Journal of
Manufacturing and Materials Processing

MDPI

Article

A Universal Framework for Skill-Based Cyber-Physical Production Systems

Max Hossfeld [1,*] and Andreas Wortmann [2]

[1] InnovationCampus Future Mobility, University of Stuttgart, 70569 Stuttgart, Germany
[2] Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, 70174 Stuttgart, Germany; andreas.wortmann@isw.uni-stuttgart.de
* Correspondence: max.hossfeld@icm.uni-stuttgart.de

Abstract: In the vision of smart manufacturing and Industry 4.0, it is vital to automate production processes. There is a significant gap in current practices, where the derivation of production processes from product data still heavily relies on human expertise, leading to inefficiencies and a shortage of skilled labor. This paper proposes a universal framework for skill-based cyber–physical production systems (CPPS) that formalizes production knowledge into machine-processable formats. Key contributions include a novel conceptual model for skill-based production processes and an automated method to derive production plans from high-level CPPS skills for production planning and execution. This framework aims to enhance smart manufacturing by enabling more efficient, transparent, and automated production planning, thereby addressing the critical gap in current manufacturing practices. The framework's benefits include making production processes explainable, optimizing multi-criteria systems, and eliminating human biases in process selection. A case study illustrates the framework's application, demonstrating its current capabilities and potential for modern manufacturing.

Keywords: automated production planning; model-driven development; cyber–physical production systems; software-defined manufacturing; skill-based manufacturing; industrial engineering

Citation: Hossfeld, M.; Wortmann, A. A Universal Framework for Skill-Based Cyber-Physical Production Systems. *J. Manuf. Mater. Process.* **2024**, *8*, 221. https://doi.org/10.3390/jmmp8050221

Academic Editor: Enkhsaikhan Boldsaikhan

Received: 28 August 2024
Revised: 18 September 2024
Accepted: 20 September 2024
Published: 2 October 2024

1. Motivation

The concept of smart manufacturing and the implementation of Industry 4.0 should provide fundamental solutions to many of today's most urgent issues in production. Central to both concepts is the intensive utilization of cyber–physical systems, IoT, big data analytics, and automation, often combined with advanced manufacturing technologies and robotics. The implementation of these two concepts has led to remarkable progress in many areas of industrial value creation, especially in the last decade.

Particular areas of tremendous progress related to manufacturing include demand forecasting [1,2], capacity planning and scheduling [3–5], and general production process optimization [6–9]. When analyzing recent advances and current research activities [10–13], a gap in the complete vision of smart manufacturing can be identified, namely the problem of automatically deriving production processes and sequences from product data.

When it comes to the question of how components should be manufactured, i.e., production planning and process engineering, the answer is still mainly human-based. Production planning and process engineering require a very high level of manufacturing expertise, as well as a comprehensive understanding of the technological, economic, logistical, and company-specific inter-relationships and constraints. As a result, these complex and time-consuming tasks are typically performed by a scarce resource, i.e., highly skilled professionals, often with decades of relevant experience. These experts extract relevant information, such as functional surfaces or tolerances, from drawings or Computer-Aided Design (CAD) models and identify and define capable processes and machines. They then

use this information to define production sequences and allocate all the resources needed to produce the component. Typically, these activities require the application of implicit, unformalized knowledge. This situation leads directly to several problems for today's manufacturing. Probably the biggest and most actively growing problem is the availability of such competencies in the context of the shortage of skilled labor [14,15]. This particularly affects small and medium enterprises (SMEs), such as contract manufacturers, which tend to have a more volatile manufacturing portfolio and, therefore, a higher proportion of such activities. Directly related to this problem is the circumstance that although planning activities are absolutely essential to subsequent value creation and are often a bottleneck themselves [16], they are often not seen as directly value-adding or worth being paid for by the customer. At the same time, these planning activities tie up the best production workers, who are, thus, unable to carry out other potentially value-creating activities [17].

But even when skilled professionals are available, finding an optimal solution is challenging. First, production planning is a complex, multidimensional optimization problem with competing objectives, such as time, quality, and cost, often with changing and unclear weighting [18,19]. What constitutes an optimal solution is, therefore, a moving target or, often enough, remains unclear. Very often, not all the necessary information is available at at time of planning and decision making, so assumptions have to be made; a number of production sequences may be approximately equivalent at the time of the decision but not shortly afterwards; decisions made by people depend strongly on their educational background and their own preferences, experiences, or competencies. Technologies evolve, and their capabilities change. Moreover, an optimal individual solution for one product does not necessarily lead to an optimal solution for the whole production system, and each individual decision affects the availability of resources for other and future products. Several current trends, such as personalization, on-demand manufacturing, and new manufacturing technologies, amplify these effects.

Target and Contribution

The target of this study is to fully automate production and process planning, i.e., answer the question of how components should be manufactured. To achieve this, all related activities performed by humans today need to be automated. This requires the representation of these activities and the required production knowledge in machine-processable form. Therefore, we propose a universal framework for skill-based cyber–physical production systems (CPPSs). In this framework, formal production skill models capture activities and the required knowledge to automatically answer the above question. The core contributions of this work include the following:

- A novel conceptual model for skill-based production processes and planning;
- A method for automatically deriving platform-specific production plans from platform-independent skills; and
- A reference software architecture that links skills with planning and plan execution.

This novel approach to skill-based production planning contributes to the vision of smart manufacturing for Industry 4.0 [20] as follows:

- By automating process and production planning;
- By making processes and their inherent decisions explainable;
- By simultaneously applying multi-criteria optimization to the whole production system
- By making implicit production knowledge explicit, easily available, and reusable;
- By providing a base for the integrated assessment of producibility in the design processes;
- By providing a base for the derivation of completely new machine designs and process combinations; and
- By eliminating human bias in the selection of processes, activities, and machines.

Section 2 introduces the foundations for our approach. Section 3 then outlines related approaches and the gap that the proposed framework fills. Then, Section 4 introduces our

conceptual framework, and Section 5 describes its methodical application, while Section 6 illustrates its application in a case study involving the production of a coffee cup. Finally, Section 7 discusses the benefits and limitations of our proposed framework and concludes this paper.

2. Foundations

Over the past few decades, the amount and importance of software in production systems have increased significantly, along with the complexity of these systems [21,22]. Today's state-of-the-art cyber–physical production systems (CPPSs) are heterogeneous, partially distributed, and often connected to a network or the Internet. Engineering software for these increasingly complex CPPSs requires concepts, methods, and tools that scale with complexity.

A major driver of the complexity of modern software systems is in the wide conceptual gap between the problem domain and the solution domain [21] of software engineering, i.e., production experts need to describe what a CPPS should do (problem domain) but need to formulate that process using programming tools focusing on solution technologies (solution domain). This gap creates so-called accidental complexities [21] that need to be addressed to create added value in software. This includes issues such as programming language peculiarities, network communication issues, data persistence, software deployment, and memory management, all of which may be conceptually irrelevant in the problem domain. These accidental complexities increase software development risks and must be reduced. To efficiently deliver reliable, reusable, and secure CPPS software, domain experts would have to become software engineering experts as well—which is not feasible. Model-driven development (MDD) [23] can increase abstraction in the problem domain to reduce the conceptual gap in the engineering of CPPS software [21,24].

2.1. Model-Driven Development

Model-driven development (MDD) is a software engineering methodology in which developers leverage domain-specific models to reduce the conceptual gap and, consequently, the accidental complexities. To achieve this, MDD employs models as primary development artifacts in various stages of software development, ranging from requirement modeling to implementation, deployment, and operations. The description of models requires corresponding notations in the form of (domain-specific) modeling languages [25] that enable the use of domain terminology, concepts, and knowledge. As modeling aims to increase abstraction, there have been many successful applications of MDD [26,27] in the context of software engineering for CPPSs [28]. Popular modeling languages for the engineering of CPPS software include the Asset Administration Shell [29], AutomationML [30], Matlab Simulink [31], OPC UA [32], and SysML [33] languages.

Model-driven development yields many benefits, ranging from the avoidance of accidental complexities, to providing a comprehensible means for communication and documentation and a formal foundation that allows for the translation of models into executable systems [26,34,35]. This requires tools that translate such models into general-purpose programming language (GPL) artifacts and (executable) software automatically while employing incorporated software engineering expertise. This frees domain experts from the need to become software engineering experts.

2.2. Model-Driven Architecture

Model-driven architecture (MDA) [36,37] is a specialization of MDD driven by standardization efforts proposed by the Object Management Group. The main rationale of MDA is that domain concepts outlive their technical realizations. For example, moving a robot from one location to another demands certain domain-specific activities, such as collision avoidance, navigation, and path planning, independent of the software components with which these activities are realized. Hence, MDA aspires to separate platform-independent and domain-specific models from their subsequent application to specific target platforms

and their software environments. Therefore, model-driven architecture divides an application into four different layers (cf. Figure 1) representing the following:

- The computation-independent model (CIM) of the application;
- Its platform-independent model (PIM);
- Its platform-specific model (PSM); and
- Its platform-specific implementation (PSI), i.e., program code.

In these terms, we propose a pervasive approach leveraging MDA for the MDD of CPPSs that separates PSMs of skills from PSMs of services that map capabilities to functions of CPPSs and from PSIs (such as G-code or IEC 61449 [38] programs) that realize these CPPS functions for automated process and production planning.

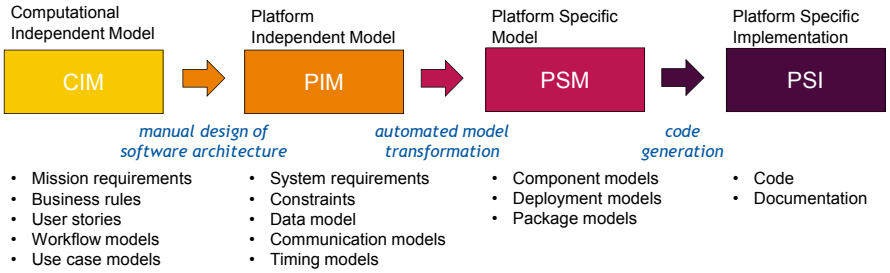


Figure 1. The four layers of MDA and related elements.

Therefore, we assume the following terminology in the remainder of this paper:

- **Capability**: The general ability of a CPPS in any form. This includes text, such as prose in documentation, tables, graphics, and more.
- **Skill**: A machine-processable representation of a (part of a) capability of a CPPS in the specific formalism presented below.
- **Service**: A (set of) accessible methods(s) provided by a platform-specific model.
- **Function**: A software artifact that realizes a service of a CPPS. It might employ other software artifacts (such as a CAM system, an MES, or an LLM) to achieve that.

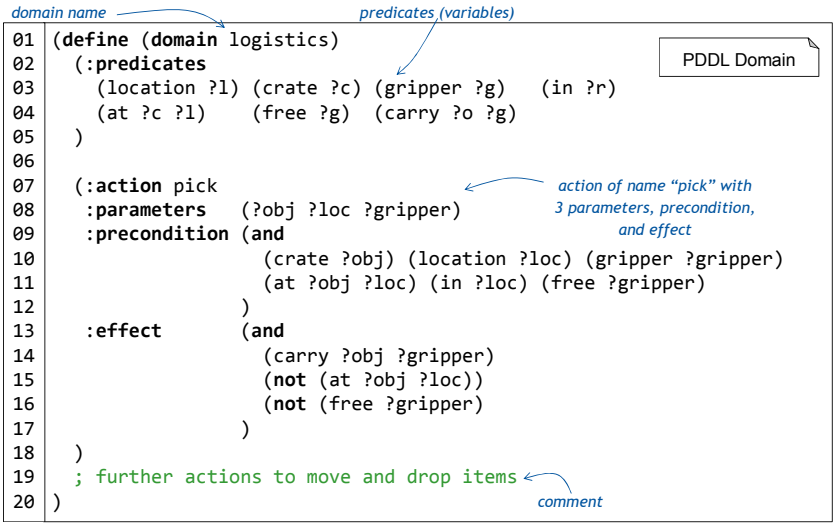
2.3. Automated Planning

Automated planning [39] is a paradigm of Artificial Intelligence (AI) in which planning software supports the finding of solutions to specified problems through search and optimization in a multidimensional problem space. To this end, planners employ a model of the domain and are provided with models of the initial state of that domain, as well as a goal to be solved. Using a variety of planning and optimization techniques, the planning software then explores the search space defined by the domain model to produce a solution that achieves the goal from the initial state or fails [40].

A popular modeling language to describe planning domains, initial states, and goals is the Planning Domain Definition Language (PDDL) [41]. Using PDDL, a *planning domain* can be described as predicates (relationships between domain elements) that hold in this domain and actions available to agents operating on this domain. Each action comprises *pre-conditions* that must hold before they can be executed and *effects* that hold after the action has been executed. Once an action is executed, its effects are applied to the overall planning domain state.

For instance, the logistics planning domain illustrated in Figure 2 (top) comprises seven predicates (ll. 3–5); six unary predicates, e.g., about whether some object of the domain is a `location` or whether the robot is `in` a specific location; and the binary predicate `carry` to denote which of the robot's grippers is carrying which object. The domain also defines the action `pick` (ll. 7ff), which features three parameters (l. 8), a pre-condition (ll. 9–12), and an effect (ll. 13–17). Both pre-condition and effect are Boolean expressions over predicates of the domain. The pre-condition states that to execute `pick`, it must hold that the `object` to be picked up is a crate, the place where it shall be picked up is a `location`, the `gripper` to

pick it up is a gripper, the `obj` is at a location `in` which the robot is present, and the gripper is `free`.

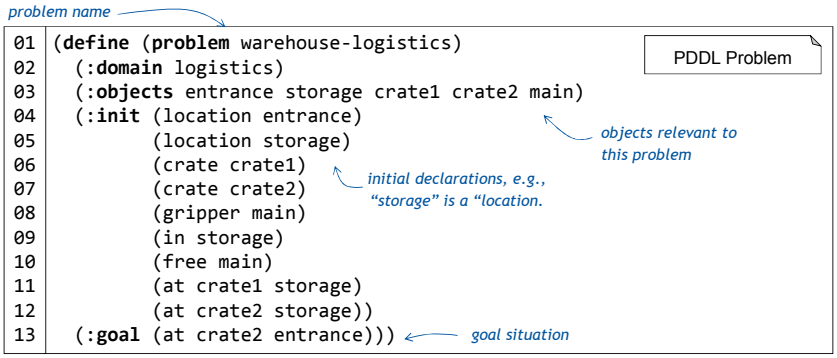



```
01 (define (domain logistics)
02   (:predicates
03     (location ?l) (crate ?c) (gripper ?g)   (in ?r)
04     (at ?c ?l)    (free ?g)  (carry ?o ?g)
05   )
06
07   (:action pick
08    :parameters   (?obj ?loc ?gripper)
09    :precondition (and
10                    (crate ?obj) (location ?loc) (gripper ?gripper)
11                    (at ?obj ?loc) (in ?loc) (free ?gripper)
12                  )
13    :effect       (and
14                    (carry ?obj ?gripper)
15                    (not (at ?obj ?loc))
16                    (not (free ?gripper)
17                  )
18   )
19   ; further actions to move and drop items
20 )
```

Figure 2. Excerpt of a PDDL planning domain describing the capability of a robot to pick something up via a PDDL action.

A planning domain specifies what is possible in general but neither initial situations nor goal situations. For better reuse, planning domains are decoupled from both. Instead, both situations are defined as a PDDL *planning problem*. A PDDL problem is defined relative to a planning domain and includes the set of existing objects, as well as initial values for selected predicates of that domain and a goal situation.

The PDDL problem illustrated in Figure 3 (bottom) is defined relative to the `logistics` domain (l. 2). It defines five objects (l. 3), such as `entrance` and `crate1`, and uses the objects to initialize the predicates of the domain (ll. 4–12). Here, among other things, `entrance` is defined as a `location`, and `crate2` is defined as a `crate`. Finally, a goal (Boolean expression over predicates of the domain) is defined that states that the planner shall derive a plan that leads to the `crate2` object being located at the `entrance`.




```
01 (define (problem warehouse-logistics)
02   (:domain logistics)
03   (:objects entrance storage crate1 crate2 main)
04   (:init (location entrance)
05          (location storage)
06          (crate crate1)
07          (crate crate2)
08          (gripper main)
09          (in storage)
10          (free main)
11          (at crate1 storage)
12          (at crate2 storage))
13   (:goal (at crate2 entrance)))
```

Figure 3. Excerpt of a PDDL planning problem for the domain of Figure 2.

In the following, we employ automated planning using the OPTIC planner [42,43] over a PDDL 3.0 production domain model comprising platform-independent skill types to find optimal solutions for production goals. These solutions are sequences of parameterized skills that are then realized through platform-specific processes.

3. Related Work

Skill-based manufacturing has been a popular research subject for decades [44], according to which skills can be understood as "manufacturer-independent standardized

automation functions" [45] that aim to decouple logical production planning from technical implementations in specific machines. Through the application of skills, researchers also seek to address manufacturing flexibility, planning efficiency, and product variability [46].

To improve production flexibility through the application of skills, research has produced various approaches that make the modeling of skills, products, processes, and resources (PPR) explicit. Many of these approaches employ concepts similar to Model-Driven Architecture (MDA) to decouple logical skills from technical implementations.

Purely logical skills. However, many approaches consider skills to be purely logical only, i.e., omit their effect on the geometry of the workpiece in processing [47–49]. While it might be possible to describe all effects of a process purely logically, it (a) is rather inefficient compared with employing CAD templating and (b) leads to inefficient planning over multiple infinite numerical domains. When skills are represented with the Resource Description Framework (RDF), e.g., [50], efficient numerical planning is impossible without any further transformations. While skills lend themselves to artificial intelligence action planning, as they abstract and discretize manufacturing activities [51], their representation matters. For instance, the representation of skills in PDDL has been investigated following various approaches [52–55]. All of these approaches to decouple abstract skills from technical implementations for the automatic derivation of production plans also consider a purely logical level. Hence, they fall short for the same reasons as the skill-based manufacturing approaches over PPRs explained above.

Other approaches employ purely logical skills at different levels of abstraction and leave it to domain experts using the skills to combine them into production plans, e.g., by orchestrating skills as UML state charts [56], UML activity diagrams [57], in Business Process Model and Notation (BPMN) format [58], or using domain-specific orchestration languages [59]. While this avoids the challenges of planning over infinite numerical domains, it (a) demands that domain experts understand the modeling languages used for orchestration and (b) reduces flexibility by fixing the available production plans a priori.

Some approaches also employ ontologies, often in the form of Web Ontology Language (OWL) or RDF knowledge bases, to model the skills of CPPSs such that existing planning systems for these formalisms can be employed for planning [50,60]. These generally focus solely on logical planning and ignore the challenges of geometric planning [60] or require that skills be modeled as deterministic processes [50] without leveraging the potential of AI action planning over skills. The submodel templates of the Asset Administration Shell (AAS), which represent various types and their relations exposed by the AAS of a CPPS, could also be exploited as a foundation for a PDDL domain model in the future.

Platform-specific skills. Skills can be realized through different implementation technologies. For instance, there are means to derive high-level skill representations from source-code implementations [61]. With OPC UA [62] becoming a de facto standard for the communication of cyber–physical production systems, it is natural to leverage OPC UA machine interfaces to provide skill implementations to their environment [59,63]. Research has even produced a mapping from CAD primitives to OPC UA services that might be applied in the context of our framework [64]. Other researchers have employed, for instance, IEC 61499 [38], to realize skills on specific machines [47]. Although this avoids the need to translate OPC UA interfaces to IEC 61499 programs, such approaches still demand means to invoke skills remotely (e.g., through the planner). However, we consider IEC 61499 an excellent choice for the realization of OPC UA service skills. Other alternatives to skill realization are MTConnect [65] or any other interfaces provided by or on top of machine controllers. All of these yield skills being tightly coupled to a specific machine hampers the reuse of skills with other machines.

Purely conceptual skill models. Existing conceptual approaches, such as those proposed in [66,67], employ a similar combination of logical skill modeling augmented by geometric state information but fail to explain how their models are integrated and how to apply AI planning to them.

Existing approaches to fully automated skill-based production planning either focus solely on logical or graphical representations, lack abstraction between the high-level representation of machine capabilities and their technical realization, or are not (yet) accessible to automated planning at all.

4. Conceptual Framework

A major challenge associated with flexible manufacturing today is the tight coupling of hardware and software. This coupling hinders the reconfiguration of production sequences, as well as the substitution of production assets, and introduces another layer of abstraction between the descriptions of *what* should be produced and *how* it should be produced. However, this coupling can be removed in various domains, including personal and mobile computing, robotics, and automotive domains. Generally, abstraction of technical details from operating systems or middleware has enabled flexibly changing of (parts of) the software operating on a PC, cellphone, robot, or vehicle to adjust their capabilities according to changing requirements. To enable similar flexibility for CPPSs, we propose the decoupling of *platform-independent production skills* (such as drilling or welding) from *platform-specific services* that describe how a specific production machine can realize one or more of these skills and *platform-specific implementations* (program code) that ultimately realizes the services on a specific machine. This decoupling enables automated production planning over platform-independent skills liberated from specific technological implementations while introducing flexibility regarding the actual CPPSs implementing a specific skill.

To achieve this, we devised the conceptual framework presented in Figure 4, which combines the following:

- Platform-independent skill-type models (left), which describe the parameters, as well as logical and geometric pre-conditions and effects of a skill;
- Platform-specific OPC UA [62] interface models providing realizations of skill types; and
- Platform-specific implementations of these skill types on specific CPPSs.

The skill-type models are defined in a novel domain-specific language that allows for the integration of logical action planning and the specification of geometric effects relative to a Unified Modeling Language (UML) class diagram (CD) [68] domain model. Both the action parts of skill types and the CD domain model are translated into PDDL. Initial situations and goal situations are specified logically as UML object diagrams [68] and geometrically as CAD models.

Given an initial situation (state of the domain) and a goal situation to be established, the planner then uses skill types with specific parameters to create *logical action plans* (sequences of skill instances) that transform the initial situation into the goal situation, e.g., applying a combination of milling skills and drilling skills to a block of aluminum to create a valve body. While iteratively building the plan as a sequence of skill instances, the planners leverage various optimizations to reduce the search effort required to find a proper plan. Moreover, some planners can optimize against heuristics, e.g., to find the shortest sequence of skill instances or (if specified by the skills), the cheapest sequence, the sequence consuming least energy, etc. After applying every skill's action, the initial CAD model is updated according to the skill's CAD template. Once a logical action plan that fulfills the geometric and logical requirements of the goal has been derived, it its translated into a *technical action plan*, comprising a sequence of OPC UA method invocations and, where necessary, manual actions. This plan is then executed to create the products specified in the goal situation.

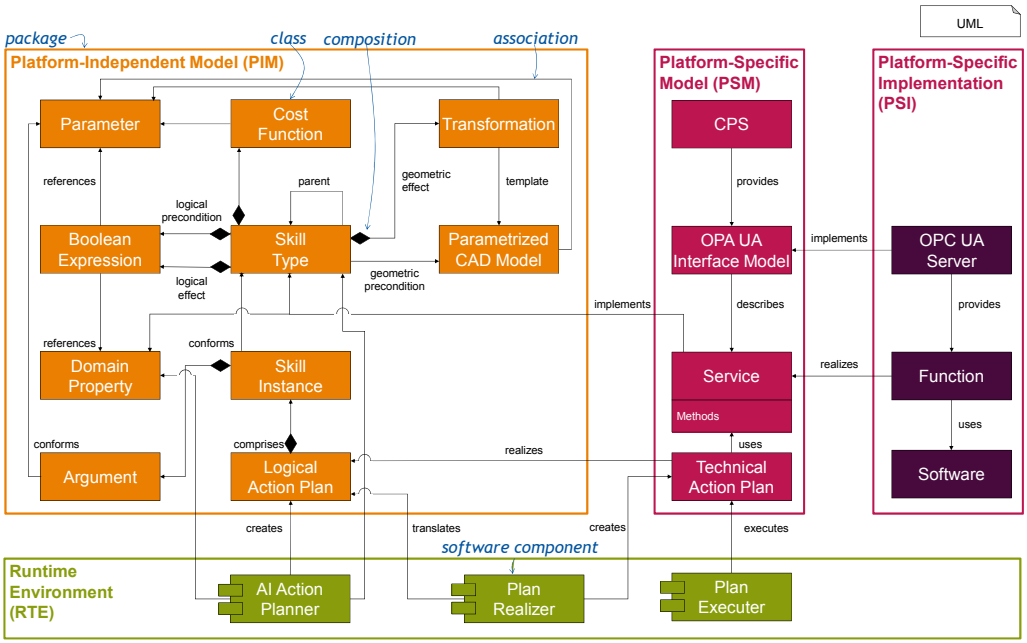


Figure 4. Conceptual framework combining platform-independent models describing skills (**left**) with platform-specific models describing services (**middle**) and their platform-specific functions (**right**) for AI action planning (**bottom**).

4.1. Platform-Independent Model

The central elements of the platform-independent model are *skill types* with parameters, costs, and logical and geometric pre-conditions and effects. The effects of a skill are defined in terms of PDDL actions and by related templates. Each skill type defines properties that hold for all instances of that particular type, i.e., a `drilling` skill type describes the parameters, cost function, etc., for all instances of `drilling`. There can be multiple skill types for different variants of drilling, each with the most appropriate and specific properties.

Parameters enable the configuration of skills. For instance, a `drilling` skill might yield parameters such as position, drilling depth, drill feed, and spindle speed. Cost functions over parameters enable the description of the multidimensional costs of executing the skill. Their realization depends on the costs relevant to the skill-type modeler and might include energy, time, and material, as well as different weights applied to the individual kinds of costs. The logical pre-conditions and effects are Boolean PDDL expressions and describe the required values of domain predicates and skill-type parameters for a skill to be executable. Logical effects describe how the domain changes after applying the skill. The geometric pre-conditions and effects are the geometric conditions that need to hold before and after applying the skill instance, respectively. For the formalization and description of the geometric transformation applied by the skill, knowledge-based design templates may be used, as they are already applied in CAD [69–71]. These may include a complete description of changes to the workpiece while simultaneously defining manufacturing processes to apply those changes. Geometric pre-conditions describe required properties of the workpieces and environments for a skill to be executable. Geometric effects describe how the workpieces and environment are changed by executing a skill.

The `AI action planner` then uses the domain model predicates and skill types to translate a production target into a logical action plan. A logical action plan is a sequence of skill instances with arguments for their parameters as identified by the planner. Therefore, it uses PDDL planning tools compatible with PDDL 2.5, such as OPTIC [42,43] or Metric-FF [72].

Figure 5 illustrates the definition of a skill type for drilling using textual syntax. Moreover, each skill type must be defined in the context of one or more UML class diagram

domain model specifying available data types, predicates, and methods. These domain models must be imported (l. 3) to make their elements (right part of Figure 5) known to the skill-type definition. Each domain model must feature at least the two classes of `CostFunctions` and `Predicates`, which may contain static methods representing costs of the `Float` data type and methods representing n-ary Boolean predicates, respectively. The `Workpiece` and `Pose` classes serve as data types to be used by the `drill` skill type.

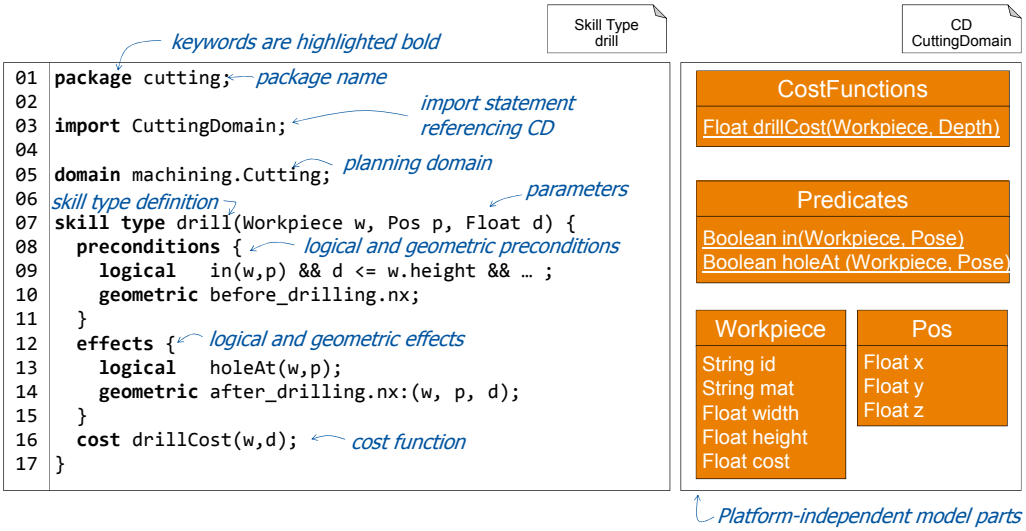


Figure 5. The class diagram domain model and skill-type model for a drilling skill.

The skill-type definition itself (ll. 7–17) begins with the skill type's name and a (possibly empty) list of parameters (l 7). It contains the logical and geometric pre-conditions (ll. 8–10) and effects (ll. 11–13) of the skill type. The logical conditions are specified as Boolean expressions over the methods defined in the `Predicates` class, and the geometric conditions refer to parametrizable CAD templates. The `cost` of the skill type refers to an arithmetic expression over the methods specified in the `CostFunctions` class. In alignment with UML conventions, method names begin with lower-case letters and use the camel-case notation.

Figure 6 illustrates an excerpt of a UML object diagram defining an instance of data types available to the planner. For each class of the domain model, multiple objects can be specified, such as `w1` and `w2` of the `Workpiece` class. Other objects may be created during planning.

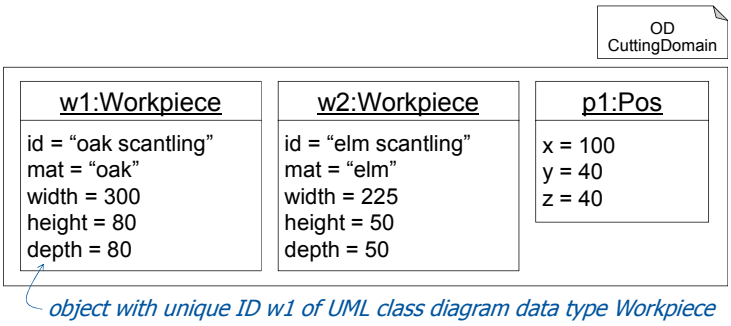


Figure 6. Instances of the drilling skill-type domain of Figure 5.

4.2. Platform-Specific Model

All predicates that are observable in the CPPS or in its environment must be grounded in reality. To this end, domain predicates related to observations, such as the position of the workpiece, need to be implemented by services. These functions do not need to be provided by a single CPPS but can be implemented by different systems (e.g., for the workpiece position, this could be a camera installed in the work cell). Similarly, the skill

types must be implemented by CPPSs such that they can change the workpiece and its environment accordingly. For platform-specific models, we assume that the services of (virtual) sensors and the CPPSs are defined in terms of OPC UA models, which expose the systems' interfaces through a URI and include a validity frame [73] that describes the extent to which, i.e., under which constraints, a specific CPPS can provide a service.

For instance, to realize the `drill` skill type, the OPC UA information model illustrated in Figure 7 might serve describes the `MillingMachine` type, which is a `MachineToolType` and specifies two variables (`Identifier, Name`), as well as two objects (`Location`, `RULDef`). The object `MyMiller` is an instance of a `MillingMachine` and has properties implementing the properties of the `MillingMachine` type. Moreover, it also yields the `drill` method, which requires a `Workpiece` and a `Depth` to operate. The `PlanRealizer` then takes the logical action plan provided by the planner as input and identifies available CPPSs to realize this. The result of this investigation is either an action plan, i.e., a sequence of services (together with their arguments) to be called, or a finding that the plan cannot be realized. The latter case also produces a hint about which capabilities are missing (e.g., a more powerful drill might be required).

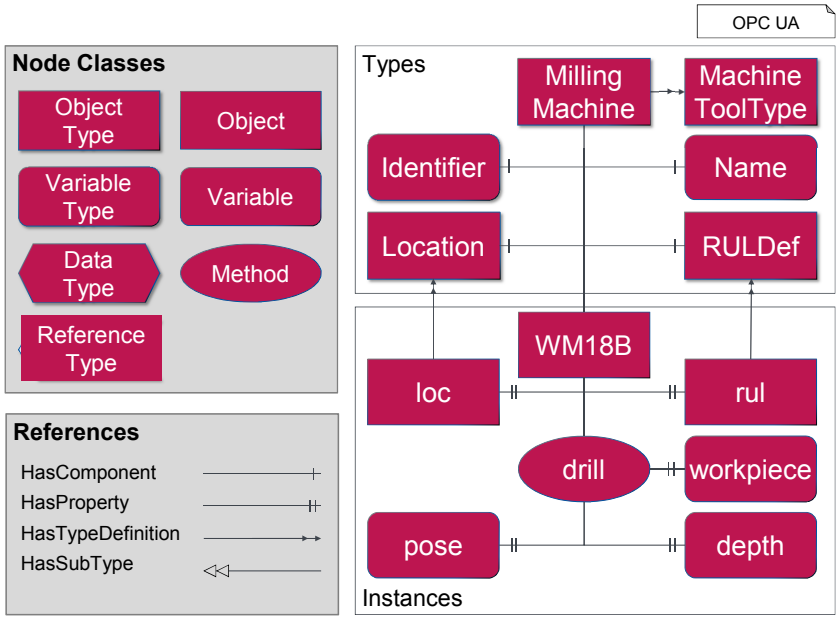


Figure 7. OPC UA information model to realize the `drill` skill type.

4.3. Platform-Specific Implementation

Each service (OPA UA endpoint) is realized through functions provided by a single CPPS. By invoking the services according to the technical action plan, the plan executor performs a sequence of system functions of participating CPPSs to achieve the production goal for which the planner produced a strategy. As plan and reality often diverge (e.g., the workpiece not being exactly in place), execution of the technical plan is iterative and checks the pre-condition of each step (skill instance) before taking the next step [74]. While this ensures that the `PlanExecutor` adheres to the plan, this also means that whenever a pre-condition of skill instance is violated, plan execution stops, and replanning from this particular situation is required.

5. Methodical Application

5.1. Instantiation

The instantiation of the conceptual framework outlined above requires a systematic method for providing platform-independent skill types, platform-specific OPC UA models that ground these skills, and platform-specific implementations of these skills on the realizing CPPSs. This section describes the roles and activities required to apply our

framework. For this, we assume the following roles, some of which might be enacted by the same person or entity:

- The machine provider is responsible for developing and commissioning the CPPS and its system functions. The latter are made accessible through an OPC UA client.
- The integrator is responsible for providing OPC UA interface models to expose selected system functions of the CPPS in a platform-independent way.
- The skill modeler describes the skill types with the logical and geometrical pre- and post-conditions and how they are implemented by the OPC UA services.
- The `ActionPlanner`, `PlanRealizer`, and `PlanExecutor` components are generic (i.e., compatible with all possible instances of the conceptual framework) and are provided by the framework developer.

Their respective activities in instantiating our framework are illustrated in Figure 8. The skill modeler collects constant and fluent domain properties and models these as a PDDL domain. Afterwards, it collects requirements regarding necessary skill types, e.g., which skills are required and how they need to be parameterized. Based on this, it models the logical pre- and post-conditions of skill types with respect to the properties in the domain. In parallel, it models the geometric pre- and post-conditions and links these from the skill types. Independent of these activities, the machine provider creates and commissions the required CPPSs and ensures that their functions are exposed through OPC UA. After both have completed their activities, the integrator creates OPC UA interface models and maps the CPPS system functions, as well as the skill types, to OPC UA services. It also configures the `PlanRealizer` and `PlanExecutor` with the resulting mappings.

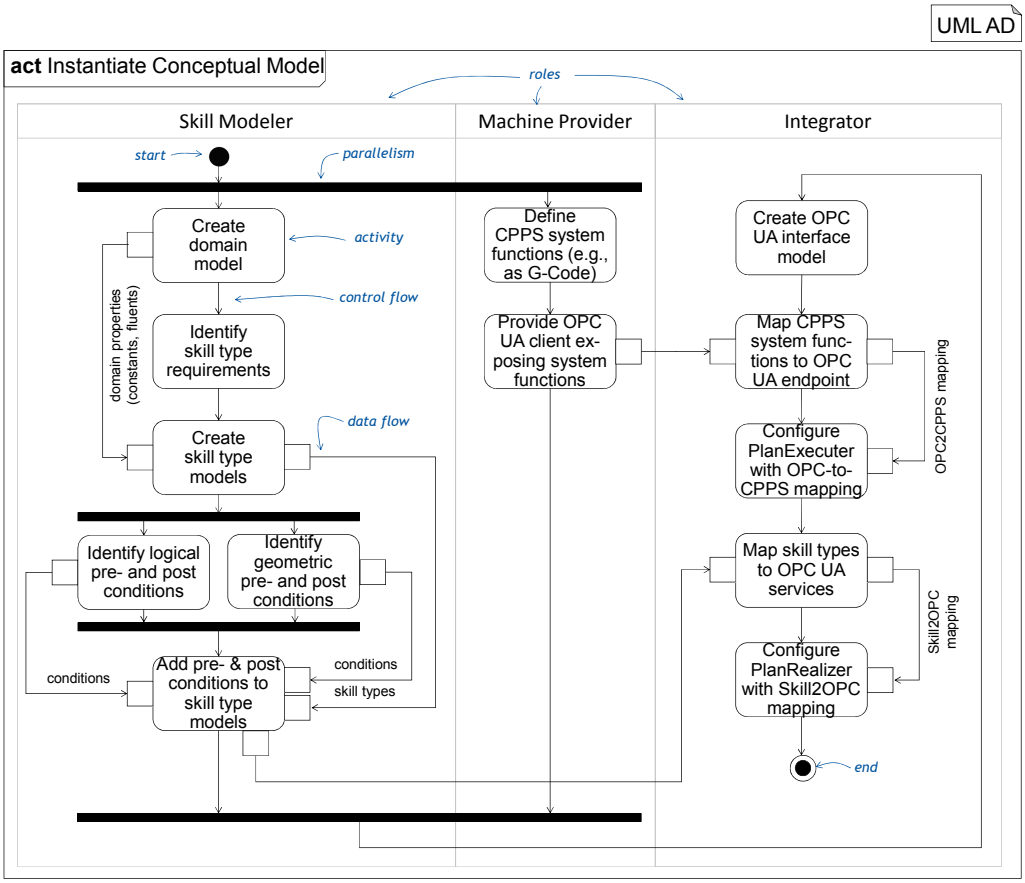


Figure 8. Activities required to instantiate our conceptual model for skill-based manufacturing.

5.2. Object and Skill Transformation

Before the planner can start deriving logical action plans, the skill-type models, the CD domain models, and the UML object diagrams describing the initial and goal situations

must be translated into a PDDL planning domain and a corresponding PDDL planning problem [75]. Given the example presented in Figure 5, as an initial step, the CD domain model and the skill types need to be translated into a PDDL planning domain by translating UML classes into relations and skill types into actions as illustrated in Figure 9.

For each attribute of a UML class in Figure 5, the transformation creates a new binary predicate in the planning domain that takes the object under consideration as a first parameter and its possible value for the respective property (ll. 4–6) as a second parameter. For instance, the `width` property of the `Workpiece` class is translated into a binary predicate `(workpiece_material ?inst ?mat)`, which relates workpiece objects to their materials (cf. Figure 9).

PDDL Domain

```
01 (define (domain mydomain)
02   (:requirements :typing :adl :durative-actions)
03   ; non-numeric properties
04   (:predicates
05     (workpiece_id       ?inst        ?id     - workpiece)
06     (workpiece_material ?inst        ?mat    - workpiece)
07     (predicates_in      ?workpiece   ?pos    - workpiece)
08     (predicates_holeAt  ?workpiece   ?pos    - workpiece)
09   )
10   ; numeric properties
11   (:functions
12     (workpiece_width           ?inst        ?width  - workpiece)
13     (pose_x                    ?inst        ?x      - pose)
14     (costsFunctions_drillCost  ?workpiece   ?depth  - workpiece)
15     ; further numeric properties
16   )
17
18   (:action cutting_drill ; prefix „cutting“ from package name
19    :parameters   (?w - workpiece ?p - pose ?d - depth)
20    :precondition (and
21                    (predicates_in ?w ?p) (workpiece_height ?w ?h)
22                    (> ?h ?d) ; further preconditions
23                  )
24    :effect       (and
25                    (predicates_holeAt(?w ?p)
26                    ; further preconditions
27                  )
28   )
29 )
```

Figure 9. Excerpt of the result of translating the example in Figure 5 to a PDDL planning domain.

For each method using the `Predicates` class, the transformation produces a predicate of the same name. Here, the first parameter is, again, the instance of the object being considered, and the remaining parameters represent the method parameters as specified in the input-class diagram (ll. 7–9). For instance, the `holeAt()` method of the `Predicates` class becomes a predicate `(predicates_holeat ?workpiece ?pos)`. Each method from the class diagram returning numerical values (including the methods of the `CostFunctions` class), the transformation produces a function following the same pattern (ll. 12–14). Afterward, for each skill type, a new `action` using the parameters, pre-conditions, and effects of the respective skill types is created (ll. 18–28). Using fully qualified class names as prefixes for names of predicates, functions, and skills avoids ambiguity.

A planning problem using the drilling skill type can be defined as illustrated in Figure 10. The planning problem enumerates the constants (ll. 3–8), as well as the objects (ll. 9–14), together with their PDDL types, and initializes the properties of the objects—in this case, of the workpieces (ll. 15–23). Afterwards, the planning domain specifies the planning goal (ll. 24–29), which entails two holes in the workpieces at certain positions and specifies which metric to minimize (which uses the cost functions, cf. Figure 5).

PDDL Problem

```
(define (problem myproblem)
  (:domain mydomain)
  (:constants
    oak_scantling - workpiece_id_name
    oak - workpiece_mat_nanme
    elm_scantling - workpiece_id_name
    elm - workpiece_mat_nanme
  )
  (:objects
    w1 - workpiece
    w2 - workpiece
    p1 - pos
    ; further objects
  )
  (:init
    (workpiece_id w1 oak_scantling)
    (workpiece_mat w1 oak)
    (= (workpiece_width w1) 300)
    (= (workpiece_height w1) 80)
    (= (workpiece_depth w1) 80)
    (workpiece_id w2 elm_scantling)
    ; further predicate initialization
  )
 (:goal
    (and
      (predicates_holeAt w1 40)
      (predicates_holeAt w2 30)
    )
  )

  (:metric minimize ; minimize costs as sum over all cost functions
    (+
      (costsFunction_drillCost w1 40)
      (costsFunction_drillCost w2 30)
    )
  )
)
```

Figure 10. PDDL planning problem obtained by translating the skill types, their class diagram domain model, and the instances specified in the corresponding object diagram to PDDL.

With this planning problem defined, the planner can try to find a sequence of skill instances that fulfills the goal while optimizing the cost of skill execution. To execute the plans in reality, additional preparation is required.

5.3. Preparation

Executing sequences of skill instances in reality demands their mapping onto relevant OPC UA information models (see the right side of Figure 4). Therefore, the skill modeler configures the plan realizer with a map from skill types and their parameters to OPC UA method calls and their parameters. If a skill type needs adaptation to be mapped to one or more OPC UA method call node, this can be achieved using Java and the generation gap pattern [76]. If a skill type cannot be mapped to an OPC UA information model at all, this suggests that the capabilities modeled by the skill are not available to the manufacturer. In this case, either the skill type needs to be removed, its cost must be updated to inflict a severe penalty of choice by the planner, or an OPC UA device with the required capability needs to be acquired. For skill types that need to be executed manually, their implementation (e.g., to send a message to a responsible person) can be realized either in the plan realizer using the generation gap pattern or on the OPC UA server.

To enable all this, the `PlanRealizer` (cf., Figure 11) must know all available skill types in the abstract `SkillType` class that yield the basic signature and infrastructure of all kinds of skills. The abstract `SkillType` class is extended by abstract classes generated from each specific skill type, e.g., `DrillSkillTypeBase` (cf., Figure 5), which contribute implementations of the methods defined in `SkillType` that are generated from the skill-type definition. This includes, e.g., checking pre-conditions and post-conditions before

skill execution. In handcrafted subclasses of the generated skill-type base classes, the skill modeler translates skill execution to OPC UA calls.

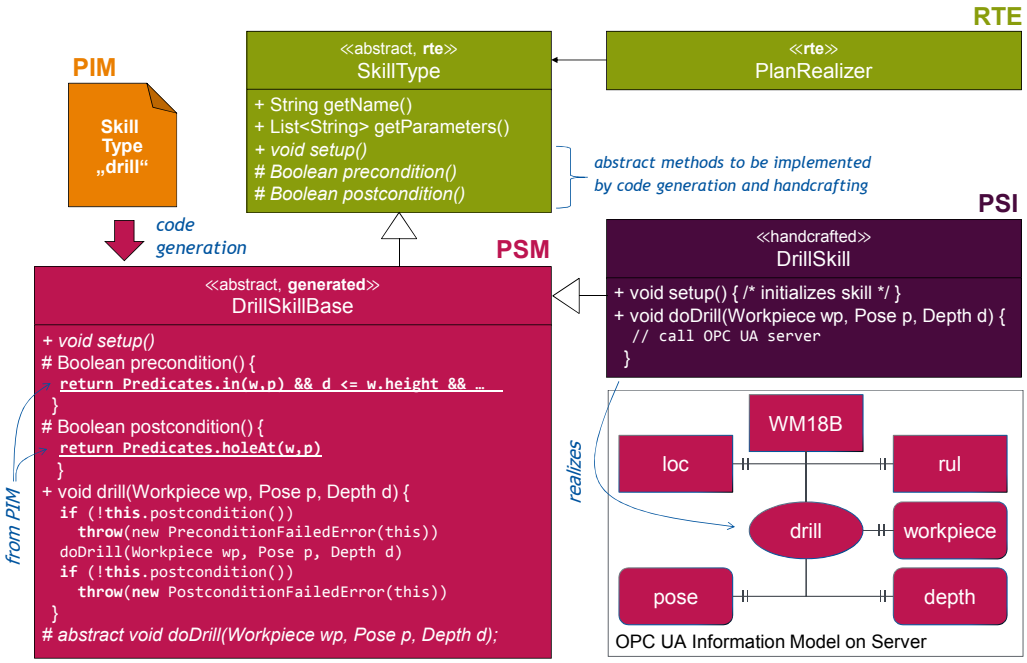


Figure 11. Outline of the software architecture of the `PlanRealizer`.

Additionally, the object diagrams describing the initial situation and final situation must be provided. As of now, both need to be created manually, but the information required for both models can—in principle—be derived from existing information. For instance, the initial situation, e.g., constants and objects with their properties, could be derived from an ERP system. The final situation (which becomes the goal expression) needs to be derived from the product specification, which is the subject to ongoing research. Once both object diagrams have been created, they can be translated to PDDL automatically [75].

5.4. Logical Action Planning

During planning time, the action planner takes the domain model as input, which includes the logical initial situation, a logical goal, a CAD model presenting the initial geometric situation, a second CAD models presenting the goal situation (i.e., the product), and a set of skill types (cf. Figure 12). The planner then sets the current intermediate logical goal (ILG) to the overall logical goal and the intermediate geometric goal (IGG) to the overall geometric goal (model). Leveraging classical backward planning, the planner first tries to find a skill type that can be used to produce the current ILG. If multiple of such skill types are found, the planner marks this planning phase as being an option for backtracking (i.e., a step to go back to and select a different skill type if subsequent planning fails), selects one of the matching skill types, and uses it. Then, the planner checks whether the instance can be applied to the current IGG. If it can, the skill is prepended to the (initially empty) plan. If it cannot be applied or the plan would contain a logical loop after adding this identified skill, the planner backtracks and tries to find alternative skills. If no such skills can be found, planning fails. Otherwise, the planner applies the skill's geometric post-condition to the IGG and its logical post-condition to the ILG, yielding an updated IGG and ILG, which are 'closer' to the initial situation. This process is iterated until either a sequence of skill instances (actions) is produced that can transform the initial situations to the goal situations by sequentially applying the skill instances or planning fails.

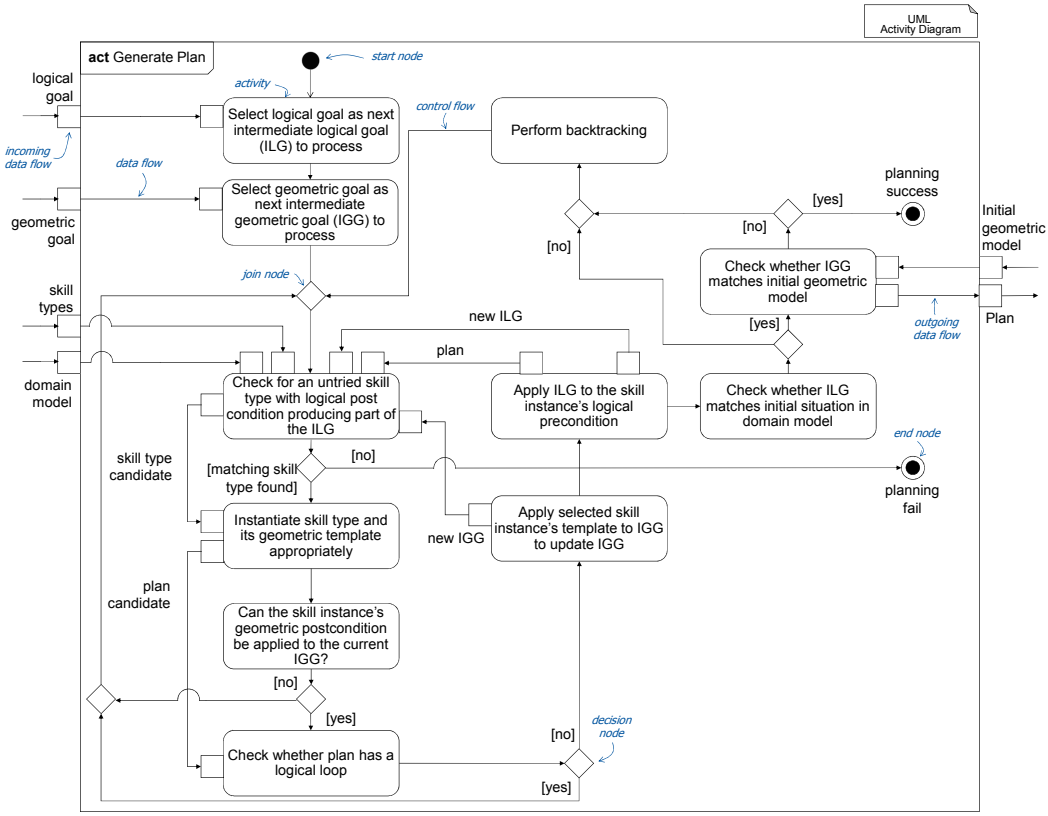


Figure 12. Overview of the skill-based generation of manufacturing plans.

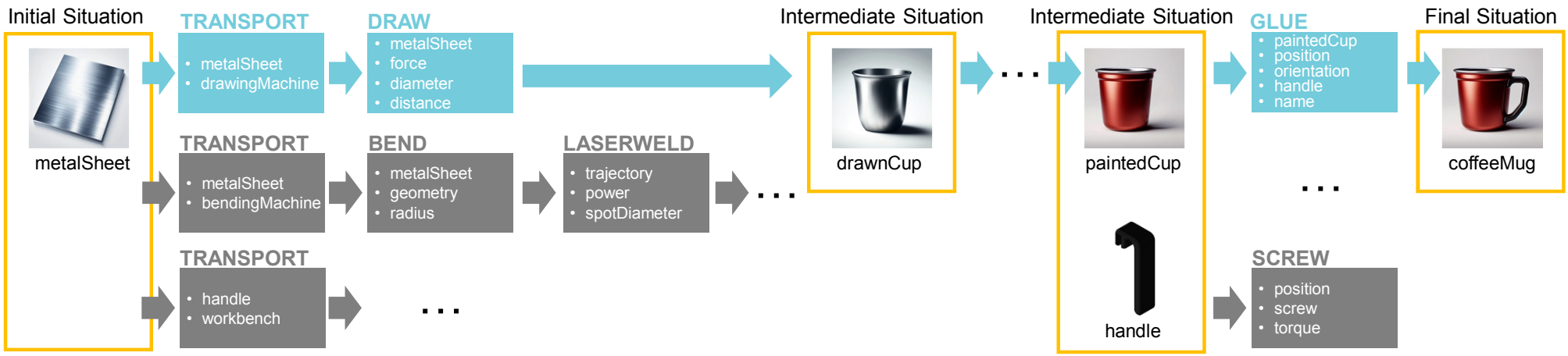
5.5. Operation

At runtime, the sequence of skill instances constituting the plan is processed by the `PlanRealizer`, which translates each skill instance into the evocation of an OPC UA service endpoint, thereby producing the technical action plan, i.e., a sequence of parameterized OPC UA calls. The `PlanExecutor` then iteratively invokes the technical action plan's OPC UA endpoints to start manufacturing. Before invoking each individual OPC UA endpoint, the `PlanExecutor` checks whether the corresponding skill instance's pre-conditions still hold. To this end, it the `PlanExecutor` uses the implementations of the domain model. In cases in which reality interferes with planning, the planning process needs to be restarted from the current state of the environment, e.g., if the skills executed in the real world change the workpiece, then the planner needs to restart with the changed workpiece as new initial situation.

6. Case Study

The following case study illustrates the approach, method, and capabilities of the proposed framework. To this end, the case study examines and evaluates various options for producing a coffee mug. The overall goal is to produce the red stainless-steel coffee mug as shown in the upper-right corner of Figure 13, which requires the following:

1. Access to the required resources, i.e., in this case, a stainless-steel sheet of sufficient size, and to the OPC UA devices required for the process;
2. A model of the relevant skill types, together with the domain model (including the initial situation, i.e., availability and properties of relevant resources);
3. A mapping of the skill types to the OPC UA devices; and
4. A problem statement of the final situation that logically represents the product to be produced.



Conceptual Representation

Logical Representation

Initial Situation (excerpt)

```
01 item(metalSheet)
02 at(metalSheet,storage
03 item(handle)
04 at(handle,storage)
05 dimensions(metalSheet, 300, 400)
06 machine(drawingMachine)
07 tool(stamp)
08 diameter(stamp, 80)
09 machine(glueingStation)
10 geometry(metalSheet, msGeo.cad)
11 geometry(drawnCup, hcGeo.cad)
12 // ext. computation of accessibility
13 accessible(Item1, Pos, Ori Item2)
14   = extern(DOMAIN.checkAccess(Item1,
15                               Pos,
16                               Ori,
17                               Item2)
```

Action TRANSPORT

```
01 action transport(Item, TargetPos)
02 {
03   pre:  at(Item, ItemPos)
04   post: !at(Item, ItemPos) && at(Item, TargetPos)
05 }
```

Action GLUE

```
01 action glue(Item1, Pos, Orientation, Item2, Name) {
02   pre:  at(Item1, glueingStation) &&
03         at(Item2, glueingStation) &&
04         accessibleWith(Item1, Pos, Orientation, Item2)
05   post: !item(Item1) && !item(Item2) &&
06         composedGeometry(Item1, Pos, Orientation, Item2, G)
07         item(Name) && geometry(Pos, G)
08         at(Name, XXX)
09 }
```

Action DRAW

```
01 action draw(Item, ItemGeometry,
02             TargetGeometry, Force,
03             Diameter, Distance)
04 {
05   pre:  at(Item, drawingMachine) &&
06         fixed(Item, drawingMachine) &&
07         diameter(stamp,Diameter) &&
08         // ...
09   post: at(Item, drawingMachine) &&
10         fixed(Item, drawingMachine)
11         geometry(Item, TargetGeometry)
12 }
```

Final Situation (excerpt)

```
01 item(coffeeMug)
02 machine(drawingMachine)
03 tool(stamp)
04 diameter(stamp, 80)
05 machine(glueingStation)
06 geometry(coffeeMug, ccGeo.cad)
07 …
```

Figure 13. Conceptual (**top**) and logical (**bottom**) representations of the case of producing a coffee mug.

For the example in question, this entails having modeled sufficient skill types to ensure that the coffee mug can be produced. Depending on the available machines, this might entail deep drawing, welding, cutting, or any other combination of skills enabling the transformation of the input metal sheet into a cup. Each skill type comes with logical and geometric pre-conditions and effects that describe the expected input and output, as well as a cost function. For the cutting skill, this might be as illustrated in Figure 5. The cost function can map to any numerical value; for example, it might represent material cost, energy cost, time cost, or a combination thereof. This needs to be realized in the respective domain model classes providing the cost function (cf. Figure 5, top right). Note that not all skills need to be executed fully automatically, e.g., the `transport` skill (Figure 13, top left), and others might be realized by issuing production tasks to human workers as well.

The desired coffee mug can be obtained through a variety of production sequences, each of which depends on the specific details of the corresponding skill types. In this case study, the `Transport`, `Draw`, `Bend`, `Laserweld` skill types, as well as a few more, are available. Within the conceptual representation of the sequences in Figure 13, they are represented by blue and gray rectangles also containing some sample parameters and functions. Each skill type refers to a class diagram domain model and geometric conditions. The bottom of Figure 5 depicts the corresponding PDDL representation, which includes excerpts of the initial and final situations and PDDL actions derived from the skill types.

Given the provided skill types, domain model, problem statement, and mapping to OPC UA, the planner evaluates the final situation. It then searches for skill types that could yield effects that would produce elements of the situation that are not already present in the initial situation (It should be noted that if the initial situation included the provision of a coffee mug with the requisite properties, no further action would be required). The planner's activities may entail the identification of skill types that color something, skill types that merge a cup and a handle, or any other skill types that contribute to the logical problem statement that constitutes the final situation. Given the cost function, the planner selects a suitable skill type while minimizing the overall cost and updates the final situation as if the selected skill instance had been applied. As a result, the logical problem statement is modified. For example, the planner may now assume that the workpiece has the correct color if the plan, which currently consists of a single skill instance, were to be applied. However, numerous additional pre-conditions would not be met in practice by the execution of the one-skill plan. Consequently, in the event that any other pre-conditions remain unfulfilled following the application of this plan fragment, the planner continues to identify skill types that could contribute to the updated problem statement. This process is repeated until one or more plans are identified that lead from the final situation back to the initial situation. Additionally, given the descriptive nature of skills and the domain model, in the event that the planner is unable to identify a solution, it can be queried about any missing steps.

Should the planner determine that the final step of mug finalization necessitates the attachment of a handle to the cup, a search for corresponding skills ensues. Given that this can be achieved by either gluing or screwing, at least two alternative paths emerge. The production of both the handle and the mug requires the use of different sequences of skill instances, ultimately leading back to the initial situation.

In this example, the planner determines that the optimal sequence of operations is to first transport the metal sheet to the deep drawing machine, then form a cup, which constitutes the base geometry. Subsequently, the cup is colored, and the handle is attached. The result of this planning activity is a sequence of skill instances with parameters that can now be sent to the PlanRealizer, which then uses the mapping to OPC UA devices or the hand-crafted action implementations to execute the plan in the real world.

7. Discussion and Summary

In its current state, the presented framework is already capable of automatically deriving production sequences from product data and, consequently, defining a manufacturing strategy. At this time, the final and initial states for the product must be known, and a set of suitable skills must be available. Without limiting the capacities of the executing framework itself, we expect the required skills to be created manually at this stage. This typical one-time effort may be performed by the machine vendor's domain experts, by the machine user, or by specialized third-party experts providing such services. These skills are the foundation for making implicit production knowledge explicit and reusable.

As underlined and demonstrated by the case study, with such skills available, the framework is capable of comparing different production processes and sequences automatically, simultaneously, and in close to real time. This is the basis for many fundamental targets of smart manufacturing, enabling simultaneous multicriteria optimization of the entire production system for its permanently moving targets, i.e., strategic and agile planning in volatile or uncertain situations such as spontaneous supply chain disruptions or availability issues. In this context, it should be emphasized that various aspects of the circular economy are very likely to significantly increase the need for such automated planning capacities or may not be feasible without such capacities at all. An example of this is the reuse of material with varying conditions as input for production. At this point, it is not (yet) our claim that our method is better than any possible domain expert on detailed questions; however,it can provide the ability to act immediately, transparently, more efficiently, and independently of this scarce resource. With the framework, planning processes become inherently explainable, since the framework itself has to optimize for a defined metric, i.e., soft or explicit targets. This eliminates unintended human bias for all production-related decisions ranging from design to manufacturing. Taken together, the framework fulfills the fundamental goal of addressing the ever-increasing shortage of skilled workers.

7.1. Realization Challenges

Our framework assumes that platform-independent skills can be ultimately mapped to a software interface to invoke a process on a machine (or interface with a human operator). Where is this is not supported, the resulting plans cannot be executed fully automatically.

We opted to use OPTIC [42,43] as the planner of choice in this framework, which allows time-dependent plans to be found under constraints. The PDDL constraints are soft goals in the sense that they do not have to be met, but they do have costs that must be minimized. This requires domain experts to make this trade-off between speed, quality, and cost explicit, which can be an organizational challenge.

In addition, the proposed conceptual framework assumes that both the initial state (in the domain model) and the intended goal state (in the problem statement) are also made explicit in the PDDL. The creation of these can be tedious and error-prone without specially trained experts. Instead, it may be desirable to derive them from CAD models of the initial state and intended goal state, as well as from models of the CPPSs. This synthesis requires significant future work [77].

Moreover, it may be that the planner cannot find suitable skill types or that all skills fail during execution. In cases where this is due to negligible differences between the planning models (domain and problem) and reality, automatically updating either to reflect these changes in reality to continue production would be beneficial. Realizing such knowledge updates is the subject to future work.

Although this work already represents a significant step towards software-defined and skill-based manufacturing, the idea of automatic generation and optimization without any further human intervention immediately comes to mind. However, the full vision of this would automatically lead to the need to extract logical and symbolic information (i.e., the skill types) from observed data. So far, there is no general method for this, but a first step towards this can be made by using various ontologies [78,79] as a basis for planning domains in which the skills act and translations from these to PDDL [80,81].

Our conceptual model uses UML class diagrams to describe the domain model. Of course, the domain model could be fully specified in terms of the underlying design technology, such as PDDL. This would raise at least two issues. (1) The platform-independent models would be tied to the chosen planning technology, and switching to a more promising technology that cannot handle PDDL would require not only changing of the transformation from UML to that particular technology space but also changing of the modeling language for the capabilities. (2) UML class diagrams are usually accessible to engineers from other technical domains [82], while their logical representation may be less accessible. The latter is also, to some extent, a challenge in creating skills, as it requires some limited understanding of Boolean logic.

Another challenge arises from the need to update plans in real-time. As none of the PDDL planners guarantees any real-time behavior, updating plans during a time-critical process requires further research regarding AI planning.

7.2. Strategic Opportunities

The framework offers further opportunities for industrial value creation. Given an early integration into the design process, it may provide a continuous virtual assessment of the producibility of the current design or even suggestions for better designs. In the long term, products and their geometries may be derived by a further enhanced production framework solely based on their functions and the target metrics. Another opportunity lies in the global comparison of production processes between industries, companies, and machines that perform similar manufacturing tasks. On the one hand, the best practices of processes and their combinations and sequences can be easily identified and continuously fed back into the framework. On the other hand, automatic and efficient allocation, segmentation, or recombination of production tasks by the framework can lead to considerable insights for machine design, process combinations, and integration.

However, the framework not only provides new approaches for the trend towards process integration and more versatile machinery in manufacturing, both of which especially target complexity [83–85]. The takeover of resource-consuming planning activities and the subsequent integration of processes and machines through connectivity by the framework also revives a familiar idea, namely the opportunity for renewed specialization and modularization of manufacturing (sub)systems, i.e., stations and machines. This could even lead to a consistent continuation of the paradigm of modularized *reconfigurable manufacturing systems* as proposed by Koren et al. [86].

In the future, new machines can be designed to easily integrate with the framework, allowing them to adapt to various production sequences and strategies. This can increase the versatility and longevity of the machines. The data and insights generated by the framework can be used to continuously improve machine design. This iterative process ensures that new machines are better suited to meet evolving production needs. By utilizing the framework's ability to optimize production processes in real time, machine designers can focus on creating machines that excel in a wider range of criteria, including efficiency, sustainability, and cost-effectiveness.

The initial creation of skills, which is a one-time effort, can be outsourced to specialized third-party experts. This enables companies to focus on their core operations while still benefiting from advanced production strategies. By automating the derivation of production sequences and manufacturing strategies, the framework reduces dependence on highly skilled workers for these tasks. This allows companies to operate efficiently with fewer specialized experts.

Author Contributions: Conceptualization, M.H. and A.W.; methodology, M.H. and A.W.; software, M.H. and A.W.; validation, M.H. and A.W.; writing, M.H. and A.W.; funding acquisition, M.H. and A.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been funded by the Ministry of Science, Research and Arts of the Federal State of Baden-Württemberg within the InnovationsCampus Future Mobility.

Data Availability Statement: The data will be shared upon request from the authors.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Kim, M.; Jeong, J.; Bae, S. Demand Forecasting Based on Machine Learning for Mass Customization in Smart Manufacturing. In Proceedings of the 2019 International Conference on Data Mining and Machine Learning, New York, NY, USA, 20–25 July 2019; pp. 6–11. [CrossRef]
2. Oh, J.; Jeong, B. Tactical supply planning in smart manufacturing supply chain. *Robot. Comput.-Integr. Manuf.* **2019**, *55*, 217–233. [CrossRef]
3. Choi, S.; Jung, K.; Kulvatunyou, B.; Morris, K.C. An Analysis of Technologies and Standards for Designing Smart Manufacturing Systems. *J. Res. Natl. Inst. Stand. Technol.* **2016**, *121*, 422–433. [CrossRef] [PubMed]
4. Serrano-Ruiz, J.C.; Mula, J.; Poler, R. Smart manufacturing scheduling: A literature review. *J. Manuf. Syst.* **2021**, *61*, 265–287. [CrossRef]
5. Qiao, F.; Liu, J.; Ma, Y. Industrial big-data-driven and CPS-based adaptive production scheduling for smart manufacturing. *Int. J. Prod. Res.* **2021**, *59*, 7139–7159. [CrossRef]
6. Scafà, M.; Marconi, M.; Germani, M. A critical review of symbiosis approaches in the context of Industry 4.0. *J. Comput. Des. Eng.* **2020**, *7*, 269–278. [CrossRef]
7. Rossmann, M.; Khadikar, A.; Le Franc, P.; Perea, L.; Schneider-Maul, R.; Buvat, J.; Ghosh, A. Smart Factories: How Can Manufacturers Realize the Potential of Digital Industrial Revolution. Available online: https://www.capgemini.com/wp-content/uploads/2017/07/smart_factories-how_can_manufacturers_realize_the_potential_of_digital_industrial_revolution.pdf (accessed on 23 September 2024).
8. Phuyal, S.; Bista, D.; Bista, R. Challenges, Opportunities and Future Directions of Smart Manufacturing: A State of Art Review. *Sustain. Futur.* **2020**, *2*, 100023. [CrossRef]
9. Tao, F.; Qi, Q.; Liu, A.; Kusiak, A. Data-driven smart manufacturing. *J. Manuf. Syst.* **2018**, *48*, 157–169. [CrossRef]
10. Wang, B.; Tao, F.; Fang, X.; Liu, C.; Liu, Y.; Freiheit, T. Smart Manufacturing and Intelligent Manufacturing: A Comparative Review. *Engineering* **2021**, *7*, 738–757. [CrossRef]
11. Son, Y.H.; Kim, G.Y.; Kim, H.C.; Jun, C.; Noh, S.D. Past, present, and future research of digital twin for smart manufacturing. *J. Comput. Des. Eng.* **2021**, *9*, 1–23. [CrossRef]
12. Qu, Y.J.; Ming, X.G.; Liu, Z.W.; Zhang, X.Y.; Hou, Z.T. Smart manufacturing systems: State of the art and future trends. *Int. J. Adv. Manuf. Technol.* **2019**, *103*, 3751–3768. [CrossRef]
13. Yang, H.; Kumara, S.; Bukkapatnam, S.T.; Tsung, F. The internet of things for smart manufacturing: A review. *IISE Trans.* **2019**, *51*, 1190–1216. [CrossRef]
14. Matt, D.T.; Orzes, G.; Rauch, E.; Dallasega, P. Urban production—A socially sustainable factory concept to overcome shortcomings of qualified workers in smart SMEs. *Comput. Ind. Eng.* **2020**, *139*, 105384. [CrossRef]
15. Agrawal, M.; Eloot, K.; Mancini, M.; Patel, A. *Industry 4.0: Reimagining Manufacturing Operations after COVID-19*; McKinsey & Company: New York, NY, USA, 2020; pp. 1–11.
16. Burbidge, J.L. Production control: A universal conceptual framework. *Prod. Plan. Control* **1990**, *1*, 3–16. [CrossRef]
17. Norman, B.A.; Tharmmaphornphilas, W.; Needy, K.L.; Bidanda, B.; Warner, R.C. Worker assignment in cellular manufacturing considering technical and human skills. *Int. J. Prod. Res.* **2002**, *40*, 1479–1492. [CrossRef]
18. Pochet, Y.; Wolsey, L.A. *Production Planning by Mixed Integer Programming*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006.
19. Reiff, C.; Buser, M.; Betten, T.; Onuseit, V.; Hoßfeld, M.; Wehner, D.; Riedel, O. A Process-Planning Framework for Sustainable Manufacturing. *Energies* **2021**, *14*, 5811. [CrossRef]
20. Martins, A.; Costelha, H.; Neves, C.; Cosgrove, J.; Lyons, J.G. Flexible Manufacturing Systems Through the Integration of Asset Administration Shells, Skill-Based Manufacturing, and OPC UA. In *Proceedings of the International Conference on Flexible Automation and Intelligent Manufacturing*; Springer: Berlin/Heidelberg, Germany, 2023, pp. 477–485.
21. France, R.; Rumpe, B. Model-driven Development of Complex Software: A Research Roadmap. In Proceedings of the Future of Software Engineering (FOSE '07), Minneapolis, MN, USA, 23–25 May 2007; pp. 37–54.
22. Vogel-Heuser, B.; Fay, A.; Schaefer, I.; Tichy, M. Evolution of software in automated production systems: Challenges and research directions. *J. Syst. Softw.* **2015**, *110*, 54–84. [CrossRef]
23. Selic, B. The pragmatics of model-driven development. *IEEE Softw.* **2003**, *20*, 19–25. [CrossRef]
24. Liebel, G.; Marko, N.; Tichy, M.; Leitner, A.; Hansson, J. Model-based engineering in the embedded systems domain: An industrial survey on the state-of-practice. *Softw. Syst. Model.* **2018**, *17*, 91–113. [CrossRef]
25. Hölldobler, K.; Rumpe, B.; Wortmann, A. Software Language Engineering in the Large: Towards Composing and Deriving Languages. *Comput. Lang. Syst. Struct.* **2018**, *54*, 386–405. [CrossRef]
26. Whittle, J.; Hutchinson, J.; Rouncefield, M. The State of Practice in Model-Driven Engineering. *IEEE Softw.* **2014**, *31*, 79–85. [CrossRef]

27. Bruel, J.M.; Combemale, B.; Ober, I.; Raynal, H. MDE in Practice for Computational Science. In Proceedings of the International Conference on Computational Science (ICCS 2015), Reykjavík, Iceland, 1–3 June 2015.
28. Wortmann, A.; Barais, O.; Combemale, B.; Wimmer, M. Modeling Languages in Industry 4.0: An Extended Systematic Mapping Study. *Softw. Syst. Model.* **2020**, *19*, 67–94. [CrossRef]
29. Tantik, E.; Anderl, R. Integrated data model and structure for the asset administration shell in Industrie 4.0. *Procedia Cirp* **2017**, *60*, 86–91. [CrossRef]
30. Lüder, A.; Schmidt, N. AutomationML in a Nutshell. In *Handbuch Industrie 4.0 Bd. 2*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 213–258.
31. Dabney, J.B.; Harman, T.L. *Mastering Simulink*; Pearson: London, UK, 2004.
32. Leitner, S.H.; Mahnke, W. OPC UA–service-oriented architecture for industrial applications. *ABB Corp. Res. Cent.* **2006**, *48*, 22.
33. Friedenthal, S.; Moore, A.; Steiner, R. *A Practical Guide to SysML: The Systems Modeling Language*; Morgan Kaufmann: Burlington, MA, USA, 2014.
34. Völkel, S. *Kompositionale Entwicklung Domänenspezifischer Sprachen*; Aachener Informatik-Berichte, Software Engineering, Band 9; Shaker Verlag: Herzogenrath, Germany, 2011.
35. Völter, M.; Stahl, T.; Bettin, J.; Haase, A.; Helsen, S.; Czarnecki, K. *Model-Driven Software Development: Technology, Engineering, Management*; Wiley Software Patterns Series, Wiley: Hoboken, NJ, USA, 2013.
36. Object Management Group. MDA Guide Version 1.0.1, 2003. Available online: https://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf (accessed on 22 September 2024).
37. Soley, R. Model driven architecture. *OMG White Pap.* **2000**, *308*, 5.
38. Thramboulidis, K. IEC 61499 in factory automation. In *Advances in Computer, Information, and Systems Sciences, and Engineering: Proceedings of IETA 2005, TeNe 2005, EIAE 2005*; Springer: Dordrecht, The Netherlands, 2006; pp. 115–124.
39. Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Pearson: London, UK, 2016.
40. Ghallab, M.; Nau, D.; Traverso, P. *Automated Planning: Theory and Practice*; Elsevier: Amsterdam, The Netherlands, 2004.
41. Fox, M.; Long, D. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.* **2003**, *20*, 61–124. [CrossRef]
42. Benton, J.; Coles, A.; Coles, A. Temporal planning with preferences and time-dependent continuous costs. In Proceedings of the International Conference on Automated Planning and Scheduling, Sao Paulo, Brazil, 25–29 June 2012; Volume 22, pp. 2–10.
43. Tierney, K.; Coles, A.; Coles, A.; Kroer, C.; Britt, A.; Jensen, R. Automated planning for liner shipping fleet repositioning. In Proceedings of the International Conference on Automated Planning and Scheduling, Sao Paulo, Brazil, 25–29 June 2012; Volume 22, pp. 279–287.
44. Brödner, P. Skill based manufacturing vs."unmanned factory"—which is superior? *Int. J. Ind. Ergon.* **1986**, *1*, 145–153. [CrossRef]
45. Dripke, C.; Schneider, B.; Dragan, M.; Zoitl, A.; Verl, A. Concept of Distributed Interpolation for Skill-Based Manufacturing with Real-Time Communication. In *Tagungsband des 3. Kongresses Montage Handhabung Industrieroboter*; Springer: Berlin/Heidelberg, Germany, 2018, pp. 215–222.
46. Froschauer, R.; Köcher, A.; Meixner, K.; Schmitt, S.; Spitzer, F. Capabilities and skills in manufacturing: A survey over the last decade of etfa. *arXiv* **2022**, arXiv:2204.12908.
47. Meixner, K.; Lüder, A.; Herzog, J.; Röpke, H.; Biffl, S. Modeling expert knowledge for optimal CPPS resource selection for a product portfolio. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1687–1694.
48. Pfrommer, J.; Schleipen, M.; Beyerer, J. PPRS: Production skills and their relation to product, process, and resource. In Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013; pp. 1–4.
49. Pfrommer, J.; Stogl, D.; Aleksandrov, K.; Schubert, V.; Hein, B. Modelling and orchestration of service-based manufacturing systems via skills. In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 16–19 September 2014; pp. 1–4.
50. Köcher, A.; Hildebrandt, C.; da Silva, L.M.V.; Fay, A. A formal capability and skill model for use in plug and produce scenarios. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1663–1670.
51. Gocev, I.; Grimm, S.; Runkler, T. Supporting skill-based flexible manufacturing with symbolic AI methods. In Proceedings of the IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society, Singapore, 18–21 October 2020; pp. 769–774.
52. Wally, B.; Vyskočil, J.; Novák, P.; Huemer, C.; Šindelár, R.; Kadera, P.; Mazak, A.; Wimmer, M. Flexible production systems: Automated generation of operations plans based on ISA-95 and PDDL. *IEEE Robot. Autom. Lett.* **2019**, *4*, 4062–4069. [CrossRef]
53. Wally, B.; Vyskočil, J.; Novák, P.; Huemer, C.; Šindelář, R.; Kadera, P.; Mazak, A.; Wimmer, M. Production Planning with IEC 62264 and PDDL. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; Volume 1, pp. 492–499.
54. Mayr-Dorn, C.; Egyed, A.; Winterer, M.; Salomon, C.; Fürschuß, H. Evaluating PDDL for programming production cells: A case study. In Proceedings of the 4th International Workshop on Robotics Software Engineering, Pittsburgh, PA, USA, 9 May 2022; pp. 17–24.

55. Rovida, F.; Crosby, M.; Holz, D.; Polydoros, A.S.; Großmann, B.; Petrick, R.P.; Krüger, V. SkiROS—A skill-based robot control platform on top of ROS. In *Robot Operating System (ROS) The Complete Reference (Volume 2)*; Springer: Cham, Switzerland, 2017; pp. 121–160.
56. Thomas, U.; Hirzinger, G.; Rumpe, B.; Schulze, C.; Wortmann, A. A new skill based robot programming language using uml/p statecharts. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 461–466.
57. Sonnleithner, L.; Wiesmayr, B.; Ashiwal, V.; Zoitl, A. IEC 61499 distributed design patterns. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–8.
58. Köcher, A.; Da Silva, L.M.V.; Fay, A. Modeling and executing production processes with capabilities and skills using ontologies and BPMN. In Proceedings of the 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), Stuttgart, Germany, 6–9 September 2022; pp. 1–8.
59. Spitzer, F.; Lindorfer, R.; Froschauer, R.; Hofmann, M.; Ikeda, M. A generic approach for the industrial application of skill-based engineering using OPC UA. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1671–1678.
60. Järvenpää, E.; Siltala, N.; Hylli, O.; Lanz, M. Capability matchmaking software for rapid production system design and reconfiguration planning. *Procedia CIRP* **2021**, *97*, 435–440. [CrossRef]
61. Köcher, A.; Hildebrandt, C.; Caesar, B.; Bakakeu, J.; Peschke, J.; Scholz, A.; Fay, A. Automating the development of machine skills and their semantic description. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1013–1018.
62. Schwarz, M.H.; Börcsök, J. A survey on OPC and OPC-UA: About the standard, developments and investigations. In Proceedings of the 2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT), Sarajevo, Bosnia and Herzegovina, 30 October–1 November 2013; pp. 1–6.
63. Zimmermann, P.; Axmann, E.; Brandenbourger, B.; Dorofeev, K.; Mankowski, A.; Zanini, P. Skill-based engineering and control on field-device-level with opc ua. In Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 10–13 September 2019; pp. 1101–1108.
64. Volkmann, M.; Legler, T.; Wagner, A.; Ruskowski, M. A CAD feature-based manufacturing approach with OPC UA skills. *Procedia Manuf.* **2020**, *51*, 416–423. [CrossRef]
65. Liu, C.; Vengayil, H.; Lu, Y.; Xu, X. A cyber-physical machine tools platform using OPC UA and MTConnect. *J. Manuf. Syst.* **2019**, *51*, 61–74. [CrossRef]
66. Zimmermann, P.; Gerber, K.; Seyler, J.R. A Concept for Selecting Suitable Resources in Automated Assembly Systems. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–8.
67. Lee, K.; Joo, S.; Christensen, H.I. An assembly sequence generation of a product family for robot programming. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Republic of Korea, 9–14 October 2016; pp. 1268–1274.
68. Rumpe, B. *Modeling with UML: Language, Concepts, Methods*; Springer International: Berlin/Heidelberg, Germany, 2016.
69. Tarkian, M. Design Automation for Multidisciplinary Optimization: A High Level CAD Template Approach. Ph.D. Thesis, Linköping University Electronic Press, Linköping, Sweden, 2012.
70. Amadori, K.; Tarkian, M.; Ölvander, J.; Krus, P. Flexible and robust CAD models for design automation. *Adv. Eng. Inform.* **2012**, *26*, 180–195. [CrossRef]
71. Wu, Y.; Zhou, Y.; Zhou, Z.; Tang, J.; Ouyang, H. An advanced CAD/CAE integration method for the generative design of face gears. *Adv. Eng. Softw.* **2018**, *126*, 90–99. [CrossRef]
72. Hoffmann, J. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *J. Artif. Intell. Res.* **2003**, *20*, 291–341. [CrossRef]
73. Mierlo, S.V.; Oakes, B.J.; Acker, B.V.; Eslampanah, R.; Denil, J.; Vangheluwe, H. Exploring validity frames in practice. In *Proceedings of the International Conference on Systems Modelling and Management*; Springer: Cham, Switzerland, 2020; pp. 131–148.
74. Dietrich, D.; Neubauer, M.; Lechler, A.; Verl, A. Iterative Planning as a Holistic Framework for Production System-Wide Optimization Control Loops. In *Proceedings of the International Conference on Flexible Automation and Intelligent Manufacturing, Porto, Portugal, 18–22 June 2023*; Springer: Cham, Switzerland, 2023; pp. 611–621.
75. Vaquero, T.S.; Silva, J.R.; Tonidandel, F.; Beck, J.C. itSIMPLE: Towards an integrated design system for real planning applications. *Knowl. Eng. Rev.* **2013**, *28*, 215–230. [CrossRef]
76. Greifenberg, T.; Hölldobler, K.; Kolassa, C.; Look, M.; Mir Seyed Nazari, P.; Müller, K.; Navarro Perez, A.; Plotnikov, D.; Reiß, D.; Roth, A.; et al. Integration of Handwritten and Generated Object-Oriented Code. In *Model-Driven Engineering and Software Development*; Communications in Computer and Information Science; Springer: Cham, Switzerland, 2015; Volume 580, pp. 112–132.
77. Fritz, C. Automated process planning for CNC machining. *AI Mag.* **2016**, *37*, 116–117. [CrossRef]

78. Horsch, M.T.; Toti, D.; Chiacchiera, S.; Seaton, M.A.; Goldbeck, G.; Todorov, I.T. OSMO: Ontology for simulation, modelling, and optimization. In Proceedings of the 12th International Conference on Formal Ontology in Information Systems (FOIS 2021): Ontology Showcase, Bolzano, Italy, 16 September 2021.
79. Horsch, M.T.; Chiacchiera, S.; Seaton, M.A.; Todorov, I.T.; Šindelka, K.; Lísal, M.; Andreon, B.; Bayro Kaiser, E.; Mogni, G.; Goldbeck, G.; et al. Ontologies for the virtual materials marketplace. *KI-Künstliche Intell.* **2020**, *34*, 423–428. [CrossRef]
80. Louadah, H.; Papadakis, E.; Mccluskey, T.L.; Tucker, G.; Hughes, P.; Bevan, A. Translating ontological knowledge to PDDL to do Planning in Train Depot Management Operations. In *Proceedings of the 36th Workshop of the UK Planning and Scheduling Special Interest Group*; AAAI Press: Cambridge, MA, USA, 2021.
81. Gocev, I.; Grimm, S.; Runkler, T.A. Explanation of action plans through ontologies. In *On the Move to Meaningful Internet Systems. OTM 2018 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22–26. 2018, Proceedings, Part II*; Springer: Cham, Switzerland, 2018; pp. 386–403.
82. Vogel-Heuser, B.; Land, K.; Bi, F. Challenges for students of mechanical engineering using UML-typical questions and faults. In Proceedings of the 2020 6th IEEE Congress on Information Science and Technology (CiSt), Agadir-Essaouira, Morocco, 5–12 June 2021; pp. 261–266.
83. Hossfeld, M. Time-dependency of mechanical properties and component behavior after friction stir welding. *Int. J. Adv. Manuf. Technol.* **2019**, *102*, 2297—2305. [CrossRef]
84. Hossfeld, M. Shoulderless Friction Stir Welding: A low-force solid state keyhole joining technique for deep welding of labile structures. *Prod. Eng.* **2022**, *16*, 389–399. [CrossRef]
85. Graf, T.; Hoßfeld, M.; Onuseit, V. A universal machine: Enabling digital manufacturing with laser technology. In *Advances in Automotive Production Technology–Theory and Application: Stuttgart Conference on Automotive Production (SCAP2020)*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 386–393.
86. Koren, Y.; Heisel, U.; Jovane, F.; Moriwaki, T.; Pritschow, G.; Ulsoy, G.; Van Brussel, H. Reconfigurable Manufacturing Systems. *CIRP Ann.* **1999**, *48*, 527–540. [CrossRef]