Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Converting semantic web services into formal planning domain descriptions to enable manufacturing process planning and scheduling in industry 4.0

Lukas Malburg [*], Patrick Klein, Ralph Bergmann

Artificial Intelligence and Intelligent Information Systems, University of Trier, Universitätsring 15, Trier, 54296, Germany
German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, Universitätsring 15, Trier, 54296, Germany

ARTICLE INFO

Keywords:
Semantic web services
Industry 4.0
Automated planning
Planning domain definition language
Cyber-physical workflows

ABSTRACT

To build intelligent manufacturing systems that react flexibly in case of failures or unexpected circumstances, manufacturing capabilities of production systems must be utilized as much as possible. Artificial Intelligence (AI) and, in particular, automated planning can contribute to this by enabling flexible production processes. To efficiently leverage automated planning, an almost complete planning domain description of the real-world is necessary. However, creating such planning descriptions is a demanding and error-prone task that requires high manual efforts even for domain experts. In addition, maintaining the encoded knowledge is laborious and, thus, can lead to outdated domain descriptions. To reduce the high efforts, already existing knowledge can be reused and transformed automatically into planning descriptions to benefit from organization-wide knowledge engineering activities. This paper presents a novel approach that reduces the described efforts by reusing existing knowledge for planning and scheduling in Industry 4.0 (I4.0). For this purpose, requirements for developing a converter that transforms existing knowledge are derived from literature. Based on these requirements, the *SWS2PDDL* converter is developed that transforms the knowledge into formal Planning Domain Definition Language (PDDL) descriptions. The approach's usefulness is verified by a practical evaluation with a near real-world application scenario by generating failures in a physical smart factory and evaluating the generated re-planned production processes. When comparing the resulting plan quality to those achieved by using a manually modeled planning domain by a domain expert, the automatic transformation by *SWS2PDDL* leads to comparable or even better results without requiring the otherwise high manual modeling efforts.

1. Introduction

The ongoing transformation and shift towards more autonomous and intelligent manufacturing within the context of I4.0 is crucial for enabling individual mass production and supporting cloud manufacturing (Lasi et al., 2014; Rüßmann et al., 2015; Kagermann and Wahlster, 2022). To achieve this, smart manufacturing systems that are easily configurable and allow a high degree of flexibility during production are needed (Lasi et al., 2014; Rüßmann et al., 2015; Cheng et al., 2017; Bergweiler, 2016). For this purpose, the use of AI methods in Cyber-Physical Production Systems (CPPSs) (Monostori, 2014) is inevitable (Lee et al., 2014; Monostori, 2014), but still in its infancy. For example, current production lines often operate isolated and have only limited capabilities to react to dynamic changes in the environment, i. e., when processes cannot be executed as previously planned and, thus, have to be adapted (Malburg et al., 2020a; Malburg and Bergmann, 2022; Malburg et al., 2023b,a). To remedy this situation, constant re-planning and scheduling to optimize the production and a closer coupling of the shop floor with high-level systems for decision support in near real-time are desirable (Rüßmann et al., 2015; Malburg et al., 2020a; Rossit et al., 2019b,a; Seiger et al., 2022).

For using and applying AI methods like automated planning (Haslum et al., 2019; Ghallab, 2004; Ghallab et al., 2016; Fox and Long, 2003) efficiently, a formal and complete planning domain description of the real world is required. The PDDL (McDermott et al., 1998) is the de facto standard for expressing planning models for state-of-the-art planners (Haslum et al., 2019). However, creating PDDL planning domain descriptions is a demanding and error-prone task that requires high manual modeling efforts even for domain experts (Nguyen et al., 2017; Zhuo et al., 2013; Jilani, 2020; McCluskey et al., 2009). Even if complete planning domain descriptions are available in real-world application scenarios, maintaining the stored knowledge encoded in plain PDDL files is tedious, resulting in additional maintenance efforts to keep knowledge representations and planning domain descriptions consistent (McCluskey et al., 2021; Wickler et al., 2015). This is especially the case in manufacturing environments, where numerous machine

* Corresponding author at: German Research Center for Artificial Intelligence (DFKI), Branch University of Trier, Universitätsring 15, Trier, 54296, Germany.
E-mail addresses: malburgl@uni-trier.de (L. Malburg), kleinp@uni-trier.de (P. Klein), bergmann@uni-trier.de (R. Bergmann).

https://doi.org/10.1016/j.engappai.2023.106727
Received 7 August 2022; Received in revised form 31 March 2023; Accepted 25 June 2023
Available online xxxx

parameters and mutual relationships between machine functionalities must be considered, and factory configurations may change regularly. To remedy the high manual modeling and acquisition efforts, several methods and tools have been proposed for automated domain model learning (Jilani, 2020). These methods, however, partly only derive incomplete planning models and do not address the maintenance of encoded knowledge. In contrast to these automated learning methods, it can be relied on already available knowledge, e. g., production-specific knowledge in the form of ontologies (Lemaignan et al., 2006; Mazzola et al., 2016), as a single source of truth (Sesboüé et al., 2022) and a shared semantic model, e. g., as required to develop digital twins (Boschert and Rosen, 2016). The advantage of using one shared semantic model is the utilization of established knowledge engineering activities of the company (Wickler et al., 2015) to ensure that the model is up-to-date and well-modeled. In addition, there is only a single initial effort in defining the transformation from the already used knowledge representation into a PDDL model required, in contrast to building and constantly updating and maintaining an isolated PDDL planning model to obtain useful and high-quality planning results.

The goal of this paper is to present a new approach to automatically convert already available knowledge into a formal PDDL planning domain description to reduce the described efforts and to use this knowledge for manufacturing process planning and scheduling in I4.0. For this purpose, (1) we derive requirements that should be satisfied by a converter and, in general, for using AI planning in Business Process Management (BPM) based on literature research, (2) we present a *SWS2PDDL* converter that transforms the already encoded knowledge in Semantic Web Services (SWSs) and ontologies to generate PDDL planning domain descriptions by supporting several PDDL language levels, and (3) we show in a practical evaluation with a near real-world application scenario how the automatically generated domain description can be applied for production planning and scheduling in I4.0. To demonstrate the suitability of the proposed approach in an experimental evaluation, we use a physical Fischertechnik (FT) smart factory for process-based research in I4.0 (Malburg et al., 2020a; Malburg and Bergmann, 2022; Malburg et al., 2023a; Seiger et al., 2022; Malburg et al., 2020c; Klein et al., 2019; Malburg et al., 2021; Klein et al., 2021; Malburg et al., 2023b). Using this physical smart factory for demonstration and evaluation of the approach strengthens the results and enables the validation of the approach in a scenario that is closer to real-world environments.

The paper is structured following the Design Science Research (DSR) methodology for information systems research proposed by Hevner et al. (2004). First, Section 2 describes the foundations that compromise the physical FT smart factory used for research, the use case of BPM and automated planning in smart manufacturing. In addition, we derive requirements based on literature research that are important for developing the research artifact. In Section 3, related approaches are presented and compared w. r. t. the derived requirements. Based on this knowledge base (*Rigor–Knowledge Base* in Hevner et al. (2004)), we can ensure that the developed research artifact consisting of a *SWS2PDDL* converter for transforming a semantic model into a formal PDDL planning domain is rigorously constructed (*Develop/Build* in Hevner et al. (2004)). To justify the utility and relevance of the artifact for the addressed problems, an experimental evaluation is conducted by using state-of-the-art planning frameworks and comparing manually modeled planning domain descriptions and the ones generated by *SWS2PDDL*. To compare the created planning domains, we use quantitative metrics such as the number of actions, the average number of parameters, or the average number of preconditions and effects. In addition, we compare the results of planning by measuring the time needed for planning and the time needed for executing the generated plans (*Justify/Evaluate* in Hevner et al. (2004)). By applying the research artifact to a real-world application scenario during the practical evaluation, it is shown that the developed artifact is useful for real environments (*Relevance–Environment* in Hevner et al. (2004)). Finally, we summarize the contributions and limitations of the proposed approach in Section 6 as well as a conclusion is given, and future work is discussed in Section 7.

2. Foundations and requirements

Since it is often difficult to conduct research with real production lines due to safety concerns and industry secrets, *Learning Factories* (Abele et al., 2017) for simulating real production environments can be used. The advantages of using such learning factories are that they enable the development and evaluation of research artifacts in a closed, protected environment but at much lower prices before transferring to real-world production (Malburg et al., 2020a). In Section 2.1, the used Fischertechnik (FT) physical smart factory for emulating real production environments is introduced. Afterwards, a Business Process Management (BPM) abstraction stack to conduct BPM related research in this context is presented in Section 2.2. The BPM abstraction stack is implemented as a service-based architecture that abstracts low-level control commands as services that can be used in higher-level systems (Seiger et al., 2022; Malburg et al., 2020c). This enables the use of BPM in smart manufacturing to control the smart factory and in turn to react to events. In addition, we use semantics to describe the capabilities of the smart factory by using a domain ontology (Klein et al., 2019). Automated planning and scheduling techniques can be used in I4.0 to re-plan the current production in case of failures. For this purpose, the foundations of automated planning and scheduling are presented in Section 2.3. In relation to the DSR (Hevner et al., 2004) methodology, the foundations represent the knowledge base consisting of all relevant theories, methods, and models. The knowledge base is used to ensure research rigor of the proposed research artifact.

2.1. Industry 4.0 physical smart factory model

For conducting applied research in the field of I4.0, small-scale physical factories are often used for the evaluation and demonstration of feasibility. This approach is useful as a first step because a real production cannot simply be interrupted for testing purposes (Polge et al., 2020) as well as there is a high potential of possible issues with major consequences, e. g., repair costs due to damages. In contrast, the models used for this purpose are called *Learning Factories* (Abele et al., 2017) since their primary purpose is to gain knowledge and estimate the potential usefulness, e. g., through the evaluation of novel artifacts, before implementing them in a complex real-world manufacturing environment. For instance, small-scale physical factories are used for AI research in the context of I4.0 for multi-agent production systems (Calà et al., 2016), predictive maintenance (Klein and Bergmann, 2019), and BPM in smart manufacturing (Malburg et al., 2020a; Seiger et al., 2022; Malburg and Bergmann, 2022; Malburg et al., 2023a,b; Kirikkayis et al., 2023).

For our research, we use a physical smart factory that consists of two similar shop floors connected for the exchange of workpieces, as shown in Fig. 1. There are four workstations on each shop floor with six identical machines: a Sorting Machine (SM) with color recognition, a multiprocessing workstation with Oven (OV), a Milling Machine (MM) and a Workstation Transport (WT) connecting the two, a High-Bay Warehouse (HBW) and a Vacuum Gripper Robot (VGR). In addition, there are individual machines on each shop floor, i. e., a Punching Machine (PM) and a Human Workstation (HW) on the first shop floor and a Drilling Machine (DM) on the second one. For control purposes, there are several light barriers, switches, and capacitive sensors on each shop floor. Moreover, the first shop floor is extended with dedicated sensors such as acceleration, differential pressure, and absolute orientation sensors. There are RFID readers/writers integrated into workstations on both shop floors and in the high-bay warehouses, creating 28 communication points. This enables tracking of each workpiece and retrieving the required manufacturing operations and parameters, which can be modified during production if necessary. Further, a camera is positioned above the two shop floors for detecting and

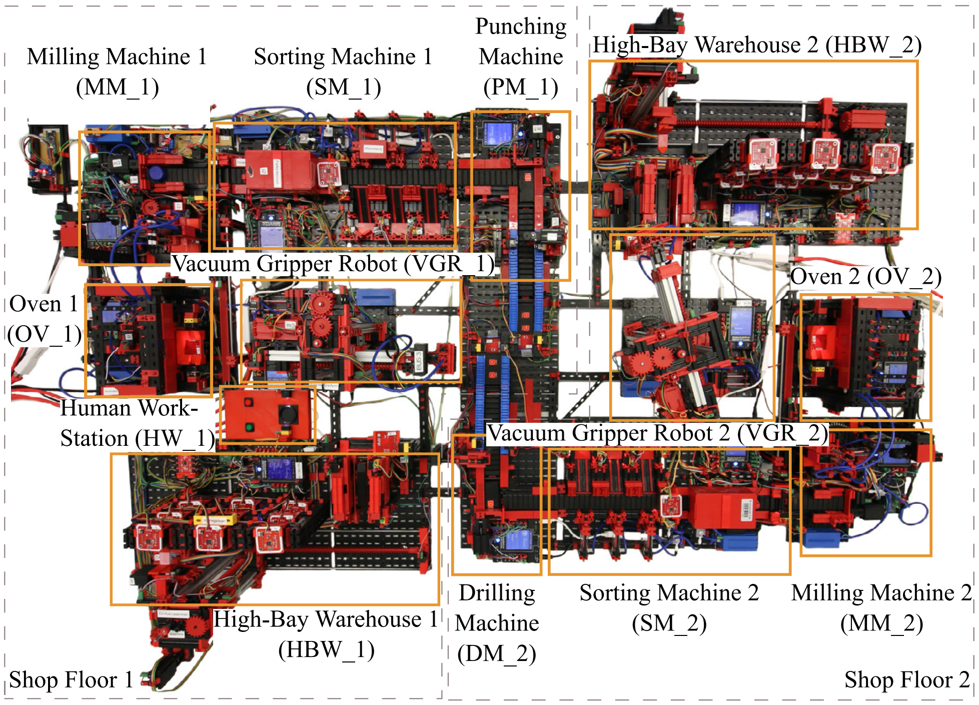

Fig. 1. The physical Fischertechnik smart factory.
Source: Malburg et al. (2020a).

tracking the workpieces during production (Malburg et al., 2021).[1] A video of the FT smart factory executing a manufacturing process and tracking workpieces by machine learning can be found in Malburg et al. (2020b).

2.2. Business process management for smart manufacturing

The combination and integration of BPM with the Internet of Things (IoT) is beneficial for both sides, since real-time data can be used in Workflow Management Systems (WfMSs) to control processes in an event-driven way and the WfMS can in turn trigger actuation in the IoT environment (Janiesch et al., 2020). To benefit from BPM solutions in smart IoT environments, the functionalities of actuators and sensors must be encapsulated to be available in a coarse-grained manner at a higher level (Malburg et al., 2020c,a; Seiger et al., 2022). In this section, we present a BPM abstraction stack based on previous work (Malburg et al., 2020a; Seiger et al., 2022; Malburg et al., 2020c) to explain how processes in smart factories can be controlled by WfMSs. In addition, we present how knowledge for the physical smart factory is represented (Klein et al., 2019; Malburg et al., 2020c). Fig. 2 depicts the abstraction stack with five individual layers and the overall *Semantics* that are presented in the following (more information about the individual layers and their implementation for the smart factory can be found in Seiger et al. (2022)).

2.2.1. Hardware layer

The *Hardware Layer* contains the individual actuators and sensors of the shop floor. Typically, the components of this layer are executing low-level and hard-wired programs, programmed in proprietary and specialized languages (e. g., G-code or C-code) with rigid and permanent routines. Interoperability between the components is limited so that the components mainly work isolated (Malburg et al., 2020c,a; Seiger et al., 2022).

[1] More information about the FT smart factory can be found at https://iot.uni-trier.de.

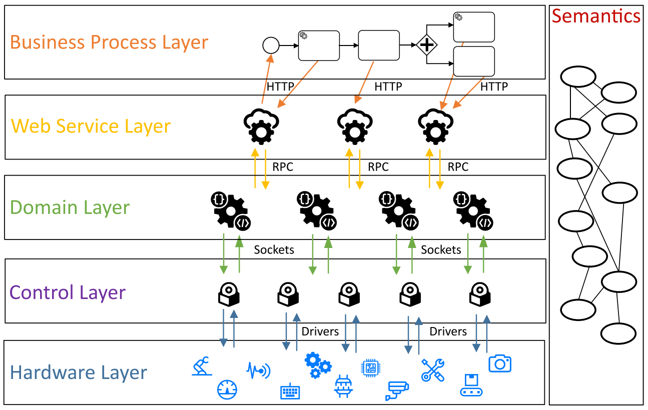

Fig. 2. Business process management abstraction layer with semantics.
Source: Based on Malburg et al. (2020a).

2.2.2. Control layer

To access the functionality of the *Hardware Layer*, very often proprietary protocols and drivers (Monostori, 2014) are used by the *Control Layer*. In industry, the *Control Layer* is often implemented as embedded control application (e. g., *Programmable Logic Controllers, PLC*) that is closely related to the corresponding hardware and especially tailored. In the smart factory model, the commands that are provided by the embedded control application from FT are, for example, methods to start or stop motors or to obtain measured values of a certain sensor (Malburg et al., 2020c,a; Seiger et al., 2022).

2.2.3. Domain layer

The rather fine-grained methods of the *Control Layer* are further abstracted as coarse-grained methods by using an object-oriented programming language for the smart factory in the *Domain Layer*. The methods from the *Domain Layer* provide the functionalities of the smart factory such as *burn*, *milling*, or *drilling*. Each of these functionalities

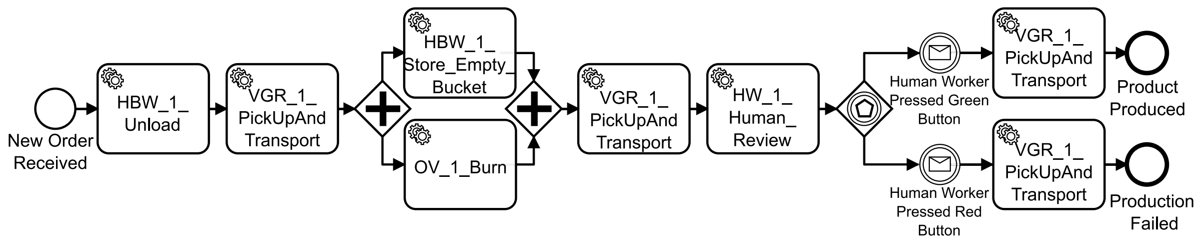
New Order Received
HBW_1_ Unload
VGR_1_ PickUpAnd Transport
HBW_1_ Store_Empty_ Bucket
OV_1_Burn
VGR_1_ PickUpAnd Transport
HW_1_ Human_ Review
Human Worker Pressed Green Button
Human Worker Pressed Red Button
VGR_1_ PickUpAnd Transport
VGR_1_ PickUpAnd Transport
Product Produced
Production Failed

Fig. 3. Sheet metal manufacturing process as BPMN 2.0 model.
Source: Malburg et al. (2023b).

consists of several smaller actions that are composed together (Malburg et al., 2020c,a; Seiger et al., 2022). An example of which actions the *burn* functionality is composed of can be found in Seiger et al. (2022).

2.2.4. *Web service layer*

The *Web Service Layer* contains RESTful web services that are built one-to-one from the provided methods of the *Domain Layer*, i. e., one functionality of the *Domain Layer* such as burn results in one corresponding web service (Malburg et al., 2020c,a; Seiger et al., 2022). In addition, the web services are semantically enriched (see Section 2.2.6).

2.2.5. *Business process layer*

At the top of the used abstraction stack, manufacturing processes can be created and executed in the FT smart factory (see Section 2.1). In our work, we use the Camunda[2] WfMS to execute Business Process Model and Notation (BPMN)[3] 2.0 processes. Fig. 3 shows a sheet metal manufacturing process modeled with BPMN 2.0 service tasks. After an order is received, an unprocessed steel slab is unloaded from the HBW. Afterwards, it is transported to the oven for burning. At the same time, the empty bucket in which the unprocessed steel slab was contained in is in turn stored in the HBW. After the burn task, the workpiece is transported to the review station and manually controlled by a human. If the workpiece is properly burned, the human worker confirms this by pressing the green button and, thus, the production of the workpiece is finished. In the other case, the human worker determines that the quality is not sufficient and indicates this by pressing the red button. This causes the workpiece to be discarded. In both cases, the produced sensor data of the factory is monitored, and a message is sent to the Camunda WfMS which button was pressed by the human worker at the human workstation. To be able to recognize such events during process execution, a Complex Event Processing (CEP) engine processes the sensor data from the smart factory by using the web services of the service layer and derives higher level events for process execution in the WfMS. In addition to the CEP engine, also a *Planning Component* could be part of the business process layer to adapt processes according to exceptions and the current state of the smart factory (Malburg and Bergmann, 2022; Malburg et al., 2023a,b). In Malburg and Bergmann (2022), we present an architectural framework for adaptive workflow management that describes which components could be used and how they interact in the business process layer.

2.2.6. *Semantics*

Since plenty of I4.0 scenarios are knowledge-intensive, a comprehensive and complete knowledge representation has an important role. Many industry initiatives have built so-called knowledge graphs to represent domain knowledge about a manufacturing environment (e. g., Kalayci et al., 2020; Hubauer et al., 2018). For this purpose, semantic technologies and especially the Web Ontology Language (OWL) (Hitzler et al., 2012) are considered as appropriate to build such an industrial information model (Kharlamov et al., 2016) that provides classes, properties, individuals, and data values to express complex knowledge about individuals, groups of individuals, and their relationships (Hitzler et al., 2012). For this reason, *Semantics* can be used by each layer in the BPM abstraction stack. In previous work, we have presented the FTOnto[4] domain ontology (Klein et al., 2019) that semantically describes the physical smart factory used. The ontology is based on well-established standards and ontologies from the Industrial Internet of Things (IIoT) area. In the FTOnto, each sensor, actuator, operation for processing and transport, and the workpieces are described semantically with the aim of modeling a digital representation of the structural properties of the whole manufacturing system.

In addition to this part of the semantic model, the domain ontology is enhanced by SWSs that represent the capabilities of the smart factory (Malburg et al., 2020c; Seiger et al., 2022). For this purpose, a self-adopted and remodeled version of OWL for Services (OWL-S) (Martin et al., 2007, 2004) is used. This has resulted in the use of only one service class that corresponds to the *Service Profile* in OWL-S in which each service's functionality is described w. r. t. its Inputs, Outputs, Preconditions, and Effects (IOPEs). An illustration of the semantic annotations of a web service as a graph is depicted in Fig. 4 in which violet rectangles represent instances of classes and classes are represented by orange ellipses. In addition, green rectangles with rounded corners depict data properties and edges between the nodes indicate relations, i. e., object and data properties. The SWS is called *pickUpAndTransport* and has two parameters (*start* and *end*) for receiving variable start and end positions. Due to the variety of pickup and drop off positions, there are 81 possible combinations for this single web service in the first shop floor and 64 in the second one. Since different parameters, e. g., positions, can have different preconditions and effects, this aspect also has to be considered for *start* and *end* in the semantic descriptions. There are five preconditions related to the service depicted in Fig. 4, which must be satisfied at different times, i. e., at start, at end, or over all of service execution. These are, for instance, that the end position of the transport (i. e., oven) is available and ready as well as the light barrier that monitors this position must not be interrupted because that indicates an empty storage space. We would like to point out that these preconditions in turn can refer to another web service. The incoming responses of the preconditions are verified outside the knowledge base to enable verification in near real-time. The benefit of this procedure is that we do not need to continuously import large amounts of raw sensor data that are required to reason within the knowledge base to verify preconditions and effects. In particular, this could lead to a significant overhead for reasoning and possibly lead to wrong and not close to real-time information for decision-making. Furthermore, this procedure is applied to effects in the same way (see Malburg et al. (2020c) for more details). As depicted in Fig. 4, the exemplary service has only a single effect to check whether the service has been executed successfully. In this case, the effect checks whether the light barrier, which was not interrupted for the corresponding precondition, has now been interrupted, i. e., it is verified that the workpiece has been transported from the first sink of the sorting machine to the oven and, thus, the execution was successful (Malburg et al., 2020c).

[2] https://camunda.com/.
[3] https://www.omg.org/spec/BPMN/2.0.2/.
[4] https://gitlab.rlp.net/iot-lab-uni-trier/ftonto.

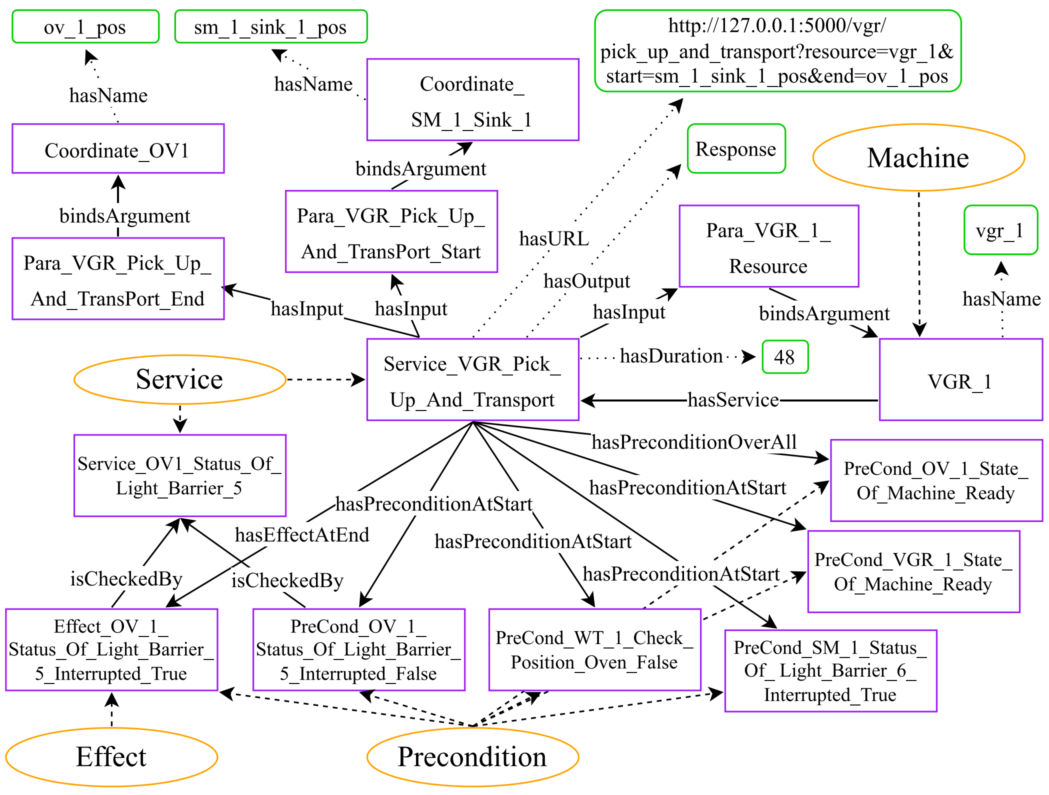
ov_1_pos
sm_1_sink_1_pos
http://127.0.0.1:5000/vgr/pick_up_and_transport?resource=vgr_1&start=sm_1_sink_1_pos&end=ov_1_pos
hasName
hasName
Coordinate_SM_1_Sink_1
Coordinate_OV1
Response
Machine
bindsArgument
bindsArgument
Para_VGR_Pick_Up_And_TransPort_Start
hasURL
Para_VGR_1_Resource
vgr_1
Para_VGR_Pick_Up_And_TransPort_End
hasOutput
hasInput
hasInput
hasInput
bindsArgument
hasName
Service
Service_VGR_Pick_Up_And_Transport
hasDuration
48
VGR_1
hasService
Service_OV1_Status_Of_Light_Barrier_5
hasPreconditionOverAll
PreCond_OV_1_State_Of_Machine_Ready
hasPreconditionAtStart
hasPreconditionAtStart
hasEffectAtEnd
hasPreconditionAtStart
PreCond_VGR_1_State_Of_Machine_Ready
hasPreconditionAtStart
isCheckedBy
isCheckedBy
Effect_OV_1_Status_Of_Light_Barrier_5_Interrupted_True
PreCond_OV_1_Status_Of_Light_Barrier_5_Interrupted_False
PreCond_WT_1_Check_Position_Oven_False
PreCond_SM_1_Status_Of_Light_Barrier_6_Interrupted_True
Effect
Precondition

Fig. 4. Semantic annotations of the *Pick Up and Transport* service from vacuum gripper robot as a graph.
Source: Based on Malburg et al. (2020c).

2.3. Automated planning in Industry 4.0

To enable more autonomous and intelligent manufacturing systems in the context of CPPSs (Monostori, 2014), it is necessary to develop methods that quickly respond to changes that occur during production and to resolve them automatically in the best case. For this reason, robust and efficient production planning and scheduling are of high importance (Monostori, 2014) and also a topic of current research (Rossit et al., 2019b,a). Automated planning[5] can be applied for solving planning problems by computers. For this reason, this technique is also becoming increasingly important in BPM, in which it can be used for several purposes and in different phases of the BPM life cycle (Marrella, 2019; Malburg and Bergmann, 2022; Rodríguez-Moreno et al., 2007; Malburg et al., 2023b). In the following, we introduce the basics of automated planning. After classical planning is introduced in Section 2.3.1, the foundations of temporal planning are explained in Section 2.3.2. Finally, Section 2.3.3 presents the basic concepts for representing planning domains by using the PDDL.

2.3.1. Classical planning

Classical planning can be employed to plan manufacturing operations by determining the production processes for each product (see Fig. 3), i. e., the necessary sequence of manufacturing tasks involved in producing a specific product. In the following, we introduce classical planning more formally, based on the definitions of Ghallab et al. (2016). A *classical planning domain* consists of a triple $\Sigma = (S, A, \gamma)$, where:

- S is a finite set of *states*,
- A is a finite set of *actions*,
- $\gamma : S \times A \rightarrow S$ is a partial *state transition function*. $\gamma(s, a)$ with $s \in S$ and $a \in A$ is defined as action a is *applicable* in the current state s.
- $\mathrm{cost} : S \times A \rightarrow [0, \infty)$ is a partial *cost function* that expresses arbitrary costs, e. g., time or money spent, for executing the action a in the current state s. The cost can also be *uniform* in the sense that the number of actions reflects the costs for executing them.

Following the definition of Ghallab et al. (2016) for classical planning, actions are instantaneous and, thus, there is no explicit time, e. g., how long a state or action holds. Classical planning problems can be solved by applying actions to an initial state that, in turn, lead to a transition from one world state to another world state by using the partial state transition function γ. To specify a planning problem

$$P = (\Sigma, s_0, S_g),$$

the initial state $s_0 \subseteq S$, the desired goal state $S_g \subseteq S$, and the available planning actions must be specified in the planning domain Σ (Haslum et al., 2019; McDermott et al., 1998; Ghallab et al., 2016). The goal of the planning process is to find a sequence of actions that transfer the initial state into the goal state. In addition, the cost for the resulting plan should be minimal. The resulting plan can be formally defined as a finite sequence of actions:

$$\pi = \langle a_1, a_2, \ldots, a_n \rangle$$

The length of a resulting plan is $|\pi| = n$, and its cost is the sum of the individual action costs: $\mathrm{cost}(\pi) = \sum_{i=1}^{n} \mathrm{cost}\left(a_i\right)$ (Ghallab et al., 2016).

2.3.2. Temporal planning

In contrast to classical planning in which the resulting plan is a sequence of actions, temporal planning (Fox and Long, 2003; Ghallab et al., 2016) must be used to combine scheduling techniques with classical planning to assign *when* which action is performed. Transferring this basic idea to production environments means that a temporal

[5] Since *Automated Planning* is a technique from the field of *Artificial Intelligence (AI)*, it is also often called *AI Planning*.

plan represents which manufacturing operations of which production machines are needed at which concrete time to produce a certain product. Based on Ghallab et al. (2016) and Fox and Long (2003), the basic concept of temporal planning are *durative actions*. A *durative action* $a \in A$ is defined as:

$$a[t, t'] : t < t'$$

The duration d is defined as a non-negative number $d > 0$ and is calculated by $t' - t$. Each planning action gets such a duration assigned. In temporal planning, the cost specified for an action is typically represented by the duration required to execute the action. Based on the definition for classical planning in Section 2.3.1, a temporal planning problem is defined as $P = (\Sigma, \phi_0)$ with $\phi_0 = (\mathcal{A}, \mathcal{S}_\mathcal{T}, \mathcal{T}, \mathcal{C})$. ϕ_0 is called a chronicle and consists of the durative planning actions $\mathcal{A}$, the goal state to achieve and the a priori supported assertions representing the initial state $\mathcal{S}_\mathcal{T}$, $\mathcal{T}$ as a set of assertions, and for expressing constraints that are conjunctive linked $\mathcal{C}$ (see Ghallab et al. (2016) for a more formally introduction). In this context, it is worth mentioning that all assertions, i. e., state descriptions, besides the durative actions, are also temporally bounded. For this reason, it is now possible to express whether conditions hold at start, at end, or over all the time of an action. Based on the definition of a temporal planning problem, a temporal plan with durative actions is a pair $(t, a[d])$ with t representing a rational-valued time and $a[d]$ expressing the durative action $a \in A$ and a non-negative rational-valued duration d. A temporal plan consists of these pairs of durative actions ordered in a sequence of time points t. In contrast to classical planning, the use of time leads to a significant increase in computational complexity. More precisely, temporal planning is EXPSPACE-complete and in general not reducible to classical PSPACE-complete planning problems (Rintanen, 2007). Consequently, using temporal planning in practical applications, e. g., for *Job Shop Scheduling*, is difficult due to the high computational complexity and, thus, currently investigated in research (Rossit et al., 2019b,a).

2.3.3. Planning domain description language

After classical and temporal planning are described formally in the previous section, the basics for representing classical and temporal planning problems by using the PDDL (McDermott et al., 1998) are introduced in this section. PDDL in its several versions (see Haslum et al. (2019) for an overview) is the de facto standard to express planning problems. In general, two files exist that are used: (1) the PDDL domain consisting of the specified planning actions that can be used and (2) the PDDL problem composed of the faced problem expressed by the initial state of the world and the goal state that should be reached by the planner. In the following, we first describe the basic components of PDDL domain descriptions and subsequently, the representation of PDDL problems.

A PDDL domain description consists of five components: (1) *Requirements* that are specified at the beginning of the domain description and serve as a hint for the planner, which specific PDDL constructs are used in the description, (2) *Types* can be specified to express class hierarchies and to perform type checking during planning, (3) *Predicates* are used to express the properties and values, i. e., *true* or *false*, of the real world, e. g., whether a certain product is located at a position or not, (4) *Constants* can be used to express properties that are not variable, such as a position within the shop floor or a concrete machine of a certain type, (5) the *planning actions* themselves that represent state transitions from one world state into another. For this purpose, preconditions and effects from the defined predicates can be combined into more complex ones. In addition, *Parameters* can be specified for each action, which can be used to configure the action, e. g., specific configuration settings of a machine in a factory. For applying temporal planning, the already described components of the domain description are extended by *Functions* that are used similarly to predicates. However, functions do not map to the binary values *true* or *false*, but to numbers (fluents). By using functions, it is possible to assign a *Duration* to otherwise instantaneous individual actions, which then depend, for example, on the selected parameter values of an action. Fluents and other numerical functions can also be used in classical planning domains for expressing other properties, such as specifying the burning temperature of an oven with a minimum and maximum value as a parameter of an action. By using numerical functions, their initial values must be specified in the initial state. For example, it is possible to specify a *total-cost* function in classical planning, representing the cost of executing an action and, thus, increasing the total cost accordingly when the action is selected during planning. For temporal planning, the *total-time* can be used to express this.

A planning problem expressed in PDDL consists of an *Initial State* in which predicates are pre-initialized. Similarly, the *Goal State* is also defined with predicates that should be satisfied after planning. Moreover, it is possible to use *Objects* in the problem that express possible parameter values for actions during planning. To guide the planning procedure and to find suitable plans, it is possible to define a *Metric* that determines when a plan is more suitable than an alternative plan. For example, it is possible to specify a metric that defines to minimize the total-cost or total-time so that plans are generated w. r. t. the lowest cost or the lowest total time (Haslum et al., 2019).

2.4. Requirements

To ensure that the already modeled knowledge about the operational capabilities of the manufacturing environment, i. e., the SWSs (Malburg et al., 2020c) and the domain ontology of the physical smart factory (Klein et al., 2019) (see Section 2.2.6), can serve as a suitable source for automatically deriving a formal planning domain description expressed in PDDL, the following requirements need to be addressed in this work. In addition to these requirements, we specify two requirements for enabling the use of AI planning in BPM in general. All requirements are based on relevant literature related to AI planning for service composition and knowledge acquisition, modeling, and representation. In addition, the requirements rely on our experiences with knowledge modeling and representation, IoT, and AI planning for BPM in the context of our smart factory (*Environment*) following the DSR methodology (Hevner et al., 2004):

R1 *Modeled Inputs, Outputs, Preconditions, and Effects:* The modeled Service-Oriented Architecture (SOA) must contain the relevant information to convert it into a corresponding planning domain description. In this context, the manufacturing capabilities have to be semantically enriched by IOPEs (Martin et al., 2004; Ďurčík and Paralič, 2011). This modeled knowledge can then be utilized to build the corresponding planning actions.

R2 *Complete Semantic Model:* The knowledge representation should be complete so that every manufacturing capability and also the IoT environment is modeled in a detailed manner. This is important as the closed world assumption of PDDL considers not modeled things as false whereas not modeled aspects in OWL underlay the open world assumption and, thus, are considered as unknown (Pieske et al., 2022). Consequently, not represented aspects, e. g., IOPEs or complete actions, in the knowledge model are not known and, thus, cannot be converted and used for planning.

R3 *Support Several Language Levels of PDDL Specifications:* PDDL (McDermott et al., 1998) is the de facto standard for representing planning domain descriptions and available in several different languages (see Haslum et al. (2019) for an overview). However, the support of planners for individual language levels is not always fully given and, thus, often only certain levels or aspects of a language are supported.[6] For example, there are only a few planners available that support

[6] There exists a comprehensive overview of several planning frameworks with information about their PDDL support at https://planning.wiki/ref/planners/atoz.

numeric expressions (i. e., fluents or functions) as parameters or in preconditions and effects. In addition, using numerical expressions in planning domains increase the problem-solving complexity and can lead to undecidability, as investigations by Helmert (2002) proved. Consequently, the planning domain converter should provide a functionality to transform advanced numeric expressions to simpler numeric functions with precalculated values. If the semantic model contains numeric values, they should be transformed into discrete value ranges (e. g., too low, low, normal, high, too high) that can then be used for planning. For example, the Fast Downward (FD) planner proposed by Helmert (2006) is one of the most prominent planners for classical planning. It supports several PDDL languages, but not always complete: FD supports the use of *action costs* from PDDL 3.1 but not the more general *numeric fluents* from PDDL 2.1.[7] In addition, the converter should provide further configuration possibilities to express properties of the planning description in different ways. For example, the *OPTIC* temporal planner (Benton et al., 2012) cannot handle negative preconditions.[8] Thus, the converter should provide a transformer that converts negative preconditions into corresponding negated predicates that can be used for planning with *OPTIC*.

R4 *Considering Temporal Aspect for Planning:* Classical planning is the most common used in several domains. However, the use of temporal planning significantly gets attraction recently, e. g., in several international planning competitions or in research (cf. Rintanen (2007), Cenamor et al. (2018), Eyerich et al. (2009) and Celorrio et al. (2015)). This is particularly the case for smart manufacturing in which the use of temporal aspects, i. e., when which production step is performed on which machine resource, is important. Therefore, the planning domain converter should be able to transform the already available knowledge into a temporal planning domain description expressed in PDDL 2.1 (Fox and Long, 2003) if the additional knowledge required for this purpose is available.

R5 *Available, Extensible, and Simple to Use:* A proposed planning domain converter should be available and extensible for other projects. For this purpose, the prototypical implementation should be available under an open-source license, enabling the reuse and modification for own purposes.

Based on the previously presented concrete requirements that should be fulfilled by a planning domain converter, we introduce in the following more General Requirements (GRs) that describe what is necessary for using AI planning in BPM.

GR1 *Reconverting Final Plans to a Common Workflow Representation Format:* After a planner is utilized to generate an appropriate solution, the final plan must be converted back into a standard that receives the names of the source domain so that the plans can be evaluated in their use case. For the application scenario used in this work, it means that the actions contained in a plan must be mapped back to their corresponding web services for execution in the physical smart factory. Since we use BPMN 2.0 service tasks for representing production processes (see Section 2.1), the actions must be transferred into corresponding process activities in a BPMN model.

GR2 *Practical Evaluation with Near Real-World Scenario:* Using AI planning in real-world scenarios can be challenging since there is a lot of knowledge needed to use AI planning efficiently and in some scenarios the computational complexity of solving the problem is high. For this reason, simulations are used in current work to prove the validity and quality of research approaches (e. g., Marrella et al., 2017). However, using simulated data for evaluation has the drawback that most of the data produced does not necessarily reflect run-time properties and behaviors and, thus, is sometimes strongly simplified. For this purpose, an approach using AI planning in context of BPM should be practical evaluated in a near real-world scenario, i. e., with a physical application scenario.

[7] https://planning.wiki/ref/planners/fd.

[8] See https://nms.kcl.ac.uk/planning/software/optic.html for more details.

3. Related work

First, we discuss related approaches that build semantic models, e. g., domain ontologies, knowledge graphs, etc., for production environments in Section 3.1. In this context, we want to highlight that knowledge engineering and knowledge management for production environments is increasingly investigated in current research. In addition, modeled knowledge about manufacturing capabilities is typically already available in practical applications. Based on that, we present related work in Section 3.2 that utilizes already modeled knowledge in ontologies or SWSs to translate this encoded knowledge into a corresponding planning domain description and, thus, limiting the typically high knowledge acquisition and modeling efforts. However, most of these approaches are not used in the context of I4.0 or in BPM and IoT. Finally, we present in Section 3.3 further approaches and methods to acquire the needed knowledge for using AI planning that go beyond techniques for reusing and translating already modeled knowledge. In general, these methods can be divided into *inductive* and *analytical* learning methods.

3.1. Knowledge representation of manufacturing capabilities

An architecture and an ontology for integrating web services for flexible manufacturing systems are presented by Cheng et al. (2017). For this purpose, a service is described semantically by which actuator provides it, which production operation is realized, its URI, and a description class, which is not further discussed. Whereas Puttonen et al. (2013) use OWL-S to describe web services semantically, similar to this work. However, they do not directly integrate them with a domain ontology of their manufacturing environment. Without integrating domain knowledge, it is difficult to automatically convert the captured knowledge into a planning domain description in later stages, since this requires converting general knowledge about the production environment, e. g., classes for planning types or properties for planning predicates, besides the services themselves. The SOA presented by Schnicke et al. (2020) describes services w. r. t. their capabilities, expenses, and quality. For finding a suitable service, matchmaking based on tag values is conducted. Unfortunately, this approach makes it difficult to automatically discover and orchestrate web services by applying AI techniques, since no established semantic approach for web service modeling is used. Among the listed related approaches, we are not aware of any work that explicitly deals with the validation of preconditions and effects of web services for CPPSs during execution. To evaluate preconditions before executing a service and effects after execution, typically the domain ontology and especially its instances must be updated according to the current state of the real world by defining queries of the Query Language for RDF (SPARQL) (e. g., Cheng et al., 2017; Puttonen et al., 2013). It can be expected that this leads to high reasoning efforts to ensure that the knowledge base is complete and up-to-date. Based on experimental investigations, we assume that continuous updates and reasoning are rather not feasible in near real-time w. r. t. the amount of data and complexity of the knowledge base due to the computational effort. Consequently, this procedure poses an issue for real-world application scenarios (cf. **GR2**). The used SOA presented in Malburg et al. (2020c) aims to make production control more flexible by using semantically enriched web services integrated into an existing knowledge base of the manufacturing environment, i. e., a domain ontology (Klein et al., 2019). The work considers the reasoning complexity of real-time applications and, thus, does not update the knowledge base continuously.

3.2. Translating semantic annotations for web service composition with automated planning

Automated planning can be used to orchestrate and compose manufacturing processes from scratch or parts of them for adaptation by using a complete domain model with corresponding actions that can be applied, an initial state, and a goal state (cf. Section 2.3). However, obtaining a complete planning domain description is a demanding and error-prone task. In addition, the maintenance of encoded knowledge in PDDL planning domain descriptions is associated with a high effort, since changes are made in plain PDDL files. Therefore, planning domain descriptions are often incomplete in real-world application scenarios and can only be completed and maintained at great expense (Nguyen et al., 2017; Zhuo et al., 2013; McCluskey et al., 2009, 2021). Since I4.0 manufacturing systems are typically built on SOAs using asset administration shells (e. g., Puttonen et al., 2013; Schnicke et al., 2020; Cheng et al., 2017; Seiger et al., 2022; Bader and Maleshkova, 2019) combined with knowledge representations (e. g., ontologies or knowledge graphs), the reuse of this already formalized knowledge can reduce the high acquisition and manual modeling effort for creating PDDL domain descriptions and is less error-prone, resulting in almost complete planning domain descriptions. In addition, the maintenance of encoded knowledge in ontologies and other knowledge representations is much easier by using well-known frameworks with graphical user interfaces (e. g., Protégé[9]) than performing changes in plain files. For this reason, approaches that use knowledge representations and translate the entailed encoded knowledge into corresponding planning domain descriptions are presented in the following.

Similar to the proposed approach, Chen and Yang (2005) use SWSs and an event calculus-based planner for process generation. For this purpose, they convert the SWSs to their planning model. Similar to that is the work of Yang and Qin (2010) in which modeled OWL-S processes based on SWSs are translated to PDDL domains for automatic web service composition. Moreover, Puttonen et al. (2013) propose an approach that uses SWSs to execute manufacturing processes using three software agents represented as web services. One of these agents, called Service Monitor, is a specialized web service that performs web service composition similar to automated planning by a web service discovery technique w. r. t. a given production goal and the current state of the world made available by a domain ontology. For this purpose, they use OWL to describe the state of the production system as well as OWL-S and SPARQL expressions to semantically describe the available web services that provide production capabilities. However, they do not convert the SWSs into a corresponding PDDL description for using established planning frameworks.

Klusch et al. (2005) present an OWL-S service composition planner called *OWLS-XPlan* that consists of several components. The first one is a converter called *OWLS2PDDL* that translates the SWSs and the corresponding domain ontology into a proprietary XML dialect named PDDXML. The second one is the automated planning component called XPlan that uses the created PDDXML to generate a solution based on an action-based FastForward planning combined with a Hierarchical Task Network (HTN) planner. In addition, the third component conducts re-planning during the execution of the plan to check whether the native plan can be properly executed or not. Similar to Klusch et al. (2005) and Kim and Kim (2007) also propose an *OWLS2PDDL* converter that converts OWL-S 1.1 service descriptions to PDDL 2.1 domains. Hatzi et al. (2009) present the *PORSCE II* system that composite SWSs by also using planning techniques. Compared to Klusch et al. (2005), they use HTN planning to solve the service composition problem as a planning problem. Their system consists of four components, namely the OWL-S parser, the transformation component, the OWL ontology manager, and a visualizer. Ďurčík and Paralič (2011) present a system in which the task of automated web service composition is converted into a corresponding planning problem. For this purpose, they use modeled OWL ontologies and OWL-S in which the conditions of web services are represented with the Semantic Web Rule Language (SWRL). In addition to this, the process specification of OWL-S is used and the goal of the process is converted into a corresponding PDDL planning problem. Similar to this approach is the work by Louadah et al. (2021). They also present a converter that transforms OWL-S services into corresponding planning actions to plan maintenance operations for trains. Daosabah et al. (2021a,b) present the *Context-Intentional Service Composition Architecture (CISCA)* for the composition of web services by using AI planning. For this purpose, one part of the architecture is a converter that uses the SWSs described in OWL and OWL-S and translates them into a correspondingPDDL planning domain description. The *Knowledge Engineering Web Interface (KEWI)* framework proposed by Wickler et al. (2015) supports users in developing and modeling planning tasks. For this purpose, knowledge can be represented and modeled by using the web interface. To enable the use of AI planning, the modeled knowledge can automatically be converted into corresponding PDDL constructs. In their evaluation, they show that the automatically generated PDDL is equal or better for the two problems compared to the PDDL modeled by an expert. Although these results are impressive, the used examples in the experiment are simplified and, thus, further more intensive evaluations are outstanding. Hoebert et al. (2020) present an approach to convert OWL ontologies into corresponding PDDL constructs. However, the approach is presented very vague and, thus, it is not described how the planning actions are automatically translated from the domain ontology into PDDL. Pieske et al. (2022) present prerequisites for their production platform system, in which they will use OWL for describing semantic knowledge about devices and capabilities. Thereafter, they plan to convert these semantic annotations into PDDL. For this reason, their prerequisites also contain the results of their comparison between the possibilities and characteristics for expressing knowledge in OWL with the possibilities and characteristics in PDDL. However, they do not propose a converter in their work, but they plan to develop one.

3.3. Acquisition of knowledge for automated planning

In addition to the approaches presented in the previous sections, there exist several further methods that can be applied to acquire knowledge for AI planning. The work of Jilani (2020) discusses several techniques for automated domain model learning. Basically, these methods can be divided into *inductive* and *analytical* learning methods. In the following, we briefly sketch these two branches of learning methods and give some references to concrete approaches.

Inductive Learning Methods are characterized by using training data that is used to derive general applicable rules that describe the correlations in the data (Jilani, 2020). One such inductive learning method is implemented in the *Opmaker* system by McCluskey et al. (2009). In their approach, they use generated plan sequences from a domain expert. Based on that, the required action knowledge is automatically derived. In contrast to reusing already available knowledge such as in the approach presented in this work, they transform knowledge encoded in the form of plans to new knowledge in the form of planning action descriptions. Similar to this work is the *HTNLearn* algorithm by Zhuo et al. (2014). In their approach, they use plan traces that are partially annotated with additional information, e. g., state information or conditions that must hold before and after the action. Based on this, they create a HTN with decomposition rules for tasks that builds the basis for hierarchical planning. The main benefit of inductive learning methods is that they generate new knowledge from available knowledge. However, one drawback is that these techniques need mostly a lot of training data to generate qualitative results and that the generated knowledge does not necessarily have to be valid (Jilani, 2020).

Analytical Learning Methods are in contrast to inductive learning methods based on a reasoning process (Jilani, 2020). In this process, new knowledge is generated by inferencing about already available knowledge. The main advantage of this process is that the new generated knowledge is valid. However, the newly generated knowledge is not really new, but derived from already existing knowledge. Analytical learning methods are based on reasoning, whereas the approach

[9] https://protege.stanford.edu/.

Table 1
Comparison of requirements addressed by related work on reusing knowledge for building planning domain descriptions. (✓) = addressed, (✓) = partially addressed, (✗) = not addressed, ⁄ = not assessable.

Related work	R1	R2	R3	R4	R5	GR1	GR2
Wickler et al. (2015)	✓	✓	✗	✗	✗	✗	(✓)
Ďurčík and Paralič (2011)	✓	✓	✗	✗	✗	✗	✗
Puttonen et al. (2013)	✓	✓	✗	✗	✗	(✓) – OWL-S	(✓)
Chen and Yang (2005)	✓	✓	✗	✗	✗	(✓) – WSL	✗
Yang and Qin (2010)	✓	⁄	✗	✗	✗	✗	✗
Klusch et al. (2005)	✓	⁄	✗	✗	(✓)	(✓) – OWL-S	✗
Kim and Kim (2007)	✓	⁄	✗	✗	✗	✗	✗
Hatzi et al. (2009)	✓	⁄	✗	✗	✗	✗ – OWL-S	(✓)
Daosabah et al. (2021a,b)	✓	⁄	✗	✗	✗	(✓) – OWL-S	(✓)
Louadah et al. (2021)	✓	✓	✗	✓	✗	✗	(✓)
Hoebert et al. (2020)	⁄	⁄	✗	✗	✗	(✓) – OWL	⁄

presented in this work is based on a pure transformation process. Both methods have one thing in common: if the knowledge available is incorrect or incomplete, the results of the analytical methods might also be inaccurate (Jilani, 2020). One system for analytical learning referenced in Jilani (2020) is the *PRODIGY* system (Carbonell et al., 1991). It uses the methodology of case-based reasoning and, thus, the stored experience to learn control rules and the domain model.

3.4. Summary and research gaps

To ensure rigorous research, it is required to build the artifact upon established methods and approaches (cf. *Rigor* cycle in DSR methodology Hevner et al., 2004). This process also includes the discussion and evaluation of current approaches based on certain criteria. Accordingly, we summarize and analyze the current approaches for reusing and transforming already modeled and encoded knowledge for AI planning (see Section 3.2) in this section. In addition, we present research gaps that are currently not investigated by related work. For this purpose, we characterize the research contributions of each approach by the derived requirements (see Section 2.4). Table 1 provides an overview to which degree the derived requirements are already addressed by related approaches. In this context, requirements marked with ✓ are completely fulfilled, requirements marked with (✓) are partially fulfilled, requirements marked with ✗ are not fulfilled, and requirements marked with ⁄ are not assessable by the descriptions in the corresponding paper.

Overall, it can be seen that all related approaches focus on certain requirements but do not examine all of them together to the extent as proposed in the approach in this work. **R1** (Modeled Inputs, Outputs, Preconditions, and Effects) related to the modeling of IOPEs is addressed by each approach, since it builds the basis for planning actions and, thus, is inevitably needed. In addition, **R2** (Complete Semantic Model) relates to the aspect that a complete semantic model is essential for using AI planning due to the differences caused by the open and closed world assumptions. In some approaches, it is described that an own ontology has been created and that this is the basis for the transformation into a planning domain description. These approaches are rated with ✓, for all others it is not possible to assess whether a semantic model is available and how complete it is, i. e., we marked them with ⁄. For **R3** (Support of Several Language Levels of PDDL Specifications), **R4** (Considering Temporal Aspect for Planning), and **R5** (Available, Extensible, and Simple to Use), it can be seen that almost none of the current approaches considers these requirements sufficiently in their approaches. The only recent work that considers temporal aspects (cf. **R4**) that are necessary for integrating scheduling in AI planning is the work by Louadah et al. (2021). All other approaches mostly support PDDL 2.1, but only instantaneous actions and not durative ones. In addition, only the approach of Klusch et al. (2005) is available by asking the author directly (cf. **R5**). However, the converter is only executable under certain conditions and packed in a JAR-file. For these reasons, it cannot be modified to own features in the semantic model and, thus, it is not easily usable. All other existing converters presented in related approaches are not publicly available and, thus, cannot be extended and used for own purposes. Considering the GRs presented in Section 2.4, some approaches convert the resulting plan back to the OWL-S process description or to some other format, such as WSL or OWL (cf. **GR1** – Reconverting Final Plans to a Common Workflow Representation Format). Even if the process description in OWL-S seems to be close to the description of processes in BPMN, there are two significant disadvantages: (1) OWL-S process descriptions cannot be executed by state-of-the-art WfMSs since processes represented in this language are not natively supported and (2) some aspects cannot be represented in OWL-S process descriptions to the same extent as it is possible in BPMN, e. g., no events can be modeled. For **GR2** (Practical Evaluation with Near Real-World Scenario), it can be determined that some approaches evaluate their approaches by using a simulation model (e. g., Puttonen et al., 2013) or by using exemplary planning problems from previous international planning competitions (e. g., Wickler et al., 2015). In addition, some approaches present case studies that are based on a real-world scenario in which the approach should be used in the future (e. g., Louadah et al., 2021; Hatzi et al., 2009; Daosabah et al., 2021b,a). All in all, there is a lack of practical evaluations with near real-world scenarios enabling the evaluation of whether the automatically generated planning domain achieves comparable planning results to a domain modeled by a domain expert. Consequently, the transfer of the proposed approaches to practical application scenarios cannot be justified. For this purpose, we evaluate the proposed approach with the physical smart factory presented in Section 2.1, which poses additional challenges due to ad-hoc interventions and runtime behavior that can only laboriously be simulated with data while additionally providing useful insights for practicability, feasibility, and for transferring the approach to real production environments (Malburg et al., 2020a).

4. Converting semantics into formal planning domain descriptions

In this section, we present how an already available semantic model (cf. *Semantics* in 2.2) can be transformed into a formal planning domain description by developing a *SWS2PDDL* converter as a research artifact of the DSR *Develop/Build* phase (Hevner et al., 2004). This transformation builds the basis to plan manufacturing processes to control the physical smart factory used and to react to unexpected behavior during runtime (Malburg et al., 2023b,a; Malburg and Bergmann, 2022; Marrella et al., 2017). In addition, the conversion into a planning problem aims to use state-of-the-art planners for production planning and scheduling in I4.0 (Rossit et al., 2019b,a). By using the PDDL standard, it is possible to flexibly choose planners, if they support the required language level, based on their properties for a certain use case. In the following, we present an overview of the architecture of the developed *SWS2PDDL* converter and how the individual components of the semantic model are mapped to corresponding counterparts in PDDL in Section 4.1. Then, Section 4.2 presents a simplified pseudocode of the converter and how the required information is gathered by using SPARQL queries. Finally, the conversion procedure is explained utilizing a running example in Section 4.3.

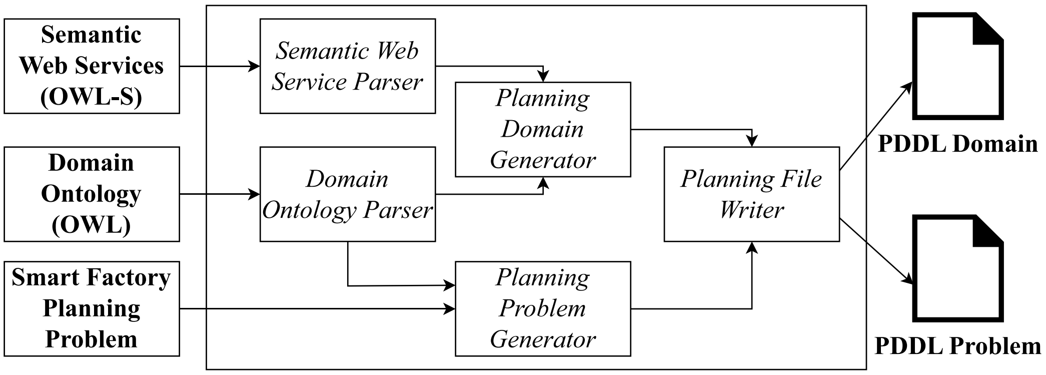
Semantic Web Services (OWL-S)
Domain Ontology (OWL)
Smart Factory Planning Problem
Semantic Web Service Parser
Domain Ontology Parser
Planning Domain Generator
Planning Problem Generator
Planning File Writer
PDDL Domain
PDDL Problem

Fig. 5. Architecture overview of the SWS2PDDL converter.
Source: According to Kim and Kim (2007).

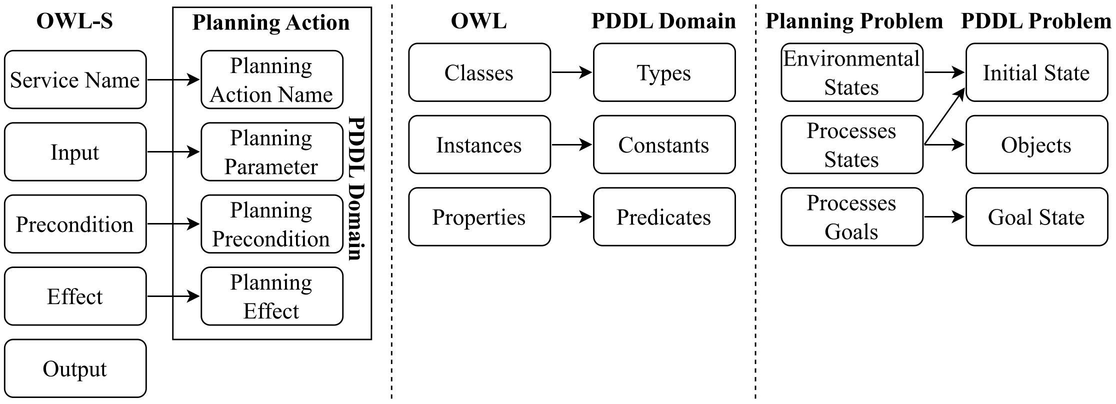

Fig. 6. Mapping of SWSs, the domain ontology, and the planning problem to PDDL constructs.
Source: According to Kim and Kim (2007).

4.1. Architectural overview and general approach

In this section, we present the components of the *SWS2PDDL* converter. Fig. 5 depicts an architectural overview of the *SWS2PDDL* components (italic font) that are needed to convert the *Semantics* presented in Section 2.2. The semantic model consisting of the SWSs and the corresponding domain ontology as well as the planning problem occurred in the smart factory are the inputs of the converter and the formal PDDL domain and problem descriptions the outputs (bold font). To generate the planning domain, the domain ontology FTOnto and the developed SWSs are processed by the *Domain Ontology Parser* and the *Semantic Web Service Parser* respectively. For this purpose, we use the open-source framework Apache Jena[10] for accessing the semantic model, i. e., the modeled knowledge represented in OWL and OWL-S. The parsers convert the stored knowledge into an intermediate format that allows to transform the SWSs into corresponding planning actions. For this purpose, the individual components of the SWSs are mapped to equivalent planning constructs (see Fig. 6). In this process, the *Service Name* is mapped to the *Planning Action Name* and the individual *Input* parameters of the SWSs are converted to *Planning Parameters* in PDDL. Similarly, the *Preconditions* and *Effects* of the service are mapped to *Preconditions* and *Effects* of an action in PDDL respectively. The *Output* of a service execution is, in most cases, a single message that is delivered to the client with information regarding the service execution time or with detailed information if a failure of the service occurs. For this reason, there is no counterpart for this in PDDL. Besides the model knowledge about the capabilities of the environment, e. g., the capabilities of resources in the physical smart factory, general knowledge about the environment represented in domain ontologies is used to specify the environment within the planning domain. *Classes* representing groups of certain things, e. g., a class of certain machines, in the ontology are converted into corresponding *Types* in PDDL. Concrete *Instances*, e. g., a machine of a certain class or the positions in the factory, are transferred to typed *Constants* in the PDDL planning domain. The *Properties*, e. g., the fact that a certain state occurs after executing an activity, are expressed by corresponding *Predicates*. Once this mapping is performed and the information is converted into the intermediate representation used, the *Planning Domain Generator* is applied to build the domain. For this purpose, we use the constructs and methods defined by the PDDL4J Java library[11] (Pellier and Fiorino, 2018). PDDL4J is also used to finally write the planning domain into a plain file (see *Planning File Writer*).

To determine the current *Planning Problem* to be solved, we need information about the *Environmental State* in which the process is executed, e. g., from the sensors and actuators of the physical smart factory and from the currently executed processes in the WfMS. This information is required to create the *Initial State* in the PDDL problem (cf. Section 2.3). It is important to note that we assume a static and deterministic state during planning. Thus, state changes such as changed sensor values during planning time are not directly considered (cf. Section 5.1). The processes currently executed in the environment by the WfMS are used to derive the current state for planning, i. e., the initial state. In addition, the processes executed by the WfMS can be uniquely identified by their process ID. This ID is converted to an *Object* in the planning problem and used to map the individual processing steps proposed in the resulting plan to the corresponding production processes. The modeled processes define the production steps for a

[10] https://jena.apache.org/.

[11] https://github.com/pellierd/pddl4j.

product and, thus, it is either possible to derive the *Goal State* of the planning problem from the respective *Processes Goals*, i. e., the process output that should be achieved after finishing the process (see Malburg et al. (2023b) for a detailed example), or to derive it from the final characteristics a workpiece should have at the end of production. Similar to the planning domain generation, the creation of the planning problem is performed by the *Planning Problem Generator* and the writing of the problem file is done with the PDDL4J library (see *Planning File Writer*).

4.2. Conversion procedure

In this section, we present the procedure of converting SWSs into PDDL, i. e., a formal planning domain description (see Section 2.3.3). The top-level pseudocode of the *SWS2PDDL* converter is shown in Algorithm 1 and details regarding the conversion for the used application scenario are presented in Section 4.3. The inputs to the algorithm are the semantic model $semModel$, i. e., a domain ontology and the modeled SWSs, the states of the environment $envStates$ in which the processes are executed, the states of the processes $processesStates$, and the goals of the processes $processesGoals$, i. e., the final products, that should be achieved by processes execution. In addition, the algorithm's output is a PDDL domain and a corresponding PDDL problem. The algorithm starts with an empty $domain$ and $problem$ (Lines 1–2 in Alg. 1). In Line 3, we iterate over each service from the ontology. Thereafter, we initiate a planning action pa and assign the name and the parameters from the service sws to the variables $pa.name$ and $pa.parameters$ from the planning action by using a SPARQL query (Lines 4–6 in Alg. 1). However, inputs of a service are only converted to planning action parameters if they cannot be directly represented by constants. The same procedure is performed for the preconditions and the effects of a service (Lines 7–8 in Alg. 1). Listing 1 depicts a simplified SPARQL query that is used to collect preconditions and effects from a service. After defining the prefixes used for executing the SPARQL query (Lines 1–5 in Lst. 1), the variables that are returned by executing the query are specified (Line 6 in Lst. 1). By using the uniquely identifiable base URL of a service, the corresponding conditions for the group of services in the ontology is queried (Line 8 in Lst. 1). Thereafter, the preconditions and effects, are retrieved (Lines 9–23 in Lst. 1). Since the SWSs define at which time a specific precondition or effect should be satisfied, we use the *UNION* statement to combine all of them. After collecting all preconditions and effects from the service groups that are relevant for the service and which can be more than one group of conditions, we search for their corresponding state that they represent (Lines 24–26 in Lst. 1). For this purpose, we express planning states with sentences composed of triples, i. e., subject, predicate, and object. Each sentence represents an *Atomic World State (AWS)* that is required for planning as a precondition or holds after executing an action as an effect. For example, an AWS could be $[workpiece, at, ov\_1\_pos]$ representing the state that a workpiece is located at the oven on the first shop floor. As the proposed converter supports temporal planning (see **R4**), the duration is gathered by a further SPARQL query from the service and thereafter a durative action is created, which is inserted into the planning domain (Lines 9–11 in Alg. 1). In this context, it is important to notice that all gathered preconditions and effects are used for defining preconditions and effects of the durative action. To be able to use PDDL 2.1 without durative actions, the corresponding parameter (cf. Parameter 1 in Alg. 1) can be set to false. In this case, the cost for executing the service is gathered by a SPARQL query and subsequently an instantaneous planning action is generated and inserted to the planning domain. In this context, not all gathered preconditions and effects are transformed into corresponding planning preconditions and effects, since some of them are only required for temporal planning (Lines 12–14 in Alg. 1). After all SWSs are transformed into corresponding planning actions, the further descriptions for the planning domain are created, if required: (1) the predicates, (2) the constants, (3) the functions, and (4) the requirements for the domain that are always inevitably needed (Lines 15–18 in Alg. 1). Thereafter, several transformers, i. e., transformation procedures, can be applied. First, a duration function in the domain and a corresponding duration metric in the problem is created if a temporal planning domain is requested (Lines 19–23 in Alg. 1). If non-temporal planning is used, a cost function in the domain and a corresponding cost metric in the problem is created (Lines 24–28 in Alg. 1). In addition, it is checked whether the requirements should be inserted into the planning problem or not, since some planners cannot handle requirements in the problem file (Lines 29–30 in Alg. 1). As several planners do not support negative preconditions or numeric fluents (cf. **R3**), we propose two further transformers (Lines 31–34 in Alg. 1).

Listing 1: Collecting Preconditions and Effects from Semantic Model.

```
1  PREFIX rdf:
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX owl: <http://www.w3.org/2002/07/owl#>
3  PREFIX rdfs:
        <http://www.w3.org/2000/01/rdf-schema#>
4  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
5  PREFIX ftonto: <http://iot.uni-trier.de/FTOnto#>
6  SELECT ?r ?so ?sp ?oo
7  WHERE {
8      ?serviceGroupConditions ftonto:onBindsBaseURL
           ?baseUrl.
9      {
10         ?serviceGroupConditions
               ftonto:preconditionsImplyAtStart ?aws.
11         BIND (ftonto:preconditionsImplyAtStart as
               ?r).
12     }
13     UNION  {
14       ?serviceGroupConditions
             ftonto:preconditionsImplyAtEnd ?aws.
15       BIND (ftonto:preconditionsImplyAtEnd as ?r).
16     }
17     UNION  {
18       ?serviceGroupConditions
             ftonto:preconditionsImplyOverAll ?aws.
19       BIND (ftonto:preconditionsImplyOverAll as
             ?r).
20     }
21     UNION  {
22       ...
23     }
24     ?aws ftonto:awsSubject ?so.
25     ?aws ftonto:awsPredicate ?po.
26     ?aws ftonto:awsObject ?oo.
27 }
```

The first one transforms the used negative preconditions into equivalent negated predicates, i. e., $(not\ (predicate(x)))$ is converted to $(not\_predicate(x))$. The other one transforms numeric fluents into other descriptions. For example, the FD planner (Helmert, 2006) does not support the full spectrum of numeric fluents. However, action costs and some simple numeric functions can be used. For this reason, the proposed converter transforms the numeric fluents and advanced functions into simpler functions by resolving parameters and precalculating all permutations by adding further functions in the converter. After applying the transformers to the created planning domain description, PDDL4J is initialized, meaning that the needed requirements are set in the exporter and some other general settings are made (Line 35 in Alg. 1). Based on that, the export function from PDDL4J is executed, which creates the planning domain and planning problem represented as PDDL (Lines 36–37 in Alg. 1). One advantage of using PDDL4J is that it is ensured that the planning domain and problem are syntactically correct and no own parser/exporter must be developed, i. e., there are no problems like forgetting a bracket. Furthermore, simple semantic errors such as dismissed to define the duration for a durative action or wrong types for constants and objects are logged and can then be easily fixed. Thereafter, the generated domain and problem is returned (Line 38).

Algorithm 1: Pseudocode of SWS2PDDL Converter

```
Input: Semantic Model $semModel$, Environmental States $envState$, Processes States $processesStates$, Processes Goals $processesGoals$
Output: PDDL Planning Domain $domain$, PDDL Planning Problem $problem$
Parameter 1: Boolean Flag $temporal$ to get the Temporal Planning Domain
Parameter 2: Boolean Flag $noNegativePreconditions$ to Remove Negative Preconditions from Planning Domain
Parameter 3: Boolean Flag $noNumericFluents$ to Remove Numeric Expressions from Planning Domain
Parameter 4: Boolean Flag $requirementsInProblem$ to write Requirements in Planning Problem
1  $domain = \emptyset$;
2  $problem = \emptyset$;
3  forall $sws \in semModel$ do
4      $init\_planning\_action(pa)$;
5      $pa.name \longleftarrow get\_Name(sws)$;
6      $pa.parameters \longleftarrow get\_Parameters(sws)$;
7      $pa.preconditions \longleftarrow get\_Preconditions(sws)$;
8      $pa.effects \longleftarrow get\_Effects(sws)$;
9      if $temporal = true$ then
10         $pa.duration \longleftarrow get\_Duration(sws)$;
11         $domain.actions \longleftarrow domain.actions \cup create\_durative\_action(pa)$;
12     else
13         $pa.cost \longleftarrow get\_Cost(sws)$;
14         $domain.actions \longleftarrow domain.actions \cup create\_action(pa)$;
15 $domain.predicates \longleftarrow create\_predicates(domain)$;
16 $domain.constants \longleftarrow create\_constants(domain)$;
17 $domain.functions \longleftarrow create\_functions(domain)$;
18 $domain.requirements \longleftarrow create\_requirements(domain)$;
19 if $temporal = true$ then
20     $domain.durationFunction \longleftarrow construct\_duration\_function(domain)$;
21     $problem.metric \longleftarrow construct\_duration\_metric(problem)$;
22     $problem.initialState \longleftarrow construct\_temporal\_initialState(domain, envStates, processesStates)$;
23     $problem.goalState \longleftarrow construct\_temporal\_goalState(domain, processesGoals)$;
24 else
25     $domain.costFunction \longleftarrow construct\_cost\_function(domain)$;
26     $problem.metric \longleftarrow construct\_cost\_metric(problem)$;
27     $problem.initialState \longleftarrow construct\_initialState(domain, envStates, processesStates)$;
28     $problem.goalState \longleftarrow construct\_goalState(domain, processesGoals)$;
29 if $requirementsInProblem = true$ then
30     $problem.requirements \longleftarrow domain.requirements$;
31 if $noNegativePreconditions = true$ then
32     $domain = transform\_preconditions(domain)$;
33 if $noNumericFluents = true$ then
34     $domain = transform\_numericFluents(domain)$;
35 $init\_pddl4j\_exporter(pddl4j)$;
36 $domain = pddl4j.export\_domain(domain)$;
37 $problem = pddl4j.export\_problem(problem)$;
38 return $domain, problem$;
```

4.3. Implementation and example from smart manufacturing application scenario

In this section, the conversion procedure is presented by using an example from the used smart manufacturing application scenario (see Section 2.1). In this context, we present how parts of the semantic model consisting of the domain ontology FTOnto (Klein et al., 2019) and the SWSs (Malburg et al., 2020c) are converted to equivalent PDDL constructs by using the proposed *SWS2PDDL* algorithm (see Algo. 1). For this purpose, the *Pick Up and Transport* service depicted in Fig. 4 is used as a running example in the following. Listing 2 depicts the planning action converted from the SWS illustrated in Fig. 4. The depicted planning action contains only one parameter (Line 4 in Lst. 2), i. e., the process ID. All other inputs of the *Pick Up and Transport* service, such as the resource executing the service, the start, and the end position can be directly specified by using planning constants, i. e., *vgr_1* and *sm_1_sink_1_pos*, *ov_1_pos*. In addition, the preconditions (Line 5–8 in Lst. 2) and effects (Line 9–12 in Lst. 2) are converted into predicates by using the collected AWS (see Lines 24–26 in Lst. 1), such as that the light barrier at the oven must be interrupted as an effect (*at ?processID ov_1_pos*). The cost for executing the service is expressed by an effect indicating that the total-cost function increases by the specified value (Line 12 in Lst. 2).

As already mentioned in Section 2.3, the consideration of temporal aspects is necessary in many environments, such as I4.0 to schedule *when* which step is performed on which machine (Rossit et al., 2019b,a). The creation of such a temporal planning domain is similar to the procedure used for non-temporal planning. Listing 3 depicts the durative planning action converted from the SWS illustrated in Fig. 4. For this reason, the duration of each service is specified as a duration in the durative action in temporal domains (see Fig. 4 and Line 5 in Lst. 3). In addition, the time-specific preconditions and effects are considered during the conversion. This leads to the fact that the used predicates for expressing preconditions and effects are extended by *at start*, *over all*, or *at end* (Fox and Long, 2003) compared to Listing 1 for classical planning (Lines 6–14 in Lst. 3).

Listing 2: Classic Planning Action for the Pick Up And Transport Service.

```
(:action Service_VGR_Pick_Up_And_Transport_       1
With_Resource_VGR_1_With_Start_Sink_1_With_       2
End_OV_1                                          3
:parameters (?processID - processID)              4
:precondition                                     5
  (and (at ?processID sm_1_sink_1_pos)            6
  (isready vgr_1)                                 7
  (not (isinactive vgr_1)))                       8
:effect                                           9
  (and (at ?processID ov_1_pos)                   10
  (not (at ?processID sm_1_sink_1_pos))           11
  (increase (total-cost) 48.0))                   12
)                                                 13
```

Listing 3: Durative Planning Action for the Pick Up And Transport Service.

```
(:durative-action Service_VGR_Pick_               1
Up_And_Transport_With_Resource_                   2
VGR_1_With_Start_Sink_1_With_End_OV_1             3
   :parameters (?processID - processID)           4
   :duration (=?duration 48.0)                    5
   :condition (and                                6
      (at start (at ?processID sm_1_sink_1_pos))  7
      (at start (isReady vgr_1))                  8
      (over all (not (isInactive vgr_1))))        9
    :effect (and                                  10
      (at start (not (isReady vgr_1)))            11
      (at start (not (at ?processID               12
          sm_1_sink_1_pos)))
      (at end (at ?processID ov_1_pos))           13
      (at end (isReady vgr_1)))                   14
)                                                 15
```

After automatically generating the planning domain description, the planning descriptions can be used to resolve failures in the smart factory (see Malburg et al. (2023b) for more details). In this context, AI planning is used to find a plan that solves the current problem and, thus, the generated plan can be executed in the smart factory. For this purpose, the previously executed process serves as a basis, and it is examined based on the generated plan which processing steps have to be inserted, deleted, or modified to resolve the failure. The resulting adapted process can then be deployed and continued in the Camunda WfMS at that point where the process was previously stopped. If changes have occurred due to the dynamic environment, and it causes that the generated plan is no longer executable, re-planning and scheduling can be performed to address this new situation (Malburg et al., 2020a; Malburg and Bergmann, 2022; Malburg et al., 2023b,a).

5. Experimental evaluation

In this section, we evaluate the use of a semantic model composed of a domain ontology and SWSs for automatically generating formal planning domain and problem descriptions expressed in PDDL (see Section 4). These domain descriptions build the basis to plan and schedule production processes in the physical smart factory. We consider a scenario in which a production component, i. e., a processing machine, is suddenly out of order and the currently running production processes must be replanned accordingly. Thus, the smart factory is used to generate five planning problems randomly to be solved by state-of-the-art planners, i. e., the FD planner (Helmert, 2006) and the Temporal Fast Downward (TFD) temporal planner (Eyerich et al., 2009). In addition to the automatically generated planning domain and problem description, we modeled another one manually based on domain expert knowledge in which the same planners solve the same tasks. In the end, the plans generated by the same planner for the same problem but by using the two different planning domain descriptions are compared by several characteristics, e. g., the plan length and the cost. The results permit conclusions whether the application of the used semantic model for transformation into a formal planning domain description in PDDL (cf. Section 4) is possible and successful. Thus, regarding the applied DSR methodology, the evaluation aims to justify the utility and relevance of the developed artifact for the addressed problems (*Justify/Evaluate* and *Relevance–Environment* in Hevner et al. (2004)). In the following, we describe the experimental setup in Section 5.1 and the results of the experiment in Section 5.2.

5.1. Experimental setup

For the conducted experiments, we use the described physical smart factory (cf. Section 2.1). We assume that each shop floor simulates an individual production line of a real-world production environment. In each production line, there are five different sheet metal processes (similar to the one shown in Fig. 3) planned and executed. To obtain near real world re-planning problems, a sudden failure, i. e., out of order, during the production is injected in one manufacturing resource of the factory. For this purpose, a failure generation engine that randomly chooses (1) one of both shop floors, i. e., the first or second one, (2) one manufacturing resource from one of these shop floors,[12] and (3) a time slot between three and 15 min to turn the component from this point off.[13] After the failure has been occurred in the smart factory, we stop all processes and capture their current state to use them as the initial planning state and their further planned production process sequence containing the final workpiece properties as the planning goal state. By using this procedure, we generate five problems that should be solved by the planning frameworks to continue production based on the current production state of the processes and by considering the failed component. This component is marked in the planning domain accordingly so that it could not be used by the manufacturing processes anymore. Thus, other components that can perform the required activities to reach the desired goal must be used by adapting the production processes, possibly even by using production capacities of the other production line (see Malburg et al. (2023b) for a similar experimental setup). In the experiments, we use FD (Helmert, 2006) as a non-temporal planner and TFD (Eyerich et al., 2009) as a temporal planner. FD can be used with different search algorithms with different search heuristics during planning. We use two different configurations: A* search with the landmark-cut heuristic (*lmcut*) (Helmert and Domshlak, 2009) and a lazy greedy best-first search with preferred operators and the queue alternation method as well as the context-enhanced additive heuristic (*hcea*) (Helmert and Geffner, 2008). TFD cannot be configured with different search algorithms and search heuristics. By default, the *hcea* is used in a modified form by TFD (Eyerich et al., 2009). In the experiments, we use a 180-second timeout for each planning run. Besides

[12] Please note that we have excluded the vacuum gripper robots and the high-bay warehouses as central components from this set.

[13] The problems generated also occur regularly in the smart factory even without explicitly generating them, e. g., a defect caused by the light barrier not working correctly.

Table 2
Characteristics of used planning domain descriptions for experimental evaluation.

Domain	Number of Actions/Types/ Predicates	Avg. Parameters per Action	Avg. Preconditions per Action	Avg. Effects per Action
Non-Temporal Expert Domain	37/34/50	3.41	6.24	4.11
Non-Temporal SWS2PDDL Domain	267/31/26	1.14	4.80	3.29
Temporal Expert Domain	37/34/50	3.46	6.27	5.32
Temporal SWS2PDDL Domain	267/31/26	1.14	4.80	4.35

the non-temporal and temporal domain generated by the *SWS2PDDL* converter, we also apply the corresponding manually modeled domains of a domain expert. Table 2 illustrates the differences between the manually modeled domain descriptions of the domain expert and the ones generated by the proposed *SWS2PDDL* converter. The domain descriptions, i. e., non-temporal and temporal, modeled by the domain expert contain 37 planning actions and required approx. 60 up to 75 h of modeling effort for each domain. The domain expert has solid experience in the field of AI planning and knowledge representation. In addition, the expert had access to recent textbooks, e. g., Ghallab (2004), Ghallab et al. (2016) and Haslum et al. (2019), and other material about automated planning and PDDL. Since the development of the planning domain started in parallel with the development of the service-based architecture and the contained SWSs (Malburg et al., 2020c), the expert could not rely on the knowledge contained therein. However, the expert is well-versed in the application scenario used (see Sections 2.1 and 2.2) and has in-depth knowledge of the possible manufacturing capabilities within the smart factory. The created domains have been created for the first time in 2020 and since then, the expert has manually maintained them in case of changes. Only after completion of the service-based architecture, the expert could access and use the existing knowledge for revisions. The goal of the manually created expert domain is to model the capabilities of the smart factory as accurately as possible. In contrast, the automatically generated domains do not require any additional manual effort since they leverage the existing semantic model (see Section 2.2.6) of the smart factory, resulting in 267 planning actions. The converter needs approximately 15 s to create one domain, including the time to read and parse the semantic model and to execute the SPARQL queries. In this context, the aim is to investigate if the provided semantic model which is converted into a planning domain description is capable of formalizing the real world similar to the expert who has modeled it manually. The different amount of generated planning actions can be explained by the fact that the domain expert has profound knowledge of how to generalize several manufacturing operations to one planning action, e. g., for reducing the modeling effort. For this purpose, the expert extends the parameterization (cf. the higher amount of average parameters per action) of planning actions. This also results in a higher number of preconditions (cf. the higher amount of average preconditions per action) checked for execution and a higher number of effects (cf. the higher amount of average effects per action) used after the execution of an action. In contrast, the *SWS2PDDL* converter generates one planning action for each possible manufacturing operation, i. e., each modeled SWS corresponds to one planning action because the knowledge needed for transferring a group of services to one planning action is currently not modeled in the ontology. Whereas the number of types is nearly similar, the number of predicates modeled by the expert is nearly twice as high as the predicates generated automatically by *SWS2PDDL*. This is because the expert relies on predicates for representing the capabilities of machines, whereas the *SWS2PDDL* converter uses different kinds of types and corresponding objects in the problem description to specify concrete manufacturing capabilities.

To compare the quality of the generated plans, we compare the plan length, the costs[14] for executing the plans, the generated and expanded nodes during search, and the average compilation and search times of the individual planners of five evaluation runs. A virtual machine with an Intel Xeon Gold 6130 CPU (8 virtual cores) with 2.10 GHz (turboboost 3.70 GHz) with 16 GB RAM, running Ubuntu 16.04 is used for the experiments. Table 3 depicts the five randomly generated problems that are used as initial state for the planners in the experiments.[15]

5.2. Experimental results

For the first experiment, we use the FD planner (Helmert, 2006) with the previously described planning domains. FD is a non-temporal planner and only generates a sequence of actions that transit the initial planning state to the desired goal state. For this reason, only the processes that are directly affected by the defect of the component are included in the planning problem, assuming that the other processes can be continued independently as planned (cf. *Number of Affected Processes* in Table 3). This means that in this case, temporal aspects such as *when* which process step is executed on which resource are not considered. Furthermore, the aspect that an adaptation of an affected process may influence another process that is not affected by the failure is also not addressed.

Table 4 lists the individual results for the experiment with FD by using the planning domain manually generated by the domain expert and the planning domain automatically generated by the *SWS2PDDL* converter. The generated plans are compared based on their length (*Plan Length*), their costs (*Total Cost*), and by determining the number of generated and expanded nodes representing the planning complexity. In addition, the average search time (*Avg. Total Time*) for each planning problem is given in seconds. For comparing the planning domain generated by the expert and the *SWS2PDDL* converter, we focus on the results generated by applying the *lmcut* heuristic during A* search because those correspond to the optimal solution in contrast to the results generated by the *hcea* heuristic from a greedy search algorithm. Comparing the two problems which take more than one second to generate a plan, i. e., problem 1 and 5, in both cases, the generated/expanded nodes and avg. total time are multiple times lower using the domain generated by *SWS2PDDL*. Whereas the number of generated/expanded nodes is still lower for the other two solvable problems, i. e., problem 2 and 3, the avg. total times are comparable since both take only a fraction of a second to solve the problem. The plans generated based on the *SWS2PDDL* domain also lead to lower or similar total costs and plan lengths compared to those obtained from the expert's domain. It is important to note that no solution could be generated for the fourth problem as all required machines for recovery are not functional, i. e., workflow executability cannot be achieved. For the results obtained by using a greedy search algorithm with the *hcea* heuristic, the average planning time decreases to less than a second and verifies the fast search speed of this type of search. However, we can observe that, except for problem 3, the resulting plans are different w. r. t. plan length and total costs compared to the optimal one found by A* search. For instance, finding the optimal plan for the first problem in the expert domain takes around 40x more time, i. e., from 0.59 to 24.72 s, but also reduces the total costs for executing the plan to nearly the half, i. e., from 1335 to 778. In the planning domain generated by *SWS2PDDL*, finding the optimal plan for problem 1 takes only around 14x more time, i. e., from 0.57 to 7.39 s, but also reduces the cost significantly from 965 to 652. Based on that, it can

[14] In the non-temporal planning this corresponds to the result of the total-cost function and in the temporal planning to the total-time for executing the plan, i. e., makespan.

[15] The used planning domains and corresponding planning problems can be found at https://gitlab.rlp.net/iot-lab-uni-trier/eaai-2023-journal.

Table 3
Overview of generated planning problems.

	Affected Production Line	Affected Component	Number of Affected Processes	Number of Running Processes
Problem 1	2	Milling Machine	2	6
Problem 2	1	Oven	1	6
Problem 3	1	Punching Machine	1	6
Problem 4	2	Sorting Machine	3	7
Problem 5	1	Workstation Transport	2	10

Table 4
Results of non-temporal production process planning.

		Planning Domain from Expert				Planning Domain from SWS2PDDL			
		Plan Length	Total Cost	Generated / Expanded Nodes	Avg. Total Time [s]	Plan Length	Total Cost	Generated / Expanded Nodes	Avg. Total Time [s]
Problem 1	*lmcut*	24	778	139409/4305	24.72	22	652	55385/1599	7.39
	hcea	33	1335	210201/5675	0.59	32	965	279357/6238	0.57
Problem 2	*lmcut*	14	599	8325/385	0.82	15	541	5123/231	0.63
	hcea	18	770	832/31	0.19	20	681	903/29	0.28
Problem 3	*lmcut*	6	320	354/16	0.19	6	320	261/11	0.26
	hcea	6	320	195/6	0.18	7	347	236/9	0.25
Problem 4	*lmcut*	—	—	—	0.15	—	—	—	0.17
	hcea	—	—	—	0.15	—	—	—	0.17
Problem 5	*lmcut*	21	803	476834/11878	99.86	19	676	53174/1368	14.00
	hcea	25	1224	70860/1773	0.38	33	1239	70863/1728	0.43

Table 5
Results of temporal production process planning and scheduling.

	Planning Domain from Expert				Planning Domain from SWS2PDDL			
	Plan Length	Makespan [s]	Generated / Expanded Nodes	Avg. Total Time [s]	Plan Length	Makespan [s]	Generated / Expanded Nodes	Avg. Total Time [s]
Problem 1	48	899.23	144885/5437	3.43	53	1008.30	323393/10058	10.93
Problem 2	47	1096.29	237059/9954	7.77	43	865.23	204017/5548	5.80
Problem 3	21	475.08	76915/4350	1.28	21	426.08	31129/3239	2.27
Problem 4	—	—	—	0.00	—	—	—	0.07
Problem 5	—	—	—	180.87	—	—	—	180.70

be observed that A* search is more appropriate for the automatically generated domain, as it has a significantly better ratio between invested computing time and resulted output. In contrast to this, the expert domain with less planning actions is better suited for a non-optimal greedy search. We checked each generated plan and determined that in some cases, the plans do not differ that much from each other. However, in some cases and in particular by using the expert domain, the plans contain some not necessarily required steps, e. g., individual transport steps or multiple trips to the same component. All in all, the results indicate the usefulness of planning domains automatically generated by the proposed *SWS2PDDL* converter for the application scenario (see Section 2.2).

Table 5 illustrates the results of the experiments with the TFD planner (Eyerich et al., 2009) and the previously described planning domains. In contrast to the first experiment, temporal aspects are considered in this experiment and, thus, all processes, even those that are not directly affected by the failure, are replanned to achieve a global near optimal allocation and utilization of the resources (cf. *Number of Running Processes* in Table 3). Fig. 7 illustrates a part of the temporal plan of problem P1.[16] Compared to the first experiment, the plan lengths increased since the schedule of actions is now considered for generating the solutions. In addition, no solution could be generated for the fourth problem as the required machines for recovery are not functional. Besides this, the fifth problem cannot be solved in the specified time, i. e., the search stopped after 180 s. Comparing the plan lengths and makespans for problem 1 to 3, it can be determined that both are comparable for the expert domain and the *SWS2PDDL* domain. For the generated/expanded nodes and avg. total time, the results in the expert domain are slightly favorable compared to those obtained in the domain generated by *SWS2PDDL*. This may occur because TFD utilizes a greedy search, which may not perform as well with a significantly higher number of planning actions (see Table 2) as for the A* search used during classical planning (see Table 4).

In summary, the automatically generated planning domain descriptions result in shorter or comparable plan lengths and the plans have better or comparable costs or makespans, which in turn leads to faster

[16] The visualization has been created with the PDDL Extension for Visual Studio Code by Jan Dolejsi. More information can be found at https://github.com/jan-dolejsi/vscode-pddl.

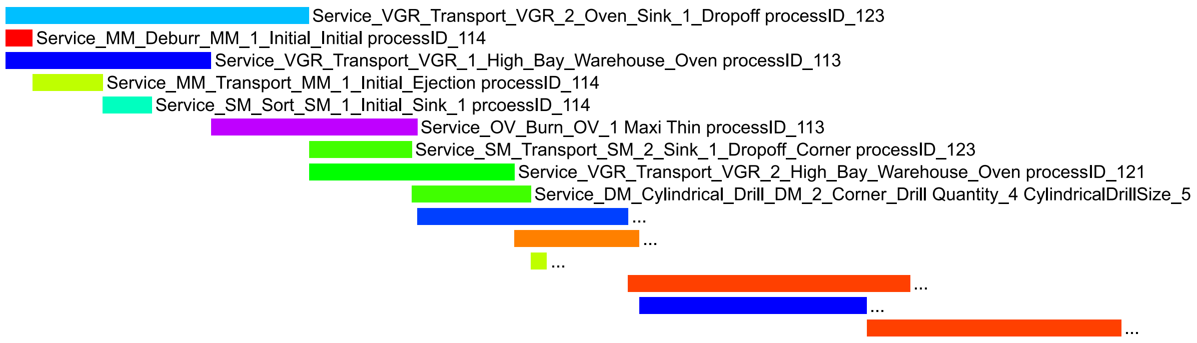

Fig. 7. Visualization of an excerpt from the temporal plan of P1.

and more efficient production planning and scheduling in I4.0. For all generated plans, we verified that those can be physically executed in the smart factory. The differences in plan length and cost between the domain created by the domain expert and the domain generated by *SWS2PDDL*, even when an optimal A* search has been used, are probably caused due to the different knowledge representations (see Table 2). This different representation of knowledge encoded in PDDL domain descriptions can result in slightly different plans. Another reason could be the lack of knowledge in the expert domain, e. g., by missing preconditions or effects. However, we have not identified any inconsistency or lack of knowledge in the expert domains during verification. Finally, the expert domains and also the planning problems are more than 1000 lines long and, thus, the complexity and error-proneness are high. At the same time, the maintainability is difficult even for domain experts and errors can easily occur. The results therefore highlight the importance of a converter that reuses existing and tested knowledge and converts it into a suitable representation for AI planning. Updating of knowledge can then easily be performed in designated knowledge engineering tools and no further modifications to the planning domains are required. Thus, the proposed converter limits the knowledge acquisition and manual modeling efforts for generating complete planning domain descriptions or supports domain experts in creating complete domain models.

6. Summary of contributions and limitations for practical application

In this section, we first discuss the results of the experimental evaluation based on the requirements defined in Section 2.4. Thereafter, limitations and barriers for practical application of the proposed approach are discussed.

The first requirement **R1** addresses modeled IOPEs and the second one **R2** a semantic model that contains all relevant information, i. e., to be complete. Based on the results of the experimental evaluation, both requirements can be considered as fulfilled, indicating that the used implementation of the SWSs and the domain ontology are complete and appropriate. The third requirement **R3** to support different PDDL language levels is addressed by providing options to apply several transformers. For instance, it is possible to transform negative preconditions into equivalent negated predicates in the planning domain description. In addition, it is possible to precalculate advanced numeric functions and, thus, to ensure that a planner without complete support for numeric fluents can handle the domain with its supported language level. Moreover, it is possible to write custom transformers that can be used for planning domain generation. The fourth requirement **R4** addresses the support of temporal aspects and, thus, the creation of PDDL planning domain and problem descriptions that can be used for temporal planning. Based on the results of the experimental evaluation, the requirement can be considered as fulfilled, as the converter can create a temporal planning domain with which the generated problems can be solved with similar results to the domain created by a domain expert. The fifth requirement **R5** demands the availability, extensibility, and simplicity of the implemented *SWS2PDDL* converter. We address this requirement by making the source code under an open-source license publicly available[17] and show how the converter's implementation is designed (cf. Section 4.1). By defining different modules, the functions are clearly structured and separated from each other. By using and implementing transformers, extensions can be easily added to customize the generated output for specific purposes. The first general requirement **GR1** requests to reconvert plans into a common workflow representation format. For our process-based research work (Malburg et al., 2023b,a; Malburg and Bergmann, 2022; Malburg et al., 2020a,c), we developed another converter that transforms generated plans into corresponding BPMN 2.0 process models (see Malburg et al. (2023b) for an example of such a converted plan). The second general requirement **GR2** addresses the practical evaluation with a near real-world scenario. As we conduct the evaluation with a physical smart factory (cf. Section 2.1) and, thus, the problems generated can be considered as near real-world problems, the requirement can be assumed as fulfilled. In addition, the use of the physical smart factory enables an easier transfer to real production environments.

As outlined in the discussion of **R5**, the implementation of the developed *SWS2PDDL* converter is publicly available, which enables other researchers to apply it and foster further research with much fewer efforts. However, some limitations and barriers exist for using the implemented converter to transform an already available semantic model into a formal PDDL planning domain description in a different setting. These include how resources required for production processes, such as machines, tools, and materials are semantically represented, as there is no standard and, thus, a need for some specific adaptions to other ontologies and semantic models can be necessary. For example, in the domain ontology FTOnto (Klein et al., 2019), we use concepts for representing machine resources by established and well-known ontologies (e. g., the *Manufacturing Concept* from *MASON* (Lemaignan et al., 2006) and the main classes of *SOSA*). Furthermore, there are several possibilities to describe IOPEs that are inevitably required for converting SWSs into a formal PDDL planning domain. For example, OWL-S can be used for representing IOPEs and is most commonly applied. However, it is also possible to represent IOPEs by using SWRL or the Web Service Modeling Ontology (WSMO) (Roman et al., 2005). Consequently, and dependent on the concrete representation and modeling of the SWSs, there could be further effort for adapting SPARQL queries in the *SWS2PDDL* converter to collect the required action name, preconditions, effects, and parameters. This highlights how important a well-structured and extensible implementation is (cf. **GR1**) to make reusing the converter for other settings possible. All in all, adapting the converter for individual research purposes leads to significantly less implementation work than developing a PDDL converter from scratch. Furthermore, the converter is based on established frameworks, e. g., Apache Jena and PDDL4J, which are well suited for the described purpose due to their runtime properties and their numerous features.

[17] The source code of the *SWS2PDDL* converter is freely available at https://gitlab.rlp.net/iot-lab-uni-trier/sws2pddl-converter.

7. Conclusion and future work

In this work, we present how already available knowledge represented as a semantic model can be converted into a formal PDDL planning domain description for automated planning and scheduling in I4.0. By reusing existing knowledge, the efforts for manual acquisition and laborious modeling of planning domain descriptions are significantly reduced. In addition, maintenance activities can be performed much easier, ensuring that the planning domain is always up-to-date and inconsistencies or mistakes are not inserted during this process. For this purpose, requirements are derived from literature that should be fulfilled by a converter. In addition, general requirements are presented that are important for applying AI planning in BPM, i. e., in the proposed smart manufacturing application scenario. The developed *SWS2PDDL* converter is applied to a semantic model composed of SWSs and a domain model of a physical smart factory. For the evaluation, the use case of a workstation failure that requires the re-planning of currently executed production processes is employed. The resulting plans to fix the misleading situations are compared to those obtained from a planning domain description that is generated manually by a domain expert. The results validate that for classical and temporal planning, the automatically generated planning domain descriptions result in similar or better plans than using the expert's one. This highlights the quality and general usefulness for limiting knowledge modeling and acquisition efforts of the proposed *SWS2PDDL* converter.

In future work, we want to investigate the topic of using AI planning to achieve more flexibility, robustness, and resilience of manufacturing processes for I4.0 further. In this context, we are currently working on the integration of AI planning with case-based reasoning to incorporate experiential knowledge during problem-solving (Malburg and Bergmann, 2022; Malburg et al., 2023a). This is especially useful since planning domains are often not complete in real-world application domains and, thus, planning from scratch is difficult to apply (Nguyen et al., 2017; Zhuo et al., 2013). In addition, repairing plans, e. g., by reusing already available plans stored in a case base for upcoming problem situations, is in most practical applications better than replanning completely from scratch (Borrajo et al., 2014; Nebel and Koehler, 1995; Babli et al., 2023). By combining case-based reasoning and AI planning, it can also be investigated whether the knowledge stored in cases can be used to learn parts of a planning domain description, similar to McCluskey et al. (2009). In this context, it can also be investigated how Large Language Models (LLMs) can be used for reducing the efforts required to create comprehensive planning descriptions and to assist experts during knowledge engineering. Currently, there is first work (e. g., Liu et al., 2023; Valmeekam et al., 2023; Guan et al., 2023) dealing with using LLMs for AI planning, but research is still in its infancy. Monitoring the smart environment to detect faults and predictions of failures from data-driven monitoring and complex event processing systems (e. g., Klein et al., 2021; Malburg et al., 2023b; Seiger et al., 2022; Malburg et al., 2023c) is a further topic for future research. In this regard, the use of semantic technologies to achieve interoperability between the different systems, i. e., planning and monitoring, is required to enable them to work together, and this research direction is still important for I4.0 (Kagermann and Wahlster, 2022). Furthermore, due to the common occurrence of incomplete planning domains in real-world applications (Nguyen et al., 2017; Zhuo et al., 2013), it could be investigated how the result of the proposed *SWS2PDDL* converter can be used for completion and correction of an existing, incomplete PDDL model. Moreover, it is possible to enhance the converter so that more than one SWS is transformed to a single planning action, as currently mostly performed by domain experts that have knowledge about dependencies between SWSs and how they can be abstracted. To use such a representation for the *SWS2PDDL* converter, further knowledge is required, e. g., about service groups, common preconditions, effects, and parameters. Future work could investigate how useful groups can be derived from the existing knowledge and which further knowledge is needed. In addition, it can be examined how well which domain representations perform on current state-of-the-art planners. In general, achieving more efficient and optimized production processes are, among the other aspects, important (research) topics for realizing the vision and corresponding benefits of I4.0 (see Use Case 2 in Malburg et al. (2020a)). However, the plan quality and the sometimes high planning times should also be considered. In this context, the already described incorporation of experience from domain experts such as in case-based reasoning (Bergmann, 2002; Malburg and Bergmann, 2022; Malburg et al., 2023a) or the use and combination with other AI methods based on deep learning (e. g., Hoffmann et al., 2022) could be beneficial as a hybrid approach. In this context, the combination of case-based reasoning with automated planning has already contributed to significantly faster problem-solving with good results in the past (e. g., Borrajo et al., 2014; Cox et al., 2005; Malburg and Bergmann, 2022; Zhuo et al., 2013; Malburg et al., 2023a).

CRediT authorship contribution statement

Lukas Malburg: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Patrick Klein:** Validation, Data curation, Writing – original draft, Writing – review & editing. **Ralph Bergmann:** Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing 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.

Data availability

We have shared the link to the data in the manuscript.

Acknowledgments

This work is funded by the Federal Ministry for Economic Affairs and Climate Action under grant No. 01MD22002C *EASY*.

References

Abele, E., et al., 2017. Learning factories for future oriented research and education in manufacturing. CIRP Ann. 66 (2), 803–826.
Babli, M., Sapena, Ó., Onaindia, E., 2023. Plan commitment: Replanning versus plan repair. Eng. Appl. Artif. Intell. 123, 106275.
Bader, S.R., Maleshkova, M., 2019. The semantic asset administration shell. In: Acosta Deibe, M., Cudré-Mauroux, P., Maleshkova, M., Pellegrini, T., Sack, H., Sure-Vetter, Y. (Eds.), Semantic Systems. In: LNCS Sublibrary, vol. 11702, Springer Open, pp. 159–174.
Benton, J., Coles, A.J., Coles, A., 2012. Temporal planning with preferences and time-dependent continuous costs. In: 22nd ICAPS. AAAI, pp. 1–10.
Bergmann, R. (Ed.), 2002. Experience Management: Foundations, Development Methodology, and Internet-Based Applications. In: LNCS, vol. 2432, Springer.
Bergweiler, S., 2016. Smart factory systems – Fostering cloud-based manufacturing based on self-monitoring cyber-physical systems. Int. J. Adv. Syst. Meas. 2, 91–101.
Borrajo, D., Roubícková, A., Serina, I., 2014. Progress in case-based planning. ACM Comput. Surv. 47 (2), 35:1–35:39.
Boschert, S., Rosen, R., 2016. Digital twin—The simulation aspect. In: Mechatron. Futur.. Springer, pp. 59–74.
Calà, A., et al., 2016. Modeling approach for a flexible manufacturing control system. In: 21st Int. Conf. on Emerg. Technol. and Factory Automat.. IEEE, pp. 1–4.
Carbonell, J., Etzioni, O., Gil, Y., Joseph, R., Knoblock, C., Minton, S., Veloso, M., 1991. PRODIGY: An integrated architecture for planning and learning. SIGART Bull. 2 (4), 51–55.
Celorrio, S.J., Jonsson, A., Palacios, H., 2015. Temporal planning with required concurrency using classical planning. In: 25th ICAPS. AAAI Press, pp. 129–137.
Cenamor, I., Vallati, M., Chrpa, L., de la Rosa, T., Fernández, F., 2018. TemPoRal: Temporal portfolio algorithm. In: Temporal Track of the International Planning Competition (IPC).

Chen, L., Yang, X., 2005. Applying AI planning to semantic web services for workflow generation. In: Int. Conf. on Semant., Knowl. and Grid. IEEE, p. 65.
Cheng, H., et al., 2017. Ontology-based web service integration for flexible manufacturing systems. In: 15th Int. Conf. on Ind. Inf.. IEEE, pp. 351–356.
Cox, M.T., Muñoz-Avila, H., Bergmann, R., 2005. Case-based planning. Knowl. Eng. Rev. 20 (3), 283–287.
Daosabah, A., Guermah, H., Choukri, I., Nassar, M., 2021a. Integrating context and intention for optimal semantic web service composition using AI planning. In: 4th CommNet. IEEE, pp. 1–9.
Daosabah, A., Guermah, H., Nassar, M., 2021b. Dynamic composition of services: an approach driven by the user's intention and context. Int. J. Web Eng. Technol. 16 (4), 324.
Ďurčík, Z., Paralič, J., 2011. Transformation of ontological represented web service composition problem into a planning one. Acta Electrotech. Inform. 11 (2).
Eyerich, P., Mattmüller, R., Röger, G., 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In: Proc. of the 19th Int. Conf. on Autom. Plan. and Sched.. AAAI, pp. 130–137.
Fox, M., Long, D., 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. J. Artificial Intelligence Res. 20, 61–124.
Ghallab, M., 2004. Automated Planning: Theory and Practice. In: The Morgan Kaufmann Series in AI, Elsevier.
Ghallab, M., Nau, D., Traverso, P., 2016. Automated Planning and Acting. Cambridge University Press.
Guan, L., Valmeekam, K., Sreedharan, S., Kambhampati, S., 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. CoRR abs/2305.14909.
Haslum, P., et al., 2019. An Introduction to the Planning Domain Definition Language. In: Synth. Lect. on Artif. Intell. and Mach. Learn., Morgan & Claypool.
Hatzi, O., et al., 2009. Semantic web service composition using planning and ontology concept relevance. In: IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology. IEEE, pp. 418–421.
Helmert, M., 2002. Decidability and undecidability results for planning with numerical state variables. In: 6th AIPS. AAAI, pp. 44–53.
Helmert, M., 2006. The fast downward planning system. J. Artificial Intelligence Res. 26, 191–246.
Helmert, M., Domshlak, C., 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In: 19th ICAPS. AAAI, pp. 162–169.
Helmert, M., Geffner, H., 2008. Unifying the causal graph and additive heuristics. In: Proc. of the 18th Int. Conf. on Autom. Plan. and Sched.. AAAI, pp. 140–147.
Hevner, A.R., March, S.T., Park, J., Ram, S., 2004. Design science in information systems research. MIS Q. 28 (1), 75–105.
Hitzler, P., et al., 2012. OWL 2 Web Ontology Language Primer. W3C Recommendation, available at https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.
Hoebert, T., Lepuschitz, W., Merdan, M., 2020. Automatic ontology-based plan generation for an industrial robotics system. In: Proceedings of the Joint Austrian Computer Vision and Robotics Workshop. Verlag der Technischen Universität Graz, pp. 27–28.
Hoffmann, M., Malburg, L., Bergmann, R., 2022. ProGAN: Toward a framework for process monitoring and flexibility by change via generative adversarial networks. In: BPM Workshops. In: LNBIP, Springer, pp. 43–55.
Hubauer, T., et al., 2018. Use cases of the industrial knowledge graph at siemens. In: Proc. of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks. In: CEUR Workshop Proc., vol. 2180, CEUR-WS.org.
Janiesch, C., et al., 2020. The internet-of-things meets business process management. A manifesto. IEEE Syst. Man Cybern. Mag. 6 (4), 34–44.
Jilani, R., 2020. Automated domain model learning tools for planning. In: Knowledge Engineering Tools and Techniques for AI Planning. Springer, pp. 21–46, Ch. 2.
Kagermann, H., Wahlster, W., 2022. Ten years of industrie 4.0. Sci 4 (3).
Kalayci, E.G., et al., 2020. Semantic integration of bosch manufacturing data using virtual knowledge graphs. In: 19th ISWC. In: LNCS, vol. 12507, Springer, pp. 464–481.
Kharlamov, E., et al., 2016. Capturing industrial information models with ontologies and constraints. In: 15th ISWC. In: LNCS, vol. 9982, pp. 325–343.
Kim, H.-S., Kim, I.-C., 2007. Mapping semantic web service descriptions to planning domain knowledge. In: World Congress on Medical Physics & Biomedical Eng.. In: IFMBE Proc., vol. 14, Springer, pp. 388–391.
Kirikkayis, Y., Gallik, F., Winter, M., Reichert, M., 2023. BPMNE4IoT: A framework for modeling, executing and monitoring IoT-driven processes. Future Internet 15 (3), 90.
Klein, P., Bergmann, R., 2019. Generation of complex data for AI-based predictive maintenance research with a physical factory model. In: 16th ICINCO. ScitePress, pp. 40–50.
Klein, P., Malburg, L., Bergmann, R., 2019. FTOnto: A domain ontology for a fischertechnik simulation production factory by reusing existing ontologies. In: Proc. of the Conf. LWDA, Vol. 2454. CEUR-WS.org, pp. 253–264.
Klein, P., Weingarz, N., Bergmann, R., 2021. Using expert knowledge for masking irrelevant data streams in siamese networks for the detection and prediction of faults. In: IJCNN. pp. 1–10.
Klusch, M., Gerber, A., Schmidt, M., 2005. Semantic web service composition planning with OWLS-Xplan. In: Agents and the Semantic Web. In: AAAI Technical Report, vol. FS-05-01, AAAI Press, pp. 55–62.
Lasi, H., et al., 2014. Industry 4.0. BISE 6 (4), 239–242.
Lee, J., Kao, H.-A., Yang, S., 2014. Service innovation and smart analytics for industry 4.0 and big data environment. Procedia CIRP 16, 3–8.
Lemaignan, S., Siadat, A., Dantan, J.-Y., Semenenko, A., 2006. MASON: A proposal for an ontology of manufacturing domain. In: Workshop on Distrib. Intell. Syst.: Collect. Intell. and Its Appl.. IEEE, pp. 195–200.
Liu, B., Jiang, Y., Zhang, X., Liu, Q., Zhang, S., Biswas, J., Stone, P., 2023. LLM+P: Empowering large language models with optimal planning proficiency. CoRR abs/2304.11477.
Louadah, H., Papadakis, E., McCluskey, T.L., Tucker, G., Hughes, P., Bevan, A., 2021. Translating ontological knowledge to PDDL to do planning in train depot management operations. In: 36th Workshop of the UK Planning and Scheduling Special Interest Group.
Malburg, L., Bergmann, R., 2022. Towards adaptive workflow management by case-based reasoning and automated planning. In: 30th ICCBR Workshops, Vol. 3389. CEUR-WS.org, pp. 211–220.
Malburg, L., Brand, F., Bergmann, R., 2023a. Adaptive management of cyber-physical workflows by means of case-based reasoning and automated planning. In: 26th EDOC Workshops. In: LNBIP, vol. 466, Springer, pp. 79–95.
Malburg, L., Hoffmann, M., Bergmann, R., 2023b. Applying MAPE-K control loops for adaptive workflow management in smart factories. J. Intell. Inf. Syst. 1–29.
Malburg, L., Schultheis, A., Bergmann, R., 2023c. Modeling and using complex IoT time series data in case-based reasoning: From application scenarios to implementations. In: Malburg, L., Verma, D. (Eds.), Proceedings of the Workshops at the 31st International Conference on Case-Based Reasoning (ICCBR-WS 2023) co-located with the 31st International Conference on Case-Based Reasoning (ICCBR 2023), Aberdeen, Scotland, UK, July 17, 2023. In: CEUR Workshop Proceedings, 3438, CEUR-WS.org, pp. 81–96.
Malburg, L., Seiger, R., Bergmann, R., Weber, B., 2020a. Using physical factory simulation models for business process management research. In: BPM Workshops. In: LNBIP, vol. 397, Springer, pp. 95–107.
Malburg, L., et al., 2020b. Demo video: Object detection for smart factory processes by machine learning. http://dx.doi.org/10.6084/m9.figshare.13240784.
Malburg, L., et al., 2020c. Semantic web services for AI-research with physical factory simulation models in industry 4.0. In: 1st IN4PL. ScitePress, pp. 32–43.
Malburg, L., et al., 2021. Object detection for smart factory processes by machine learning. Procedia Comput. Sci. 184, 581–588.
Marrella, A., 2019. Automated planning for business process management. J. Data Semant. 8 (2), 79–98.
Marrella, A., Mecella, M., Sardiña, S., 2017. Intelligent process adaptation in the SmartPM system. ACM Trans. Intell. Syst. Technol. 8 (2), 25:1–25:43.
Martin, D.L., et al., 2004. OWL-S: Semantic markup for web services - W3C member submission. URL https://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
Martin, D.L., et al., 2007. Bringing semantics to web services with OWL-S. World Wide Web 10 (3), 243–277.
Mazzola, L., et al., 2016. CDM-core: A manufacturing domain ontology in OWL2 for production and maintenance. In: 8th KEOD. pp. 136–143.
McCluskey, T.L., Cresswell, S., Richardson, N.E., West, M.M., 2009. Automated acquisition of action knowledge. In: 1st ICAART. INSTICC Press, pp. 93–100.
McCluskey, T.L., Louadah, H., Papadakis, E., Tucker, G., Bevan, A., Hughes, P., 2021. Knowledge engineering for planning and scheduling in the context of ontological engineering: An application in railway rolling stock maintenance. In: Knowledge Engineering for Planning and Scheduling Workshop.
McDermott, D.V., et al., 1998. PDDL - The Planning Domain Definition Language: Technical Report CVC TR-98-003/DCS TR-1165.
Monostori, L., 2014. Cyber-physical production systems: Roots, expectations and R&D challenges. Procedia CIRP 17, 9–13.
Nebel, B., Koehler, J., 1995. Plan reuse versus plan generation: a theoretical and empirical analysis. Artificial Intelligence 76 (1–2), 427–454.
Nguyen, T.A., Sreedharan, S., Kambhampati, S., 2017. Robust planning with incomplete domain models. Artificial Intelligence 245, 134–161.
Pellier, D., Fiorino, H., 2018. PDDL4J: a planning domain description library for java. J. Exp. Theor. Artif. Intell. 30 (1), 143–176.
Pieske, S., Herfs, W., Zenke, M., Storms, S., Brecher, C., 2022. Semantic modeling of a cyber-physical biological production platform. In: 2022 27th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, pp. 1–8.
Polge, J., Robert, J., Traon, Y.L., 2020. A case driven study of the use of time series classification for flexibility in industry 4.0. Sensors 20 (24), 7273.
Puttonen, J., et al., 2013. Semantics-based composition of factory automation processes encapsulated by web services. IEEE TII 9 (4), 2349–2359.
Rintanen, J., 2007. Complexity of concurrent temporal planning. In: 17th ICAPS. AAAI, pp. 280–287.
Rodríguez-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A., 2007. Integrating planning and scheduling in workflow domains. Expert Syst. Appl. 33 (2), 389–406.
Roman, D., et al., 2005. Web service modeling ontology. Appl. Ontol. 1 (1), 77–106.
Rossit, D.A., Tohmé, F., Frutos, M., 2019a. Industry 4.0: Smart scheduling. Int. J. Prod. Res. 57 (12), 3802–3813.

Rossit, D.A., et al., 2019b. Production planning and scheduling in cyber-physical production systems: a review. Int. J. Comput. Integr. Manuf. 32 (4–5), 385–395.

Rüßmann, M., et al., 2015. Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries, Vol. 9. Boston Consulting Group, pp. 54–89, (1).

Schnicke, F., et al., 2020. Enabling industry 4.0 service-oriented architecture through digital twins. In: Softw. Archit.. In: CCIS, vol. 1269, Springer, pp. 490–503.

Seiger, R., Malburg, L., Weber, B., Bergmann, R., 2022. Integrating process management and event processing in smart factories: A systems architecture and use cases. J. Manuf. Syst. 63, 575–592.

Sesboüé, M., Delestre, N., Kotowicz, J., Khudiyev, A., Zanni-Merk, C., 2022. An operational architecture for knowledge graph-based systems. In: 26th KES. In: Procedia Computer Science, vol. 207, Elsevier, pp. 1667–1676.

Valmeekam, K., Sreedharan, S., Marquez, M., Olmo, A., Kambhampati, S., 2023. On the planning abilities of large language models (A critical investigation with a proposed benchmark). CoRR abs/2302.06706.

Wickler, G., Chrpa, L., McCluskey, T.L., 2015. Ontological support for modelling planning knowledge. In: Knowledge Discovery, Knowledge Engineering and Knowledge Management. In: CCIS, vol. 553, Springer, pp. 293–312.

Yang, B., Qin, Z., 2010. Composing semantic web services with PDDL. Inform. Technol. J. 9 (1), 48–54.

Zhuo, H.H., Muñoz-Avila, H., Yang, Q., 2014. Learning hierarchical task network domains from partially observed plan traces. Artificial Intelligence 212, 134–157.

Zhuo, H.H., Nguyen, T.A., Kambhampati, S., 2013. Model-lite case-based planning. In: 27th AAAI. AAAI Press, pp. 1077–1083.