Leverage Asset Administration Shells to Support Artificial Intelligence Planning

Alexis T. Bernhard, Benjamin Blumhofer,
Martin Ruskowski, Achim Wagner
Innovative Factory Systems
Deutsches Forschungszentrum
für Künstliche Intelligenz GmbH (DFKI)
Kaiserslautern, Germany
{*firstname.lastname*}*@dfki.de*

Andreas Luxenburger, Daniel Porta
Cognitive Assistants
Deutsches Forschungszentrum
für Künstliche Intelligenz GmbH (DFKI)
Saarbrücken, Germany
{*firstname.lastname*}*@dfki.de*

***Abstract*—Modern Digital Twin standards and technologies, designed to enable smart factories in line with the Industry 4.0 vision, have not yet addressed all the challenges associated with contemporary digital twins. This paper shows how to leverage the Asset Administration Shell as an Industry 4.0 compliant Digital Twin technology to provide all necessary data for Artificial Intelligence Planning, represented via the Planning Domain Definition Language. For this purpose, a model for preconditions and effects of planning actions is defined. This model is applied to three different use cases showing different problems of fetching data from different partially standardized shell templates including three defined actions of transport, assembly, and storage.**

***Index Terms*—Asset Administration Shell, Planning Domain Definition Language, Intralogistics, Digital Twin, Industry 4.0**

I. Introduction

Over the past few years, the Asset Administration Shell (AAS) [1] has established as a constituent technology for digital twins in modern Industry 4.0 systems. In particular, a product AAS represents a digital twin of the respective product and is used in various forms throughout the whole production process and beyond. This includes applications such as digital product passport, as a handler for product representations (e.g., in CAD format) or for the production process. In the last case, one typical task is to automatically create an executable plan for a production process. Planning [2] belongs to the broad family of Artificial Intelligence (AI) methods focused on the problem of finding a (production) plan as a sequence of actions that, when sequentially applied, transitions from a current state to a desired goal state [3].

In order to derive such a plan, all required product and factory data must be gathered. This paper examines the applicability of AAS to gather all planning data. Specifically, we focus on defining an input format for an AI planning domain which contains all relevant static and also contextual information for planning in a symbolic and logical form. Given a goal to achieve, e.g., a newly produced product instance, a valid sequence of production steps will be found using a backward chaining approach. These steps represent or will be mapped to existing skills of available production resources. This, however, requires that all preconditions as well as the effects of the skill application are properly formalized. In this context, a skill is an executable implementation of an encapsulated (automation) function [4] and treated as a black-box only knowing its preconditions and effects without any information about the detailed implementation. This approach follows the characteristic of independent encapsulated modules in modern cyber-physical production systems (CPPS), reduces the planning complexity and increases the flexibility and reconfiguration of modules inside a CPPS. As a part of the applicability of AAS for production planning, this paper tackles the following research questions:

RQ1 Can existing AAS submodels be reused or extended to support the planning process, and / or new AAS submodels need to be defined?

RQ2 Do standardized AAS templates help to define new or existing AAS models for artificial intelligence planning instances?

RQ3 How can particularly preconditions and effects as core items of AI planning problems be represented?

To deal with question 3, we apply a new structure in the AAS, which is related to the grammar of the Planning Domain Definition Language (PDDL) [5]. PDDL is a widely accepted language to represent planning problems and there are ready-to-use PDDL solvers for generating a valid production plan.

To answer all research questions, the paper is structured as follows. In Section II, the current state of the art in the topics of AAS, skill-based production systems, intralogistics, and PDDL is introduced. Section III defines a model for actions and shows the currently defined general precondition and effect model defined for different capabilities: Pick&Place, transport, and assembly. Section IV and V show the precondition and effect model in three different processes on real testbeds. On this basis, Section VI discusses the answer of the research questions of the use cases. Finally, Section VII gives an overview of potential research steps in the future, and Section VIII summarizes the findings of the paper.

II. State of the Art

The AAS in its latest specification (V3) offers a uniform modelling and provision of different information facets of a production asset (e.g., the product itself, its manufacturing

process or involved machines and robots) in terms of standardized submodels. This benefits not only collaborating domain experts and partners along the entire value chain, but also AI-based value-added services in particular, such as online process planners and orchestration systems which plan production and ensure its execution. In [6] it was shown how online process planners in highly dynamic human-robot collaboration (HRC) scenarios can be provided with planning-relevant information before and during production in near real-time on the basis of an AAS-based service and information architecture [7]. In this context, submodels were developed to describe the hierarchical structure of the production facility (topology), route networks, access points and other geometric characteristics (layout), as well as product properties, related manufacturing processes and the location of product parts in corresponding warehouses and their stocks. While established industry standards, such as the VDA-5050 communication standard between automated guided vehicles (AGVs) and a master control system, were already considered the planning itself was a back-box. In this paper, we aim to integrate the PDDL standard by means of creating PDDL-based planning domains and problem statements from relevant AAS submodels to make this transparent.

To be able to utilize the AAS for planning tasks, it is necessary to describe the functionality of the production systems in the factory accordingly. To accomplish that, the concept of Capabilities, Skills, and Services (CSS) was published by [8] as the so-called CSS model. This model extends the Product-Process-Resource (PPR) model published in [9] by incorporating functions in the concept of production. These functions are encapsulated on three levels of abstraction. The terms' capability, skill, and service follow the definitions from [4] and the extensions from [8] and [10]. These definitions can be briefly summarized as follows. Services are the most abstract descriptions, including commercial properties. Capabilities (IDTA-02020) are implementation-independent specifications of functions. Skills implement the capabilities for a specific machine and may belong to a Control Component (IDTA-02015, IDTA-02016) with standardized control interfaces. Within a capability, skill, and service description, preconditions and effects can be included, which is essential for planning. According to [11] the CSS model can significantly enhance intralogistics flexibility and interoperability. These transport systems can utilize the skill concept as shown in [12] and [13].

PDDL is an action-centered language, inspired by the well-known strips formulations of planning problems [14]. Each action is expressed with preconditions and effects to describe the applicability and the consequences of an action. One main design decision in PDDL is to split the defined domain actions and predicates as well as problem definition with its object instances, initial- and goal-conditions into two files. PDDL is factored into a subset of features, called requirements, and thus each PDDL solver can specify in detail which input features it supports. More complex features like numeric fluents, derived predicates or events [14], [15] are typically computationally more expensive. In this paper, we focus on a common baseline of features that are supported by many PDDL solvers. For our validation, we specifically rely on the Fast Downward planner [16].

```
(define (domain on_off_switch)

(:requirements :adl :typing)
(:types switch)
(:predicates (on ?s1 - switch))

(:action turnOff
    :parameters (?s1 - switch)
    :precondition (on ?s1)
    :effect (not (on ?s1))
)
(:action turnOn
    :parameters (?s1 - switch)
    :precondition (not (on ?s1))
    :effect (on ?s1)
))
```

```
(define (problem turnOffSwitch)
    (:domain on_off_switch)

(:objects
    on_off_switch - switch
)

(:init
    (on on_off_switch)
)

(:goal (and
    (not (on on_off_switch))
)))
```

Fig. 1. The sample PDDL instance of turning off an on-off-switch with the domain part (left) and the problem part (right).

Figure 1 shows an example of the use case of turning off a switch in PDDL. The domain part describes two actions of turning the switch on and off. The problem part shows particularly the use case for the plan to turn the switch off.

The presented combination between AAS and PDDL was not explicitly treated in the literature. However, other approaches exist that are capable of using and transforming digital twin representations into PDDL instances. De Giacomo et al. [17] created a transformation between different digital twin languages such as Azure Digital Twins, Eclipse Ditto and Bosch IoT Things into PDDL. Novák et al. [3] use a problem generator to define input from a digital twin to a PDDL problem and generate live production plans and schedules without supporting AAS notations. Both papers do not particularly tailor their solutions to the AAS technology. In addition, [18] shows the usage of planning and scheduling tasks in the AAS context. However, they do not particularly cover the problem of skill / capability sequencing solved with PDDL.

III. AAS-based Information Representation for AI Planning

As described in RQ1 in Section I, the scope of the paper is to use data from AAS submodels to represent and handle the various contents of PDDL instances. To reduce the initial complexity of this problem, this paper focuses on PDDL parts with a marked high usage based on the definition in [19]. This includes requirements, object types, predicates, and actions of a PDDL domain, as well as objects, initial- and goal-conditions in a PDDL problem description. As basis for the PDDL domain part, a new submodel called planning description is added to the demonstrator AAS as displayed in Figure 2.

This submodel defines the name of the domain part in PDDL as a property, a SubmodelElementCollection (SMC) for all necessary PDDL requirements and an SMC for included object types, listing all object types such as product or module necessary for the execution. Underneath the object types SMC, all predicates are defined in a respective SMC. Each predicate

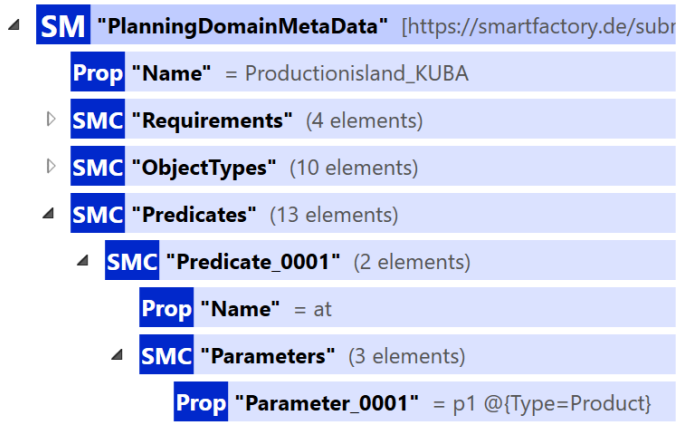

Fig. 2. The structure of the PDDL specific submodel in the demonstrator asset administration shell.

is also modeled as SMC including the name of the predicate and all parameters including their object types.

The last and main part of the domain part are the domain actions as displayed in Figure 1. In the considered modular use cases, all actions are not centrally managed in one AAS, but each module has an own AAS consisting of a submodel listing all machine skills, as a similar concept to actions in PDDL. For the transformation to actions as the main focus of RQ3, the existing skills need to be extended by SMCs for skill preconditions and effects. Both types of conditions share in these SMCs the same condition structure, displayed in Figure 3.

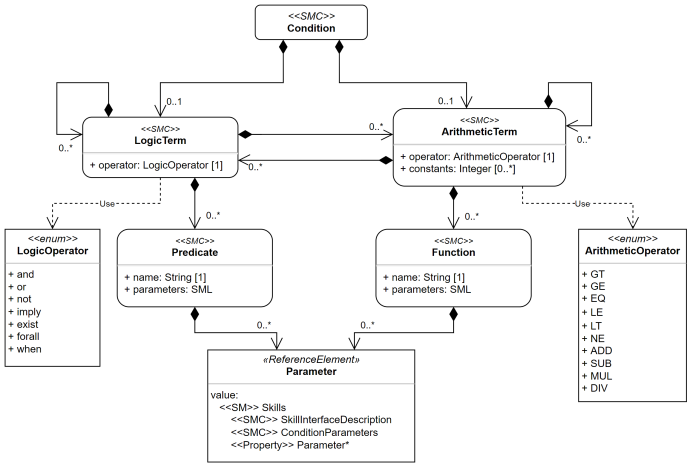

Fig. 3. The structure of a SubmodelElementCollection for Conditions.

This figure shows that conditions are modelled as either logical terms or arithmetic terms. A logic term consists of a logical operator. In plain PDDL version 1 without any requirements, the logic operator *and* is allowed in conditions. The other logical operators are added in additional requirements such as disjunctive preconditions for the operators *imply*, *or*; negative preconditions for the operator *not* (only available in PDDL 2.1); existential preconditions for the operator *exist*; universal preconditions for the operator *forall* and conditional effects for the operator *when* in effects. In the AAS, all operators are available by default for preconditions and effects, although the usage of added operators in requirements is restricted to their defined condition type. Currently, they are implemented in the AAS via using FormChoices, however a more suitable way to implement operators might be advisable in the future. One idea might be to use the Function Submodel Element, which is currently not supported in AAS implementations. Besides one logic operator, a logic term can consist of an arbitrary number of logic subterms, arithmetic subterms or atomic boolean predicates. Each predicate has a name and multiple parameters. A parameter in an action is a reference to one particular resource parameter stored in an SMC in the Skill Submodel (and SkillInterfaceDescription SMC) called ConditionParameters SMC. Each parameter has a unique idShort and requires a valid object type as a type.

In contrast to logic terms, arithmetic terms expect numeric planning enabled by (numeric) fluents. Arithmetic terms contain arithmetic operators such as comparative operators (`Less Than`, `Less Equal`, `Equals`, `Not Equals`, `Greater Than` and `Greater Equal`). As remark, the comparative operator `Equals` is already enabled by the requirement Equality. Besides, the comparative operators, the four basic mathematical operators `add`, `subtract`, `multiply`, and `divide` are supported. Similar to logic terms, arithmetic terms consist of an arbitrary number of logic and arithmetic terms and use functions as analogues concept to predicates as atomic building blocks.

Based on the given model, this paper discusses *three different kinds of skills* in the domain description as main building blocks for all use cases in the following chapters. The first skill kind is the storage skill kind, including the `store` and `deplete` skills, each implemented via a pick&place operation. The `store` skill requires a resource with storage capabilities. This resource picks a product from an external storage at a for the resource accessible location. Afterward, the resource places the product in another internal storage at another internal location. As a side effect, the external storage space is empty. Thereby, a location is a physical place which is accessible by one or multiple modules and/or transport systems. The `deplete` skill works the other way around and picks a product from an internal storage at an accessible location and places the product to another external storage at an external location. As for the store skill, the deplete skill leaves an empty internal storage space.

The second skill kind is transport. For the transport task, four different skills are supported. `GetTransport` orders a transport vehicle containing a certain product in its internal storage to a location. As preconditions, the respective transport system of the vehicle must be able to access the location, the location must be free of other transport vehicles and the product is not already at the desired location. `GetEmptyTransport` is similar to `GetTransport`, but orders an empty transport vehicle to a location and the product related precondition is omitted. The opposite to get transport

skills are the release transport skills. `ReleaseTransport` itself releases a transport vehicle containing a certain product from an accessible location and frees the vehicle to fulfill other tasks. `ReleaseEmptyTransport` releases a transport vehicle without a product on it.

The third skill kind is assembly. The created `assembly` skill description including its preconditions and effects originates from the original `assembly` action defined as one domain in AIPS-98 planning competition for PDDL. The `assembly` action enables the assembly of an arbitrary number of parts together into subproducts and afterward the assembly of these subproducts together into a product in a hierarchical manner. To do so, a precondition is required which defines that a subproduct is part of an assembled product and an assemble-order predicate defines the order between two subproducts to assemble the assembled product. Besides that, all products for the assembly must be in the same storage to assemble them together.

In addition to the domain description and the presented skill kinds, the given AAS should also support problem descriptions. The problem definition is implementation-specific and differs depending on the factory structure. In this paper, the idea is to present different ways to find module information, to get information about storages inside each module, to define certain transport or product locations, to define all active resources or capabilities of the products and to define all (sub-)products in the process. As basic structure throughout all use cases, each use case consists of three different types of AASs. Product AASs contain information about a product. Production System AASs provide information about the respective system topology and layout, including information about modules, storages, their current capacities and locations. The module AAS contains module skills and capabilities and their conditions. Out of these information sources, all initial- and object information needs to be built. Goal-conditions depend on the product configuration of each order and need to be set depending on the order information from the product AAS.

IV. Use Case: 3D-printed Battery Pack Assembly

This section introduces the use case of assembling a battery pack at SmartFactory-KL[1]. The battery pack consists of 3D-printed model batteries, and a battery case including a preassembled inlet to insert the batteries. The battery pack is displayed in Figure 4.

The battery pack in this use case is assembled at a modular demonstrator called productionisland_KUBA shown in Figure 5. The productionisland consists of four production modules. The Conveyor Module is a transport system that uses a linear motor to convey shuttles on a track. The Connector Module contains an input and output storage for incoming and outgoing parts, and a robot arm to pick and place parts from the Conveyor Module to this storage and vice versa.

Fig. 4. The battery pack including 3D-printed black model batteries and the respective Bill-of-Material of the Battery Pack.

The Assembly Module stores and depletes products from the Conveyor Module and assembles two parts together. The Quality Control Module uses AI-supported image processing to carry out a quality inspection of the manufactured products. However, quality control is not integrated in the production process. In addition, productionisland_KUBA contains the partial productionisland_SYLT that includes a 3D printer, a label-printer module, a robot to pick and place products at each module, and a manual assembly module.

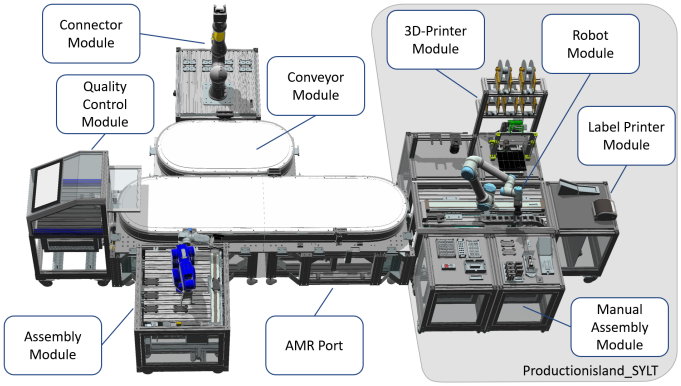

Fig. 5. The modular demonstrator productionisland_KUBA.

To answer RQ1, most objects and initial preconditions of the PDDL problem connected to the use case are fetched from existing AASs. For the product related initial conditions and products, this use case uses the bill of material (BOM) submodel of the product AAS [2]. The product AAS is, in general, used for all order and related product and process information. For example, this skill sequencing step has the goal to generate a planning submodel and adds this submodel to the product AAS. Before this planning step, the BOM submodel is already defined based on the ordered product configuration. A reduced version of the BOM of the battery use case is displayed in Figure 4.

Each entity in the BOM submodel maps to exactly one object of type product in the PDDL object definition. Furthermore, all hierarchically embedded entity relations map

[1]All PDDL instances and the battery pack AASs can be found in https://github.com/SmartFactory-KL/Leverage_AAS_PDDL

[2]The BOM follows the reference model in https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/AAS_Reference_Modelling.pdf

to one part-of predicate in the initial conditions. For example, this includes the predicate `part-of battery_black battery_pack`. Furthermore, two entities on the same hierarchy level lead to an assemble-order initial condition. This means that Battery_Case is added to the Battery_Pack assembly before all six Battery_Black.

Besides product-related information, also resource-related information needs to be modeled. As a basis, all entities as defined in the BOM submodel of productionisland_KUBA shown in Figure 6 need to be mapped to modules in the object model.

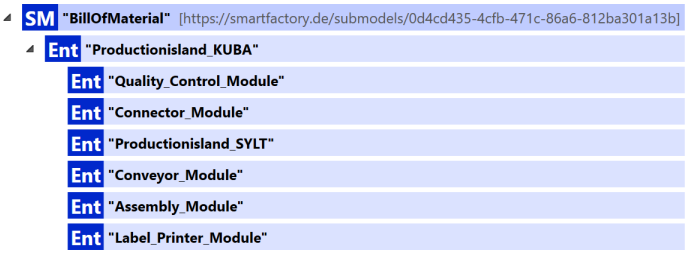

Fig. 6. The BOM submodel of productionisland_KUBA.

Next up are the storage and the location data. Since synchronizing all data to the AAS is a lot of work, we have decided to gather the data directly from the skill interface. For this purpose, the standardized AAS templates provide the standardized submodel "Asset Interface Description" (IDTA-02017). However, for productionisland_KUBA, the technology OPC UA is used as communication language as described in Jungbluth et al. [12]. Neither, the asset interface description in its current version supports OPC UA interfaces, nor a standardized submodel for OPC UA is published yet. This is why a non-standardized solution as a part of the skill submodel is currently used.

Examining the OPC UA NodeSet of each module, each NodeSet contains all locations and all currently active storages of a module. Since this demonstrator does not implement different storage types, all storages have the same external_storage_type. The mapping between locations and storages and modules, is ensured by the predicates `contains` and `accessibleBy`. The corresponding module to each storage and the location is given by their contained module name in OPC UA. For all `stock` and `capacity` predicates connecting storages to products, all storages contain either battery_packs or battery_cases, since they are always transported on top of picked workpiece carriers. On the other hand, productionisland_SYLT contains one storage for batteries, so they cannot be stored elsewhere. Last, but not least, the resources are set based on the capabilities the resources offer. All possible options of modules involving certain capabilities are given in the capability submodel of the resources. These options are restricted to the capabilities the resource can actually provide based on a given product configuration estimated in a capability matching between required capabilities of an order and provided capabilities of productionisland_KUBA. The results of the capability matching are given in a ProcessChain submodel in the product AAS. Before the presented skill sequencing step, a feasibility check can ensure the feasibility of the sequenced skills. In this use case, for example, only the manual assembly station is feasible of assembling 3D-printed batteries into the AAS.

With the initial conditions generally set, this use case particularly requires batteries in a storage at the manual assembly station at productionisland_SYLT and a preprinted 3D-battery case consisting of an inlet for the batteries inside the storage of the Connector Module. As an overview, the process of the use case is presented, however the generation of the production plan by a PDDL solver is not part of the paper.

1) $Connector$ Load battery case to the conveyor module
2) $Conveyor$ Transport battery chassis to productionisland_SYLT
3) $SYLT$ Transport the battery case to the manual assembly station, assemble the batteries into the battery chassis, and transport the battery pack to the conveyor module
4) $Conveyor$ Transport the battery pack to the connector module
5) $Connector$ Load battery pack to the output storage

To generate such a production plan in PDDL out of the transformed PDDL, the PDDL solver Fast Downward is used [16]. As a goal condition for this use case, an assembled battery pack should be available at the connector module. Besides the general goal condition, the secondary goal of freeing all allocated resources during the process is set.

V. Use Case: AMR Transport at SmartFactory^{KL} and RICAIP Saarbrücken Testbeds

In the following, we describe the setup of our demonstrator located at the RICAIP Saarbrücken testbed covering the production of a raspberry pi with housing including manufacturing and transport processes. The target product is depicted in Figure 7 and consists of four raw parts: a top and bottom case, a fan unit and the raspberry pi circuit board. What is also displayed is the manufacturing process of the product in terms of required production skills and their relative order. In this respect, for the sake of simplicity, we assume that all product components are pluggable, neglecting the connection of cables of the fan and screwing processes for the case. The structure of the product (BOM) and its manufacturing process are described in respective submodels of the product type AAS according to previous work of [7]. The manufacturing process submodel declares required steps, their preconditions in terms of needed product parts with respective links to the BOM submodel, relative order and corresponding capability requirements (like picking, placing or joining of different product part types) and their parameters in the form of a priority graph. This graph is used as input for PDDL like the Capability Matching described in Section IV.

Figure 8 shows the structure and layout of our demonstrator environment in which the execution of the production process will happen. A bin picking module is supposed to provide

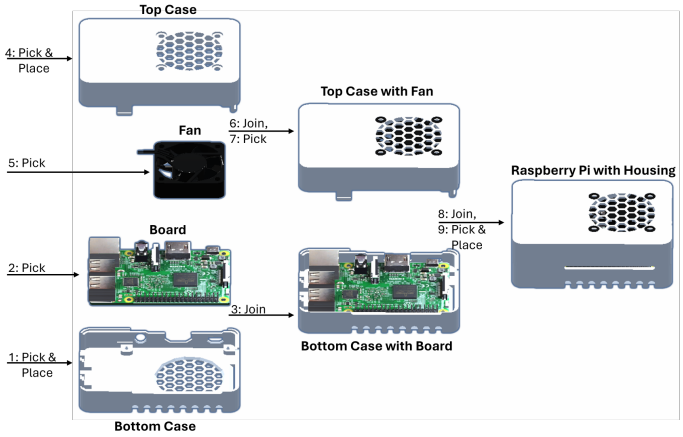

Fig. 7. Target product: a Raspberry Pi with housing, consisting of an upper and lower case, a circuit board, and a fan. Depicted is the corresponding manufacturing process with required capabilities like pick & place or join in a respective order.

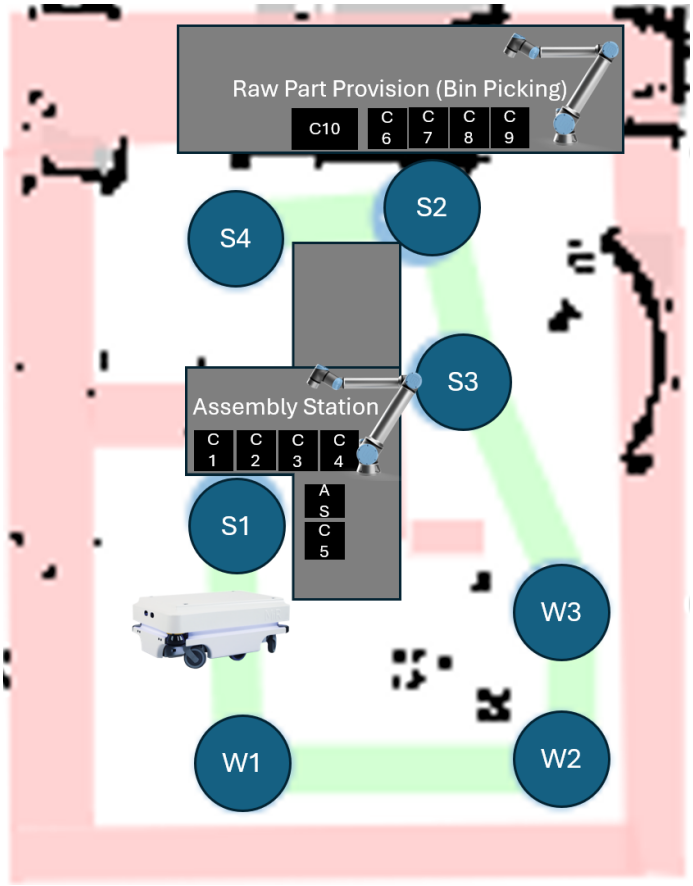

Fig. 8. The RICAIP Saarbrücken testbed demonstrator, consisting of a bin picking module for part provisioning and a station for product assembly with different storages (C1-C10). Route networks are made up by access points for stations (S1-S4) and waypoints (W1-W3).

the different product components as raw part materials in respective bins (C6-C9) which serve as storages this way. The module can be accessed by a mobile transport unit (MiR100) in terms of an access node (S3), constituting a station for the module. These access nodes, together with further waypoints and connecting edges (green lines), are defined in a layout submodel and manifest a valid navigation graph for the corresponding AMR type. Besides its symbolic name, known to the AMR, and its 2D pose in the associated map, a node declares its type and connection to topology elements of the production site such as the bin picking module. By node S1 the mobile robot can then deliver raw parts to the assembly station, which are then stored in associated cashes C1-C4 serving as temporary storage. Another storage type at the assembly station is constituted by the assembly stage for the assembly robot (UR10e), an internal storage type for the module where actual assembly steps will be conducted and which is supposed to be primarily used by the robot itself. Similarly, also the storage of an AMR is declared as internal storage type. Regarding topology, we follow the approach of [7] and declare the mobile robot and both one arm robots as WorkUnits together with their relationships to respective modules (topologically represented as WorkCenters), as well as associated storages in terms of EquipmentModules. A WorkUnit, as self-managed Entity, holds a reference to the respective AAS of the asset describing its production skills in a Capability submodel. This way, via said relationships, orchestration systems can infer the set of skills of a certain module in terms of the capabilities of its associated work units. Finally, the demonstrator setup is completed by temporary and final storage modules for finished product delivery at the assembly station (C9) and bin picking module (C10), respectively. Concerning our storage submodel of the AAS of the production site, we model storage as an annotated relationship between a certain product or part type (referencing respective elements in the BOM submodel of the product) and respective storage EquipmentModules in the topology submodel while annotating the current amount of stored objects and the maximum capacity of the storage.

In order to create a PDDL problem from said information about the production site and the intended manufacturing process, we proceed as follows. For participating objects, we need to derive, which modules, locations, product parts, storages and storage types exist, as well as which production resources by means of assets with certain capabilities are available. From the topology submodel of the production site, we identify both available workcenters together with their associated work units as modules and resources, respectively. Locations are then derived from nodes in the layout submodel. As mentioned before, the Capability submodels of the assets encode the skills of the resources with respective parameters, e.g., the ability of the mobile robot to autonomously navigate to known symbolic positions and that a one-arm assembly robot can pick, place and join specific object types. Note that in this scenario, we equipped the mobile robot with simulated capabilities of picking and dropping the mentioned product parts. This part will be then executed in a simulation environment by means of a virtual commissioning [6]. Storages and their type by means of internal or external

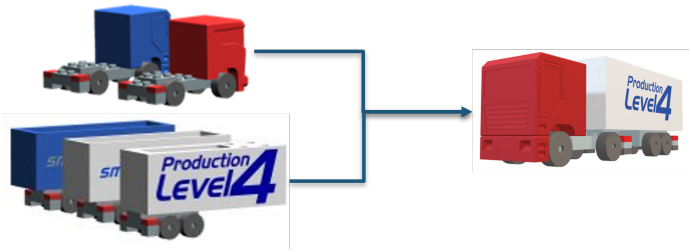

Fig. 9. Assembly of SmartFactory-KL model truck

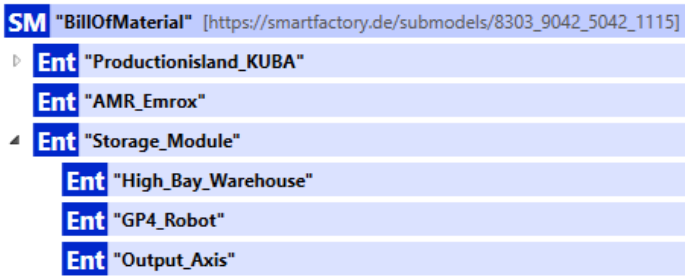

Fig. 10. BOM submodel of the SmartFactory-KL intralogistics testbed

storage, are inferred from the topology and storage type submodel, respectively. Analyzing the BOM submodel of the product type AAS gives us the different product components, while the manufacturing process submodel tells us in which order they need to be assembled as well as needed capabilities. Having all PDDL objects at hand, we can define the initial state of the problem. From the layout submodel, we infer from nodes of type "AccessNode", referencing a workcenter, which modules are accessible by which location. Storage is assigned to the corresponding modules ('Contains' attribute) using respective annotated Relationship elements ("BelongsTo") from the topology submodel and provided with its corresponding storage type from the storage type submodel. The product types that can be stored in a storage ("capacity") and quantities actually stored in it ("stock") result from the corresponding annotated relationships and quantity properties of the Storage submodel, respectively.

As a second evaluation with AMRs, we describe a use case in the SmartFactory-KL demonstrator. The goal of this use case is the final assembly of the SmartFactory-KL model truck. The truck consists of two subassemblies, the semitrailer and the semitrailer truck, as shown in Figure 9. For each subassembly, there are multiple configurations available.

As a demonstrator for this use case, the productionisland_KUBA introduced in Section IV is used in combination with a storage module and an autonomous mobile robot. The layout for this use case is shown in Figure 11. The BOM submodel of the factory is shown in Figure 10. The storage module consists of 3 core components, a high bay warehouse with a linear axis system to store and deplete products, a robot for pick and place applications and an output axis to make products available for external access. Additional components such as trolleys are simplified and not considered in this use case. As initial state for this use case, both subassemblies are located in the high bay warehouse within the storage module. The goal is to deliver the assembled truck to the label printer module. The products must be picked out of the storage slot by the axis system and placed on a pick position of the robot, which moves it onto the output axis. The output axis brings the product to a location from which the AMR picks up the product. The AMR consists of a robotic arm, and an internal storage where the robot places multiple products during the transportation. To bring the product into

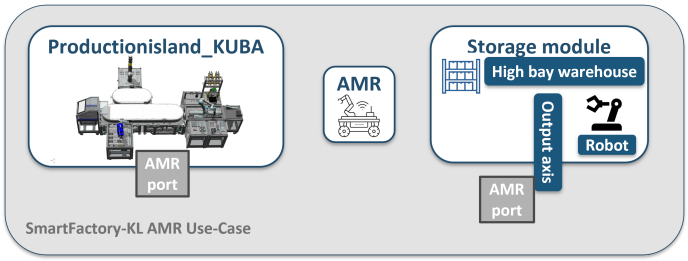

Fig. 11. Scenario for final truck assembly at SmartFactory-KL

the productionisland_KUBA the AMR moves to the open port on the conveyor module. The positions of the ports for the AMR are modelled in the production modules' AAS and read during navigation directly from the AMR. For each product, a shuttle has to drive to this port, so the AMR can place the product onto the shuttle. From there, the products are brought to the assembly module. Thereafter, the final assembly is executed, and the truck depletes to the conveyor module and a quality control is executed. After a successful quality control, the product is transported to the delivery module of productionisland_SYLT. This marks the end of the process and the achievement of the plan's goal state.

VI. Discussion

When considering the presented use cases in terms of the research questions, the given PDDL domain and problems are based on different data originating from a combination of existing submodels. This data is fetched from partially existing AAS elements including product and resource BOM submodels, or skill submodels for action data. On the other hand, additional information is required and introduced. This applies in particular to PDDL domain data, which is represented in an own new submodel (SM PlanningDomainMetaData). This also applies to the preconditions and effects of PDDL actions as the main topic of RQ3. For this purpose, Section III extends the skill submodel with a condition structure for preconditions and effects. In addition, the definition of conditions showed that further AAS concepts might help to cope with problems such as conditions and constraints more easily (e.g., via functions). Concerning RQ2, already defined standardized submodels could mainly be used for data gathering. This particularly includes the submodels Bill-of-Material, capability, control component interface, and asset

interface description submodel, which is flexibly applicable for different communication technologies to automatically fetch data from different architectural levels such as a OPC UA-based skill interface.

Using multiple use cases broadens the scope to include additional applications. At first, it shows that AMRs can be used with the same planning data structure as conveying systems by utilizing a skill-based approach. Second, it indicates that PDDL can also be used to plan skills on a component level within a production module. Third, it shows the requirement for additional concepts such as the concept of storage types.

VII. Future Work

Due to the fact that this is the first pilot test to combine the technologies of PDDL and AAS, there is a high potential to continue the work based on this paper. One potential option is to include other production actions and capabilities in the PDDL domain description and define additional related predicates to them (for example, typical manufacturing capabilities such as additive manufacturing). This also includes a usage of the PDDL instance for the representation of scheduling or optimization tasks in combination with action costs and metrics. Furthermore, another topic is the treatment of alternative action sequences, that can be changed dynamically during production execution based on resource and skill states, enabling a replanning or rescheduling. For example, a successful or failed quality check at the quality control module in the battery use case can be used to show this. Besides, the utilization of PDDL version one limits the representable planning problems. This is why, one promising expansion is to investigate numerical conditions such as storage sizes or scheduling with durative actions supported in higher versions of PDDL and other extensions such as PDDL expansions or axioms. Moreover, more research is desirable to automatically create and design goal conditions in PDDL based on data fetched from order-related submodels in the AAS. The same holds for the problem of flexibly changing domain data and automatically fetching / searching for data in the AAS to define the PDDL problem file. As a next step, an autonomous transformation between the AAS definition and the PDDL is planned based on the theoretical description given in this paper.

VIII. Conclusion

In conclusion, this paper demonstrates the applicability of the asset administration shell in supporting AI planning with PDDL. For this purpose, we discussed three research questions based on three different use cases. As a result, some new PDDL related AAS submodels, and a concept for conditions in AASs are developed and information originating from existing partially standardized submodels is reused.

Acknowledgment

This work has been supported by the European Union's Horizon 2020 research and innovation program under the grant agreement No 857306, the project RICAIP (Research and Innovation Centre on Advanced Industrial Production).

References

[1] Industrial Digital Twin Association, "Specification of the Asset Administration Shell Part 1: Metamodel," (Visited on 28.06.2023). [Online]. Available: https://industrialdigitaltwin.org/wp-content/uploads/2023/06/IDTA-01001-3-0_SpecificationAssetAdministrationShell_Part1_Metamodel.pdf

[2] M. Ghallab, D. Nau, and P. Traverso, *Automated planning and acting*. Cambridge University Press, 2016.

[3] P. Novák, J. Vyskočil, and B. Wally, "The digital twin as a core component for industry 4.0 smart production planning," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 10 803–10 809, 2020.

[4] A. Köcher, C. Hildebrandt, L. M. Vieira Da Silva, and A. Fay, "A Formal Capability and Skill Model for Use in Plug and Produce Scenarios," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2020, pp. 1663–1670.

[5] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson *et al.*, "Pddl - the planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[6] A. Luxenburger, J. Mohr, D. Merkel, S. Knoch, D. Porta, C. Paul, J. Widenka, P. Schäfers, M. Baumann, S. Lehnhoff, and J. Schwab, "Interactive digital twins for online planning and worker safety in intralogistics and production," in *Proceedings of the 6th IEEE International Conference on Artificial Intelligence and Extended and Virtual Reality (AIxVR-2024)*. IEEE Computer Society Press, 2024.

[7] A. Luxenburger, D. Porta, S. Knoch, J. Mohr, and T. Schwartz, "A service infrastructure for industrie 4.0 testbeds based on asset administration shells," in *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023, pp. 1–8.

[8] C. Diedrich, A. Belyaev, R. Blumenfeld, Juergen Bock, S. Grimm, J. Hermann, T. Klausmann, A. Köcher, M. Maurmaier, K. Meixner, J. Peschke, M. Schleipen, Siwara Schmitt, B. Schnebel, G. Stephan, M. Volkmann, A. Wannagat, K. Watson, M. Winter, and P. Zimmermann, "Information Model for Capabilities, Skills & Services," 2022. [Online]. Available: https://publica.fraunhofer.de/handle/publica/428536

[9] M. Schleipen, A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite, "Requirements and concept for Plug-and-Work: Adaptivity in the context of Industry 4.0," *atp magazin*, vol. 63, no. 10, pp. 801–820, 2015.

[10] A. Köcher, A. Belyaev, J. Hermann, J. Bock, K. Meixner, M. Volkmann, M. Winter, P. Zimmermann, S. Grimm, and C. Diedrich, "A Reference Model for Common Understanding of Capabilities and Skills in Manufacturing," *at-Automatisierungstechnik*, 2022.

[11] B. Blumhofer, M. Simon, A. Ritter, L.-M. Weil, T. Legler, and M. Ruskowski, "Capability Skill Service Model as enabler for Intralogistics 4.0: A Review," 2024, presented at 2024 7th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS).

[12] S. Jungbluth, T. Barth, J. Nußbaum, J. Hermann, and M. Ruskowski, "Developing a skill-based flexible transport system using OPC UA," *atp magazin*, vol. 71, no. 2, pp. 163–175, 2023.

[13] B. Blumhofer, A. Ritter, S. Jungbluth, J. Hermann, and M. Ruskowski, "Skill-basierte intralogistik: Transport von produkten an produktionsmodule durch mobile roboter," *atp magazin*, vol. 65, no. 8, pp. 48–57, 2023.

[14] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[15] A. Gerevini and D. Long, "Plan constraints and preferences in pddl3," Technical Report 2005-08-07, Department of Electronics for Automation ..., Tech. Rep., 2005.

[16] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[17] G. De Giacomo, D. Ghedallia, D. Firmani, F. Leotta, F. Mandreoli, and M. Mecella, "Iot-based digital twins orchestration via automated planning for smart manufacturing," in *Workshop on Generalization in Planning (GenPlan)*, 2021.

[18] Z. Mueller-Zhang, P. O. Antonino, and T. Kuhn, "Integrated planning and scheduling for customized production using digital twins and reinforcement learning," *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 408–413, 2021.

[19] A. Green, B. J. Reji, ChrisE2018, C. Muise, E. Scala, F. Meneguzzi, F. M. Rico, H. Stairs, J. Dolejsi, M. Magnaguagno, and J. Mounty, "Planning.wiki - The AI Planning & PDDL Wiki." [Online]. Available: https://planning.wiki/ref/pddl