Production Planning with IEC 62264 and PDDL

Bernhard Wally[1], Jiří Vyskočil[2], Petr Novák[2], Christian Huemer[1],
Radek Šindelář[3], Petr Kadera[2], Alexandra Mazak[3] and Manuel Wimmer[3]
[1]*Information Systems Engineering*, *TU Wien*, Vienna, Austria, {lastname}@big.tuwien.ac.at
[2]*CIIRC, CTU in Prague*, Prague, Czech Republic, {firstname.lastname}@cvut.cz
[3]*Business Informatics – SE*, *JKU Linz*, Linz, Austria, {firstname.lastname}@jku.at

***Abstract*—Smart production systems need to be able to adapt to changing environments and market needs. They have to reflect changes in (i) the reconfiguration of the production systems themselves, (ii) the processes they perform or (iii) the products they produce. Manual intervention for system adaptation is costly and potentially error-prone. In this article, we propose a model-driven approach for the automatic generation and regeneration of production plans that can be triggered anytime a change in any of the three aforementioned parameters occurs.**

***Index Terms*—Manufacturing Automation, Process Planning, Model-driven Engineering**

I. Introduction

Smart manufacturing environments require high flexibility on various levels, such as flexibility of the production system itself, flexibility on the products that are to be produced, flexibility with respect to material routing, etc. This agility is driven by the need for an *adaptive organization* [1], and it can be achieved by establishing well-integrated IT-systems [2]. This integration will have to be implemented by internal IT-systems, but also in accordance with external partners; it can be achieved by providing models for the data to be exchanged. These models are further of importance for many other aspects of Industry 4.0, such as simulation, optimization and data analytics [3] as well as the *digital twin* [4]. Such models can be implicitly computed from operations data, or they can be explicitly modeled in a traditional top-down fashion. Either way, many concepts established in the software engineering community in the field of *model-driven software engineering* [5] can be beneficial in the context of smart manufacturing, such as metamodeling, queries, views, and model transformations [6].

In this work, we are presenting a model-driven approach for generating production plans based on structural descriptions of production systems, their processes, and the material to be consumed and produced. It is a step towards the *model-driven smart factory* [7]. The production plans can be recomputed whenever a change in the system occurs, be it the production system itself (e.g., new machinery, machine breakdown, etc.), or the products that are being produced. Further, established processes can be examined whether there exist better ways to produce a certain good, or whether certain aspects of a production plant are not required and could be shut down.

Industrial control programs are traditionally hard-coded in the sense that they are programmed for producing a specific product or sets of products and need manual reprogramming in case a new kind of product is to be produced. In our prototype, the production recipe is derived dynamically from a production plan. The production plan, in turn, is generated automatically from a production system model that describes the available production resources and their capabilities, as well as the material to be handled and produced.

The remainder of this article is structured as follows: after a discussion of related work in Sec. II, we are providing background information in Sec. III and presenting our approach and implementation in Sec. IV. Then, we are describing our evaluation in the context of a real use case in Sec. V, before concluding and providing future research directions in Sec. VI.

II. Related Work

Already in [8], sophisticated production planning and scheduling software was seen to play a key role for companies to stay competitive, yet the availability and quality of corresponding software was rather limited. Today, there exist several planning and scheduling solutions, but as pointed out in [9], these products have not been adopted on a wider scale. Instead, it is noted in [9] that advanced production technology could enable companies to be profitable even if they produce very small lot-sizes with automated machinery by implementing flexible production systems that allow "rapid changeovers among different product". This is one aspect that we want to tackle with our approach: quickly creating production plans for new, potentially customer-defined products.

A genetic algorithm approach for the planning of flexible manufacturing systems has been presented in [10]. There, the production problem was numerically encoded and then solved using a genetic algorithm method. Different to our approach, their production system was manually crafted towards their genetic algorithm encoding, whereas we are using a standardized production system modeling format for the specification of the production system and generate planning information thereof. Unfortunately, [10] also does not make any statements on runtime and memory consumption of the algorithm with respect to the problem and domain size.

Evolvable production systems is another term used to describe the dynamic nature of future-proof production systems [11]. Such systems should be able to facilitate the shift from long-term vague production planning to short-term precise production planning, thus increasing fast reaction times, which is a requirement in today's fast paced world.

In [12], an automated assembly planning system is presented, that uses CAD assembly models to generate assembly plans using minimal manual interventions. That work focuses on the product design and the computation of a good sequence of assembly steps. This is very much related to our work, and the information retrieved by their work could be of significant use to our approach when it comes to describing the relations of the assembly parts. What is not included in [12] are the capabilities of the production system and the planning of the different assembly steps in a multi-machine shop-floor.

One approach to implement production flexibility is using multi-agent systems, as depicted in [13]. In such a system, multiple agents, situated on various levels of granularity of the production system negotiate feasible production plans. Such multi-agent systems are valid approaches for tackling various planning and reconfiguration scenarios, as shown in [14]. In our work, we implement such flexible behavior on the basis of a logics scheme, and we expand it from the pure transportation system scenario to a more complex situation involving transportation and product assembly.

Various systems for executing production plans have been proposed, some of them are discussed in [15]. However, model-driven generation of PDDL fragments from production system models following international standards is not discussed there. Using PDDL for industrial problems is further presented in [16], however, our approach differs in that it utilizes PDDL as an *intermediate* format rather than a tool for direct modeling by experts—the PDDL code is auto-generated from a production system model.

III. Background

In this section, we are presenting the concrete semantic models and languages that are used for the modeling of the production system and for the planning task: (i) an industry standard for the description of the production system: IEC 62264, and (ii) a well-established technology for planning tasks: the *planning domain description language* (PDDL).

A. Industry Standard IEC 62264

IEC 62264 is a series of international standards that have been originally published by the American National Standards Institute and the International Society of Automation (ISA) as early as 2000 under the name ISA-95 [17]. It comprises five parts that have been released over the years. IEC 62264 has gained additional momentum recently [18]–[20], by being adopted by AutomationML [21], which is considered a key enabling technology/standard for smart manufacturing by many experts. Model-driven integration of IEC 62264 with AutomationML has been described in [22].

For this work, we are using definitions from the latest revisions of the international standards parts 2 and 4, which have been standardized in [23] and [24], respectively. Part 2 defines metamodels, mostly in terms of Unified Modeling Language (UML) class diagrams, which allow the specification of entities and classes-of-entities that are relevant for specifying production systems and production processes. The most relevant metamodels for this work are the ones for equipment, material, process segments and operations definitions. Part 4 provides more detailed metamodels for the manufacturing operations management, but also some general purpose metamodels, such as resource relationship networks. We are using the latter for describing certain advanced aspects of a production system.

With the following IEC 62264 concepts we are able to describe a manufacturing facility in a detailed enough manner in order to set up a workflow for automatically generating production plans, as we will show later.

1) Equipment: describes the production sites, production lines, machinery, etc. that is available, in an hierarchical order. Equipment can be categorized by equipment classes and can define properties (which are, in essence, named attributes of type String).

2) Material: allows the specification of entities that are consumed and produced during production. A specific kind of material (e.g., a product, or a raw material) is defined by a material definition, which can be categorized by material classes. Specific instances of material definitions, i.e., pieces of material that have a specific location where they can be found, that have a mass, a serial number, etc., are modeled in terms of material lots. A material definition may define from which other material definitions it is assembled from, thus providing a “bill of material” (BOM).

3) Process segments: describe the procedural capabilities of a factory, or in other words, the composable production steps that can be carried out with the available machinery and personnel on a specific set of material. Each process segment declares which kind of personnel or which person, which kind (or piece) of equipment and which kind (or piece) of material, and how much of each of it, is required to produce a certain output material.

4) Operations definitions: when used for the configuration of production steps (as opposed to maintenance, quality control or inventory related tasks), the term “product definition” may be used synonymously. In this case (which we are mostly assuming in this work), an operations definition defines which production steps (i.e., process segments) are required to produce a certain product and their product-specific parameters, and the dependencies of these steps. As such, a product definition describes a “bill of processes” (BOP) for making a certain product.

5) Resource relationship networks: provide structures to model connections between any kind of basic resource. In our work these resources are equipment, material and process segments. Connections specify a connection type so that they can be semantically distinguished from each other.

B. Planning Task Language PDDL

PDDL was introduced in 1998 as the standardized Planning Domain Definition Language (PDDL) for the 1st International Planning Competition (IPC) [25]. Among its features was the separation of the planning domain (i.e., the “vocabulary” available) from the planning problem (i.e., a concrete problem instance). Multiple planning problems can be defined for a single planning domain. The latest revision of PDDL (3.1) dates back to 2011 and supports important planning requirements such as timing information and numeric properties [26]. Today, there exist several derivations of PDDL that have forked from various versions of PDDL—they add specific features for certain use cases. However, these derivations have not been adopted widely and only some their extensions have been introduced to PDDL in later revisions. PDDL is traditionally defined in terms of the Backus-Naur-Form (BNF) [27] and has been updated over the years to accommodate additional use cases to be competed for in the IPCs.

PDDL provides concepts for the description of planning domains, including types, constants, predicates, functions and actions, as well as for initializing problem instances, including objects, initializations, a goal and a metric. PDDL solvers try to find sequences of actions to reach the goal state from a defined initial state.

1) Types: enable the definition of a single-inheritance type-hierarchy. The default type is "object", which is assumed for all elements that have not defined an explicit type. Typing is optional, nevertheless it is supported by most PDDL solvers.

2) Constants: object instances that are available already in the domain definition (and later on in the problem definition).

3) Predicates: boolean statements that are bound to tuples of objects at runtime. In the domain definition each predicate is defining its name and the number and type of parameters.

4) Functions: definitions of numeric statements about tuples of objects. In the domain definition a function defines the types of objects allowed as parameters.

5) Actions: named steps a plan can rely on. Only through actions the system state can be changed. An action defines a set of optionally typed parameters that are then used in preconditions and postconditions, which are ultimately predicate statements on the provided parameters that can be (depending on the supported extensions) combined using *and*, *or*, *not*, *imply*, *exists* and *forall*. The extension *durative-actions* enables actions to last a certain time and to clarify at which point in time the pre- and postconditions shall hold.

6) Objects: are defined in problem statements and define (in addition to the constants of the domain definition) the known objects of the system under observation. Like constants, they can be typed.

7) Initializations: a list of predicate and function initializations, representing the *start* system state.

8) Goal: a list of predicate instantiations that must be satisfied. The goal declares the *end* system state.

9) Metric: an optional metric. Typically this is a function that should be minimized or maximized. Based on the metric PDDL solvers decide whether a newly found plan is superior compared to an already computed plan.

IV. Implementation

We are using two distinct kinds of technology to solve the automated planning of production steps and intermediate inventory movements based on structurally defined production systems: (i) we are defining the metamodels for the required domains, and (ii) we are defining a workflow that incorporates a PDDL solver for automatically finding production plans.

A. Creating the Metamodels

In a first step, we are creating what is called the "metamodels" in model-driven engineering (MDE) [28] for the technologies in use. Such a metamodel (i.e., a model of a model) defines the "vocabulary" that can be used to express certain occurrences in a specific domain.

1) IEC 62264: for the metamodel of IEC 62264 we are following the standards documents of for part 2 [23] and part 4 [24] as closely as possible. There, most of the domain knowledge is encoded in UML class diagrams, which corresponds to the required format of metamodeling. Nevertheless, the class diagram images from the standard have to be implemented in terms of a metamodel, and not all information is encoded in the class diagrams.

2) PDDL: the metamodel for PDDL is more difficult to create, as it is only specified using BNF. Specifically, we are using the PDDL BNF presented in [26], which represents PDDL version 3.1. Since the automated generation of class diagrams (and thus of metamodels) from BNF is not easily achievable, as depicted in [29], we are manually crafting a corresponding metamodel. We are not completely implementing PDDL 3.1, but a large subset that allows modeling of the required features of our use case. We include support for *typing*, *fluents*, *universal-preconditions*, *disjunctive-preconditions*, *existential-preconditions*, *equality*, *conditional-effects*, *duration-inequalities*, *durative-actions* and *numeric-fluents*. With all that, we can implement our envisioned domain and problem descriptions for production planning.

B. Tooling Support

To realize the proposed approach, the following tool decisions have to be taken: (i) the MDE framework and (ii) the PDDL solving environment.

1) Model-driven engineering: metamodels are created using the Eclipse Modeling Framework (EMF) and Ecore (as the meta-metamodel). For the generation of "flat" files (i.e., non-Ecore-model files, which are required at the interface between the modeling tools and the PDDL solver), we rely on the Xtend framework. With this basic tooling, it is very natural to use Java as the main programming language, which we have done.

2) PDDL solver: in order to create meaningful production plans, we have to consider production time. Thus, we are targeting PDDL solvers with support for *durative actions*, so that the production plan can be optimized towards the lowest lead time. We are using the LPG-td PDDL solver [30], which is described in detail in [31]. It supports durative actions, being the main requirement. Unfortunately, it does not support *conditional effects*, which would allow elegant formulations with respect to setting mutexes on certain items. Instead, we need to pass them as arguments to some of the actions, which significantly increases the search space. Switching to another PDDL solver with broader capabilities would be an option that is to be explored in future research endeavours.

C. Software Implementation

The software implementation comprises (i) a general workflow that orchestrates the diverse transformation and computation steps and (ii) the transformation tasks from IEC 62264 to PDDL and vice versa.

1) Workflow: we are implementing the workflow depicted in Fig. 1, which can be described in more detail as follows:

1) Initially, the production system model needs to be created, in terms of an IEC 62264 model instance. In our approach, this creation itself is a multi-step process that involves a model-to-model transformation from a proprietary transportation system model into an IEC 62264 model as well as handcrafting additional equipment, material and process segment information.
2) In a second step, one or more goal states have to be modeled in a separate IEC 62264 model.
3) In the first "Convert" task, the production system model and the goal state models are used as input to produce a PDDL domain model and one PDDL problem model per goal state model. The PDDL models are then serialized into "flat" PDDL files to be read by standard PDDL solvers. More information on this conversion process is given in the next subsection "Transformations" (Sec. IV-C2).
4) The "Calculate" task invokes (for each problem file) a PDDL solver process in order to compute a plan. The computed PDDL plans (or `null`, if no plan could be found) are then handed over to the next task.
5) The second "Convert" task parses the "flat" plan files into PDDL plan models and creates one IEC 62264 operations definition per plan. For each plan its action calls are converted operations segments and added to the operations definition. The operations definitions are then injected into the initial production system model.
6) The output of this workflow is an IEC 62264 model that contains one or more *automatically generated operations definitions*.

2) Transformations: due to space constraints, we are focusing on the first "Convert" task of the workflow; from an IEC 62264 model to PDDL models, which is the most important and innovative part of the proposed approach.

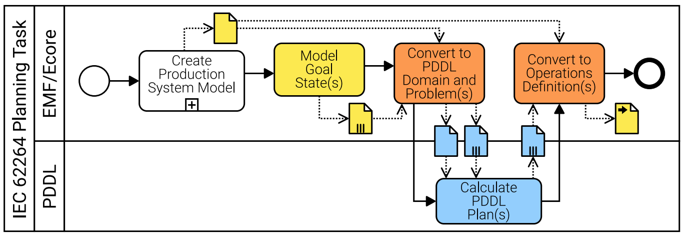
IEC 62264 Planning Task
EMF/Ecore
PDDL
Create Production System Model
Model Goal State(s)
Convert to PDDL Domain and Problem(s)
Convert to Operations Definition(s)
Calculate PDDL Plan(s)

Fig. 1: Implemented workflow (expressed in Business Process Model and Notation [32]). Yellow data objects (in the upper swimlane) depict Ecore models, blue data objects (on the swimlane border) represent "flat" files that need to be generated and parsed as part of the orange "Convert" tasks.

Since we are using durative actions, we need to discuss how the duration of the actions is determined. In our case, this is realized twofold: (i) pick-and-place processes are defining their duration using the standard duration field of IEC 62264 process segments, (ii) inventory movement duration is calculated depending on the length of the movement and the underlying transportation system topology. For instance, in monorail intra-logistics transportation systems, such as the one that we are using in our evaluation (cf. Sec. V), the moving shuttles need considerably more time to move over a switch than they would need for moving the same distance on a straight line. We are using PDDL functions to accommodate for this special timing behavior. Line 4 in Lst. 1 depicts the application of the `shuttle-time` function in the context of the `MoveShuttle` durative action. Concrete timing information is created in corresponding PDDL problem files in terms of function initializations, as depicted in Lst. 2.

In order to discuss the conversion from an IEC 62264 model to a PDDL domain file, we pick the `Assemble` process segment of our evaluation as our use case. This `Assemble` process segment describes the situation that a robot, which carries a "pickable" material lot, places this material lot on top of another one, thus performing an assembly.

The "header" of the corresponding durative action is depicted in Lst. 1. The process segment itself is transformed into a durative action (line 6), the duration attribute (specifying 5 s) is converted to a duration statement with a fixed length of `5` (line 11). Note, that all timing information is converted to seconds. The resource segment specifications of the process segment are transformed into parameters of the durative action (lines 7–10). Eventually given resource segment specification properties are used as pre- or postconditions, if they are tagged with the terms `pddl:pre` or `pddl:post`, respectively.

The conversion tasks adds another parameter to the durative action, which has not been specified in the IEC 62264 model: `SUBCARRIER`. It is added based on the knowledge that the production system of the use case is based on an intra-logistics system where the moving shuttles move between "positioning units" (PUs), which need to be physically locked in order to perform assembly operations—this issue is discussed below.

Lines 14–24 in Lst. 1 depicts the part of the precondition that deals with the question, whether a given `PICKABLE` material lot may be mounted onto a `BASE` material lot as part of the given `ASSEMBLY` material lot. This question is answered by looking for material definitions (lines 15–17) that are the material definitions of the given material lots (lines 19–21) and for which the corresponding relations are defined (lines 22–24). The used predicates do not change over time in the given use case, as such the time specification could be set to anything; we chose `start` (line 14).

Lines 26–33 of Lst. 1 clarifies how the previously discussed `SUBCARRIER` parameter is used: in line 27 it is made clear, that the following clauses are only required to hold if the `CARRIER` is of equipment class `Shuttle` (the other possibility would be `Table`). (i) the `SUBCARRIER` (which is the equipment which is underneath the `CARRIER`)

Listing 1: Excerpts of the generated PDDL domain file.

```
 1 ;... [several lines left out] ...
 2 (:durative-action MoveShuttle
 3   :parameters (?SHUTTLE ?FROM ?TO - Equipment)
 4   :duration (= ?duration (shuttle-time ?FROM ?TO))
 5 ;... [several lines left out] ...
 6 (:durative-action Assemble
 7   :parameters (?ROBOT ?CARRIER - Equipment
 8     ?PICKABLE ?BASE ?ASSEMBLY
 9     ?BASEPLATE - MaterialLot
10     ?SUBCARRIER - Equipment)
11   :duration (= ?duration 5)
12   :condition
13 ;... [several lines left out] ...
14   (at start
15     (exists ( ?AMD - MaterialDefinition
16       ?PMD - MaterialDefinition
17       ?BMD - MaterialDefinition)
18       (and
19         (MaterialDefined ?ASSEMBLY ?AMD)
20         (MaterialDefined ?PICKABLE ?PMD)
21         (MaterialDefined ?BASE ?BMD)
22         (MaterialDefinitionAssembly ?AMD ?PMD)
23         (MaterialDefinitionAssembly ?AMD ?BMD)
24         (Mountable ?PMD ?BMD))))
25 ;... [several lines left out] ...
26   (over all
27     (imply (EquipmentClassed ?CARRIER EC_Shuttle)
28       (and
29         (EquipmentClassed ?SUBCARRIER
30           EC_PositioningUnit)
31         (EquipmentLocation ?CARRIER ?SUBCARRIER)
32         (ReachesTo ?ROBOT ?SUBCARRIER)
33         (PositioningUnitLocked ?SUBCARRIER))))
34   :effect (and
35   (at end (not (RobotOccupied ?ROBOT)))
36   (at start (EquipmentMutex ?SUBCARRIER))
37   (at end (not (EquipmentMutex ?SUBCARRIER)))
38   (at end (MaterialLocation ?PICKABLE ?CARRIER))
39   (at end (not (MaterialLocation ?PICKABLE
40     ?ROBOT)))
41   (at end (Mounted ?PICKABLE ?BASE))
42   (at end (not (MaterialRaw ?PICKABLE)))
43   (at end (not (MaterialRaw ?BASE)))
44   (at end (MaterialLotAssembly ?ASSEMBLY
45   ?PICKABLE))
46   (at end (MaterialLotAssembly ?ASSEMBLY ?BASE))
47   (at end (Mounted ?ASSEMBLY ?BASEPLATE))
48   (at end (MaterialLocation ?ASSEMBLY ?CARRIER))))
```

Listing 2: Excerpt of the generated PDDL problem file, showing a function initialization for moving the shuttle from a positioning unit to a track switch.

```
(= (shuttle-time E_PositioningUnit-MachiningCenter-3
    ↪ E_TrackSwitch-Join-02) 3.04)
```

must be of equipment class `PositioningUnit`, (ii) the `CARRIER` must be located at this `SUBCARRIER`, (iii) the `ROBOT` performing the assembly must be able to reach to the `SUBCARRIER` and (iv) the `SUBCARRIER`, which is a `PositioningUnit`, must be locked.

We are skipping the rest of the preconditions due to limited space and now attend to the effect of the durative action, which is depicted in lines 34–48 of Lst. 1. Most of the effect is equal to the effect of the `Place` durative action that is defined in accordance with the equally named process segment. The `Assemble` specific effects are listed in lines 41–48: (i) the `PICKABLE` is now mounted on the `BASE` material, (ii) both materials are tagged as being no longer "raw", (iii) the `ASSEMBLY`, which is the item to be built, now physically consists of the two materials, (iv) and the `ASSEMBLY` is correctly located.

V. Evaluation

We are evaluating our approach based on a real industrial system that has been built at the Czech Institute of Informatics, Robotics, and Cybernetics (CIIRC) for studying Industry 4.0 scenarios, the *Industry 4.0 Testbed* (cf. Fig. 2).

A. Use Case

The layout of the testbed is presented in Fig. 3; it comprises an intralogistics monorail transportation system by montratec GmbH and four 6-axis robots. Each of the robot has a dedicated table where it can manipulate material. In addition, each robot can handle material that is located at adjacent PUs. A PU is a hardware device that is mounted to the monorail track—it provides a well-defined location for shuttles running on the monorail track to stop. Further, PUs can activate a hardware lock, physically holding a contained shuttle in place, which is required for precise material handling. This implies that the lock must be open in case a shuttle is entering or exiting a PU.

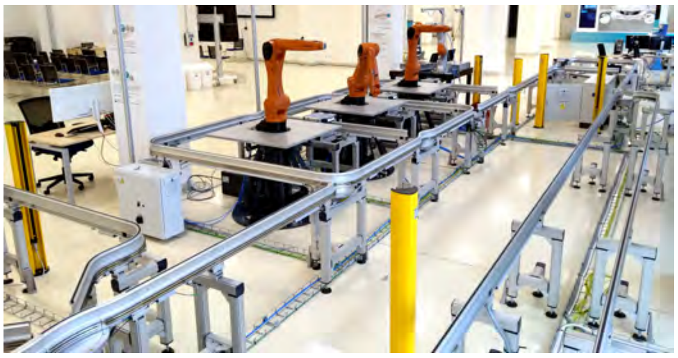
Fig. 2: The Industry 4.0 testbed located at CIIRC.

In order to simplify the transportation system for the planning task, we have abstracted it to only consist of positioning units and switches (nodes) that are connected via edges. The weights of the edges between transportation system switches and positioning units are computed from (i) the lengths of the participating elements that form the basis of this abstracted topology, (ii) the average shuttle movement speed, and (iii) a penalty of $2\,\mathrm{s}$ when traversing a switch. We are assuming a fixed average speed of the shuttles of $0.56\,\mathrm{m/s}$, which corresponds to the mean of the maximum ($0.92\,\mathrm{m/s}$) and the reduced running speed ($0.2\,\mathrm{m/s}$), as given in [33].

We are building a very simple LEGO pickup car as out production use case, comprising three parts, as depicted in Fig. 4. We consider each of the parts already pre-assembled by one of the three 6-axis robots (`R1-3`) in the production system; these parts are located on the tables (`T1-3`) in front of these robots. The goal is to have the finally assembled pickup ready for take-away at table `T10`. This goal statement is expressed

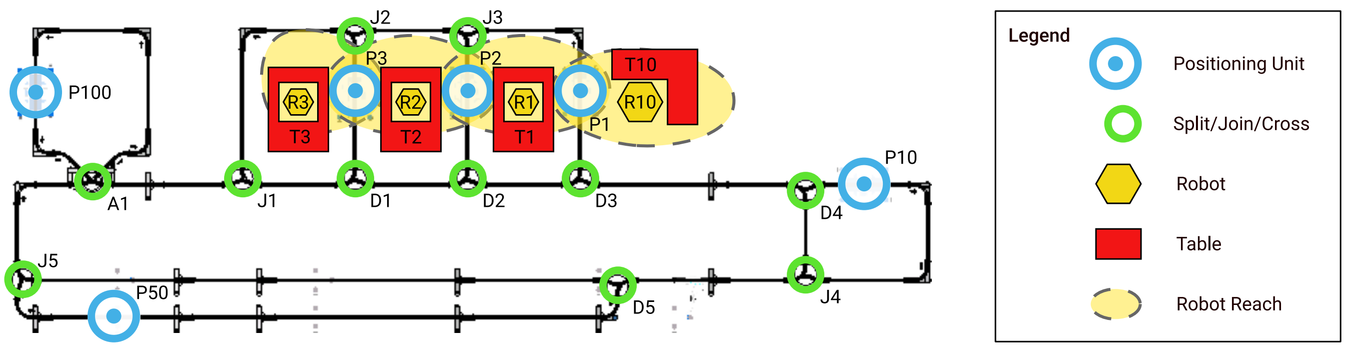

Fig. 3: Layout of the Industry 4.0 Testbed. Blue nodes depict positioning units, green nodes depict switches. The working area of each robot is marked with a yellow ellipse.

in terms of a small IEC 62264 model that is interpreted in the "Convert" task that produces the domain and problem files.

B. Production System Variations

In our evaluation, we are testing the feasibility of our approach with respect to the generation of PDDL code that can be solved by off-the-shelf PDDL solvers in due time. In order to assess the behavior in different scenarios, we create three variants of the production system and product:

1.1 Variant *Pickup* (PP): the production system as depicted in Fig. 3, and the product as depicted in Fig. 4. The transportation system comprises three shuttles, that are initially located at `P100`, `P10`, and `P50`.

1.2 Variant *One Shuttle* (OS): the production system as depicted in Fig. 3, and the product as depicted in Fig. 4. Only one shuttle is available, it is located at `P100`.

1.3 Variant *Large Pickup* (LP): the production system as depicted in Fig. 3, and the product a variation of Fig. 4: a large pickup consists of a chassis with two cabins and two bodies. Shuttle configuration as in (1).

In order to further test the performance of the PDDL solvers, we generate three different kinds of PDDL code with respect to the durative aspect of the planning problem:

2.1 Variant *Durative Actions* (DA): we are rendering PDDL code containing durative actions (Sec. IV).

2.2 Variant *Non Durative* (ND): same as (1), but normal actions are used instead of durative ones. The duration is encoded as cost of the action.

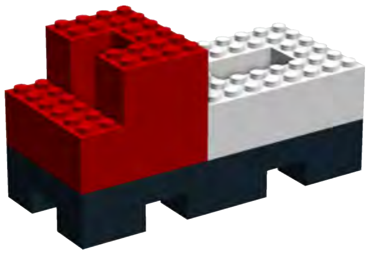

Fig. 4: The pickup model to be built, made up from three prebuilt sub-modules: (black) chassis, (red) cabin, and (white) body. Model is used in the "PP" and "OS" use cases.

2.3 Variant *No Cost* (NC): same as (2), but durations/cost are not encoded at all; instead the number of actions is used to assess the quality of a plan.

Further, we create two variants of the PDDL domain description with respect to the locking and unlocking of the PUs:

3.1 Variant *Verbose* (VB): PUs can be physically locked. In a direct conversion from IEC 62264 to PDDL, locking and unlocking are two distinct actions that can be triggered anytime.

3.2 Variant *Optimized* (OP): unlocking stays a separate action, but locking is built into a variation of the `move shuttle` action: `move shuttle and lock`. This speeds up the solving process, as it prevents the solver from attempting lock-unlock-lock-unlock-... chains.

After the IEC 62264 models are converted to PDDL files in different variations, they are fed into the PDDL solvers. Durative action (DA) variations are solved by LPG-td, for the non-durative actions we are using Fast Downward [34], as it performs better than LPG-td in all non-durative scenarios. Fast Downward is configured with A* as search engine, and four evaluator settings are competing against each other: (1) *astar(ipdb)*, (2) *astar(ff, ipdb)*, (3) *astar(sum([cea, ff]), ipdb)*, and (4) *astar(sum([cea, g]), ipdb)*.

C. Results and Findings

Computation is realized on a standard PC from 2008 @2.83 GHz with 4 GB RAM and 1369 GB swap space (of which 345 GB on SSD and 1024 GB on HDD) for the `DA` use cases and on a standard laptop from 2013 @2.4 GHz with 16 GB RAM for the `ND` and `NC` use cases. This makes the results not directly comparable with respect to the search time, but the massive amount of memory required for the `DA` use cases implied heavy use of swap space, which slowed down the computation process, making the time-related results not directly comparable anyway.

Tbl. I depicts the results from running the different variations on the respective solvers. The first column, *Scenario*, depicts the selected variant composition—for instance, `PP-DA-OP` means "Pickup" with "Durative Actions" and "Optimized" locking code. Subsequent rows in the table highlight what has changed in comparison with the previous row. The second column, *Evaluator*, shows which evaluator setting for the A* search has provided the best result—in case multiple evaluator settings have provided equally costly results, the one that reached it in the shortest time is selected. Durative action (DA) use cases are run on LPG-td with standard evaluator, depicted as "–". The *Time* column shows the time (in seconds) that was required to compute the plan, *Memory* depicts the peak memory (in megabytes) that was consumed. *Cost* shows the cost of the plan. In case of durative (DA) and non-durative (ND) use cases, this corresponds to the duration of executing the plan in seconds, in case of non-cost (NC) use cases, it shows the number of steps.

An example for a computed temporal PDDL plan is given in Lst. 3: it resembles the best solution that could be computed in a 4 hours time frame for scenario `PP-DA-VB`. This particular plan was found after 4617.05 s.

The following findings can be obtained from the results:

1) Approach: using MDE techniques proved useful in our use cases. Meaningful PDDL representations could be found, and they could be used to run standard PDDL solvers upon.

2) Durative Actions: in general, temporal PDDL solvers can be used for the planning of small problems, but they are slow and memory demanding. Instead of that, a more feasible solution would be to use a PDDL planner for the planning of more simplified problems (without durative actions) and then use some scheduling algorithm to perform a given set of actions (already found by planner) using a limited number of resources in a limited amount of time as it is recommended in [35]. Another approach would be to solve the planning problem in a sequential fashion and then post-process the plan by parallelizing subsequent actions.

3) Optimization: smart design of actions in the PDDL domain can significantly increase computational efficiency. Except for one case, where a fast yet rather approximate evaluator setting successfully found the best plan for the given problem, the optimized `OP` variants are usually less memory demanding and much faster for the tested sequential planner. The temporal planner was not strongly affected by these optimization efforts.

4) Summary: usage of sequential PDDL planners seems to be feasible on small and middle sized production lines but needs to be further tested on larger use-cases. Temporal planning using durative actions, however, seems to be unfeasible, even for small scenarios.

Scenario	Evaluator	Time (s)	Memory (MB)	Cost
`PP-DA-OP`	–	4865.03	582 800.00	84.49
`-VB`	–	4617.05	583 400.00	83.49
`-ND-OP`	1	4.18	70.30	153.00
`-VB`	2	30.80	90.60	153.00
`-NC-OP`	2	1.31	36.00	31.00
`-VB`	4	0.73	35.54	34.00
`OS-DA-OP`	–	937.90	210 300.00	125.37
`-VB`	–	962.16	210 200.00	123.87
`-ND-OP`	1	0.02	34.22	153.00
`-VB`	2	0.91	36.35	153.00
`-NC-OP`	1	0.02	34.22	32.00
`-VB`	2	0.91	36.09	35.00
`LP-DA-OP`	–	14 232.18	1 130 800.00	105.83
`-VB`	–	12 762.90	1 131 300.00	104.83
`-ND-OP`	2	12.62	55.06	240.00
`-VB`	4	174.88	185.25	240.00
`-NC-OP`	2	5.63	42.01	44.00
`-VB`	4	19.13	52.35	47.00

TABLE I: Performance of PDDL solvers in different scenarios.

VI. Conclusion

We have presented an approach for the automated generation of production plans based on structural descriptions of the production system and the products to be produced thereon. The production system model was created as an instance of an industry standard based metamodel: IEC 62264. The model was transformed into PDDL format, production plans have been generated and fed back into the production system model.

We have evaluated our approach on real hardware, the CIIRC Industry 4.0 Testbed, in various scenarios, incorporating several variations of PDDL code generation and compared the performance of PDDL solvers on these variants. The main findings are: (i) quality of the generated code matters and (ii) complex PDDL features such as durative actions demand *extremely* much from the underlying processing hardware, especially with respect to available memory.

Future work will include (i) further evaluation of durative actions on processing hardware comprising more physical RAM in order to better compare search speed and involvement of different temporal solvers and (ii) setting up a workflow for the post-processing of sequential plans for *ex post* action parallelization.

Acknowledgment

Financial support by the Austrian Fed. Ministry for Digital and Economic Affairs and the Nat. Foundation for Research, Technology and Development is gratefully acknowledged, as well as support by the DAMiAS project funded by the Technology Agency of the Czech Republic, by the H2020 project DIGICOR, and by the OP VVV DMS project Cluster 4.0.

References

[1] J. Davis, T. Edgar, J. Porter, J. Bernaden, and M. Sarli, "Smart manufacturing, manufacturing intelligence and demand-dynamic performance," *Computers & Chemical Engineering*, vol. 47, pp. 145–156, 2012.
[2] R. Anderl, "Industrie 4.0—advanced engineering of smart products and smart production," in *Proc. 19th Int. Seminar on High Technology*, 2014.
[3] A. Kusiak, "Smart manufacturing," *International Journal of Production Research*, vol. 56, no. 1–2, pp. 508–517, 2018.
[4] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *Int. J. Advanced Manufacturing Tech.*, vol. 94, no. 9–12, pp. 3563–3576, 2018.
[5] S. Kent, "Model driven engineering," in *Proc. 3rd Int. Conf. on Integrated Formal Methods (IFM)*, ser. LNCS, vol. 2335, 2002, pp. 286–298.
[6] D. C. Schmidt, "Model-driven engineering," *COMPUTER*, vol. 39, no. 2, pp. 25–31, 2006.
[7] J. Cadavid, M. Alférez, S. Gérard, and P. Tessier, "Conceiving the model-driven smart factory," in *Proc. Int. Conf. on Software and System Processes (ICSSP)*, 2015, pp. 72–76.
[8] E. A. Silver, D. F. Pyke, and R. Peterson, *Inventory Management and Production Planning and Scheduling*, 3rd ed., 1998.

Listing 3: Generated production plan for use case PP-DA-VB, including timing information (in seconds). Line format: <start>: (<action> <parameters>*) [<duration>].

```
 0.0002: (MoveShuttle E_Shuttle-1 E_PositioningUnit-IO-1 E_TrackSwitch-Arena-01) [4.18]
 0.0002: (MoveShuttle E_Shuttle-3 E_PositioningUnit-Buffer E_TrackSwitch-Join-05) [9.69]
 0.0002: (Pick E_Robot-2 E_Table-2 ML_Body1 ML_LegoBaseplate-Table-2) [5.00]
 0.0002: (Pick E_Robot-3 E_Table-3 ML_Chassis1 ML_LegoBaseplate-Table-3) [5.00]
 4.1805: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Arena-01 E_TrackSwitch-Join-01) [5.92]
 9.6905: (MoveShuttle E_Shuttle-3 E_TrackSwitch-Join-05 E_TrackSwitch-Arena-01) [6.21]
10.1007: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Join-01 E_TrackSwitch-Divide-01) [4.97]
15.0710: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Divide-01 E_PositioningUnit-MachiningCenter-3) [3.04]
15.9007: (MoveShuttle E_Shuttle-3 E_TrackSwitch-Arena-01 E_TrackSwitch-Join-01) [5.92]
17.6115: (LockPositioningUnit E_PositioningUnit-MachiningCenter-3) [0.50]
18.1117: (Place E_Robot-3 E_Shuttle-1 ML_Chassis1 ML_LegoBaseplate-Shuttle-1 E_PositioningUnit-MachiningCenter-3) [5.00]
21.8210: (MoveShuttle E_Shuttle-3 E_TrackSwitch-Join-01 E_TrackSwitch-Divide-01) [4.97]
23.1122: (UnlockPositioningUnit E_PositioningUnit-MachiningCenter-3) [0.50]
23.6125: (MoveShuttle E_Shuttle-1 E_PositioningUnit-MachiningCenter-3 E_TrackSwitch-Join-02) [3.04]
26.6528: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Join-02 E_TrackSwitch-Join-01) [8.89]
26.7912: (MoveShuttle E_Shuttle-3 E_TrackSwitch-Divide-01 E_TrackSwitch-Divide-02) [4.97]
31.7615: (MoveShuttle E_Shuttle-3 E_TrackSwitch-Divide-02 E_PositioningUnit-MachiningCenter-2) [3.04]
34.3020: (LockPositioningUnit E_PositioningUnit-MachiningCenter-2) [0.50]
34.8023: (Place E_Robot-2 E_Shuttle-3 ML_Body1 ML_LegoBaseplate-Shuttle-3 E_PositioningUnit-MachiningCenter-2) [5.00]
35.5430: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Join-01 E_TrackSwitch-Divide-01) [4.97]
39.8025: (Pick E_Robot-1 E_Shuttle-3 ML_Body1 ML_LegoBaseplate-Shuttle-3 E_PositioningUnit-MachiningCenter-2) [5.00]
40.5133: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Divide-01 E_TrackSwitch-Divide-02) [4.97]
45.4835: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Divide-02 E_TrackSwitch-Divide-03) [4.97]
50.4538: (MoveShuttle E_Shuttle-1 E_TrackSwitch-Divide-03 E_PositioningUnit-MachiningCenter-1) [3.04]
52.9943: (LockPositioningUnit E_PositioningUnit-MachiningCenter-1) [0.50]
53.4945: (Pick E_Robot-10 E_Shuttle-1 ML_Chassis1 ML_LegoBaseplate-Shuttle-1 E_PositioningUnit-MachiningCenter-1) [5.00]
58.4948: (Place E_Robot-1 E_Shuttle-1 ML_Body1 ML_LegoBaseplate-Shuttle-1 E_PositioningUnit-MachiningCenter-1) [5.00]
58.4948: (Place E_Robot-10 E_Table-10 ML_Chassis1 ML_LegoBaseplate-Table-10) [5.00]
63.4950: (Pick E_Robot-10 E_Shuttle-1 ML_Body1 ML_LegoBaseplate-Shuttle-1 E_PositioningUnit-MachiningCenter-1) [5.00]
63.4950: (Pick E_Robot-1 E_Table-1 ML_Cabin1 ML_LegoBaseplate-Table-1) [5.00]
68.4953: (Assemble E_Robot-10 E_Table-10 ML_Body1 ML_Chassis1 ML_Pickup1 ML_LegoBaseplate-Table-10 E_Table-10) [5.00]
68.4953: (Place E_Robot-1 E_Shuttle-1 ML_Cabin1 ML_LegoBaseplate-Shuttle-1 E_PositioningUnit-MachiningCenter-1) [5.00]
73.4955: (Pick E_Robot-10 E_Shuttle-1 ML_Cabin1 ML_LegoBaseplate-Shuttle-1 E_PositioningUnit-MachiningCenter-1) [5.00]
78.4958: (Assemble E_Robot-10 E_Table-10 ML_Cabin1 ML_Chassis1 ML_Pickup1 ML_LegoBaseplate-Table-10 E_Table-10) [5.00]
```

[9] E. A. Silver, D. F. Pyke, and D. J. Thomas, *Inventory and Production Management in Supply Chains*, 4th ed., 2017.

[10] J.-H. Chen and S.-Y. Ho, "A novel approach to production planning of flexible manufacturing systems using an efficient multi-objective genetic algorithm," *Int. J. Machine Tools and Manufacture*, vol. 45, no. 7, 2005.

[11] H. Akillioglu and M. Onori, "Evolvable production systems and impacts on production planning," in *IEEE Int. Symp. on Assembly and Manufacturing (ISAM)*, 2011.

[12] L. D. Xu, C. Wang, Z. Bi, and J. Yu, "AutoAssem: An automated assembly planning system for complex products," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 3, pp. 669–678, 2012.

[13] L. Monostori, J. Váncza, and S. R. T. Kumara, "Agent-based systems for manufacturing," *CIRP Annals*, vol. 55, no. 2, pp. 697–720, 2006.

[14] W. Lepuschitz, B. Groessing, and M. Merdan, "Automation agents for controlling the physical components of a transportation system," in *Industrial Agents*, 2015, pp. 323–339.

[15] T. Niemueller, T. Hofmann, and G. Lakemeyer, "CLIPS-based execution for PDDL planners," in *Proc. WS on Integrated Planning, Acting, and Execution (IntEx)*, 2018.

[16] A. Rogalla, A. Fay, and O. Niggemann, "Improved domain modeling for realistic automated planning and scheduling in discrete manufacturing," in *Proc. 23rd IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2018, pp. 464–471.

[17] American National Standards Institute (ANSI), *Enterprise-Control System Integration Part 1: Models and Terminology*, ANSI Standard, 2000.

[18] B. Wally, C. Huemer, and A. Mazak, "Entwining plant engineering data and ERP information: Vertical integration with AutomationML and ISA-95," in *Proc. 3rd IEEE Int. Conf. on Control, Automation and Robotics (ICCAR)*, 2017.

[19] B. Wally, "Provisioning for MES and ERP," TU Wien and AutomationML e.V., Application Recommendation, 2018.

[20] B. Wally, C. Huemer, A. Mazak, and M. Wimmer, "IEC 62264-2 for AutomationML," in *Proc. 5th AutomationML User Conference*, 2018.

[21] International Electrotechnical Commission (IEC), *Engineering data exchange format for use in industrial automation systems engineering—Automation Markup Language—Part 1: Architecture and general requirements*, 2018.

[22] B. Wally, C. Huemer, and A. Mazak, "A view on model-driven vertical integration: Alignment of production facility models and business models," in *Proc. 13th IEEE Int. Conf. on Automation Science and Engineering (CASE)*, 2017.

[23] International Electrotechnical Commission (IEC), *Enterprise-control system integration—Part 2: Objects and attributes for enterprise-control system integration*, 2013.

[24] ——, *Enterprise-control system integration—Part 4: Object model attributes for manufacturing operations management integration*, 2015.

[25] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, *PDDL—The Planning Domain Definition Language*, APIS-98 Planning Competition Committee, 1998.

[26] D. L. Kovacs, "Complete BNF description of PDDL 3.1," Department of Measurement and Information Systems, Budapest University of Technology and Economics, Language Specification, 2011.

[27] D. E. Knuth, "Backus normal form vs. backus naur form," *Commun. ACM*, vol. 7, no. 12, pp. 735–736, 1964.

[28] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, "Model-driven architecture," in *Advances in Object-Oriented Information Systems*, J.-M. Bruel and Z. Bellahsene, Eds., 2002, pp. 290–297.

[29] A. Bergmayr and M. Wimmer, "Generating metamodels from grammars by combining transformation and by-example techniques," in *Proc. 1st Int. WS on MDE By Example (MDEBE)*, 2013.

[30] A. Gerevini, A. Saetti, I. Serina, and P. Toninelli, "Planning in PDDL 2.2 domains with LPG-TD," in *Booklet of the Int. Planning Comp.*, 2004.

[31] A. Gerevini, A. Saetti, and I. Serina, "An approach to temporal planning and scheduling in domains with predictable exogenous events," *Journal of Artificial Intelligence Research*, vol. 25, pp. 187–231, 2006.

[32] Object Management Group (OMG), *Business Process Model and Notation (BPMN)*, Specification formal/13-12-09, Rev. 2.0.2, 2013.

[33] montratec GmbH, "montrac design guide 2017/2018," Manual, 2018.

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

[35] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, ser. Artificial Intelligence. Morgan Kaufmann, 2004.