IoT-based Digital Twins Orchestration via Automated Planning for Smart Manufacturing

Giuseppe De Giacomo[1] , **David Ghedalia**[2] , **Donatella Firmani**[3] , **Francesco Leotta**[1] , **Federica Mandreoli**[2] and **Massimo Mecella**[1]

[1]Sapienza Università di Roma, Italy
[2]Università degli Studi di Modena e Reggio Emilia, Italy
[3]Roma Tre University, Italy

Abstract

Recently, the concept of Digital Twin (DT), meant as a digital replica of a physical asset, emerged in smart manufacturing applications. Due to the growth of the Internet-of-Things (IoT), an increasing number of software solutions implementing DTs appeared. Nevertheless, the potential of DTs has been only partially explored. In particular, the current employment of DTs is mainly limited to simulation, whereas they can be also employed for enacting and monitoring manufacturing processes. In this paper, we show how automated planning can be employed in such a scenario to synthesize a manufacturing process according to a specific production goal and to adapt to failures. We provide a running implementation based on a commercial DT platform and a state-of-the planner, and propose a research outlook towards a more general employment of such an approach.

1 Introduction

The continuous evolution of digital technologies applied to the more traditional world of industrial automation led to *smart manufacturing*, which envisions *production processes* subject to continuous monitoring and able to *dynamically respond to changes* that can affect the product life cycle at any stage (*resilient factory*). In the so-called digital factory, the involved actors that can fall in different categories, being humans (i.e., final users or participants in the production process), information systems or industrial machines, must be able to communicate and interact at the digital level while operating in the physical world. The Industrial Internet-of-Things (IIoT) represents one of the technological pillars to this end. By covering the domains of machine-to-machine (M2M) and industrial communication technologies, IIoT is the computing concept that enables efficient interaction between the physical world and its digital counterpart [Sisinni *et al.*, 2018]. Thanks to IIoT, physical entities can have a faithful representation in the digital world, usually defined as *digital twins*. Digital Twins (DTs) are up-to-date digital descriptions of physical objects and their operating status [Qi *et al.*, 2019]. Modern information systems and industrial machines may natively come out with their digital twin; in other cases, especially when the approach is applied to already established factories and production processes, digital twins are obtained by wrapping actors that are already in place.

DTs are commonly known as a key enabler for the digital transformation in manufacturing. Nevertheless, their potential has been only partially explored. In particular, the current employment of DTs is mainly limited to simulation, whereas they can be also employed for monitoring and enacting manufacturing processes. In this paper we advocate that is possible, by leveraging techniques from automated planning, to automatically orchestrate DTs in order to reach specific production goals and respecting expected Key Performance Indicators (KPIs) [Catarci *et al.*, 2019], by totally or partially replacing rules hand-made by human experts. AI planning has indeed proven its potential in conjunction with different kinds of business processes (e.g., emergency management processes [Marrella *et al.*, 2016]), and could have a huge impact on manufacturing processes in digital factories too, by allowing automatic recovery and optimization, and even automatic orchestration of the intermediate steps for achieving a production goal, thus achieving manufacturing *resilience*.

Contribution: The availability of DTs, and the possibility to describe them in terms of their software interfaces via knowledge representation (KR) approaches, opens up to new scenarios where DTs are orchestrated automatically in order to pursue specific goals and perform adaptation to failures in the production chain. In this paper, we show how a standard DT description language can be easily extended for DT orchestration and we present ALTO - AgiLe Twin Orchestrator, a system prototype that leverages a DT platform and automated planning to compute and continuously monitor sequences of DT actions ensuring to achieve the given production goal. The efficiency and scalability of the proposed approach is proven against a synthetic dataset and a virtual machine with a running version of our prototype and all the code used in our experiments is available to the reader [1].

This paper represents a first step towards a more general employment of this approach. Describing software interfaces of complex machines may require the employment of advanced KR approaches to be matched with planning techniques complexity, which in turn influences efficiency and

[1]See https://drive.google.com/drive/folders/1gKiSIKFQz3NKRHCWLUcU5b3XHoLGBt2h?usp=sharing

scalability. As a consequence, our aim is also to outline a research outlook for the involved AI research communities.

Outline. Section 2 presents an overview of DTs in the recent literature. Section 3 describes the architecture of the system prototype ALTO. Performances are discussed in Section 4. Finally, Section 5 concludes the paper with overall considerations and a possible research outlook.

2 Digital Twins in Smart Manufacturing

Digital Twins, DTs, are today among the most promising technologies for smart manufacturing and Industry 4.0. Even though there is no common understanding concerning this term, different digital twin definitions agree on features such as *(i)* connectivity, i.e., the ability to communicate with other entities and DTs, *(ii)* autonomy, i.e., the possibility for the DT to live independently from other entities, *(iii)* homogeneity, i.e., the capability, strictly connected to the autonomy, that allows to use the same DT regardless of the specific production environment, *(iv)* easiness of customization, i.e., the possibility to modify the behavior of a physical entity by using the functionalities exposed by its DT, and *(v)* traceability, i.e., the fact that a DT leaves traces of the activity of the physical entity.

We review recent literature based on four research questions: *(R1)* What are the opportunities and challenges for Industrial IoT? *(R2)* What are the modelling efforts towards DT interoperability? *(R3)* What are the DT-based architectural paradigms for smart manufacturing? *(R4)* What are the available software solutions?

(R1) Industrial IoT and DTs. The IoT paradigm has made its way into the industry marketplace. The recent paper [Sisinni *et al.*, 2018] clarifies the related concepts of IoT, Industrial IoT, and Industry 4.0, highlighting challenges in energy efficiency, interoperability, security, privacy and real-time performance. The paper [Tao *et al.*, 2018b] focuses on the review of DT applications in industry, including production, health management and product design, outlines current challenges and future work. The authors of [Tao *et al.*, 2019] provides a review of DTs in Industrial IoT from a different perspective, focusing on their origin, engineering practices and development. The work in [Qi *et al.*, 2019] summarizes the current state of the art of technologies enabling DTs to mirror the different facets of industrial processes and products. Finally, the paper [Kamath *et al.*, 2020] explores results and limitations of current open-source IoT platforms to achieve DT functionalities, such as virtual representation, real-time data acquisition and analytics.

(R2) Models for DTs. Authors in [Ameri and Sabbagh, 2016] describe the Manufacturing Service Description Language (MSDL), that is, a descriptive ontology for representing capabilities of manufactures with a service-oriented paradigm. MSDL decomposes the manufacturing capabilities into different levels of abstraction (e.g., supplier-level, machine-level and process-level) and enables automated supplier discovery in distributed environments. The work of [Tao and Zhang, 2017] provides a conceptual model and specific operation mechanisms for a DT *shop-floor*, that is, a basic unit of manufacturing, where data from both physical and virtual sides as well as the fused data are provided to drive all the steps of the production process. The work in [Schroeder *et al.*, 2016] and the more recent [Bao *et al.*, 2018] propose the usage of the popular Automation Markup Language (AutomationML) to model DT's physical and logical components with an object-oriented paradigm, and discuss its application towards the data exchange. Finally, authors in [Zehnder and Riemer, 2018] introduce a semantic representation for industrial data streams in DTs and the authors of [Tao *et al.*, 2018a] discuss more in general how to generate and use converged DT data to drive product design and manufacturing. For our solution, we will not enforce any specific modeling language, relying on the very generic description language provided by the Bosch IoT Things platform.

(R3) DT-based architectures. The group around Reference Architecture Model Industry 4.0 – RAMI – developed a first implementation of the DT, the open Asset Administrative Shell, and a conceptual architecture [Hankel and Rexroth, 2015]. RAMI is a three dimensional reference architectural framework leveraging EU guidelines and explicitly defined as a Service Oriented Architecture (SOA), in order to realize composition and interoperability for adaptable applications. More recently, authors in [Weber *et al.*, 2017] provide an overview of Industry 4.0 features in multiple reference architectures and develops a maturity model for software architectures for data-driven manufacturing. Authors in [Wang and Wang, 2018] introduce a SOA architecture for the waste electrical and electronic equipment recovery in which both the electronic device and its DT are considered as the services. Finally, authors in [Brosinsky *et al.*, 2018] introduce the concept of a DT-based control center architecture based on a dynamic simulation engine and [Bicocchi *et al.*, 2019] propose a methodology to coordinate the actors of a smart factory based on Business Process Management.

(R4) DT platforms. There is a wide variety of software solutions for implementing DTs. We consider three representative solutions among those offering cooperation functionalities between twins: Azure Digital Twins, Eclipse Ditto and Bosch IoT Things. Azure Digital Twins creates models of physical environments with the notion of *spatial intelligence graphs*. Spatial intelligence graphs use different object models to describe domain-specific concepts including Spaces (i.e., virtual or physical locations), Devices, Sensors and space-occupant Users to send signals to devices. Spatial intelligence graph can be managed via REST APIs. In Eclipse Ditto, each DT is represented as a *Thing* entity with a simple JSON-based description language. Features of such Thing can either represent states and properties or DT functionalities. Physical devices can update their corresponding Thing entity by sending commands. If a command is successful, the Thing entity changes, and an event is generated. Applications can request operations to devices by the mean of messages. Eclipse Ditto provides a REST-like HTTP API to manage DTs and communicate with their real-world counterparts, a built-in search engine to find Things and a policy system to configure fine-grained access control. Bosch IoT Things is a cloud DT platform based on Eclipse Ditto.

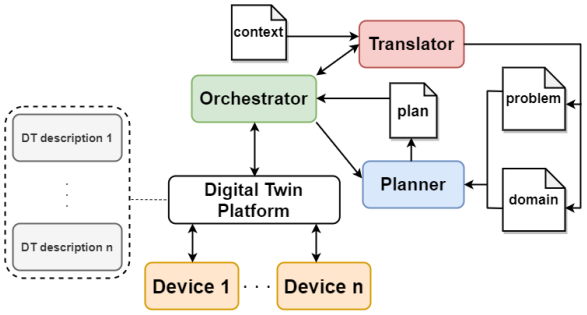


Figure 1: The architecture of ALTO.

The main differences with Ditto are: built-in integration with Eclipse IoT projects such as Eclipse Hono and Eclipse hawkBit; the possibility to add functionalities of other services of the Bosch IoT suite (e.g., Bosch IoT Permissions); and built-in cloud platforms functionalities (e.g., security).

3 ALTO Architectural Framework

In this paper, we propose a resilient and scalable approach for the agile orchestration of DTs aimed at achiving a production goal and to adapt to failures. To this end, we introduce the architectural framework of ALTO (AgiLe Twin Orchestrator). ALTO relies on Bosh IoT as reference DT platform and leverages a state-of-the-art planning method to compute a sequence of DT actions leading to the goal. Actions are dispatched to the DTs through Bosh IoT and are executed by the physical devices while the production process is monitored to detect failures and possibly generate recovery plans.

Example 1 ("MyFactory") *Let us introduce a simple example that will be used in the rest of the paper to illustrate our approach. Consider a smart factory with three manufacturing devices, dubbed a 3D printer, a laser cutter and an assembler, each associated to a DT. The basic production goal of MyFactory is to "manufacture a single product and place it in the specific storage location". The resilient plan will correspond to a main production plan where products are manufactured via 3D-printing, and – in case of printer failure – a recovery plan where products are manufactured by cutting and assembling two pieces ("typeA" and "typeB").*

ALTO main components are depicted in Figure 1. Besides *(i)* the DT platform to wrap and mediate the available IoT manufacturing devices and *(ii)* the planner to generate the plan and adapt to faulty instances, the other components are *(iii)* an *orchestrator* to enact plans and possibly managing exceptions, (iv) a *translator* to enable automatic interaction between the DT platform and the planner via the orchestrator. Devices can be connected to the DT platform through MQTT (Message Queue Telemetry Transport), a lightweight messaging protocol for the IoT supported by many DT platforms. The orchestrator can interact with the DT platform through a WebSocket API, which allows to receive change notifications.

In order to leverage existing planning systems, the core component of ALTO is the *translator* that converts the DTs' descriptions and the context, i.e. the production goal and the current production environment that is usually monitored by sensors, into a planning problem stored in the Planning Domain Definition Language (PDDL) domain and problem files.

A *planning problem* [Ghallab *et al.*, 2016] (A, I, γ, O) consists of a set of state variables A, a description (i.e., a valuation over A) of the initial state of the system I, a goal γ represented as a formula over A, and a list of operation (or, actions) O over A that can lead to state transition. States variables A and action O constitute the *planning domain* of the planning problem. An *automated planner* aims at finding automatically a sequence of operations that, applied to the initial state I, leads to a state s such that $s \models \gamma$. In this framework, an operation $o \in O$ is defined as a tuple $o = \langle \chi, e \rangle$ where χ is the *precondition* and e is the *effect*, both expressed as conjunctions of literals (positive or negated atomic sentences) over A. χ defines the states in which o can be executed, whereas e defines the result of executing o. For each effect e and state s, the change set $[e_s]$ is a set of state variables whose value is modified upon the operation. The *successor state* of s with respect to the action o is the state s' with $s' \models [e_s]$ and $s'(v) = s(v)$ for all state variables v not mentioned in $[e_s]$.

The Planning Domain Definition Language (PDDL) [Fox and Long, 2003] is a standard language to describe planning domains and problems. Each planner supports specific features of the different versions of PDDL. The current prototype of ALTO is based on the FastDownward planner [Helmert, 2006], which supports PDDL 2.2 level 1. Currently, we only employ deterministic features of PDDL.

Input. The actual input to the ALTO prototype are a context file expressed in PDDL and a set of Bosh IoT DT descriptions. Considering our example, the context file contains a conjunction of PDDL atoms that expresses the basic production goal and that must be true at the end of the process.

Each DT in Bosch IoT Things is represented as a *Thing* entity and described in JSON format. Figure 3 shows a possible description of the 3D printer in our MyFactory example. The different aspects of the DT are represented as attributes (lines 3-4) and features (lines 5-31). *Features* can either represent a state with properties (e.g., `status`) or a functionality of a DT (e.g., `printing`), while *attributes* (e.g., `type`) hold values that do not change, or that change less frequently than the Features property values.

We assume the context file to be static. Values of state fields in the DT descriptions, instead, may vary over time (e.g., `at` in line 23 of Figure 3) and are therefore monitored during the production process.

Workflow. The main steps of the ALTO workflow are below.

(1) The orchestrator retrieves the DT descriptions from the DT platform and feeds them to the translator which takes in input the context file and generates the PDDL files for the planning *domain* and *problem*.

(2) The orchestrator gets the domain and problem files and submits them to the planner. Upon planning completion, the orchestrator gets the resulting plan file for downstream execution. Figure 2a shows a sample of such a file for MyFactory. Note that this plan, i.e., the main plan, uses the 3D printer, as it is the cheapest way to achieve the production goal.

```
(print pr p pos32)                                  1
(move forklift p pos32 pos33)                       2
```

(a) Main plan

```
(move forklift ta pos11 pos21)                      1
(move forklift tb pos11 pos21)                      2
(cut laser ta pos21)                                3
(move forklift ta pos21 pos23)                      4
(cut laser tb pos32)                                5
(move forklift tb pos21 pos23)                      6
(assemble assembler ta tb p pos23)                  7
(move forklift p pos23 pos33)                       8
```

(b) Recovery plan

Figure 2: Plans in the MyFactory example

(3) The orchestrator sends the sequence of actions in the plan to the corresponding DTs for execution, one by one, via the API of the DT platform. For example, the action `(move forklift p pos32 pos33)` becomes a message in Bosch IoT Things with subject `move`, the ID of its recipient is `forklift`, while `pos32` and `pos33` form the message body. Between an action and the next one, the orchestrator checks the state fields of the involved DTs in order to verify that the output is equal to the expected one. Let us assume, for instance, that operation (1) of the main plan times out. Then the orchestrator will set the DT status of the 3D printer (line 29 in Figure 3) to `broken`, update the problem and domain files and ask the planner to generate a recovery plan that starting from the current state of the system can lead to the production goal. Figure 2b shows an example of recovery plan, involving the cutter and the assembler in place of the 3D printer.

Extending the DT description language. In order to enable automatic translation of DT descriptions to PDDL, we introduce an extension of the description language of the DT platform. Such an extension allows indeed the identification of the features that constitute the planning domain, i.e., the set of *state variables* and *actions*. Field `type` is added to the feature properties of a DT to this end: for instance, `type: state` at line 25 of Figure 3 denotes that the feature `at` described in lines 23-28 (i.e., the device location) is a state variable; whereas `type: operation` at line 8 denotes that the feature `printing` described in lines 6-22 (i.e., the printing operation of the 3D printer) is an action.

State variables describe the properties of the physical device that are gathered from the sensor data and can evolve over time. The JSON object `value` (e.g., line 26) describes the state variable current value and value type.

Actions represent the operations that the device is able to perform in the planning domain. The field `command` (e.g. `print` in line 9) stores the operation name, required by the DT platform to invoke it. Actions are expressed in terms of parameters, preconditions and effects.

- the field `parameters` represents the typed variables that must be given as input: in our example, one parameter of type `product` and another of type `position`, representing the target position x of the product p to be printed;
- the field `requirements` represents the preconditions that must be true (cf. `positive`) and the states that must be false (cf. `negative`) for the operation to be applicable. For instance, the requirements at lines 14-18 state that the product must not be already processed and the printer's position needs to be x;
- the object `effects` stores the lists of states that will be set to true (cf. `added`) and the states that will be set to false (cf. `deleted`) after the operation execution, provided that the operation terminates successfully;

In addition, actions have a field `cost` (line 10).

```
{"thing":{                                                  1
    "thingId":"com.myThings:pr",                            2
    "attributes":{                                          3
      "type":"printer"},                                    4
    "features":{                                            5
      "printing":{                                          6
        "properties":{                                      7
          "type":"operation",                               8
          "command":"print",                                9
          "cost":1,                                         10
          "parameters":[                                    11
            "product - p",                                  12
            "position - x"],                                13
          "requirements":{                                  14
            "positive":[                                    15
              "at:x"],                                      16
            "negative":[                                    17
              "p.processed"] },                             18
          "effects":{                                       19
            "added":[                                       20
              "p.processed",                                21
              "p.at:x"] } } },                              22
      "at":{                                                23
        "properties":{                                      24
          "type":"state",                                   25
          "value":{                                         26
            "type":"position",                              27
            "current":"pos32" } } },                        28
      "status":{                                            29
        "properties":{                                      30
          "value":"free" } } } } }                          31
```

Figure 3: Sample JSON decription of a DT.

Finally, as mentioned, each DT has a `status` feature, which is used by the orchestrator for monitoring purposes. Its value can be either `free`, if the device is waiting for commands, `busy`, if the device is performing a task, or `broken`, if the device is in an error state and needs intervention.

Result of translation. The translator component of ALTO transforms the DT descriptions and the context file into *(i)* a PDDL domain file containing types, predicates and actions and *(ii)* a PDDL problem file containing the objects, initial states and the goal. The translation result is detailed below.

Types encode the objects in the working environment and the IoT devices and are organized in inheritance hierarchies. Specifically, for the purpose of DT specification, we introduce types `service` for DT objects and `capability` for the operations they provide. In our example, the PDDL domain file contains the following object types:

```
(:types                                                 1
  position                                              2
  typeA typeB - piece                                   3
  piece product - movable                               4
  printer cutter assembler - service                    5
  printing cutting assembling - capability              6
  movable service  - object)                            7
```

Each line specifies type names and type-subtype relationships. For instance `piece` and `product` are subtypes of the `movable` type (i.e., all those objects that may be moved around the factory), which in turn belongs to `object`.

Predicates apply to specific types of objects or to all objects and are either true or false at any point in the plan. In our example, the PDDL domain file contains the following ones:

```
(:predicates (at ?o - object ?p - position)          1
             (cut ?p - piece)                        2
             (processed ?p - product))               3
```

For instance, predicate `at` takes two arguments, the first one of type `object` and the second of type `position`.

Actions correspond to the operations offered by the available DTs. In our example, DTs provide four actions: `move`, `print`, `cut` and `assemble`. For instance, operation `print` in Figure 3 translates to the following PDDL fragment:

```
(:action PRINT                                       1
  :parameters   (?srv - service                      2
                 ?p - product                        3
                 ?x - position)                      4
  :precondition (and (provides ?srv printing)        5
                 (not(processed ?p))                 6
                 (at ?srv ?x))                       7
  :effect       (and (processed ?p)                  8
                 (at ?p ?x)))                        9
```

Objects constitute the possible arguments to the predicates in the problem instance. They are obtained from both the context file and DT descriptions. In particular, each Thing becomes an object of type `service` and each feature an object of type `capability`. An excerpt of this section follows:

```
(:objects                                            1
    p - product                                      2
    ta - typeA                                       3
    pr - printer                                     4
    printing - capability                            5
    pos11 pos12 pos13 pos21 pos22 ... - position     6
    ...)                                             7
```

The data that contributes to the initial state are collected from the features of type `state` of the DTs and the context file. With reference to Figure 3, the printer pr contains a state named `at`. Therefore, the initial state contains the following grounded predicate:

```
(at pr pos11)                                        1
```

In addition, the fact that the printer pr contains the operation `printing` leads to the inclusion of the following grounded predicate in the initial state:

```
(provides pr printing)                               1
```

In our example, the PDDL problem file contains the following goal that requires the production of product "p" and its storing in position "pos33":

```
(:goal (and (processed p)(at p pos33))               1
```

4 Performance Assessment

We now provide empirical results about the scalability of our approach and demonstrate that plans can be generated in reasonable time (e.g., few minutes) for up to 100 DT operations in the platform. As frame of comparison, our experience suggests that typical scenarios involve few dozens of operations available in the smart factory, possibly due to the presence of many simple DTs (i.e., with few operations each) or few complex ones (i.e., with many operations each). The total amount of time spent to perform the entire process is made up of three components: *(i)* the time spent by the devices to perform the assigned tasks, *(ii)* the communication time and *(iii)* the time dedicated to planning. Typical tasks in a manufacturing chain are usually estimated in the order of minutes (e.g., a 3D printer takes several minutes in order to produce an output). The time spent to exchange messages for any single operation is usually far below one second. In order to show the overhead introduced by planning, we randomly generate different PDDL domains/problems and measure the corresponding planning time as described below.

Domain generation. We consider different amounts $n = 10, 20, \ldots 100$ of available DT operations and generate a corresponding random directed graph with n nodes, where every node represents a DT operation and every edge (u, v) represents that the effect of the operation u is a preconditions of the operation v. More specifically, let p an input probability value, the graph generation consists of the main steps below.

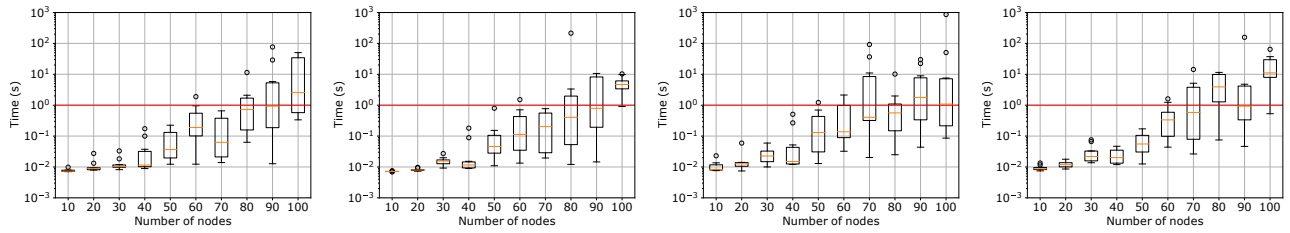
1. We sort the nodes in decreasing order of their default identifier (e.g., in lexicographic order). Let $u_0, u_1, \ldots, u_{n-1}$ the resulting sorted list.
2. For every node u_i, we add the edge (u_i, u_{i+1}) with probability p. Then, if (u_i, u_{i+1}) is inserted, we add the edge (u_i, u_{i+2}) with probability $p/2$. Otherwise we add the edge (u_i, u_{i+2}) with probability p, and so on.

By construction the graph *(i)* is acyclic; *(ii)* it has at least one source node with no in-going edges (i.e., u_0) *(iii)* it has at least one sink node with no out-going edges (i.e., u_{n-1}). Source nodes represent operations with no preconditions, e.g. raw material provision. The resulting graph is finally translated into a PDDL domain and problem as follows.

- Each node becomes an action with no parameters. The name of such action is $processL$, where L is the node identifier. The effect consists of only one predicate, $processedL$. The precondition is a conjunction over the effects of the predecessors of node L (if there are no predecessors, the precondition is empty), e.g., if node L has two predecessors, node P and node Q, the precondition becomes $and(processedP)(processedQ)$;
- For each sink node u we add the predicate $processedL$ (where L is the identifier of u) to the goal clause.

Finally, we set the initial state of the PDDL problem as empty and check if a planning solution exists, otherwise we discard the graph. We note here that a PDDL domain built using this strategy is definitely more complex than a typical manufacturing scenario, as operations are much more interconnected than in the reality. Thus, obtained performance results represent an upper bound for those in real world cases.

Results. Figure 4 shows planning time in the first and in two different scenarios, for increasing values of n and two values of p, dubbed $p = 0.25$ and $p = 0.50$. For every n we repeat the process 50 times, every time generating a new graph. In the first scenario, we measure the time for the optimal planner to compute the main plan. In the second scenario, in order to simulate the malfunctioning of a device, we eliminate a



(a) scenario 1, $p = 0.25$ (b) scenario 2, $p = 0.25$ (c) scenario 1, $p = 0.50$ (d) scenario 2, $p = 0.50$

Figure 4: Planning times for random factory domains. Y-axes are in log scale. The horizontal line corresponds to 1 second.

random node and its associated edges from the current graph, and measure the time for computing the recovery plan. For each n, the box shows the lower and upper quartile values, with a line at the median. The whiskers and the outlier points show the complete range of the data.

We first observe that planning is known to require worst case exponential time in the input size, and thus as n increases the maximum time grows rapidly. However, most of the instances can be processed successfully in less than 10 seconds (which is a negligible time in the context of production processes) and almost all instances require less than 3 minutes. We also note that re-planning can take more time, as recovery plans can be longer than main plans, but on average our two scenarios run in comparable times. Of course, multiple failures and consequent recovery plans could occur during the process. However, in real-world applications failures are infrequent and there is typically time to fix broken machines between one failure and another, so that they become available again for future re-planning.

5 Research Outlook

In this paper, we have proposed a technique, based on automated planning, to be applied in smart manufacturing scenarios that, given a set of descriptions of available digital twins in terms of available actions, preconditions and effects, provides a sequence of actions suitable to pursue a production goal and, in case failures are observed, finds a recovery plan. We have shown how such an approach can be realized and how observed performances are suitable with respect to typical manufacturing chain sizes.

Limitations. The proposed approach represents a first step towards a more general application of AI planning to smart manufacturing. As such, it suffers of the some limitations:

1. Preconditions and effects of DT actions only contain logical conjunctions of boolean predicates;
2. Different DTs share a common vocabulary, s.t., for instance, the same predicate is denoted by the same name in all the DTs in the platform;
3. Planning takes into account neither non-determinism nor probabilities in the execution of DT actions, thus resulting in potentially non-optimal or risky plans.

Research outlook. The *first limitation* can be addressed by applying optimization techniques, such as *discretization* and *grounding*, in order to handle complex task preconditions and effects without compromising performances. Discretization is a powerful tool to support non boolean properties, such as data coming from sensor measurements. A reader might note that this technique has been already applied in our example for handling positions, and in future we plan to leverage advanced discretizazion strategies to handle also more complex physical measures. Grounding [Gnad *et al.*, 2019] is instead a common approach in planning to reduce the state space, thus improving solution search performance. State-of-the-art planners already transparently implement grounding, but we plan to make an explicit use of it in the planning domain in order to cope with data integration issues.

The *second limitation* can be addressed instead by applying data integration techniques [Lenzerini, 2002; Bernstein *et al.*, 2011], such as *schema matching*, for handling vocabulary ambiguities, such as different names for the same predicate or measurement units conversion.

Both the first and the second limitations can be addressed at the *translator* level in Figure 1, in order to avoid PDDL features requiring the employment of less efficient planners, which would compromise the application of the proposed approach in a realtime scenario such as smart manufacturing.

Concerning the *third limitation*, addressing uncertainty in planning is a known task [Blythe, 1999]. Researchers must anyway find a trade-off between the expressive power of planning languages and the complexity required to machinery manufacturers in order to describe DTs. In such a scenario, approaches such as the one proposed in [Ciolek *et al.*, 2020] could be helpful to bridge the gap between the planning research community and manufacturers.

Another important research direction is represented by the integration of *experimentable DTs*, i.e., DTs equipped with prediction (or, *what-if*) functionalities. This would allow, for example, to promote production plans with greater likelihood of success. Additionally, even though the paper presents only examples involving a single factory, the approach can be extended to a more general setting where different companies are involved in different stages of the product life cycle. In this case, prediction capabilities of DTs and providing different solutions, for example, for mass- and custom- productions, assume a prominent role.

Finally, as decision taken by AI components can be potentially harmful, the proposed approach should be integrated in a human-in-the-loop approach, where knowledge and feedbacks from human experts are employed to improve automatic decisions.

References

[Ameri and Sabbagh, 2016] Farhad Ameri and Ramin Sabbagh. Digital factories for capability modeling and visualization. In *IFIP Intl. Conf. on Advances in Production Management Systems*, pages 69–78. Springer, 2016.

[Bao *et al.*, 2018] Jinsong Bao, Dongsheng Guo, Jie Li, and Jie Zhang. The modelling and operations for the digital twin in the context of manufacturing. *Enterprise Information Systems*, pages 1–23, 2018.

[Bernstein *et al.*, 2011] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011.

[Bicocchi *et al.*, 2019] Nicola Bicocchi, Giacomo Cabri, Federica Mandreoli, and Massimo Mecella. Dynamic digital factories for agile supply chains: An architectural approach. *J. of Ind. Information Integration*, 2019.

[Blythe, 1999] Jim Blythe. Decision-theoretic planning. *AI magazine*, 20(2):37–37, 1999.

[Brosinsky *et al.*, 2018] Christoph Brosinsky, Dirk Westermann, and Rainer Krebs. Recent and prospective developments in power system control centers: Adapting the digital twin technology for application in power system control centers. In *2018 IEEE International Energy Conference (ENERGYCON)*, pages 1–6. IEEE, 2018.

[Catarci *et al.*, 2019] Tiziana Catarci, Donatella Firmani, Francesco Leotta, Federica Mandreoli, Massimo Mecella, and Francesco Sapio. A conceptual architecture and model for smart manufacturing relying on service-based digital twins. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 229–236. IEEE, 2019.

[Ciolek *et al.*, 2020] Daniel Ciolek, Nicolás D'Ippolito, Alberto Pozanco, and Sebastian Sardiña. Multi-tier automated planning for adaptive behavior. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 66–74, 2020.

[Fox and Long, 2003] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[Ghallab *et al.*, 2016] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[Gnad *et al.*, 2019] Daniel Gnad, Alvaro Torralba, Martín Domínguez, Carlos Areces, and Facundo Bustos. Learning how to ground a plan–partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7602–7609, 2019.

[Hankel and Rexroth, 2015] Martin Hankel and Bosch Rexroth. The reference architectural model industrie 4.0 (rami 4.0). *ZVEI, April*, 410, 2015.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Kamath *et al.*, 2020] Vignesh Kamath, Jeff Morgan, and Muhammad Intizar Ali. Industrial iot and digital twins for a smart factory: An open source toolkit for application design and benchmarking. In *2020 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2020.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, 2002.

[Marrella *et al.*, 2016] Andrea Marrella, Massimo Mecella, and Sebastian Sardina. Intelligent process adaptation in the SmartPM system. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):1–43, 2016.

[Qi *et al.*, 2019] Qinglin Qi, Fei Tao, Tianliang Hu, Nabil Anwer, Ang Liu, Yongli Wei, Lihui Wang, and AYC Nee. Enabling technologies and tools for digital twins. *Journal of Manufacturing Systems*, 2019.

[Schroeder *et al.*, 2016] Greyce N Schroeder, Charles Steinmetz, Carlos E Pereira, and Danubia B Espindola. Digital twin data modeling with automationml and a communication methodology for data exchange. *IFAC-PapersOnLine*, 49(30):12–17, 2016.

[Sisinni *et al.*, 2018] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial Internet of Things: Challenges, opportunities, and directions. *IEEE Tran. on Ind. Informatics*, 14(11):4724–4734, 2018.

[Tao and Zhang, 2017] Fei Tao and Meng Zhang. Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing. *Ieee Access*, 5:20418–20427, 2017.

[Tao *et al.*, 2018a] Fei Tao, Jiangfeng Cheng, Qinglin Qi, Meng Zhang, He Zhang, and Fangyuan Sui. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94(9-12):3563–3576, 2018.

[Tao *et al.*, 2018b] Fei Tao, He Zhang, Ang Liu, and Andrew YC Nee. Digital twin in industry: State-of-the-art. *IEEE Tran. on Ind. Informatics*, 15(4):2405–2415, 2018.

[Tao *et al.*, 2019] Fei Tao, Qinglin Qi, Lihui Wang, and AYC Nee. Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, 5(4):653–661, 2019.

[Wang and Wang, 2018] Xi Vincent Wang and Lihui Wang. Digital twin-based WEEE recycling, recovery and remanufacturing in the background of industry 4.0. *International Journal of Production Research*, pages 1–11, 2018.

[Weber *et al.*, 2017] Christian Weber, Jan Königsberger, Laura Kassner, and Bernhard Mitschang. M2DDM – a maturity model for data-driven manufacturing. *Procedia CIRP*, 63:173–178, 2017.

[Zehnder and Riemer, 2018] Philipp Zehnder and Dominik Riemer. Representing industrial data streams in digital twins using semantic labeling. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4223–4226. IEEE, 2018.