Computers in Industry 149 (2023) 103916

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.sciencedirect.com/journal/computers-in-industry

Digital twin composition in smart manufacturing via Markov decision processes

Giuseppe De Giacomo [a], Marco Favorito [a,b,1], Francesco Leotta [a,*], Massimo Mecella [a], Luciana Silo [a]

[a] Dipartimento di Ingegneria informatica, automatica e gestionale Antonio Ruberti Sapienza Università di Roma, Italy
[b] Banca d'Italia, Italy

ARTICLE INFO

Keywords:
Service composition
Roman model
Digital twins
Industry 4.0
Smart manufacturing
Markov decision processes

ABSTRACT

Digital Twins (DTs) are considered key components in smart manufacturing. They bridge the virtual and real world with the goal to model, understand, predict, and optimize their corresponding real assets. Such powerful features can be exploited in order to optimize the manufacturing process. In this paper, we propose an approach, based on Markov Decision Processes (MDPs) and inspired by Web service composition, to automatically propose an assignment of devices to manufacturing tasks. This assignment, or policy, takes into account the uncertainty typical of the manufacturing scenario, thus overcoming limitations of approaches based on classical planning. In addition, obtained policies are proven to be optimal with respect to cost and quality, and are continuously updated in order to adapt to an always evolving scenario. The proposed approach is showcased in an industrial application scenario, and is implemented as a freely available tool.

1. Introduction

The term *Industry 4.0* which denotes the fourth industrial revolution, was first introduced in Germany in 2011 at the Hanover fair, where it was used for denoting the transformation process in the global chains of value creation (Kagermann et al., 2011). At present Industry 4.0 is a result of the emergence and distribution of new technologies – digital technologies and Internet technologies – which allow the development of fully automatized production processes, in which only physical objects that interact without human participation take part (Popkova et al., 2019). *Smart Manufacturing* is nowadays a term highly used in conjunction with the concept of Industry 4.0. Smart Manufacturing aims at improving the manufacturing processes in order to increase productivity and quality, to ease workers' lives, and to define new business opportunities. This is enabled by leveraging innovative techniques like Artificial Intelligence (AI), big data analytics, Machine Learning (ML), and Business Decision Support Systems (BDSS). The employment of these technique has made it possible to create new possibilities of interoperability, modularity, distributed scenario processing, and integration in real time with other industrial processes.

Essentially in the same period, the concept of Digital Twin (DT) was introduced as a key technology used in the industrial context. A lot of different definitions for digital twin can be found in literature, mainly caused by various application areas. The first clear definition was given by NASA in 2012 (Shafto et al., 2012). They define digital twin as "an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin". Authors in VanDerHorn and Mahadevan (2021), after a thorough literature review, propose a consolidated and generalized definition for a Digital Twin as "a virtual representation of a physical system (and its associated environment and processes) that is updated through the exchange of information between the physical and virtual systems".

The application of DT in the manufacturing sector impacts the way the products are designed, manufactured and maintained. On a high level, the DT can evaluate the production decisions, access the product performance, command and reconfigure machines remotely, handle the troubleshoot equipment remotely and connect systems/processes to improve monitoring and optimize their control (Kitain, 2018). DTs can also be applied for process control, process monitoring, predictive maintenance, operator training, product development, decision support, real-time analytics and behavior simulation (Pires et al., 2019). A single manufacturing process may include hundreds of different actors (humans, equipment, organizations), together with their digital twins. Such actors may suddenly fail or provide bad performance as, due to their continuous use, they can wear out and therefore have not only a

* Corresponding author.
E-mail address: leotta@diag.uniroma1.it (F. Leotta).
[1] Opinions expressed are solely the author's own and do not express the views or opinions of his employer.

https://doi.org/10.1016/j.compind.2023.103916
Received 19 April 2022; Received in revised form 14 December 2022; Accepted 3 April 2023
Available online 12 April 2023

greater probability of breakage but also higher costs to complete their job. Notice that these information can only be available if a digital twin is available for each actor to compute them. At any moment in time, in order to provide resilience, manufacturing process should be able to automatically adapt to new conditions, considering new actors (with lower cost and low probability of breaking) for the fulfillment of the manufacturing goals. This task cannot be performed manually when actors span multiple organizations possibly separated from both the geographical and organizational points of view. For this reason it is crucial to have a plan for the manufacturing process able to manage several actors, taking into the account their possible failures and costs.

In that regard, very little research effort has been put in defining automatic techniques to orchestrate manufacturing actors towards a final goal. Authors in Catarci et al. (2019) argued that an important step towards the development of new automated techniques in smart manufacturing is modeling DTs in terms of the provided services. This would allow to partly reuse the results obtained in the area of Web services, such as the automatic composition and orchestration of software artifacts.

The inherent limitation of such approaches, though, is the assumption that the available services, i.e., the services that can be used to realize the target service, behave *deterministically*. This assumption is unrealistic in the case of DTs for smart manufacturing, because in practice the underlying physical system modeled as a set of services might show a stochastic behavior due to the complexity of the domain, or due to an inherent uncertainty on the dynamics of such a system. In these cases, the deterministic service model is not expressive enough to capture crucial facets of the system under consideration.

In this paper, we adopt service composition techniques to orchestrate digital twins in order to generate a plan for a manufacturing process reducing the costs while preserving the quality of the final outcome. In particular, we propose a generalization of service composition techniques in a stochastic setting, in which the services, corresponding to DTs and underlying manufacturing actors (both machines and humans) have an unpredictable behavior and are subject to wear. We find an optimal solution by solving an appropriate probabilistic planning problem (solving a Markov decision process—MDP), taking into account the probability of breaking and the cost of employing specific actors. To this aim, we leverage the capability of DTs to assess the status and the wearing of the underlying physical entity (Melesse et al., 2020; Aivaliotis et al., 2019). In this way, we are able to *autonomously*, i.e., without human intervention, obtain a production planning which is *adaptive*, as it changes every time that the manufacturing of a new product or batch is started, and *context-aware*, as it is dependent from the current status of involved actors.

The rest of the paper is structured as it follows. Section 2 compares our approach to other planning approaches in the literature. Section 3 introduces the conceptual architecture of the proposed approach, introducing key components and actors and explaining their roles. Section 4 formalizes the problem in terms of Markov Decision Processes (MDPs) and proposes a solution finding an optimal production plan. Section 5 shows the application of the formal framework to an industrial case study. Section 6 describes how the approach has been implemented and the results obtained on the case study. Finally Section 7 concludes the paper with final remarks and future works.

2. Related works

The work that we propose is inspired by service composition, and in particular to the "Roman model" (Berardi et al., 2003, 2005). Here, each available service is modeled as a finite-state machine (FSM), in which at each state, the service offers a certain set of actions and each action changes the state of the service in some way. A new service, the target, is generated from the set of existing services, specified as FSM too. The goal is to build a scheduler, called the orchestrator that will use actions provided by existing services to implement requests of the target. This solution does not fit the requirements of smart manufacturing, as services (i.e., manufacturing machines and human workers) do not exhibit a deterministic behavior. The approach proposed by Brafman et al. (2017), while providing a stochastic model for service composition, does not solve this problem. In fact, the authors employ MDP to provide sub-optimal solution, capturing only the non-deterministic behaviors of the target but not of the available services. In this paper, we solved this issue by proposing an algorithm computing an optimal composition of stochastic DTs in order to execute a specific manufacturing process.

Our framework can be contextualized in the broader research area that applies automated planning techniques to the world of smart manufacturing and, in general, of Cyber-Physical Systems and business processes (Marrella, 2019). In this area, we can distinguish between classical planning, which deals with deterministic scenarios, and so called decision-theoretic planning (Blythe, 1999), which deals with non-determinism and stochastic behavior. MDPs can be classified indeed as an approach to decision-theoretic planning. There are several application of MDPs to DTs. For example authors in Lee et al. (2020) develop an optimal planning and control policy of the surveillance system considering pro-active targets, heterogeneous sensory data, and dynamic environmental factors. They use MDP to represent dynamic changes in environmental effects in the surveillance area. The work (Xu et al., 2022) proposes a DT framework for crane dynamic scheduling problem, modeled as an MDP with the corresponding double deep Q-network (DDQN) to interact with the logic simulation environment. The trained DDQN is embedded into the DT framework, and connected with the physical workshop. The authors in Ghosh et al. (2019) describe the utilization of the hidden Markov models to construct a digital twin of the surface roughness of a ground surface. Here, the surface heights given in the form of a time series are used to construct a Markov chain. Then, they used a Monte Carlo simulation process that simulates the states in accordance with the constructed Markov chain. In all these works, as in our case, the MDP is used for devising optimal planning of the problem. The crucial novelty of our approach lies in ensuring adaptiveness to change, in fact our system evolves continuously and continuously is able to find an optimal planning. In detail, each digital twin that depicts a machine in the manufacturing process updates its values at each execution, i.e., the probability to break and the cost increases.

Moreover, in this context it is useful to make a distinction between automated planning and automated scheduling applied to smart manufacturing. Automated planning is the application of artificial intelligence technologies to the problem of generating a correct and efficient sequence of actions (Marrella, 2019) (in this paper the term action is a synonym of task). On the other hand, scheduling (Braccesi et al., 2004) is applied to the related problem of efficiently, or sometimes optimally, allocating time and other resources to an already-known sequence of actions (plan).

An example of application of classical planning to smart manufacturing is provided in Marrella et al. (2016). Here, authors employ automated planning in order to cope with exceptional and unanticipated events. In particular, planning is employed to fix a process instance, restoring the conditions to continue with the standard, manually defined process. Our approach is different both because of the probabilistic behavior of DTs and because we reason on the entire manufacturing process and not only on fixing it.

Example of classical planning applied to the entire manufacturing process are provided by Fernández et al. (2005), Krueger et al. (2019), Carreno et al. (2020). Here, again, the solutions provided are deterministic, without taking into account the uncertainty typical of manufacturing.

Non-deterministic planning is employed by authors in Ciolek et al. (2020), where a set of degrading planning domains are defined. The planner tries to find a solution in the most restrictive, optimal domain. If during execution, assumptions of a plan are not verified, due for

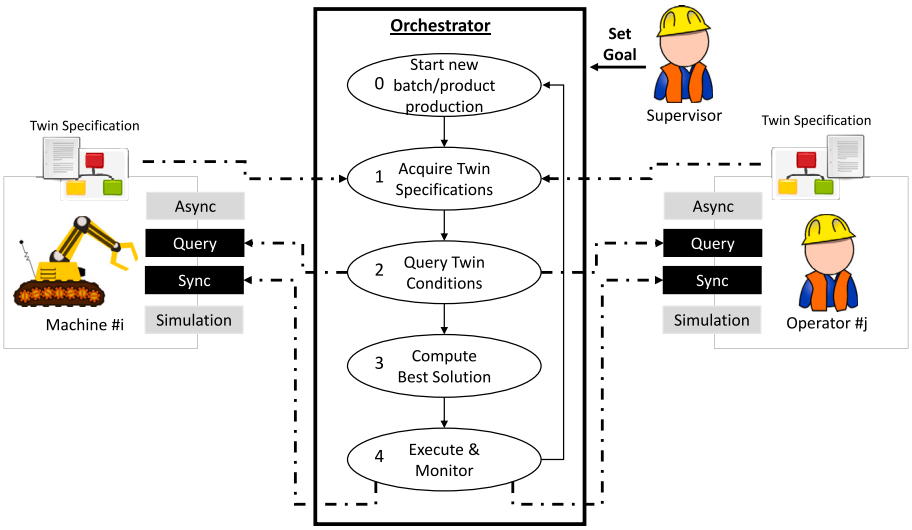

Fig. 1. The proposed architecture.

example to failures, more and more sub-optimal domains are employed. The approach focuses on the entire process and non-determinism of manufacturing actors is modeled, but it does not consider the numerical information that can be provided by the DT about the wear level and probability of failure of the underlying physical actor, as in our approach.

In general, all the approaches based on classical planning and non-deterministic planning requires the definition of a planning domain. Even though our approach relies on models, expressed as MDPs, of the DTs and of the manufacturing process, such models are much less time consuming to be produced. On the other hand, a proper planning domain allows to specify complex conditions, for example, on the data involved in the process.

3. Conceptual architecture

In order to explain the proposed approach, Fig. 1 shows a conceptual architecture, complying with the general one provided in Catarci et al. (2019), where the main components are the supervisor, the orchestrator and the DTs of involved actors, the latter ones including manufacturing machines, human operators/workers and external suppliers.

The term DT was originally intended to denote a digital model, that faithfully reproduces a physical entity, and allows to perform physical simulations (e.g., mechanical solicitations). In the last years, the term has been used to more generically denote a digital interface allowing not only for simulations but also for an all-round control of the physical entity during run time (Negri et al., 2017; Durão et al., 2018). Following this intuition, a DT exposes an Application Programming Interface (API) consisting, in general, of three parts (Catarci et al., 2019): synchronous, query and asynchronous. The synchronous interface allows to give instructions to the physical entity. These instructions include actions to be executed by the physical entity that changes its status. The query interface allows to ask information to the physical entity about its state and related information; noteworthy, these latter ones can be obtained by applying diagnostic and prognostic functions results of machine learning. The asynchronous interface generates events available to subscribers. A fourth interface, called simulation interface, is provided to access the simulation functionalities provided by the DT.

In addition, each DT is equipped with a specification of the provided functionalities. This specification may take a form that depends on the specific framework employed to implement the DT (see for example Section 6).

For the sake of space, Fig. 1 depicts a single machine #*i* and a single operator #*j*, but the reader should imagine a factory as including a multitude of them. In addition, each action (task) of a manufacturing process can be performed by different machines or humans. The choice of the specific actor to be employed for a specific action is driven by different factors, including the cost and the potential quality loss due to wear or obsolescence. As an example, humans are usually more expensive than machines for a specific action in terms of economic and time cost, but a worn machine could negatively impact the quality of the final product.

The *orchestrator*, as discussed in Catarci et al. (2019) (there, authors use the more generic name of *mediator*), is a software program intended to guarantee that the manufacturing process fulfills the goals imposed by a human *supervisor*. In addition, the orchestrator selects services to be employed maximizing a set of Key Performance Indicators (KPIs). In this paper, the goal consists of a description of a manufacturing process, which we aim to execute while reducing the total cost, where the concept of cost embodies both the economic cost and quality loss due to the specific choices made by the orchestrator.

The orchestrator is called every time the production of a new product or a batch of products is started. At this point, the orchestrator:

1. Gathers the specifications of all the available DTs.
2. According to these specifications, the orchestrator employs the query interface of each DT to obtain the current status of the machine. The status is continuously kept updated by the DT and may include the probability of breaking and the wear level of the correspondent device.
3. Computes the optimal solution given the current status and capabilities of each available actor, as reported by the respective DTs. A solution, or plan, consists of a sequence of manufacturing actions, consistent with the description provided by the supervisor, and of an assignment of a specific actor (machine or human) to each action.
4. Automatically executes the manufacturing process following the obtained plan, by leveraging the synchronous interface of each involved twin. In addition, it monitors the execution, by taking countermeasures when needed (e.g., when a specific machine breaks).

After a product or batch is completed, the orchestrator starts over, waiting for a new production start event.

Noticeably, every time that a new production starts and the orchestrator is invoked, the decision this latter will make will be influenced by the production history, as the DT behind each service involved in

the production will update information about costs and likelihood of a breaking event.

Noteworthy, the definition of orchestrator provided in this section is very generic and it is open to several different implementation strategies. In this paper, in particular, we will implement the orchestrator as a tool that finds an optimal policy to a Markov Decision Process (MDP) which is constructed by combining in an innovative way the different MDPs modeling available actors/services.

4. Composing DTs

In this section, we formalize the orchestrator defined in Section 3 and we introduce an algorithm with formal optimality guarantees to compute the manufacturing plan.

As DTs and corresponding physical actors can be modeled in terms of their service interface, their composition to obtain a manufacturing goal can be performed similarly to what has been done for Web service composition in the area of classical information systems. The problem of service composition, i.e., the ability to generate new, more useful services from existing ones, has been considered in the literature for over a decade (Hull, 2008; Medjahed and Bouguettaya, 2011; De Giacomo et al., 2014). The goal is, given a specification of the behavior of a (complex) target service, to build a controller, known as an *orchestrator*, that uses existing (composing) services to satisfy the requirements of the target service. In our case, the target service is the manufacturing process, whereas the composing services are the DTs wrapping physical actors.

In particular, we took inspiration from the approach known as the "Roman model" (Berardi et al., 2003, 2005). In the Roman model, each available Web service is modeled as a finite-state machine (FSM), in which at each state, the service offers a certain set of actions, where each action changes the state of the service in some way. The designer is interested in generating a new service (specified using a FSM too), referred to as the target service, from the set of existing services.

The first limitation of such an approach, when applied to the world of smart manufacturing, is that whereas Web services behavior is predictable, i.e., the execution of an action in a specific state deterministically takes the service from one state to another, the execution of an action from an industrial actor (either machine or human) can have unpredictable effects (e.g., the machine breaks), which must be taken into account while computing a solution. In addition, the behavior of a physical actor in manufacturing may degrade over time due to wearing.

Moreover, it is not always possible to synthesize a service that fully conforms to the requirement specification. This zero–one situation, where we can either synthesize a perfect solution or fail, is often restrictive. Rather than returning no answer, we may want the notion of the "best-possible" solution. A solution to this last issue has been proposed in Brafman et al. (2017), where the authors discuss and elaborate upon a probabilistic model for the service composition problem, first presented in Yadav and Sardina (2011). In this model, an optimal solution can be found by solving an appropriate probabilistic planning problem (e.g., a Markov Decision Process) derived from the services and requirement specifications. Still, the proposed solution is applicable only to deterministic and non-degrading services such as Web services.

The solution proposed in this paper relies on the concept of Markov Decision Process (MDP). An MDP $\mathcal{M} = \langle \Sigma, A, P, R, \lambda \rangle$ is a discrete-time stochastic control process containing *(i)* a set Σ of states, *(ii)* a set A of actions, *(iii)* a transition function $P : \Sigma \times A \to Prob(\Sigma)$ that returns for every state s and action a a distribution over the next state, *(iv)* a reward function $R : \Sigma \times A \to \mathbb{R}$ that specifies the reward (resp. the cost), a real value received (resp. paid) by the agent when transitioning from state σ to state σ' by applying action a, and *(v)* a discount factor $\lambda \in (0, 1)$. A solution to an MDP is a function, called a *policy*, assigning an action to each state, possibly with a dependency on past states and actions. The *value* of a policy ρ at state σ, denoted $v_\rho(\sigma)$, is the expected sum of rewards when starting at state σ and selecting actions based on ρ. This expected sum of rewards could possibly be discounted by a factor λ, with $0 \leq \lambda < 1$. Typically, the MDP is assumed to start in an initial state σ_0, so policy optimality is evaluated w.r.t. $v_\rho(\sigma_0)$. Every MDP has an optimal policy ρ^*. In discounted cumulative settings, there exists an optimal policy $\rho : \Sigma \to A$ that is Markovian, i.e., that depends only on the current state, and deterministic (Puterman, 1994). Among the techniques for finding an optimal policy of an MDP, there are *value iteration* and *policy iteration* (Sutton and Barto, 2018).

4.1. Modeling DTs as stochastic services

In order to overcome the limitations of the Roman model when applied to smart manufacturing, *we model each DT and the underlying physical actor as a stochastic service*. A *stochastic service* is a tuple $\tilde{S} = \langle \Sigma_s, \sigma_{s0}, F_s, A, P_s, R_s \rangle$, where Σ_s is the finite set of service states, $\sigma_{s0} \in \Sigma$ is the initial state, $F_s \subseteq \Sigma_s$ is the set of the service's final state, A is the finite set of services' actions, $P_s : \Sigma_s \times A \to Prob(\Sigma_s)$ is the transition function, and $R_s : \Sigma_s \times A \to \mathbb{R}$ is the reward function.

The stochastic service is the stochastic variant of the service defined in the classical Roman model, and it can be seen as an MDP itself. Such a solution allows for the flexibility required to model a physical machine operating in manufacturing environments. As an example, specific states can be defined to model unavailability conditions (e.g., a broken machine) and the probability of ending in such states. In addition, rewards can be used to model the degradation of service quality in time. Repair costs to recover from unavailability states can also be modeled to take into account in the solution the possibility to fix broken devices if they guarantee an high quality. All of these parameters can be computed and continuously refreshed by the DTs by using models trained by the equipment manufacturers. A case study demonstrating these aspects will be presented in Section 5.

We also define, to simplify the discussion in the next sections, the *stochastic system service* as the community of stochastic services $\tilde{C} = \{\tilde{S}_1, \ldots, \tilde{S}_n\}$. It is a stochastic service itself defined as $\tilde{\mathcal{Z}} = \langle \Sigma_z, \sigma_{z0}, F_z, A, P_z, R_z \rangle$ where:

- $\Sigma_z = \Sigma_1 \times \cdots \times \Sigma_n$,
- $\sigma_{z_0} = (\sigma_{10}, \ldots, \sigma_{n0})$,
- $F_z = \{(\sigma_1, \ldots, \sigma_n) \mid \sigma_i \in F_i, 1 \leq i \leq n\}$,
- $A_z = A \times \{1, \ldots n\}$ is the set of pairs (a, i) formed by a shared action a and the index i of the service that executes it,
- $P_z(\boldsymbol{\sigma}' \mid \boldsymbol{\sigma}, (a, i)) = P(\sigma_i' \mid \sigma_i, a)$, for $\boldsymbol{\sigma} = (\sigma_1 \ldots \sigma_n)$, $\boldsymbol{\sigma}' = (\sigma_1' \ldots \sigma_n')$ and $a \in A_i(\sigma_i)$, with $\sigma_i \in \Sigma_i$ and $\sigma_j = \sigma_j'$ for $j \neq i$,
- $R_z(\boldsymbol{\sigma}, (a, i)) = R_i(\sigma_i, a)$ for $\boldsymbol{\sigma} \in \Sigma_z$, $a \in A_i(\sigma_i)$.

Intuitively, the stochastic system service represents, in a single MDP, all the stochastic services, i.e., all the DTs and underlying physical actors. As a consequence, its status includes the current status of all the composing services. A specific action performed on the system service changes only one component of the current state, corresponding to the service selected to execute that action.

4.2. Modeling the manufacturing process

In order to model the manufacturing process we use the concept of *target service* introduced by the Roman model. The term denotes a complex service that can be obtained by composing simpler services. In our case, the manufacturing process must be obtained by composing the functions of available DTs. In particular, we will use the definition of target service as adapted by Brafman et al. (2017) to the stochastic setting.

A *target service* is defined as a tuple $\mathcal{T} = \langle \Sigma_t, \sigma_{t0}, F_t, A, \delta_t, P_t, R_t \rangle$, where Σ_t is the finite set of service states, $\sigma_{t0} \in \Sigma$ is the initial state, $F_t \subseteq \Sigma$ is the set of the service's final state, A is the finite set of services' actions, $\delta_t : \Sigma \times A \to \Sigma$ is the service's deterministic and

partial transition function, $P_t : \Sigma_t \to \pi(A) \cup \emptyset$ is the action distribution function, $R_t : \Sigma_t \times A \to \mathbb{R}$ is the reward function.

Noticeably, the target service itself, as the stochastic services modeling the DTs, is a particular case of MDP. In the vast majority of cases, manufacturing processes (differently from manufacturing actors) are deterministic. The employment of such definition, is only to be intended as functional to our formal framework, with $P_t \in \{0, 1\}$ and R_t constant.

4.3. The composition problem

We define the set of joint histories of the target and the system service as $H_{t,z} = \Sigma_t \times \Sigma_z \times (A \times \Sigma_t \times \Sigma_z)^*$. A joint history $h_{t,z} = \sigma_{t,0}\sigma_{z,0}a_1\sigma_{t,1}\sigma_{z,1}a_2 \dots$ is an element of $H_{t,z}$. The projection of $h_{t,z}$ over the target (system) actions is $\pi_t(h_{t,z}) = h_t$ ($\pi_z(h_{t,z}) = h_z$). An orchestrator $\gamma : \Sigma_t \times \Sigma_z \times A \to \{1, \dots, n\}$, is a mapping from a state of the target-system service and user action $(\sigma_t, \sigma_z, a) \in \Sigma_t \times \Sigma_z \times A$ to the index $j \in \{1, \dots, n\}$ of the service that must handle it. Since the being stochastic comes also from the services, the orchestrator *does* affect the probability of an history $h_{t,z}$. Moreover, in general, there are *several* system histories associated with a given target history.

A target history h_t is realizable by an orchestrator γ if for all joint histories $h_{t,z}$ such that $h_t = \pi_t(h_{t,z})$, the orchestrator is well-defined, i.e. it can perform all the actions requested by the target for every possible (stochastic) evolution of the system service. The orchestrator γ is said to *realize* a target service $\mathcal{T}$ if it realizes all histories of $\mathcal{T}$.

Let $P_\gamma(h) = \prod_{i=0}^{\infty} P_t(\sigma_{t,i}, a_{i+1}) P_z(\sigma_{z,i+1} \mid \sigma_{z,i}, \langle a_{i+1}, \gamma(\sigma_{t,i}, \sigma_{z,i}, a_{i+1})\rangle)$ be the probability of a (joint) history $h = \sigma_{t0}\sigma_{z0}\langle a_1, j_1\rangle\sigma_{t1}\sigma_{z1}\langle a_2, j_2\rangle \dots$ under orchestrator γ. Intuitively, at every step, we take into account the probability, determined by P_t, that the user does action a_{i+1} in the target state $\sigma_{t,i}$, in conjunction with the probability, determined by P_z, that the system service does the transition $\sigma_{z,i} \xrightarrow{(a_{i+1}, j)} \sigma'_{z,i+1}$, where j is the choice of the orchestrator at step i under orchestrator γ, i.e. $j = \gamma(\sigma_{t,i}, \sigma_{z,i}, a_{i+1})$.

The value of a joint history under orchestrator γ is the sum of discounted rewards, both from the target and the system services: $v_\gamma(h) = \sum_{i=0}^{\infty} \lambda^i \Big(R_t(\sigma_{t,i}, a_{i+1}) + R_z(\sigma_{z,i}, \langle a_{i+1}, \gamma(\sigma_{t,i}, \sigma_{z,i}, a_{i+1})\rangle) \Big)$. Intuitively, we take into account both the reward that comes from the execution of action a_{i+1} in the target service, but also the reward associated with the execution of that action in service j chosen by orchestrator γ. Now we can define the expected value of an orchestrator to be:

$$v(\gamma) = \mathbb{E}_{h_{t,z} \sim P_\gamma} \left[v_\gamma(h_{t,z}) \cdot realizable(\gamma, \pi_t(h_{t,z})) \right] \tag{1}$$

where $realizable(\gamma, \pi_t(h_{t,z}))$ is 1 if $h_t = \pi_t(h_{t,z})$ is realizable in γ (i.e. all the possible target histories are processed correctly), and 0 otherwise. That is, $v(\gamma)$ is the expected value of histories realizable in γ. Finally, we define an optimal orchestrator to be $\gamma = \arg\max_{\gamma'} v(\gamma')$.

We have the following theorem:

Theorem 1. *Assuming that (1) the target is realizable, and (2) every target-system history has strictly positive value, if the orchestrator γ is optimal, then γ realizes the target.*

Proof. The claim in its contrapositive form is that any orchestrator that does not realize some history is non-optimal. By assumption (2), if the set of target histories realizable using orchestrator γ contains the set realizable using orchestrator γ', then $v(\gamma) \geq v(\gamma')$. Moreover, if the set of histories realizable by γ but not by γ' has positive probability, then $v(\gamma) > v(\gamma')$. If a target history h_t is not realizable by γ', there exists a point in h_t where γ' does not assign the required action to a service that can supply it. Thus, any history that extends the corresponding prefix of h_t is not realizable, and the set of such histories has non-zero probability. Since we assume all histories have positive value, the optimal orchestrator would always prefer realizing all possible target histories (which, by assumption (1), are all the ones to realize), possibly optimizing for the rewards coming from the services' actions, and therefore realize the target. Note that by definition of $v(\gamma)$ all the joint histories $h_{t,z}$ whose associated target history $h_t = \pi_t(h_{t,z})$ is not realizable in γ do not contribute to the value of an orchestrator (even the ones where γ is well-defined), thanks to the factor $realizable(\gamma, h_t)$. □

Note that, in our setting, we can only guarantee that the optimality of an orchestrator γ implies that γ is a realization of the target service, but not vice versa (e.g. see the sufficient condition of Theorem 1 of Brafman et al. (2017)); this is because our definition of realizability does not take into account the rewards coming from the system service, and therefore an orchestrator that realizes the target might choose sub-optimal services despite being a target realization. Moreover, note that any detour from a solution that realizes all the (infinite) target histories, that prefers rewards coming from the services at the cost of not realizing some target history h_t, is penalized, since by definition the missed realization of such history is considered of value 0. Nonetheless, as discussed in the following section, we are able to provide an optimal solution to the problem.

Additionally, assumption (2) is without loss of generality. Indeed, one can shift all the rewards with an additive constant c, both from the target service R_t and from the system service R_z, such that they become all positive. Having only positive rewards is a sufficient condition for having histories with strictly positive value. Therefore, in case the minimum target (system) service reward $R_{t,\min}$ ($R_{z,\min}$, respectively) is smaller than zero, we can use $c = |R_{t,\min}| + |R_{z,\min}|$. Crucially, this reward transformation is policy invariant, which means that the set of the optimal policies remains the same. Hence, the value function for some policy ρ with the original rewards is the same as that with the new rewards, except for a constant addend.

4.4. The solution technique

The solution technique is based on finding an optimal policy for the *composition MDP*. The composition MDP is a function of the system service and the target service as follows: $\tilde{\mathcal{M}}(\tilde{\mathcal{Z}}, \tilde{\mathcal{T}}) = \langle S_{\tilde{\mathcal{M}}}, A_{\tilde{\mathcal{M}}}, T_{\tilde{\mathcal{M}}}, R_{\tilde{\mathcal{M}}}\rangle$, where:

- $S_{\tilde{\mathcal{M}}} = \Sigma_{\tilde{\mathcal{Z}}} \times \Sigma_{\tilde{\mathcal{T}}} \times A \cup \{s_{\tilde{\mathcal{M}}0}, s_{\text{sink}}\}$,
- $A_{\tilde{\mathcal{M}}} = \{a_{\tilde{\mathcal{M}}0}, 1, \dots, n\}$,
- $T_{\tilde{\mathcal{M}}}(s_{\tilde{\mathcal{M}}0}, a_{\tilde{\mathcal{M}}0}, (\sigma_{z0}, \sigma_{t0}, a)) = P_t(\sigma_{t0}, a)$,
- $T_{\tilde{\mathcal{M}}}((\sigma_z, \sigma_t, a), i, (\sigma'_z, \sigma'_t, a')) = P_t(\sigma'_t, a') \cdot P_z(\sigma'_z \mid \sigma_z, \langle a, i\rangle)$, if $P_z(\sigma'_z \mid \sigma_z, \langle a, i\rangle) > 0$ and $\sigma_t \xrightarrow{a} \sigma'_t$ and 0 otherwise,
- $T_{\tilde{\mathcal{M}}}((\sigma_z, \sigma_t, a), i, s_{\text{sink}}) = 1 - \sum_{\sigma'_z, \sigma'_t, a'} T_{\tilde{\mathcal{M}}}((\sigma_z, \sigma_t, a), i, (\sigma'_z, \sigma'_t, a'))$
- $T_{\tilde{\mathcal{M}}}(s_{\text{sink}}, i, s_{\text{sink}}) = 1$, for all i,
- $R_{\tilde{\mathcal{M}}}((\sigma_z, \sigma_t, a), i) = R_t(\sigma_t, a) + R_z(\sigma_z, \langle a, i\rangle)$, if $(a, i) \in A(\sigma_z)$ and 0 otherwise,
- $R(s_{\text{sink}}, i) = 0$ for all i.

Noteworthy, even if the composition MDP is obtained by combining the system service and the target service, it has completely different characteristics. Here, the next action to perform is part of the state of the MDP, whereas the "action" is the selection of a specific service to execute that action ($A_{\tilde{\mathcal{M}}} = \{a_{\tilde{\mathcal{M}}0}, 1, \dots, n\}$). This means that by solving the composition MDP, we find an assignment of manufacturing actors to actions (manufacturing tasks) as well as a sequence of actions.

This definition is pretty similar to the construction proposed in Brafman et al. (2017), with the difference that now, in the transition function, we need to take into account also the probability of transitioning to the system successor state σ'_z from σ_z doing the system action $\langle a, i\rangle$, i.e. $P_z(\sigma'_z \mid \sigma_z, \langle a, i\rangle)$. Moreover, in the reward function, we need to take into account also the reward observed from doing system action $\langle a, i\rangle$ in σ_z, and sum it to the reward signal coming from the target. The state s_{sink} is an absorbing state, that transitions only to itself and that generates only rewards of value 0, and it is needed to make the transition function of the MDP well-defined. If a trajectory reaches that state, it means it represents an unrealizable (joint) history.

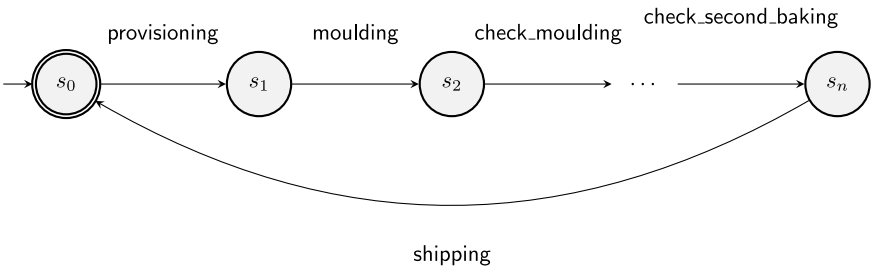

Fig. 2. The state machine of the target. Some steps are omitted.

Theorem 2. *Let γ_ρ, defined as $\gamma_\rho(\sigma_t, \sigma_z, a) = \rho(\langle \sigma_z, \sigma_t, a \rangle)$, be the orchestrator associated to a policy ρ. Assume that for all policies ρ and target histories h_t, we have that $realizable(\gamma_\rho, h_t) = 1$. If ρ is an optimal policy, then the orchestrator γ_ρ is an optimal orchestrator.*

Proof. Observe that for realizable joint histories $h_{t,z}$, for some ρ and γ_ρ, there is an obvious one-to-one relationship between $h_{t,z}$ and non-failing trajectories τ of the composition MDP (to be more precise, we first have to drop $s_{\tilde{\mathcal{M}}0}$ and $a_{\tilde{\mathcal{M}}0}$ from τ). By construction, for any joint history $h_{t,z}$ and policy ρ, we have that $v_{\gamma_\rho}(h_{t,z}) = \frac{1}{\lambda} G_t$, where G_t is the total return of a trajectory τ obtained following policy ρ; the discount factor at the denominator is because the MDP requires an initial auxiliary action $a_{\mathcal{M}0}$ that we have to take into account to get the equality.

Then, we can show that the value of an orchestrator is proportional to the value of the initial state of the MDP by following policy ρ, i.e. $v(\gamma_\rho) \propto v_\rho$:

$$
\begin{aligned}
v(\gamma_\rho) &= \mathbb{E}_{h_{t,z} \sim P_{\gamma_\rho}} \left[v_{\gamma_\rho}(h_{t,z}) \cdot realizable(\gamma_\rho, \pi_t(h_{t,z})) \right] && (2)\\
&= \mathbb{E}_{h_{t,z} \sim P_{\gamma_\rho}} \left[v_{\gamma_\rho}(h_{t,z}) \right] && (3)\\
&= \mathbb{E}_{\tau \sim \mathcal{T}_{\tilde{\mathcal{M}}}, \rho} \left[\frac{1}{\lambda} G_t \right] && (4)\\
&= \frac{1}{\lambda} v_\rho && (5)\\
&\propto v_\rho
\end{aligned}
$$

where step (2) is by definition (see Eq. (1)), step (3) by assumption on the value of the predicate *realizable*, step (4) by the equivalence between joint history and trajectory of the composition MDP, and by construction of $\mathcal{T}_{\tilde{\mathcal{M}}}$, and step (5) by linearity of expectation and by definition of value of a policy.

Given the above, we have that the thesis holds because $\arg\max_\gamma v(\gamma) = \arg\max_\rho v_\rho$. □

To summarize, given the specifications of the set of stochastic services and the target service, the orchestrator first computes the composition MDP, then finds an optimal policy for it, and then deploys the policy in an orchestration setting and dispatch the request to the chosen service according to the computed policy.

5. Case study

In order to show the suitability of the proposed approach, in this section we present the real-world application scenario of a ceramics manufacturing company [2]. Fig. 2 depicts a snippet of the process to be automated and monitored expressed as a target service.

The process is a deterministic sequence of actions (as discussed in Section 4.2). The complete sequence is the following: *(1)* `provisioning`, *(2)* `moulding`, *(3)* `drying`, *(4)* `first_baking`, *(5)* `enamelling`, *(6)* `painting`, *(7)* `second_baking`, and *(8)* `shipping`. Some of the actions are followed by the corresponding *checking* actions,

[2] At the moment of writing, the case study has not been deployed yet in a real factory but it represents a real manufacturing scenario.

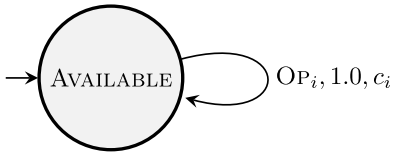

Fig. 3. A prototype of the service's MDP we consider to model simplest services.

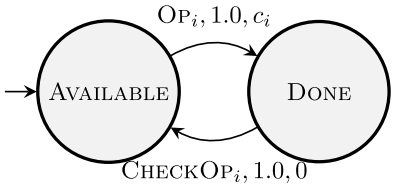

Fig. 4. A prototype of the service MDP that we consider to model services performed by a human worker.

which verify the correctness of the output (e.g., the check_moulding action checks the outcome of moulding).

Each action in the manufacturing process can be executed by different machines or human workers. In particular it is possible that the same action is provided by different models of the same machine, and that these machines can be replaced by human operators. The DTs corresponding to these actors must be modeled as stochastic services as shown in Section 4.1. Each actor is associated with a unique identifier i, which allows to specify the parameters of the associated MDPs. As an example, in the following Op$_i$ is the action implemented by actor i. We classify services into three categories, according to their complexity and provided actions.

Simplest services like those exposed by actors with no possibility to break and not provide any checking functionality. Actors in this category are external suppliers, which are seen as black boxes providing an action with a specific cost. The prototype of such services is provided in Fig. 3. Such services have a single state and a self-loop deterministic (with probability 1.0) transition triggered by the Op$_i$ action. In our case study, for simple services, we have Op$_i$ ∈ {`provisioning`, `shipping`}. The transition is associated with a cost c_i to perform the action.

Services that represent human workers have two states and no possibility to break. The prototype of such services is provided in Fig. 4. The service starts in the Available state from the Op$_i$ action is available. Executing Op$_i$ the service deterministically (with probability 1.0) ends in the Done state, with a certain cost $c_i < 0$. In the Done state the CheckOp$_i$ action is available, assumed to be executed by the target right after Op to make the service available again after it has completed an action. Noticeably CheckOp$_i$, has cost 0.0 as there is no possibility for the service to break (being a human worker) once it has performed the action correctly it can simply return available again and move on to performing the next action without being repaired with a certain cost.

Fig. 5 shows the prototype of a complex service that has the possibility to break. The service is initially in the Available state. The execution

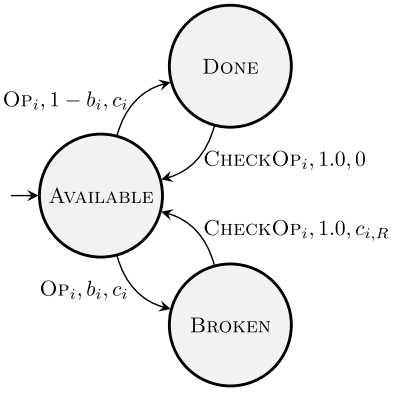

Fig. 5. A prototype of the service's MDP we consider to model complex services (i.e. services that can break).

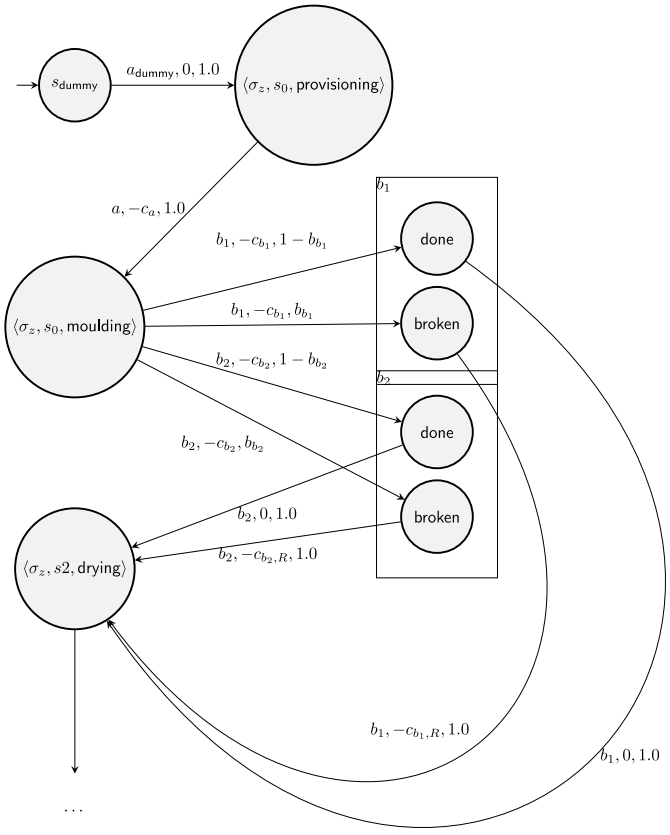

Fig. 6. The initial part of an example of composition MDP for the use case.

of the action $\textsc{Op}_i$ takes with probability b_i to the Broken state, and with probability $1-b_i$ to the Done state. In both cases, the cost of performing $\textsc{Op}_i$ is $c_i < 0$. The probability b_i models the chances of the machine to break while performing $\textsc{Op}_i$. The action $\textsc{CheckOp}_i$ is assumed to be executed by the target right after $\textsc{Op}_i$ in order to make the service available again, and additionally, to force the repairing in case the service is in the Broken state. In this latter case, $c_{i,R} < 0$, is the repair cost for the service i.

Fig. 6 depicts the initial (for the sake of space) portion of the composition MDP (see Section 4.4). The system starts dispatching the provisioning request to a simple service (forced choice); then, to process the request moulding, the orchestrator can choose between services b_1 and b_2, taking into account the costs c_{b_1}, c_{b_2} and the probability of breaking b_{b_1}, b_{b_2}. The execution continues after checking the action

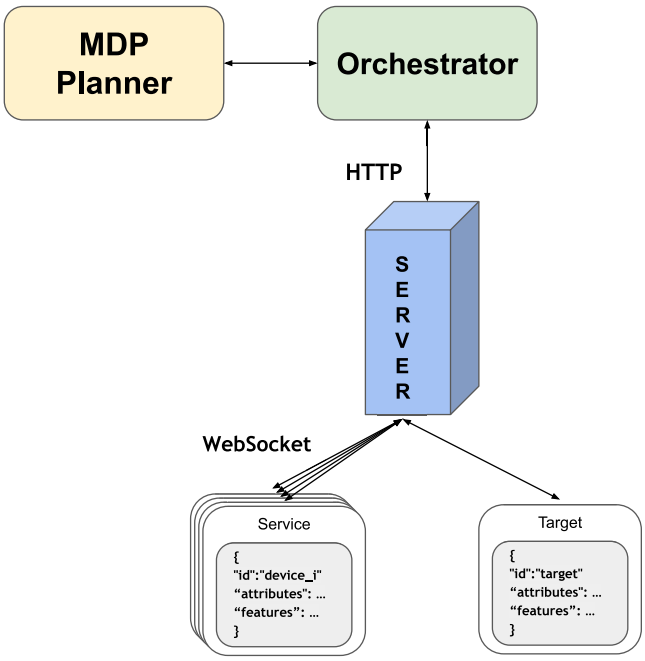

Fig. 7. Software architecture.

on the service previously chosen (a forced orchestration choice, by construction).

The goal of the orchestrator is to first find a plan such that the overall expected sum of rewards is maximized (or, equivalently, that the expected sum of costs is minimized), even if the orchestration is not guaranteed to succeed in all the cases. The plan assigns an actor to each action taking into account breaking probabilities and action and repair costs provided by the DTs. It is not straightforward indeed to determine a-priori which service a certain action must be assigned to. For example, it might be the case that despite the action cost of a machine is low, its breaking probability might be high, and considering the repair cost it might let us to prefer a human worker for that action.

6. Implementation

An implementation of the proposed case study, together with a reusable implementation of the solution techniques are freely available.[3] The communication between the orchestrator and the devices (available services and target) is managed by a *server*, a sort of platform for IoT devices that supports queries to find them, and requests for remote task execution. This server is representative of a category of middleware supporting the deployment, management and interconnection between DTs. This category also includes Amazon Digital Twin Builder, Microsoft Azure IoT, and Bosch IoT Suite. Fig. 7 depicts the software architecture of our case study. The server allows devices (both available services and target service) to connect to the server via WebSocket, while the orchestrator can interact with the server via HTTP requests. The services can connect to the server in order to register themselves and then wait for requests of action execution or maintenance tasks. The orchestrator can interact with the server in the following ways: query the server to retrieve both the specification and the current state of the available services and the current active target service, request an action from the target, request the execution of an action to be performed by a service, and request maintenance of the services.

[3] See https://github.com/luusi/digital-twins-composition-in-smart-manufacturing-via-markov-decision-processes

```
{
  "id": "complex_service",
  "attributes": {
    "type": "service",
    "transitions": {
      "available": {
        "OP": [
          {
            "done":
               1-broken_probability,
            "broken":
               broken_probability
          },
          operation_cost
        ]
      },
      "broken": {
        "CHECKOP": [
          {
            "available": 1
          },
          repair_cost
        ]
      },
      "done": {
        "CHECK_OP": [
          {
            "available": 1
          },
          0
        ]
      }
    },
    "initial_state": "available",
    "final_states": [
      "available"
    ]
  },
  ... continue to the right ...
```

```
  "features": {
    "transition_function": {
      "available": {
        "OP": [
          {
            "done":
               1-broken_probability,
            "broken":
               broken_probability
          },
          operation_cost
        ]
      },
      "broken": {
        "CHECKOP": [
          {
            "available": 1
          },
          repair_cost
        ]
      },
      "done": {
        "CHECK_OP": [
          {
            "available": 1
          },
          0
        ]
      }
    },
    "current_state": "available"
  }
}
```

Fig. 8. The definition of a complex DT device.

We implemented each actor, the target process (service), and the orchestrator as separate Python processes. The different processes do not communicate directly one each other. Their behavior specification is recorded by the server at registration time, while the features of the current state are updated during the execution of the process. The description of the services is provided as a JSON document containing:

- an **id** that uniquely identifies the device;
- a set of **attributes** containing the *static* properties of a DT. In our case, these include the MDPs of the actors, with nominal transition costs and probabilities;
- a set of **features** modeling the *dynamic* properties of a DT. In our case we use features to model probabilities and costs, which change over time because of the usage of the DT.

An example of a complex DT device is reported in Fig. 8.

The orchestrator works as a client for the server and as previously mentioned, communicates with it through the HTTP protocol. The Server dispatches then messages from the orchestrator and the DTs and vice versa.

Fig. 9 depicts the message exchange between the orchestrator and a specific service/DT_i. Here, the communication with the server is implied. Before the actual execution starts, the orchestrator collects DT descriptions, composes the MDPs, calculates the optimal policy and issues a command to ask to the target DT to request the action . At this point, the orchestrator listens to the event originating from the target service, which is a DT itself. Events produced by the target service, are requests for actions. For each event, the orchestrator sends a request to execute the action to the more convenient actor, following the computed policy. The chosen actor performs the action and updates its state and its costs and probabilities (i.e., the parameters of the MDP) for the next repetition of the manufacturing process. Once the actor completes the execution of the action, the orchestrator updates the current state of the MDP, downloads the updated MDP parameters from the actor, updates the optimal policy, and waits for the new action from the target service.

We can highlight some important implementation aspects. Every time that an available actor is used to perform a certain action, it undergoes a slight wear. Obviously, this does not happen for the services that cannot break, like services with a single state and services provided by human workers. After the service executes the action its MDP parameters change, in particular the probability that the service will end in a broken state grows.

Moreover, we know that a machine that is wearing out is less performing, so the cost of executing an action also increases. In particular, we assume that at the beginning of the manufacturing production every machine starts from a low broken probability (0.05) and a low cost to perform a certain action (−1), as it is not worn. At each iteration, i.e., at each use of the machine, it will gradually start to degrade, so we assume that the broken probability increases by 0.05 and the cost increases by 1. As we discussed in the previous section, we can have different services (i.e., actors) that can perform the same action. As an example, in our case study, the painting action can be executed both by a machine and by a human. The orchestration is able to execute actions following the optimal policy that allows the choice of the best service that has a low cost and a minimal chance of breaking. Since at every call the probability of breaking and the cost increase gradually, the optimal policy must be recalculated at every repetition of the manufacturing process. What the provided implementation shows,

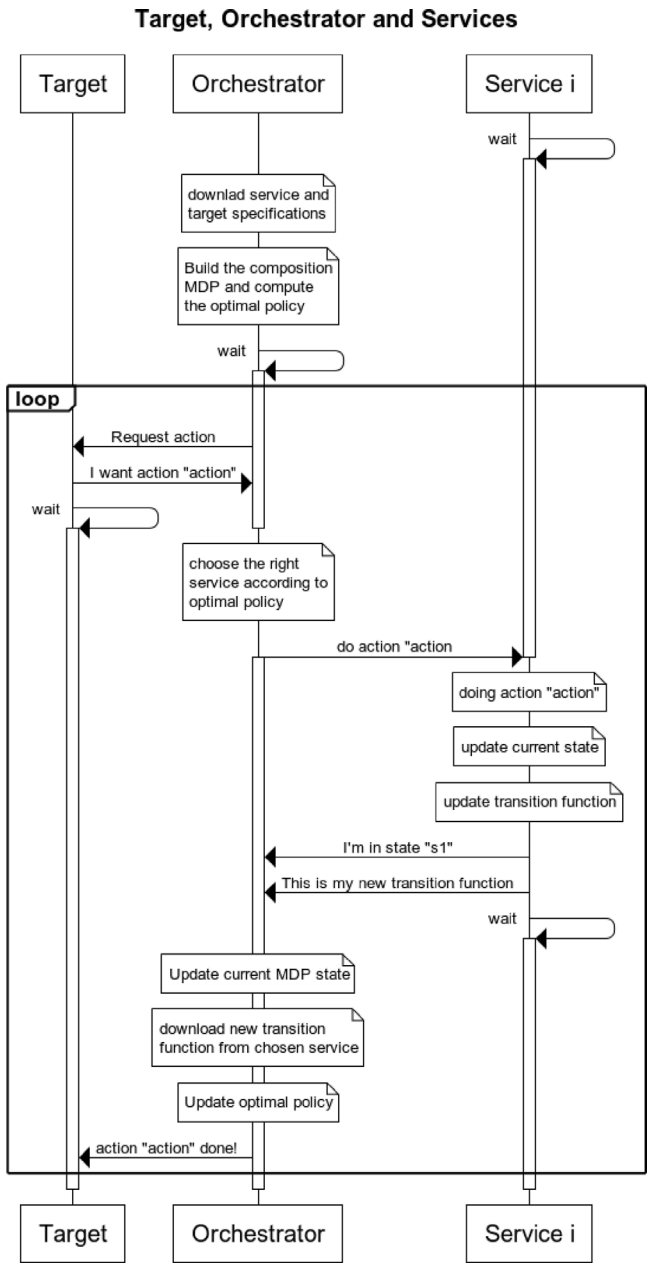

Fig. 9. Communication between Target, Orchestrator and Services.

is that despite initially the machine is chosen for the painting action (because has low-cost respect to the human), at a certain point the human will become more convenient.

The interested reader can reproduce the entire experiment by following the instructions provided with the tool code repository.

Every machine that breaks can be repaired, returning available, at a certain cost. Beside this aspect, which is taken into account while computing the new policy, we implemented a *scheduled maintenance* strategy. In particular, the orchestrator periodically sends a maintenance event to each actor, which restores its quality, i.e., resetting the breaking probability and the action execution cost. In this way, all the machines that have reached a significant state of wear can be restored and return to their initial status. This allows to overcome the problem that once a machine degrades it will no longer be chosen. Through scheduled maintenance, all the machines are checked and repaired periodically, in such a way as to ensure their optimal functioning.

7. Conclusions

In this work, we have proposed a stochastic service composition, in which also the services are allowed to have stochastic behavior and rewards on the state transitions. We formally specified the problem and proposed a solution based on a reduction to MDPs, showing how it is well-suited for a realistic Industry 4.0 scenario. We proved that our solution, under mild assumptions, guarantees that optimal policies for the MDP are also orchestrators that realize the target manufacturing process.

Besides the development of this work we considered several future research directions, aiming to enrich the theoretical framework.

In first place, the manufacturing process is described in this paper as an MDP itself, which is a stochastic finite state machine. In many cases, it is required to describe a manufacturing process from different aspects. As an example, certain choices can be driven by conditions on data, as in the case of customized (or mass-customized) products. This can be achieved in our model by providing different target services, but the inclusion of conditions and guarded transitions in the process automaton would ease the supervisor's work.

Additionally, in certain cases, a manufacturing process can be easily expressed in terms of precedence relations between manufacturing actions. This can be achieved by allowing to specify the target process using, for example, temporal logic formalisms such as Linear Temporal Logic on finite traces (LTLf) (De Giacomo and Vardi, 2013).

Another interesting aspect to be explored is the definition of safety constraints and, in general, Key Performance Indicators (KPI) to be respected from the process (Qu et al., 2019).

Exception handling is also a key feature in smart manufacturing. The proposed approach is able to cope with negatively impacting events if they are properly managed (e.g., a machine breaking), but it cannot cope with unexpected exceptions (e.g., in our case study, the unavailability of the shipping service). Techniques could be employed, inspired for example to service discovery (Rodriguez et al., 2010), to automatically cope with these situations.

All of the aforementioned research directions suggest that, in order to fully exploit DTs towards resiliency, efficiency and sustainability, a layered definition of the manufacturing process should be proposed.

Data integration aspects will be also taken into account. In our work, we suppose the employment of a common action vocabulary used by the supervisor, who specifies the manufacturing process, and DTs. This is not always the case, as different vendors could use different terms for the same actions, thus making twin specifications incompatible. Again, literature from Semantic Web services could represent an initial reference (Klusch et al., 2016).

Moreover, another interesting direction to explore is the integration of learning techniques in order to achieve greater scalability and resilience.

The theoretical contributions from this paper potentially impact many other research areas. In particular, the same methodology can be applied to other scenarios where agents can be modeled by using MDPs, such as for example collaborative robots applications and Human-Robot collaboration.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

The work of Giuseppe De Giacomo, Francesco Leotta, Massimo Mecella and Luciana Silo is partially funded by the ERC project WhiteMech (no. 834228), the PRIN project RIPER (no. 20203FFYLK), the Electrospindle 4.0 project (funded by Ministero per lo Sviluppo Economico, Italy, no. F/160038/01-04/X41). This study was carried out within the MICS (Made in Italy – Circular and Sustainable) Extended Partnership and received funding from the European Union NextGenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1551.11-10-2022, PE00000004). In particular authors were partly supported by by PE1 (CUP B53C22003980006) and PE11 (CUP B53C22004130001) initiatives. This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

Aivaliotis, P., Georgoulias, K., Chryssolouris, G., 2019. The use of Digital Twin for predictive maintenance in manufacturing. Int. J. Comput. Integr. Manuf. 32 (11), 1067–1080.

Berardi, D., Calvanese, D., De Giacomo, G., Hull, R., Mecella, M., 2005. Automatic composition of transition-based semantic web services with messaging. In: VLDB. 5, pp. 613–624.

Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M., 2003. Automatic composition of e-services that export their behavior. In: International Conference on Service-Oriented Computing. Springer, pp. 43–58.

Blythe, J., 1999. Decision-theoretic planning. AI Mag. 20 (2), 37.

Braccesi, L., Monsignori, M., Nesi, P., 2004. Monitoring and optimizing industrial production processes. In: Proceedings. Ninth IEEE International Conference on Engineering of Complex Computer Systems. IEEE, pp. 213–222.

Brafman, R.I., De Giacomo, G., Mecella, M., Sardina, S., 2017. Service composition in stochastic settings. In: Conference of the Italian Association for Artificial Intelligence. Springer, pp. 159–171.

Carreno, Y., Pairet, È., Pétillot, Y.R., Petrick, R.P.A., 2020. Task allocation strategy for heterogeneous robot teams in offshore missions. In: Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS. pp. 222–230.

Catarci, T., Firmani, D., Leotta, F., Mandreoli, F., Mecella, M., Sapio, F., 2019. A conceptual architecture and model for smart manufacturing relying on service-based digital twins. In: 2019 IEEE International Conference on Web Services. ICWS, IEEE, pp. 229–236.

Ciolek, D., D'Ippolito, N., Pozanco, A., Sardiña, S., 2020. Multi-tier automated planning for adaptive behavior. In: Proceedings of the International Conference on Automated Planning and Scheduling. 30, pp. 66–74.

De Giacomo, G., Mecella, M., Patrizi, F., 2014. Automated service composition based on behaviors: The roman model. In: Web Services Foundations.

De Giacomo, G., Vardi, M.Y., 2013. Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI'13 Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. ACM, pp. 854–860.

Durão, L.F., Haag, S., Anderl, R., Schützer, K., Zancul, E., 2018. Digital twin requirements in the context of industry 4.0. In: IFIP International Conference on Product Lifecycle Management. Springer, pp. 204–214.

Fernández, S., Aler, R., Borrajo, D., 2005. Machine learning in hybrid hierarchical and partial-order planners for manufacturing domains. Appl. Artif. Intell. 19 (8), 783–809.

Ghosh, A.K., Ullah, A.S., Kubo, A., 2019. Hidden Markov model-based digital twin construction for futuristic manufacturing systems. (AI EDAM) Artif. Intell. Eng. Des. Anal. 33 (3), 317–331.

Hull, R., 2008. Artifact-centric business process models: Brief survey of research results and challenges. In: OTM Confederated International Conferences. Springer, pp. 1152–1163.

Kagermann, H., Lukas, W.-D., Wahlster, W., 2011. Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. VDI Nachrichten 13 (1), 2–3.

Kitain, L., 2018. Digital twin—The new age of manufacturing. Online Blog Originally Posted on Seebo.

Klusch, M., Kapahnke, P., Schulte, S., Lecue, F., Bernstein, A., 2016. Semantic web service search: a brief survey. KI-Künstliche Intelligenz 30 (2), 139–147.

Krueger, V., Rovida, F., Grossmann, B., Petrick, R., Crosby, M., Charzoule, A., Garcia, G.M., Behnke, S., Toscano, C., Veiga, G., 2019. Testing the vertical and cyber-physical integration of cognitive robots in manufacturing. Robot. Comput.-Integr. Manuf. 57, 213–229.

Lee, S., Jain, S., Zhang, Y., Liu, J., Son, Y.-J., 2020. A multi-paradigm simulation for the implementation of digital twins in surveillance applications. In: IIE Annual Conference. Proceedings. Institute of Industrial and Systems Engineers (IISE), pp. 79–84.

Marrella, A., 2019. Automated planning for business process management. Journal on Data Semantics 8 (2), 79–98.

Marrella, A., Mecella, M., Sardina, S., 2016. Intelligent process adaptation in the SmartPM system. ACM Trans. Intell. Syst. Technol. 8 (2), 1–43.

Medjahed, B., Bouguettaya, A., 2011. Service Composition for the Semantic Web.

Melesse, T.Y., Di Pasquale, V., Riemma, S., 2020. Digital twin models in industrial operations: A systematic literature review. Procedia Manuf. 42, 267–272.

Negri, E., Fumagalli, L., Macchi, M., 2017. A review of the roles of digital twin in CPS-based production systems. Procedia Manuf. 11, 939–948.

Pires, F., Cachada, A., Barbosa, J., Moreira, A.P., Leitão, P., 2019. Digital twin in industry 4.0: Technologies, applications and challenges. In: 2019 IEEE 17th International Conference on Industrial Informatics. INDIN, 1, IEEE, pp. 721–726.

Popkova, E.G., Ragulina, Y.V., Bogoviz, A.V., 2019. Industry 4.0: Industrial Revolution of the 21st Century. vol. 169, Springer.

Puterman, M.L., 1994. Markov Decision Processes.

Qu, Y., Ming, X., Liu, Z., Zhang, X., Hou, Z., 2019. Smart manufacturing systems: state of the art and future trends. Int. J. Adv. Manuf. Technol. 103 (9), 3751–3768.

Rodriguez, J.M., Crasso, M., Zunino, A., Campo, M., 2010. Improving web service descriptions for effective service discovery. Sci. Comput. Program. 75 (11), 1001–1021.

Shafto, M., Conroy, M., Doyle, R., Glaessgen, E., Kemp, C., LeMoigne, J., Wang, L., 2012. Modeling, simulation, information technology & processing roadmap. National Aeronaut. Space Adm. 32 (2012), 1–38.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction.

VanDerHorn, E., Mahadevan, S., 2021. Digital Twin: Generalization, characterization and implementation. Decis. Support Syst. 145, 113524.

Xu, Z., Chang, D., Sun, M., Luo, T., 2022. Dynamic scheduling of crane by embedding deep reinforcement learning into a digital twin framework. Information 13 (6), 286.

Yadav, N., Sardina, S., 2011. Decision theoretic behavior composition. In: AAMAS.