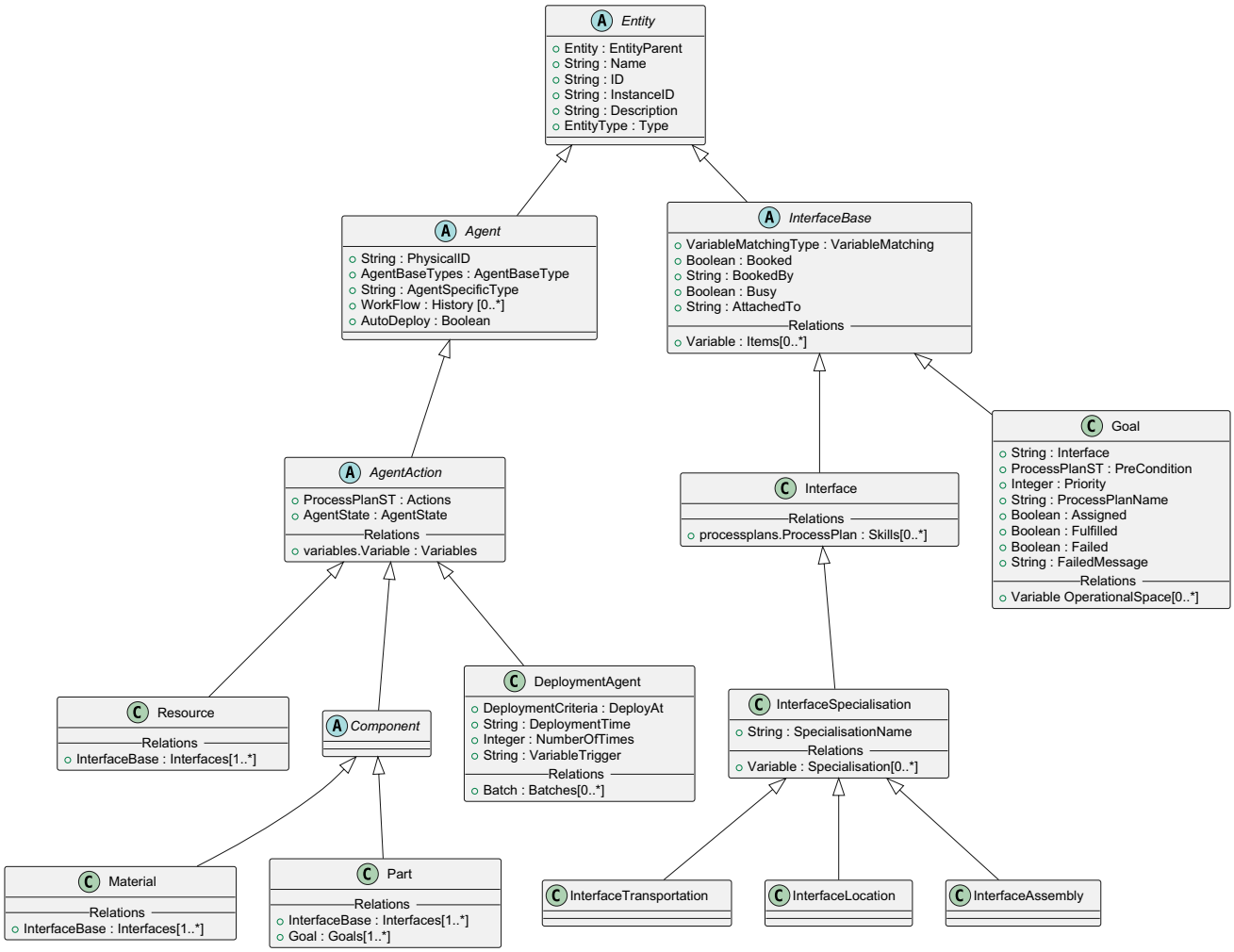

Fig. 1 C-Mas ontology part 1 with the Entity, Agents, Interface, and Goal classes

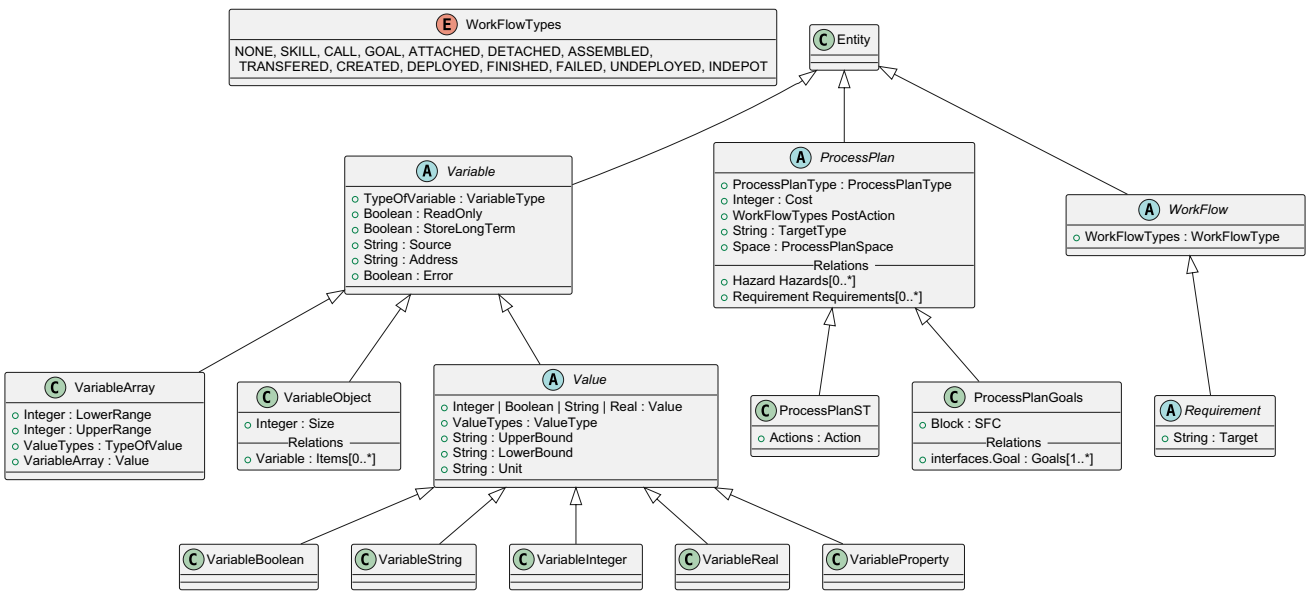

Fig. 2 C-MAS ontology part 2, including the Variable, Process plan, and Workflow classes

Requirements can trigger the automatic planner by defining conditions that must be satisfied before a skill can be executed. These requirements establish a relationship between the skill and the necessary preconditions for the *WorkFlowTypes* ATTACHED, DETACHED, and ASSEMBLED. When an agent is attached by workflow ATTACHED to one of another agent's interfaces, it remains a separate entity, ready to detach again, which is practical for pick-and-place operations. When an agent is assembled into another agent by the workflow ASSEMBLED, it ceases to exist as a separate entity. This is useful in scenarios like assembly and kitting, where materials become a merged part when transferred to the part. Part agents can have a sequence of either requirements, goals, or a combination of both. The distinction between goals and requirements lies in their complexity, where requirements are more straightforward. For instance, a requirement might specify that an agent must be assembled or attached to a specific part, thereby triggering the automatic planner to find a solution. In contrast, goals require predefined process plans or skills that specify how to achieve them, and these plans can indeed include requirements. Resource agents may also have requirements to fulfill before utilizing a skill on a specific interface.

A new class for this paper is *InterfaceSpecialization*, defined upon the Interface class to facilitate automatic planning (Fig. 1). *InterfaceSpecialization* is inherited by three interface specialization classes: *InterfaceSpecialization*, *InterfaceLocation*, and *InterfaceAssembly*. An instantiated *InterfaceSpecialization* object encapsulates the variables and process plans required to fulfill a specific task or satisfy a given requirement. *InterfaceLocation* variables are not included in the standard interface matching and negotiation process. Variables in objects of the *InterfaceLocation* type are intended to contain location details. During negotiations, agents use skills and standard variables to reach an agreement. However, location information specifies where the interface is in space. The name of the specialization specifies the intended use of the information. Hence, specialization variables can be used to store locations for detaching, attaching, assembly, and transportation. Objects of InterfaceTransportation are supposed to contain skills for detaching, transporting, and attaching agents. Agents that are transferred between the interfaces of other agents are defined as passenger agents. Part agents, material agents, and resource agents can all be transferred between agent interfaces that the automatic planner initializes. The *InterfaceAssembly* is a special class for assembling a material agent into a part agent, in which the material agent is merged into the part agent and ceases to exist.

3.2 Automatic generation of problem description

The planning algorithm must have a comprehensive understanding of all possible transfer locations and performers divided into transport edges within the domain, hereafter referred to as a map. This map must either be predefined during the design phase or generated dynamically at runtime to ensure the planner has access to all potential options. Given the dynamic nature of Plug & Produce, a generative approach at runtime is preferred and presented in this paper.

The dominant method for automated planning involves generating a route from node A to node B using a graph-based map of adjacent nodes structured in edges [51]. Each edge typically connects two adjacent nodes and a cost, often represented as a numeric constant or vector, that quantifies the effort required for the transfer. An edge in a map typically represents a transition that moves the system from one state (node) to another, referred to as a transfer operation. Edges that connect two nodes and have a cost are suitable for determining the shortest or cheapest path between two locations on a map. A machine-level transfer operation involves three nodes when the transporter (e.g., a robot or conveyor) is included. The transporter is determined at runtime in a Plug & Produce system and is therefore neither obvious nor predefined. A novel Transport Edge with three nodes and some additional information is introduced in this paper. Standard two-node edges typically require two edges for a transport: one from the source to the transporter and one from the transporter to the destination. Two standard edges can be replaced by a single three-node transport edge, thereby halving the total number of edges on the map. A transport edge also includes the agent to be transported, the passenger agent, and an edge ID. The edge ID will pass through the automatic planner and be included in the resulting plan. That will enable C-MAS to trace back to the agents who shall perform the transfer task.

The transport edge for a passenger agent that will be transferred is denoted as *transportEdge* and is defined by a five-tuple (1).

$$transportEdge =< passengerAgent, nodeSource, nodeTransport, nodeTarget, edgeID > \quad (1)$$

where the *passengerAgent* is the agent of any kind, resource, part, or material, targeted for transportation for the specific *transportEdge*. The *nodeSource* is the agent interface from which the *passengerAgent* can be detached and transported. The *nodeTransport* is the agent interface responsible for executing the transport of the passengerAgent between *nodeSource* and *nodeTarget*. The *nodeTarget* is the agent

interface where the *passengerAgent* will be attached after the transport. The *edgeID* is a unique identifier associated with the *transportEdge*.

A *node* represents an agent interface that is compatible and negotiated with the *passengerAgent* (2).

$$Node =< agent, interface, skill, location > \quad (2)$$

where the *agent* is a remote entity that has agreed to participate in the planning process. The *interface* is the remote interface for an agent compatible with the *passengerAgent*. The *skill* is an ability of the agent to execute the node (attach, detach, or transport). The *location* is a specialization variable representing the interface position.

All agents in C-MAS have access to and can create an instance of an automatic planner when needed. The method for problem generation, automated planning, and execution is carried out through the following steps:

1. Generate a map that includes all possible transfer operations within the domain.
2. Select and instantiate an automatic planner.
3. Generate a problem (PDDL or SMT) based on the map and the goal state.
4. Let the automatic planner solve the problem.
5. Execute the solution.

A map of transport edges is auto-generated based on the information given by the C-MAS agent configuration and performed by the function *generateMap* (Algorithm 1).

Algorithm 1 The function that generates a map of transport edges.

```
function generateMap(domain, passengerAgent)
  for each agent in findAllAgents(domain) do
    iAll := getAllSpecializationInterfaces(agent)
    iLocationOrTrans := getAllLocationOrTransfer(iAll)
    iList := getCompatebleAgentsInterfaces(passengerAgent, iLocationOrTrans)
    for each iS in iList do
        if iS has a skill s that can do DETACH then sourceList:= sourceList + [iS, s]
        if iS has a skill s that can do ATTACH or ASSEMBLY then
                                    targetList := targetList + [iS, s]
         if iS has a skill s that can do TRANSFER then transferList:= transferList + [iS, s]
     end for each iS
  end for each agent
  for each nodeTarget in targetList do
    for each nodeTransfer in transferList do
      for each nodeSource in sourceList do
       if agentNegotiation(nodeSource, nodeTransfer, nodeTarget ) then
            map := map + createEdge(passengerAgent, nodeSource,
          nodeTransfer, nodeTarget, generateID())
       end if
      end for each nodeTarget
    end for each nodeSource
  end for each nodeTransfer
  return map
end function generateMap
```

The automatic planning is based on the map generated by the *generateMap* function, which contains all agents in the domain with a specialization, attaching, detaching, or assembling. The object variable domain contains all agents that are related to the actual workstation. The *passengerAgent* is the agent that is requested to be transferred. A searching process will find all interface specializations *iS* of the type LOCATION, which in turn are sorted into different lists: the *fromList* that includes all location specialization interfaces to pick from, the *toList* that contains all location specialization interfaces to place, and the *transferList* including location-specialization interfaces for all agents that can transfer passenger agents. Once the lists are complete, an agent negotiation will add the agreement as transport edges to the *map*.

Once the map of all transport edges is completed, the next step is to generate a problem description. The function **planner** generates a PDDL or SMT problem description based on the map of transport edges, the goal location, and the passenger agent (Algorithm 2).

Algorithm 2 The function planner() generates a problem description for the automatic planner based on the map of transport edges.

```
function planner(domain, passengerAgent, goalLocation)
for each agent in gettAllAgents(domain) do
    map := map + generateMap(domain, agent)Function
end for each
solver := Solver( SMT | PDDL)
solver.generateProblem(map, goalLocation, passengerAgent)
plan := solver.findShortestPlan()
execute(plan)  / Execute all steps in the plan for attach , transfer, and detach/
end function
```

When the problem description is generated, the selected automatic planner is invoked to produce a plan with a minimum of operations to reach the goal location. The plan contains a sequence of skills to be executed to fulfill the initial requirement. Observe: During plan execution, the agents involved may need to initiate new planning sessions due to their relationships with other requirements. This recursive planning approach within C-MAS enables the formulation of intelligent strategies to fulfill multiple requirements, beginning with an agent's need for another agent, the passenger agent, to be attached or assembled. The planning problems are decomposed into simpler subproblems, reducing each complex task into flat, faster-to-solve problems. A flat problem is one where only a few actions are needed to reach the goal state (Fig. 3).

A requirement in the Part Agents Sequence of Goals Chart triggers the function planner, which then begins planning according to Algorithm 2. When the plan is complete, the part agent begins executing skills on the proposed and negotiated resource agent. The resource agent can, in turn, have requirements that trigger the function planner again, resulting in a recursive planning approach. Any resource agent can be configured to have a requirement that triggers the function planner. The attach skill is simplified and consists of a handover procedure in which the passenger agent is detached from one agent and attached to the agent who will perform the transport, and vice versa, in the Detach skill. There is a theoretical risk that one skill in an agent can trigger the same agent again in an infinite loop, but that cannot happen in a real implementation. This paper is built around a kitting task in which bolts are picked and placed in a box (Fig. 5). Using that kitting task as an example, the recursive planning works as follows: The Sequence of Goals Chart for the part agent Kit is configured with a requirement to assemble a bolt (Fig. 3). That requirement triggers the function planner, which begins planning the bolt's transport. It is an agent for a jaw-gripping tool configured with an interface that fits the bolt. The transport skill of the jaw gripper, in turn, requires attachment to a robot capable of transporting both the gripper and the bolt, which initiates the function planner again. If the robot has another tool attached, the planner will perform a tool change before transporting the bolt, as specified in the first plan.

3.3 Modeling the automatically generated problem formulation

The following section presents a formal framework for modeling a problem description applicable to the automatic planning of transfer operations. The theory of synchronized automata provides a solid foundation for problem generation, enabling precise representation of temporal behaviors

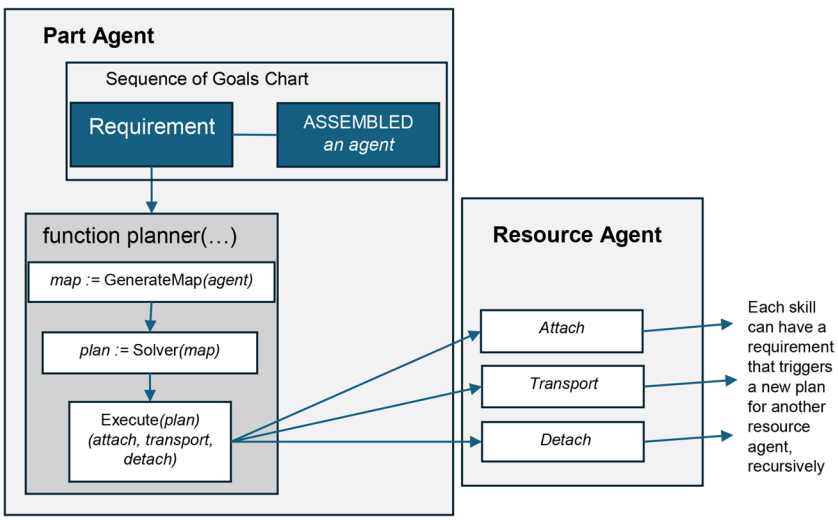

Fig. 3 Simplified description of the recursive planning approach started with a requirement in the Sequence of Goals Chart to have an agent assembled to the part agent

and interactions within transfer operations. In this work, transfer consists of pick, transport, and place operations. Pick-and-place operations involve handovers of the transferred agent between the agents' interface locations. Interface locations are generated as nodes in the map of transport edges. Agents operate independently of each other and must be synchronized during handover operations, like the way two different systems are coordinated by shared and local transition labels, according to Hoare's full synchronous composition [52]. Having synchronized nodes will enable the automatic planner to propose fewer actions from the initial state to the goal state than would otherwise be the case. Three finite synchronized automata are needed for the tool-change problem of two tools and serve as a formal model for the upcoming SMT and PDDL problem descriptions (Fig. 4).

The tool change problem consists of a tool stand with two tool holders, two different gripper tools, and an industrial robot. The two tool holders and the robot are treated as location-specialization interfaces here, as nodes n_1, n_2, and n_3. The two gripper tools are passenger agents p_1 and p_2. These tools can be positioned either at the tool holders or on the robot itself, but only one tool is assigned to a single node at any given moment. Observe that this is only an example; the automata model and the generated problem description coincide with the actual C-MAS configuration generated by the *generateProblem* function (Algorithm 2).

Generally, the passenger agent p_1 and p_2 are the agents that are transferred and can be of any kind, material, part, or resource. One of the nodes, in this case, n_3, must have one transport skill configured on an InterfaceSpecialization for a complete transport operation. The two nodes n_1 and n_2 consist of agents that have a *variableLocation* configured on an InterfaceSpecialization. The states for each node are unoccupied ∅ or occupied by an agent, in this case p_1 and p_2. The predicate variables e_1 to e_4 synchronizes the three automata. The number of nodes and agents to be transported is scalable with the size of the map of transport edges. The map represents the current digital configuration, agent agreement, the connected resources, and the number of active material and part agents. These synchronized finite automata focus exclusively on handover operations, but skills are activated for detaching, transporting, and attaching the transferred agent. Figure 5 shows the implementation of this scenario, with the nodes and passenger agents marked.

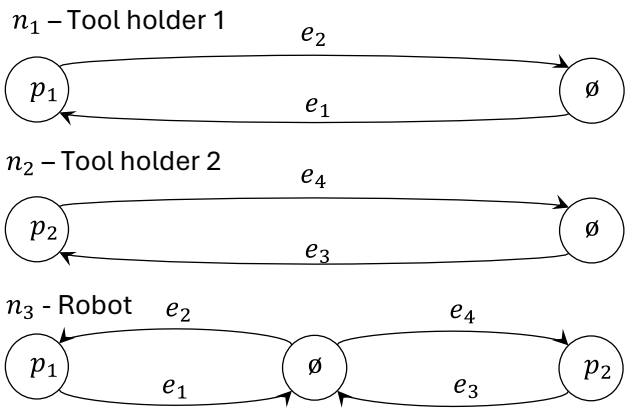

Fig. 4 An example of a finite synchronized automata model of the three nodes $\boldsymbol{n}_1$, $\boldsymbol{n}_2$, and $\boldsymbol{n}_3$ and two passenger agents $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, where ø is equal to an empty node

3.4 Automatic generation of problem description

The result of the automatic planner consists of actions that lead from the initial state to the goal state. In the case of C-MAS, initiated by the agent that has the requirement that another agent, the passenger agent, should be attached or assembled. Agents involved in the plan can recursively initiate other plans. The solver must be able to move other agents aside to transfer the passenger agent, which is necessary for a robot tool change operation. The tool the robot holds must first be placed in a tool holder before the robot can pick up the requested tool. A solver capable of handling partial plans and solving tasks incrementally is needed. Most SMT solvers support incremental solving by the Push and Pop methods. However, this work uses incremental model-checking against the goal state. The result will be a sequence of states reaching the goal state. Some SMT solvers retain valuable computational information, such as learned clauses and scores between calls,

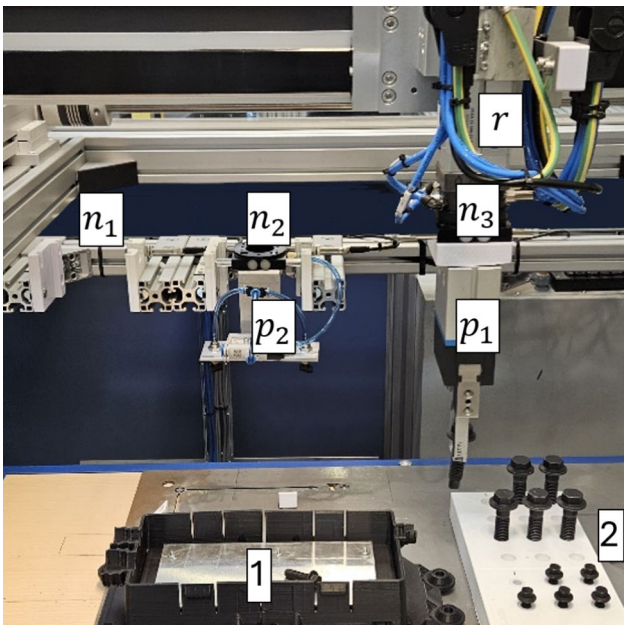

Fig. 5 The gantry robot-based kitting application that produces kits of a box and bolts

thereby eliminating the need to restart the search from scratch each time and reducing solving time [53]. An example of a generated problem to solve for three nodes, n_1, n_2, and n_3, and two passenger agents, p_1 and p_2, that are targeted to be transferred. Visible as pseudo-code in Algorithm 3.

Algorithm 3 A generated SMT problem for the three synchronized nodes, n_1, n_2, and n_3, and the passenger agent a_1 and a_2. The constraints are synchronized by the edge[i] value from 1 to 4.

```
s := solver()
s.add(And(n3[0] = p2, n1[0] = p1, n2[0] = Ø))  /Initial state/
for i := 1 to MAX_ATEMPTS do /Incremental check against the goal/
  if sat = s.check(and(n3[i] = p1)) then return  /Model checking/
  s.add( and( /Constraints of possible transfers generated from the map of transport edges/
    or(and( n1[i] = p1, n3[i] = Ø, n1[i + 1] = Ø, edge[i] = 2),
        and( n1[i] = Ø, n3[i] = p1, n1[i + 1] = p1, edge[i] = 1),
        and( n1[i] = n1[i + 1], not(edge[i] = 1), not(edge[i] = 2))),
    or(and( n2[i] = p2, n3[i] = Ø, n2[i + 1] = Ø, edge[i] = 4),
        and( n2[i] = Ø, n3[i] = p2, n2[i + 1] = p2, edge[i] = 3),
        and( n2[i] = n2[i + 1], not(edge[i] = 3), not(edge[i] = 4))),
    or(and( n3[i] = p1, n1[i] = Ø, n3[i + 1] = Ø, edge[i] = 1),
        and( n3[i] = Ø, n1[i] = p1, n3[i + 1] = p1, edge[i] = 2),
        and( n3[i] = p2, n2[i] = Ø, n3[i + 1] = Ø, edge[i] = 3),
        and( n3[i] = Ø, n2[i] = p2, n3[i + 1] = p2, edge[i] = 4),
        and( n3[i] = n3[i + 1], not edge[i + 1], not(edge[i] = 1),
          not(edge[i] = 2), not(edge[i] = 3), not(edge[i] = 4)))))
end for
```

Initially, p_1 is attached to n_1 and p_2 to n_3. The goal is to have p_1 attached to n_3 instead of p_2. The result of the SMT solver will be:

p_2 transfers from n_3 to n_2 by using $edge = 3$
p_1 transfers from n_1 to n_3 by using $edge = 2$.

The edge assignments have two functions: (1) as an identifier to be able to trace back to the map to find the involved agents after the solution is found, and (2) as a predicate or event e variable to synchronize the operations according to the finite synchronized automata (Fig. 4).

For PDDL, the approach differs because it is a description language for planning problems. A PDDL description is divided into two parts, the Domain and Problem descriptions. The domain description outlines the overall aspects of a problem and is implemented as fixed in C-MAS for all automatically planned transfer tasks when a PDDL planner is selected (Algorithm 4).

Algorithm 4 The fixed C-MAS PDDL domain description that utilizes variables applied uniformly across all automatically generated transfer tasks.

```
(define (domain cmas)
 (:requirements :strips :typing :negative-preconditions)
 (:types node transporter passenger instanceId)
 (:predicates (adjacent ?from - node ?t - transporter
        ?to - node ?p - passenger ?id - instanceId)
        (at ?n - node ?p - passenger)
        (allocate ?n -node))
 (:action move :parameters(
     ?from - node ?t - transporter
     ?to - node ?p - passenger ?id -   instanceId)
      :precondition ( and (adjacent ?from ?t ?to ?p ?id)
                     (at ?from ?p) (not (allocate ?to)))
      :effect ( and (at ?to ?p) (not (at ?from ?p))
            (allocate ?to)(not (allocate ?from)))))
```

The predicate adjacent takes four variables that are all related to a transport edge in the generated map. The variables *?from* is the source node in the map where the passenger agent is picked, ?*t* is the transporter agent, *?p* is the passenger agent, ?to is the target node where the passenger agent will be placed (the agent that requested the passenger agent), and *?id* is a unique identifier associated with the actual enhanced edge. The PDDL problem description is auto-generated by the *generateMap* function. The problem description is generated from the map of transport edges and the requirements from the passenger agent, such as for SMT solvers. The problem description has four sections: *domain*, which connects to the domain description; objects, which declare objects; *init*, which includes the map of transport edges and the initial state; and *goal,* describing the goal state (Algorithm 5).

Algorithm 5 An example of a generated PDDL problem description of the three synchronized nodes, n_1, n_2, and n_3 and the passenger agent p_1 andp_2.

```
(define (problem PDDLProblemDescription)
 (:domain cmas)
 (:objects
    n1 - node
    n2 - node
    n3 - node
    t1 - transporter
    p1 - passenger
    p2 - passenger
    edge_1 - instanceId
    edge_2 - instanceId
    …
 (:init
    ;Map of transport edges
    (adjacent n1 t1 n3 p1 edge_1)
    (adjacent n3 t1 n1 p1 edge_2)
    …
    ;Initial state
    (at n1 p1)
    (allocate n1)
    (at n3 p2)
    (allocate n3)
  (:goal (and (at n3 p1))))
```

This PDDL problem is the same tool-change problem explained using synchronized automata in Fig. 4. Initially, p_1 is attached to n_1, and p_2 is attached to n_3. The nodes, n_1 and n_3 are also marked as allocated, indicating to the planner that no more agents can be attached to these nodes. The goal is to have p_1 attached to n_3 instead of p_2. The transport agent t_1 represents the agent that can do the transport and has a transport specialization skill and, in this case, also a localization specialization, node n_3.

4 Verification and results

A demonstrator based on a gantry robot was built to verify and validate the automatic generation, solving, and execution of planning problems (Fig. 5). The test case is a kitting application in which a box is transferred from a buffer area to a kitting area using a vacuum gripper. Thereafter, large and small bolts are placed in the box with a jaw gripper, and thereupon, an automatic planned tool change is triggered.

The nodes n_1 and n_2, in Fig. 5**,** depict the tool holders for the passenger agents p_1 and p_2, corresponding to the jaw gripper tool for bolts p_1 and the vacuum gripper tool for the box p_2. Node n_3 denotes the tool fastening surface of the automatic tool changer on the gantry robot r, which can take both p_1 and p_2. The node n_3 has the skills to transfer tools by utilizing the gantry robot. The nodes n_1, n_2, n_3, and the passenger agents p_1 and p_2 are related to the earlier-described problem example that started with the finite synchronized automata model in Fig. 4, followed by the problem descriptions for SMT in Algorithm 3 and the PDDL descriptions in Algorithm 5. Further, in Fig. 5, the number 1 is a box for the kit located in the kitting area, and number 2 is a buffer area for small and large bolts and the box. One small bolt has already been picked, and a large bolt is currently held in the jaw gripper tool attached to the gantry robot. The box and bolts are also passenger agents in the full kitting scenario.

Agents must be configured for the specific application. The following resource agents were created: one robot agent, two tool agents, one tool holder agent with an interface for each tool, and one buffer agent with several interfaces that match the different material agents. The material agents are created for two bolt sizes and the box, one agent per material type. The kit, one box, and a configurable number of bolts are modeled as a part agent with requirements for automatic planning of box and bolt transfers. The kit is modeled graphically by a sequence of goals chart [23]. The first requirement is to place the box with the compatible vacuum gripper. The following requirement is to place a bolt in the box with the compatible jaw gripper. To place bolts in the box, the automatic planner must plan for: (1) placing the vacuum gripper in the tool stand to the right, (2) picking up the jaw gripper in the left tool stand, and (3) picking a bolt from the buffer and placing it in the box. The tool change planning problem was tested across different search space sizes by increasing the number of bolts from 2 to 72, with each bolt corresponding to a node in the search space. In addition to the bolt nodes, the search space includes two tool-holder nodes, one kitting location, and one robot node, yielding 6 to 76 nodes. The automatically generated map of transport edges becomes large when the number of *variableLocation*—nodes and InterfaceTransportation—skills of the *InterfaceSpecialization* class increases, see the ontology in Fig. 1. The map includes transport edges representing all possible combinations of compatible nodes for each possible transport of passenger agents, multiplied by the number of resource agents capable of performing the transport. The number of transport edges is determined using the following Eq. (3)

$$\sum_{p \epsilon P} n_p * (n_p - 1) * R_p \tag{3}$$

where p is an agent in the set of all passenger agents P that can be transported, n_p is the number of nodes that are compatible with p and R_p is the number of all resource agents that can transfer passenger agents. The number of automatically generated transport edges depends on how agents and their interfaces are modeled. The factor $n_p * (n_p - 1)$ is valid if it is possible to both pick and place a passenger agent p from and to each node n, which may not always be the case. But in this test case, the configuration admits pick-and-place in both directions to stress the system and increase the flexibility of the Plug & Produce. Mean values of automatic planning times for tool changes across various solvers and planners for different search spaces were plotted in a graph (Fig. 6).

Since times can vary across hardware, interpret the values as relative comparisons rather than absolute benchmarks. All tests were run on a Dell Precision 5560 with an Intel® Core™ Ultra 9 285H 2.90 GHz processor and 64 GB of internal memory, running Windows 11 and Java version 23. The Java Virtual Machine (JVM) heap size was set to 80% of available physical memory and remained unchanged across all experiments. Default configurations were used for the tested planners and solvers.

All planners and solvers evaluated in this study consistently found plans with the minimum possible number of actions for the specified tool-change problem. The test case workstation was designed to always have a solution available. No explicit runtime timeouts were observed, and the success rate was 100% except for the PDDL planners that could not handle search spaces with more than 11 nodes. The PDDL planners ran out of memory, which explains why the graphs end there. All SMT planners could solve 76 nodes, but solving time increased significantly. Only MatSAT5 could consistently solve 76 within 10 s. Solving times above 10 s are excluded from the graph because they are irrelevant in industrial applications. Due to the Plug & Produce concept, the solvers operate at runtime. Few manufacturers will see a robot standing still for calculations before taking the next action. To provide statistical evidence, the mean, median, minimum, maximum, interquartile range, standard deviation, and 95% confidence interval were computed from 30 runs of each search space, PDDL planner, and SMT solver (Table 1).

All samples were run in the full test cell (Fig. 5), starting by picking the box before the tool change, as shown in the graph (Fig. 6) and table (Table 1) reported for bolt picking. The time to complete a pick-and-place task was about one-third of the solving times for the tool-change problem across all solvers and search spaces, except for Z3, which solved the bolt-picking task as fast as MatSAT5.

The PDDL planners FF, GSP, and HSP were quite stable up to 8 nodes, but their instability gradually increased until all three PDDL planners ran out of memory, which happened for more than 11 nodes. MathSAT5 was the most stable and fastest SMT solver, outperforming the other solvers, especially in larger search spaces. For 72 nodes, the mean solving time was 6.5 s, with a maximum of 6.8 s and a standard deviation of ± 65 ms in 95% of runs. The SMT solver, Princess, needs slightly more than 10 s to solve 72 nodes.

Automatic planning times exceeding 10 s are irrelevant for industrial manufacturing applications at runtime. In addition, automatic planning times are not the only time needed to plan a transfer task in this proposed C-MAS concept. The total end-to-end planning time T_{E2E} can be divided into the following computational stages (4).

$$T_{E2E} = T_{negotiate} + T_{map} + T_{encode} + T_{solve} + T_{decode} \tag{4}$$

where $T_{negotiate}$ is the time for all agents to either agree to or refuse the requested skill. T_{map} is the time required to

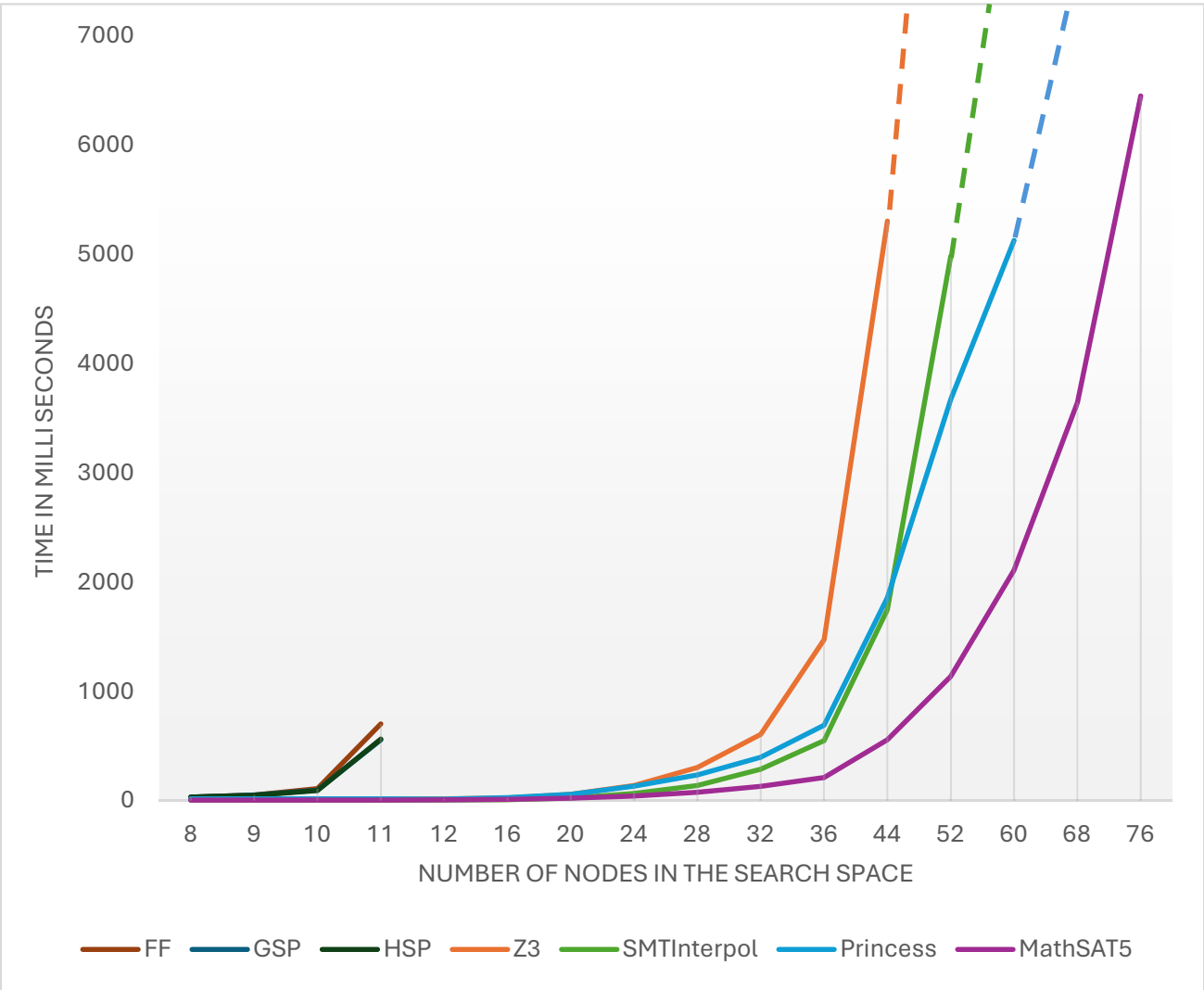

Fig. 6 Times for the automatic planning of the tool-change problem in search spaces of 6 to 76 nodes (30 to 5700 transport edges), based on mean values of 30 runs. The dashed lines indicate that the next value is outside the graph

generate a map of all possible transport edges and the problem description in PDDL or SMT format. T_{solve} is the time it takes for the automatic planners and solvers to plan the task, which is the focus of this paper. T_{decode} is the time it takes to decode the result and initiate the involved agents. $T_{negotiate}$ and T_{solve} are the most significant times. $T_{negotiate}$ are 3 ms for search spaces of 8 nodes, but 12.8 s for 72 nodes. This time can be reduced, but it is not the focus of this paper. In this benchmark, all agents negotiated with one another to stress the system. Making restrictions here will reduce both $T_{negotiate}$ and T_{solve}. The time T_{map} ranged from 0 up to 4 ms for the largest search spaces. The decoding part is more complicated for SMT than for PDDL, which results in T_{decode} being less than 1 ms in the PDDL case. When one of the SMT solvers were used, T_{decode} ranging from 1 ms for 8 nodes up to 352 ms for 72 nodes.

5 Discussion

C-MAS is fundamentally based on manually configured process plans, as industry feedback indicates a desire to maintain control over the parts processes. The implemented automatic planning generates problem descriptions for tasks such as material supply, tool changes, and product transfers.

The proposed TransportEdge is specifically designed for the class of machine-level transfer operations considered in this work, where each transfer consists of three functional roles: a source interface, a transport interface, and a target interface. These are typically pick-and-place tasks with three nodes: one to pick from, one transporter node, and one to place into, which cover all transfer operations. Under these assumptions, let M denote the number of feasible transfer operations. A conventional pairwise graph

Table 1 Statistical evidence in milliseconds across the tested PDDL planners and SMT solvers for a sample of node counts to plan the tool-change problem

Number of nodes	Value	FF	GSP	HSP	Z3	SMT Interpol	Princess	MathSAT5
	Mean	32	32	34	16	4.0	17	4.8
	Median	31	30.5	31	15	3.0	12	5.0
8	Min	25	25	24	14	2.0	8	4.0
	Max	51	50	65	18	17	66	7.0
	Interquartile	8.5	5.0	9.2	1.0	1.0	6.8	1.0
	Standard Deviation	6.3	6.6	10	0.9	2.9	12	0.7
	95% Confidence Interval	2.3	2.3	3.7	0.3	1.0	4.3	0.2
	Mean	702	556	563	16	5.0	14	7.1
	Median	610	374	373	16	4.0	13	7.0
11	Min	270	262	262	15	3.0	4.0	6.0
	Max	1520	1157	1192	18	14	34	8.0
	Interquartile	717	488	270	1.0	1	3.5	0.0
	Standard Deviation	393	325	341	0.8	2.2	7.2	0.6
	95% Confidence Interval	141	116	122	0.3	0.8	2.6	0.2
	Mean				58	25	58	23
	Median				57	25	57	23
20	Min	Out of memory			45	23	51	21
	Max				88	31	93	26
	Interquartile				16	2.0	5.5	2.0
	Standard Deviation				11	2.0	8.1	1.1
	95% Confidence Interval				4.0	0.7	2.9	0.4
	Mean				606	288	397	130
	Median				600	288	380	131
32	Min	Out of memory			341	249	334	123
	Max				1061	343	498	140
	Interquartile				317	30	72	6.0
	Standard Deviation				181	25	49	4.8
	95% Confidence Interval				65	8.8	18	1.7
	Mean				5305	1748	1857	557
	Median				5874	1708	1847	550
44	Min	Out of memory			2435	1211	1309	435
	Max				9154	2392	2471	613
	Interquartile				3180	118	389	39
	Standard Deviation				1896	233	280	33
	95% Confidence Interval				960	83	100	12
	Mean				16579	4982	3673	1138
	Median				15559	4854	3780	1129
52	Min	Out of memory			9142	3197	2484	1089
	Max				24676	6859	4571	1212
	Interquartile				6484	1155	898	50
	Standard Deviation				4658	789	588	35
	95% Confidence Interval				1667	282	210	13
	Mean				39437	10074	5127	2110
	Median				38186	10079	5124	2109
60	Min	Out of memory			25026	7205	3806	2054
	Max				65253	13199	6996	2165
	Interquartile				11583	2702	1676	44
	Standard Deviation				10930	2025	913	33
	95% Confidence Interval				5531	1025	327	12
	Mean							6449
	Median							6460
76	Min	Out of memory			Irrelevant long solving time			5948
	Max							6775

Table 1 (continued)

Number of nodes	Value	FF	GSP	HSP	Z3	SMT Interpol	Princess	MathSAT5
	Interquartile							222
	Standard Deviation							181
	95% Confidence Interval							65

representation requires $2M$ transfer edges, whereas the proposed TransportEdge representation requires M transport edges. Thus, the number of transfer-representation edges is reduced by 50%, while preserving the set of feasible transfer operations.

General planning domains may require different representations and planning abstractions. However, this work is limited to the workstation level, where entities such as materials, resources such as tools, fixtures, and products must be transferred between entities. Within this context, transfer planning rather than production scheduling is the primary concern, and the TransportEdge is designed specifically for this purpose.

Furthermore, the primary motivation for introducing TransportEdge is not only edge reduction. In a multi-agent Plug & Produce system, it is necessary to explicitly represent which agent is responsible for detaching, transporting, and attaching the transferred entity. The TransportEdge therefore captures both the transfer relationship and the responsible transport agent. This information supports agent negotiation, automatic planning, execution traceability, dynamic resource allocation, and runtime reconfiguration. For purely reactive systems based solely on local sensing, an explicit transport representation may not be required, since resources may simply react when an entity arrives. However, this approach does not provide the information needed to plan and reason about how the entity reaches its destination. In contrast, the proposed approach enables the automatic planners to reason about transport resources, handover operations, and alternative execution paths before execution.

Regarding state explosion, the proposed representation does not increase the number of reachable transfer states compared to a conventional pairwise representation. The planner still has access to the same feasible transfers. The TransportEdge changes only how transfer operations are represented and traced back to the responsible agents. For the evaluated transfer-planning domain, this reduces the number of planner-visible graph elements while preserving the available solution space. Each agent capable of performing transport requires its own set of transport edges, and the same applies to parallel operations. In directional operations, each transfer operation has one transport edge. If multiple transporters are selectable for one transfer operation, there will be one transport edge per transporter. The same applies if two or more transporters serve two or more passenger agents. Intermediate handovers do not change anything in our implementation. Still, one place operation to the next transporter and one pick operation by that transporter. In all these cases, the number of transport edges is reduced by 50% compared to pairwise edges. Parallel tasks are possible, but in the current implementation, parallel transporters cannot handle the same passenger agent.

However, pick-and-place and tool-change problems apply to many robot applications beyond kitting, such as assembly, sorting, inspection, and quality control. This is because the proposed planning approach operates on a generic transfer abstraction rather than on application-specific process knowledge. The planner reasons only about passenger agents, compatible interface locations, and transport resources. The transfer operations represented as pick, transport, and place actions. Consequently, from the automatic planner's perspective, there is no fundamental difference between transferring a bolt, a tool, a camera, a sensor, or another resource, provided the ontology describes the corresponding interfaces and transfer skills. The aerospace industry, for example, uses advanced quality controls with various changeable sensors and test methods across a high mix of low-volume parts. Another example is the manufacturer of pre-manufactured wooden houses, which also has a high mix of parts on a low volume of complete houses. For example, robotized wall construction requires many tool changes and material transfers of screws, nails, and fasteners.

All planners and solvers tested in this paper consistently found the minimum number of actions needed to reach the goal state for the evaluated tool-change and pick-and-place problem. The fewest actions to reach the goal state are assumed to be the fastest, since it runs on a limited-sized robotized workstation. This is because the robot mostly accelerates and decelerates, rarely reaching full speed. SMT solvers using bounded model checking (BMC) will, by their structure, guarantee that they find the plan with the fewest possible actions. BMC implies that the solver will first consider all possibilities to find a solution that requires only one action to reach the goal. If not, the solver looks for a solution that takes two actions, and so on until it reaches the goal state. The tested PDDL planners cannot guarantee they always find the plan with the fewest actions because different planning strategies prioritize finding a solution

quickly. But that is for deep and complex problems, which is beyond the scope of this paper. In this benchmark, even the tested PDDL planners regularly find the plan with the fewest actions.

State-space planners that can handle partial plans, such as the tested PDDL planners FF, HSP, and GSP, are the first choice for the planning of transfer tasks, especially the tool change problem. Examples such as the Tower of Hanoi and the Box World problem are readily available as PDDL implementations. Although SMT solvers were initially designed for planning, they are more often used for model checking, software verification, and mathematical problems rather than planning. Both approaches addressed the problem excellently, with no unnecessary tool changes or bolt pickings, and the robot behaved as expected. The SMT solvers were more efficient on this flat problem, requiring only a few actions to reach the goal. Deeper problems should benefit PDDL planners that use heuristic search to explore deep problem spaces, but that is beyond the scope of this paper.

The recursive planning approach proposed in this paper avoids deep problems. The purpose of the recursive planning strategy is not to improve a given planner's performance on an identical planning problem, but to decompose a potentially complex planning request into a sequence of smaller planning problems. Consequently, a direct comparison between recursive and non-recursive planning is not straightforward because the two approaches operate on different problem formulations. Recursive planning also aligns with multi-agent approaches that divide large problems into smaller fractions across several agents. In a non-recursive formulation, requirements related to tool availability, transport resources, and material transfers would need to be represented in a single planning problem, increasing both the search space and the number of interacting constraints. In the proposed approach, these dependencies are instead resolved by recursively generating separate planning instances when requirements are encountered during plan execution. The resulting planning problems therefore remain comparatively shallow and can be solved independently.

SMT solvers can handle larger search spaces than the tested PDDL-based planners. The maximum number of nodes was set to 74 because additional nodes were not relevant in a small robot cell, and the time to generate and solve the plans exceeded acceptable limits in the manufacturing industry. In addition to automatic planning time, it takes time to create the planning problem in a way the planner and solvers can understand, and time to interpret the result, which has been shown to be significant in the larger search spaces.

Other automatic planners were also considered. For example, Constraint Programming-Satisfaction (CP-SAT) solvers are well known for their efficiency. They require trimming and adjustment of the problem formulation to be efficient and stable, which is unsuitable for automatic problem generation implemented in an industrial environment. LLMs were also considered, but they are too slow and do not consistently yield the same optimal result.

Reducing the map of TransportEdge to improve solving efficiency can be achieved by eliminating rarely used skills and optimizing the negotiation logic. Nevertheless, the risk of erasing required solutions is considerable. Although not all nodes are relevant to the solution, they should still be included in the map to maintain high Plug & Produce flexibility. All possibilities must be visible to the automatic planner if, for example, this test case of automatic tool change is to be feasible. A tool change would be impossible to plan automatically if not all bolts, tools, and tool holders were visible on the map when the bolt should be picked, and an incompatible picking tool is attached to the robot. Additionally, in a Plug & Produce system, it is essential to have all alternatives available to ensure the most flexible solutions at any given time.

Beyond the ontology, maintaining semantic consistency across agents is challenging in a multi-agent system. For example, a drilling operation must mean the same thing to the part agent who needs it and the resource agent who performs it. In C-MAS, a skill or interface must be described in a standardized, unambiguous way to ensure that all agents can understand and use it consistently. This typically requires a high degree of responsiveness to the application being modeled. Accordingly, the skills used by the automatic planner must operate as expected and produce predictable outcomes to ensure success.

This study demonstrates the effectiveness of distributed intelligence and recursive planning in addressing the challenges of high-mix, low-volume Plug & Produce production. The novel ontology, which integrates configurable agents with automatic planners, enables operators and process planners in the manufacturing industry to configure their processes and automatically plan all transfer tasks. The automatic planner proposes skills, which capable agents negotiate before execution. If the negotiation procedure fails, the automatic planner in C-MAS will attempt to identify an alternative solution to ensure reliable production. If not, C-MAS will not give up. C-MAS will just wait until a solution is available because conditions are dynamic in a Plug & Produce system.

6 Conclusion

Two research questions were initially stated. RQ1 was about which ontologies and algorithms most efficiently support the dynamic generation and execution of planning problems for transfer tasks in a Plug & Produce manufacturing

system. RQ2 was about which planning approach, SMT or PDDL, is most efficient for automatic planning of machine-level transfer operations within a dynamic Plug & Produce manufacturing system.

Answer to RQ1:

This study shows that the proposed extension of the C-MAS ontology, combined with automatic planning algorithms, enables efficient, dynamic planning of machine-level transfer operations in Plug & Produce manufacturing. The introduced ontology extends C-MAS with requirements, workflow concepts, interface specializations, and transport-related abstractions that enable automatic generation of planning problems directly from the digital configuration and current shop-floor state. A key contribution is the three-node TransportEdge, which explicitly models source, transport, and target roles while preserving information about the responsible transport resource. This representation reduces the number of planner-visible transfer edges compared with conventional pairwise graph representations and improves execution traceability. In addition, the proposed recursive planning mechanism decomposes complex transfer requests into smaller and computationally manageable subproblems. Combined with synchronized automata for modeling handover operations, the approach enables automatic generation, planning, and execution of transfers of materials, parts, and resources without requiring users to manually specify detailed transport sequences.

Answer to RQ2:

The experimental results show that SMT-based planning is the most efficient approach for the investigated machine-level transfer-planning problems. All evaluated planners and solvers generated correct plans with the minimum number of required actions, but SMT solvers clearly outperformed the PDDL planners in both search-space scalability and execution time. While the PDDL planners FF, GSP, and HSP could solve only relatively small search spaces and ran out of memory beyond approximately 11 nodes, all SMT solvers successfully handled substantially larger problems. Among the evaluated alternatives, MathSAT5 delivered the best overall performance, maintaining low and stable solving times even for the largest tested search spaces. The superior performance of SMT can be attributed to the bounded model checking formulation and synchronized automata representation used in this work. These methods are particularly effective for the relatively shallow transfer-planning problems created by the recursive planning strategy. Consequently, SMT solvers, and especially MathSAT5, are recommended for runtime planning of machine-level transfer operations in Plug & Produce manufacturing systems where fast response times and scalability are essential. Although heuristic PDDL planners remain relevant for deeper and more complex planning domains, SMT-based planning proved most suitable for the recursive transfer-planning problems addressed in this study.

The C-MAS framework represents a forward-thinking approach to modern manufacturing by combining modular design, multi-agent systems, and advanced planning technologies, offering a flexible and intelligent alternative to traditional manufacturing control systems. Human-centric configurable multi-agent systems with agents that have access to automatic planners lay a strong foundation for the future of adaptive, autonomous manufacturing.

Author contributions All authors contributed to the conception, study, and design. Material preparation, experimental setups, test-bed design, data collection, and analysis were performed by Anders Nilsson. The main part of the C-MAS implementation was performed by Fredrik Danielsson. Bengt Lenartson contributes his extensive knowledge of automated planners and solvers. The first draft of the manuscript was written by Anders Nilsson, and all authors commented on earlier versions. All authors read and confirmed the final manuscript.

Funding This work was supported by the KK-foundation, Dnr. 20230032.

Data availability Data from test runs are all presented in this paper. A description of C-MAS is available at https://www.hv.se/en/research/research-environments/production-technology/flexible-automation/C-Mas/
C-MAS, documentation, and a simulated Plug & Produce gantry robot are available and free to use at: https://github.com/UniversityWest-IndustrialAutomation/CMAS C-MAS version 2.11 was used for all verifications in this paper.

Declarations

Competing interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Safeea M, Neto P (2022) Precise positioning of collaborative robotic manipulators using hand-guiding. Int J Adv Manuf Technol 120(7–8):5497–5508. https://doi.org/10.1007/s00170-022-09107-1
2. Mayr-Dorn C, Winterer M, Salomon C, Hohensinger D, Ramler R (2021) Considerations for using Block-Based Languages for Industrial Robot Programming - a Case Study, in *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*, pp. 5–12. https://doi.org/10.1109/RoSE52553.2021.00008
3. Saleem Z, Gustafsson F, Furey E, McAfee M, Huq S (2025) A review of external sensors for human detection in a human robot collaborative environment. J Intell Manuf 36(4):2255–2279. https://doi.org/10.1007/s10845-024-02341-2
4. Lyu G, Brennan RW (2025) Multi-agent modelling of cyber-physical systems for IEC 61499-based distributed intelligent automation. Int J Comput Integr Manuf 38(5):596–622. https://doi.org/10.1080/0951192X.2023.2294442
5. Mayer A et al (2025) Digital twins, extended reality, and artificial intelligence in manufacturing reconfiguration: a systematic

literature review. Sustainability 17(5):2318. https://doi.org/10.3390/su17052318
6. Shi C et al (2025) Automatic path planning for pelvic fracture reduction with multi-degree-of-freedom. Comput Methods Programs Biomed 261:108591. https://doi.org/10.1016/j.cmpb.2025.108591
7. Fu B et al (2025) Digital twin-based smart manufacturing: dynamic line reconfiguration for disturbance handling. IEEE Trans Autom Sci Eng 22:14892–14905. https://doi.org/10.1109/TASE.2025.3563320
8. Gharibvand V et al (2024) Cloud based manufacturing: a review of recent developments in architectures, technologies, infrastructures, platforms and associated challenges. Int J Adv Manuf Technol 131(1):93–123. https://doi.org/10.1007/s00170-024-12989-y
9. Du B, Zhang H, Guo J (2025) Cooperative-based multi-agent reinforcement learning for dynamic job shop scheduling. Int J Comput Integr Manuf 1–20. https://doi.org/10.1080/0951192X.2025.2515389
10. Schirmer F, Kranz P, Rose CG, Schmitt J, Kaupp T (2025) Towards dynamic human–robot collaboration: a holistic framework for assembly planning. Electronics (Basel) 14(1):190. https://doi.org/10.3390/electronics14010190
11. Haslum P et al (2022) An introduction to the planning domain definition language, vol 13. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-031-01584-7
12. Bofill M, Palahí M, Suy J, Villaret M (2012) Solving constraint satisfaction problems with SAT modulo theories. Constraints 17(3):273–303. https://doi.org/10.1007/s10601-012-9123-1
13. Saffar F, Moghaddam SK, Huang S (2025) Dynamic layout design for scalable reconfigurable manufacturing systems. Int J Prod Res 1–23. https://doi.org/10.1080/00207543.2025.2474215
14. Ramu A (2024) A review of manufacturing technologies on the industry: categories, integration and impacts. J Enterprise Bus Intell 232–241. https://doi.org/10.53759/5181/JEBI202404024
15. Arai T, Aiyama Y, Maeda Y, Sugi M, Ota J (2000) Agile assembly system by 'plug and produce.' CIRP Ann 49(1):1–4. https://doi.org/10.1016/S0007-8506(07)62883-2
16. Labbi H, Sekkat S (2024) "From flexible manufacturing systems to evolvable production systems: review and implementation,." Networks and systems, vol vol. 1098. Springer, Cham., pp 496–505. https://doi.org/10.1007/978-3-031-68650-4_47
17. Rocha AD et al. (2024) Human-centric digital twin-driven approach for Plug-and- Produce in modular cyber-physical production systems, in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, pp. 1–7. https://doi.org/10.1109/ETFA61755.2024.10710730
18. Kovalenko I, Balta EC, Tilbury DM, Barton K (2023) Cooperative product agents to improve manufacturing system flexibility: a model-based decision framework. IEEE Trans Autom Sci Eng 20(1):440–457. https://doi.org/10.1109/TASE.2022.3156384
19. Luzolo PH, Elrawashdeh Z, Tchappi I, Galland S, Outay F (2024) Combining multi-agent systems and artificial intelligence of things: technical challenges and gains. Internet Things 28:101364. https://doi.org/10.1016/j.iot.2024.101364
20. Lim J, Vogel-Heuser B, Kovalenko I (2024) Large language model-enabled multi-agent manufacturing systems, in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, IEEE, pp. 3940–3946. https://doi.org/10.1109/CASE59546.2024.10711432
21. Zhao Z et al (2026) A large language model-based multi-agent manufacturing system for intelligent shopfloors. Adv Eng Inform 69:103888. https://doi.org/10.1016/j.aei.2025.103888
22. Farahani MA, Khan MI, Wuest T (2026) Hybrid agentic AI and multi-agent systems in smart manufacturing. J Manuf Syst 86:612–623. https://doi.org/10.1016/j.jmsy.2026.04.002
23. Nilsson A, Danielsson F, Svensson B (2023) Customization and flexible manufacturing capacity using a graphical method applied on a configurable multi-agent system. Robot Comput Integr Manuf 79. https://doi.org/10.1016/j.rcim.2022.102450
24. Nilsson A, Danielsson F, Svensson B (2023) From CAD to plug & produce. Int J Adv Manuf Technol. https://doi.org/10.1007/s00170-023-12280-6
25. Sapel P, Molinas Comet L, Dimitriadis I, Hopmann C, Decker S (2024) A review and classification of manufacturing ontologies. J Intell Manuf. https://doi.org/10.1007/s10845-024-02425-z
26. Onori M, Semere D, Lindberg B (2011) Evolvable systems: an approach to self-X production. Int J Comput Integr Manuf 24(5):506–516
27. Cavin S, Ferreira P, Lohse N (2013) Dynamic skill allocation methodology for evolvable assembly systems, in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, IEEE, pp. 218–223. https://doi.org/10.1109/INDIN.2013.6622885
28. Danny P, Ferreira P, Lohse N, Dorofeev K (2018) An Event-Based AutomationML Model for the Process Execution of Plug-and-Produce' Assembly Systems, in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, IEEE, pp. 49–54. https://doi.org/10.1109/INDIN.2018.8471955
29. Reichler A-K, Gabriel F, Timmann F, Steil J, Dröder K (2019) An architecture for AutomationML-based constraint modelling and orchestration of incremental manufacturing. Procedia CIRP 86:173–178. https://doi.org/10.1016/j.procir.2020.01.020
30. Järvenpää E, Siltala N, Hylli O, Lanz M (2019) The development of an ontology for describing the capabilities of manufacturing resources. J Intell Manuf 30(2):959–978. https://doi.org/10.1007/s10845-018-1427-6
31. Heuss L, Gebauer D, Reinhart G (2024) Concept for the automated adaption of abstract planning domains for specific application cases in skills-based industrial robotics. J Intell Manuf 35(8):4233–4258. https://doi.org/10.1007/s10845-023-02211-3
32. Aguado E, Gomez V, Hernando M, Rossi C, Sanz R (2024) A survey of ontology-enabled processes for dependable robot autonomy. Front Robot AI 11. https://doi.org/10.3389/frobt.2024.1377897
33. Fikes RE, Nilsson NJ (1971) Strips: a new approach to the application of theorem proving to problem solving. Artif Intell 2(3–4):189–208. https://doi.org/10.1016/0004-3702(71)90010-5
34. Grundke C, Röger G, Helmert M (2024) Formal representations of classical planning domains. Proc Int Conf Automated Plann Sched 34:239–248. https://doi.org/10.1609/icaps.v34i1.31481
35. Taitler A et al (2023) The 2023 International Planning Competition. AI Mag 45(2):280–296. https://doi.org/10.1002/aaai.12169
36. Davis M, Logemann G, Loveland D (1962) A machine program for theorem-proving. Commun ACM 5(7):394–397. https://doi.org/10.1145/368273.368557
37. Marques-Silva J, Malik S (2018) Propositional SAT solving. Handbook of model checking. Springer International Publishing, Cham, pp 247–275. https://doi.org/10.1007/978-3-319-10575-8_9
38. Rintanen J (2012) Planning as satisfiability: heuristics. Artif Intell 193:45–86. https://doi.org/10.1016/j.artint.2012.08.001
39. Elahi M, Rintanen J (2024) Optimizing the optimization of planning domains by automatic action schema splitting. Proc AAAI Conf Artif Intell 38(18):20096–20103. https://doi.org/10.1609/aaai.v38i18.29988
40. de Moura L, Dutertre B, Shankar N (2007) A tutorial on satisfiability modulo theories. Computer science, vol vol. 4590. Springer, Berlin, Heidelberg, pp 20–36. https://doi.org/10.1007/978-3-540-73368-3_5
41. Biere A, Cimatti A, Clarke E, Zhu Y (1999) Symbolic Model Checking without BDDs. 193–207. https://doi.org/10.1007/3-540-49059-0_14
42. De Giacomo G, M Vardi Y (2000) Automata-theoretic approach to planning for temporally extended goals. 226–238. https://doi.org/10.1007/10720246_18

43. Tappler M, Aichernig BK, Lorber F (2022) Timed Automata Learning via SMT Solving. 489–507. https://doi.org/10.1007/978-3-031-06773-0_26
44. Ghallab M, Nau D, Traverso P (2016) Automated planning and acting. Cambridge University Press
45. Pellier D, Fiorino H (2018) PDDL4J: a planning domain description library for java. J Exp Theor Artif Intell 30(1):143–176. https://doi.org/10.1080/0952813X.2017.1409278
46. Azhar S (2011) Building information modeling (BIM): trends, benefits, risks, and challenges for the AEC industry. Leadersh Manag Eng 11(3):241–252. https://doi.org/10.1061/(ASCE)LM.1943-5630.0000127
47 Bonet B, Geffner H (2001) Heuristic search planner 2.0. AI Mag 22(3):77
48. Hoffmann J (2001) FF: the fast-forward planning system. AI Mag 22(3):57. https://doi.org/10.1609/aimag.v22i3.1572
49. Baier D, Beyer D, Friedberger K (2021) JavaSMT 3: Interacting with SMT Solvers in Java, in *Computer Aided Verification*, Springer, pp. 195–208. https://doi.org/10.1007/978-3-030-81688-9_9
50. Dobos-Kovács M, Vörös A (2022) Evaluation of SMT solvers in abstraction-based software model checking, in *Proceedings of the 11th Latin-American Symposium on Dependable Computing*, New York, NY, USA: ACM, pp. 109–116. https://doi.org/10.1145/3569902.3570187
51. Ko J, Ennemoser B, Yoo W, Yan W, Clayton MJ (2023) Architectural spatial layout planning using artificial intelligence. Autom Constr 154:105019. https://doi.org/10.1016/j.autcon.2023.105019
52. Hoare C (1985) Communicating sequential processes. Prentice-Hall Int Ser Comput Sci 21(8):666–677
53. Cimatti A, Griggio A, Schaafsma BJ, Sebastiani R (2013) The mathsat5 smt solver, in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 93–107

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.