Orchestration of services in Smart Manufacturing through automated synthesis

Flavia Monti*, Luciana Silo*†, Marco Favorito‡, Giuseppe De Giacomo*§, Francesco Leotta* and Massimo Mecella*
* Sapienza Università di Roma, Rome, Italy
Email: {monti, silo, degiacomo, leotta, mecella}@diag.uniroma1.it
† Camera dei Deputati, Italy
‡ Banca d'Italia, Italy
Email: marco.favorito@bancaditalia.it
§ University of Oxford, England, UK

***Abstract*—In recent decades, manufacturing practices have undergone a significant transformation, with the integration of computers and automation playing a central role. Concurrently, there has been a growing interest in utilizing intelligent techniques to effectively manage manufacturing processes. These processes entail the seamless integration of various activities across the supply chain. Given the diverse range of actors in a supply chain, each one with distinct characteristics such as cost, quality, and probability of failure, task assignment becomes a crucial challenge. In such a complex scenario, manual decision-making becomes impractical, necessitating the adoption of automated techniques to effectively address these challenges in a resilient and adaptive manner. This paper proposes a service-oriented approach to model each manufacturing actor within the supply chain. Furthermore, it categorizes automated synthesis approaches for smart manufacturing on the basis of *(i)* the characteristics of each actor, which are retrieved by their Industrial API, and *(ii)* the goal(s) of the manufacturing process. Finally, the paper evaluates three distinct approaches that implement automated synthesis techniques for composing services and generating operational plans.**

***Index Terms*—Smart manufacturing, Industrial API, Automated reasoning, Markov Decision Processes, Planning**

I. Introduction

Manufacturing has undergone a profound transformation in recent decades, increasingly incorporating computers and automation. This evolution, known as *smart manufacturing* [1], has taken advantage of the integration of several approaches and technologies, including Artificial Intelligence (AI), Internet-of-Things (IoT), Big Data and Cloud computing. They have significantly enhanced the efficiency of manufacturing processes by enabling automated, adaptive, optimized, and precise decision-making solutions [2]–[6]. Depending on the specific technologies which are employed, terms such as Industry 4.0 or, more recently, Industry 5.0, are also frequently used. In the following, we consider the concept of smart manufacturing as being broader and more widely accepted, and we will refer to it.

In recent years the concept of manufacturing has shifted from isolated processes to interconnected ones across multiple entities along a *smart supply chain* [7], [8]. A supply chain is typically described as a network of companies engaged in different procedures and operations that generate value for the ultimate customer in the form of goods and services [9]; it becomes smart when now it incorporates smart manufacturing technologies. This integration generates a wealth of data, providing comprehensive insights into supply chain behavior and facilitating accurate and resilient decision-making. The term *digital thread* is also used to describe the data stream produced in a supply chain [10].

Supply chains are expected to exhibit certain quality properties. The Triple-A strategy [11] emphasizes three conceptual aspects: *adaptability* – modifying the process design to account for changes in the market; *alignment* – encouraging partners to enhance overall performance; and *agility* – quickly responding to short-term changes in supply or demand. The idea of *resilience*, which is the capacity to function well in difficult situations, is also increasingly becoming important [12]. Throughout this paper, we use the term resilience to generally refer to the expected quality of a smart supply chain.

Manufacturing processes require the multi-layered integration of numerous operations across the supply chain [13]. Individual organizations operate as autonomous entities, dynamically adjusting their structures to fulfill required tasks [14]. However, these activities are susceptible to unexpected disruptions [15], which can be caused by internal malfunctions or external events such as pandemics. Designing solutions to anticipate and respond to such disruptions is crucial to mitigate negative impacts on revenues and costs.

Given the multitude of actors involved in a supply chain, ranging from humans (e.g., operators) to devices (e.g., actuators), robots (e.g., cobots), and software (e.g., business information systems), the size and diversity of supply chains are substantial. Each actor possesses unique properties, such as quality and costs, and is designed to perform specific tasks, potentially affecting the status and progress of the overall (supply chain's) process execution. Additionally, supply chains often include multiple instances or duplicates of identical actors distributed across various locations, each with distinct characteristics but producing identical outputs.

On the basis of these considerations, the optimal assignment of tasks and resources along the supply chain is essential for achieving resilience. Manual planning, implementation, and

control of supply chains require significant effort. Moreover, defining optimality in such contexts can be ambiguous due to conflicting objectives, such as cost versus quality.

Automated optimal task assignment is a common problem in logistics, often addressed using numerical optimization methods. While effective, formalizing these problems, particularly when expressing preconditions between actors, can be challenging. Symbolic AI offers automated synthesis techniques (e.g., automated planning, Markov Decision Processes (MDPs)) to model optimization problems using formal languages and achieve the best solutions according to predefined goals. However, these techniques can be computationally intensive, with costs growing exponentially with problem size.

In this paper, we *(i)* propose a service-oriented approach to model each manufacturing actor involved in the supply chain, *(ii)* categorize automated synthesis strategies for smart manufacturing based on actors' characteristics and goals, and *(iii)* present, discuss and evaluate three different approaches implementing automated synthesis techniques for composing services and generating operative plans in a complex supply chain. In such a way, we demonstrate that a service-oriented approach (i.e., modular) can achieve practical outcomes.

The paper is structured as follows. Section II presents the service-oriented modeling of manufacturing actors (and resources) accessible through an *Industrial Application Programming Interface (Industrial API)* [16]–[19]. Section III reviews and categorizes state-of-the-art techniques in the area. Section IV introduces a realistic case study. Sections V and VI demonstrate how to map the proposed categorization to selected automated synthesis techniques addressing common requirements in smart supply chains. Section VII *(i)* evaluates the performance of three selected automated synthesis techniques with respect to the problem size, *(ii)* discusses the practical applicability of the different approaches, and *(iii)* briefly introduces the AIDA tool realizing the proposed approaches. Finally, Section VIII concludes the paper outlining open challenges.

II. A Service-Oriented Approach

We propose applying a service-oriented approach to industrial *supply chains*. For simplicity, we use the term supply chain to refer to (the collection of) industrial manufacturing processes across different organizations and production sites, along with the actors involved in producing goods/services. The word *actor* refers to machines, humans, or entire organizations that interact with industrial information systems.

Following the suggestion in [16], all actors in the supply chain can be modeled in terms of the digital services they offer (e.g., a REST service). These services can be coordinated by implementing an *orchestrator* to achieve the intended goals (sometimes also referred to as mediator, composition engine, etc.). We refer to the collection of digital services provided by an actor as its *Industrial API*. Each digital service within the Industrial API offers a set of operations for instructing the actor to perform physical or virtual actions. Our work assumes that all actor operations can be described using:

- *Preconditions*. These are boolean conditions that enable the execution of an action. They are defined over a set of variables, including sensor values, configuration settings, phases, and other contextual information.
- *Postconditions*. They indicate the effects of executing an action by the actor and are expressed in terms of the variables that change. Notably, *(i)* postconditions may include a change in the machine's phase, *(ii)* effects can be probabilistic (but the actor's API should provide this information).
- *Rewards*. These indicate the contribution of executing the action to the overall supply chain. They can represent negative effects, such as quality loss associated with action execution.

We formalize the aforementioned concepts as follows.

Definition II.1. An actor is defined as $A = (X, s, s_0, O)$, where $X = \{x_1, \ldots, x_n\}$ represents the set of variables (e.g., sensor variables and machine phases), s denotes the current state of the machine (i.e., the current assignment of the variables in X), s_0 is the initial state and $O = \{o_1, \ldots, o_m\}$ represents the set of the actions (or operations) executable by the actor. Each variable $x_i \in X$ has a corresponding domain $D(x_i)$, thus $s \in S = D(x_0) \times \cdots \times D(x_n)$. An action $o \in O$, applicable for an actor in state $s \in S$, is defined as a tuple $o = \langle PRE_o, POST_o(s) \rangle$. Specifically, PRE_o is a logical expression representing the preconditions necessary for executing the action, meaning the action can be executed if the actor's state satisfies these preconditions (i.e., $s \models PRE_o$). The multifunction $POST_o(s) = \{(s'_1, r_{s'_1}, p_{s'_1}) \ldots (s'_n, r_{s'_n}, p_{s'_n})\}$ denotes the effects of the action, including the next state s'_k, the associated reward $r_{s'_k}$ and the probability $p_{s'_k}$ of reaching s'_k, where $\sum_k p_{s'_k} = 1$.

In essence, we assume that actions in O can be executed using the Industrial API provided by the actor. Furthermore, there is a continuous update of action specifications (i.e., rewards r and probabilities p) to reflect the current conditions of the actors. In the case of humans, information can be extracted, for example, from smart wristbands or information systems that report injuries [20]. In the case of machines this information can be estimated using automated techniques (e.g., remaining useful life and quality estimation) [21]. Obtaining such information is simplified when digital twins of actors are available [22]. However, these detailed aspects are beyond the scope of this paper.

According to the previous definition, we distinguish two types of actors: deterministic and stochastic. For a *deterministic actor*, actions result in a unique state and we assume that their execution is always correct, yielding exactly the expected effects. This implies that, for deterministic actors, from a state s it leads to a single s'_k such that $p_{s'_k} = 1$. In contrast, for a *stochastic actor*, the destination state depends on a variety of conditions [23], including internal factors such as malfunctions, wear level, device age, and temporary limitations, as well as external factors like environmental conditions. Cumulatively, all these factors and conditions induce a stochastic behavior which cannot be analyzed in details, but can be observed (hence the probability in the model).

We propose representing a supply chain process as a composition of Industrial APIs, where composition is achieved

by modeling the services as automata. The final goal of the process corresponds to a target service, which is a system of automata resulting from the synthesis of the available services. This approach, known in the literature on *automatic service composition* as the *Roman Model* [24], provides a framework for effectively orchestrating complex supply chain processes. In the original Roman Model, 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 an FSM too. However, this solution does not fit the requirements of smart manufacturing, as services (i.e., the manufacturing actors) do not exhibit a deterministic behavior.

Whereas the goal of this paper is the categorization of automated synthesis techniques to obtain resilience in smart manufacturing and to prove that, despite their computational complexity, they can still be used in practical settings when the size of the problem grows, it is worth discussing the relationship with the previous research on service composition. The more recent survey in this area is provided in [25], where a taxonomy of service composition approaches is provided. With respect to this taxonomy, the relevant categories are:

- Concerning the *language* dimension, we compose the *application logic* of different manufacturing actors at *run-time*, described through *diagrams*, with a *pull* mechanism in order to target a *business process*. Proposed approaches follow an *Event Condition Action (ECA)-based* paradigm to extract *control flow patterns*. *Cross cutting concerns* driving the selection include *exceptions* and *Service Level Agreements (SLAs) & Quality of Service (QoS)*.
- With respect to *knowledge reuse*, *components*, i.e., actors, are reused through *copy/paste*.
- Categorized approaches fall in the *synthesis* and *planning* categories.
- The execution engine is supposed to be a *business process engine*.
- The target users are supposed to be *domain experts*.

Finally, it is worth noting that while we model actors as services, we do not endorse any specific technology for their implementation. Services can be implemented in various ways, including synchronous/asynchronous, blocking/non-blocking, and using diverse technologies. Our modeling formulation abstracts their behavior, allowing flexibility in implementation choices.

III. Resilience in Smart Manufacturing

Resilience and flexibility of processes (not only in manufacturing) are discussed in [26], where the authors focus on general processes, with no specific reference to manufacturing processes. The deployment of resilient manufacturing processes requires, in particular, specialized supporting systems and technologies. Among them, we mention Process-aware Information Systems (e.g., MES and ERP) coupled with key-enabling technologies such as AI and Process Mining (PM) [27]. The adoption of such an approach enables the emergence of AI-augmented Business Process Management Systems (ABPMSs), a new class of process-aware information systems powered by reliable AI technology [28]. ABPMSs improve the execution of business processes, making them more adaptive, proactive, explainable, and context-sensitive.

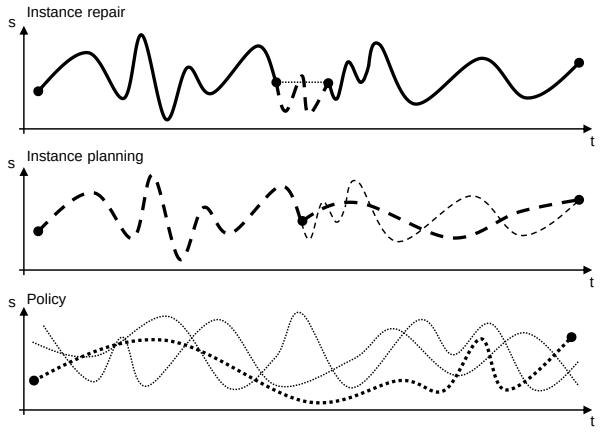


Fig. 1: Intuitive representation of a controller outcome.

Various strategies for enhancing resilience are outlined in the literature [29]. These strategies can be conceptualized as black box systems, referred to as *controllers*, which receive inputs specifying the resources involved (such as a set of services representing manufacturing entities) and the desired outcome (in our case, a manufacturing goal), and generate a resilience plan as output.

We present a conceptual framework for categorizing these approaches based on the different possibilities for inputs and targets/outcomes [30]. Concerning inputs, specifically the services, a distinction can be made between deterministic and stochastic behaviors exhibited by the entities. Additionally, the definition of the target, i.e., the manufacturing goal, can vary. It may be fully specified or loosely specified, allowing for flexibility in adaptation strategies.

Figure 1 presents an intuition of the three potential strategies derived from a controller by utilizing automated reasoning techniques. Across all strategies, the horizontal axis illustrates the progression of time, while the vertical axis encapsulates the collective state of resources (represented as a tuple) in a numerical format. Each action undertaken is denoted by a pair (a, r), where a signifies the action and r denotes the resource executing the specific action. As the sequence of actions is chosen, resources transition from an initial state to a final one, signifying the end of the process, potentially fulfilling the goal(s). To achieve the end goal, the several strategies that are generated as outputs representing processes can focus on either the process instance or the coherence of data. The various categories are outlined below.

Definition III.1 (Instance Repair)**.** This strategy involves well-defined process instances within the supply chain. In the event of an unexpected disruption (e.g., machinery failure), automated reasoning techniques are utilized to restore the state of resources to the expected condition within the original process instance. Adaptation is localized, ensuring that the overall forthcoming process remains unaltered. In Figure 1,

the process model is depicted with a solid line, while the adaptation is illustrated with a thin dashed line.

Definition III.2 (Instance Planning)**.** In this strategy, automated reasoning techniques are employed each time a new process instance is required. These techniques utilize the latest information about resources to generate an entire process instance. If any unforeseen event (e.g., a malfunctioning resource) obstructs the execution of the plan, automated reasoning is reapplied to produce a new process instance which should achieve the same goal(s) of the original one (cf. the same final point). In Figure 1, the segment of the process that cannot be executed is indicated by a thin dashed line, whereas the executed process is depicted with a thick dashed line.

Definition III.3 (Policy-based)**.** Here, automated reasoning techniques are utilized to derive a policy, which is essentially a function that suggests the next action for each state. In case of unexpected occurrences, there is no need to reapply the approach since all potential actions from each state have already been computed. In Figure 1, dashed lines represent all feasible legal instances of the process. Among these instances, based on the state of various resources, a specific one (represented as a thick dashed line) is selected.

Although theoretically, all combinations of input services and output policy representations are feasible, the literature has only explored certain configurations. In our study, we concentrate on methodologies harnessing automated reasoning techniques, a specialized field of AI tailored for representing and reasoning about dynamic domains, to facilitate adaptation in smart manufacturing, particularly within the supply chain context. Our research aligns within the broader scope of studies that apply automated planning techniques. It is pertinent to differentiate between automated planning and automated scheduling. Automated planning refers to the utilization of AI technologies to devise a correct and efficient sequence of actions [31]. Conversely, scheduling [32] addresses the task of optimally allocating time and other resources to a predefined sequence of actions. Furthermore, there is a distinction between classical planning, which handles deterministic scenarios, and decision-theoretic planning [33], which deals with stochastic behavior scenarios.

Several works exemplify the application of classical planning in smart manufacturing, specifically in implementing instance repair strategies. The authors in [34] employ automated planning to address unexpected events, employing it to rectify process instances and restore conditions to facilitate the continuation of predefined processes. Similarly, authors in [35] evaluate the use of classical planning in a physical smart factory context to manage detected exceptional situations and ensure uninterrupted process execution. Furthermore, authors in [36] introduce a variant of the classical planning approach to address the challenge of reacting to a dynamic environment like manufacturing, where plans may fail due to changing contexts. Moreover, classical planning is utilized in various works to realize instance planning strategies. In [37], the authors demonstrate the planning of small truck assembly using available components and allocating specific production operations to existing production resources. Additionally, authors in [38] combine temporal planning with computer vision to automate manufacturing tasks using low-cost robots. However, it is important to emphasize that all methods based on classical planning ignore a significant aspect of manufacturing production: the inherent uncertainty that exists throughout the manufacturing actors and the production process as a whole.

To address this limitation, employing decision-theoretic planning methodologies with the integration of MDPs could offer a viable solution. While MDPs are extensively utilized to model decision-making boundaries, their application within manufacturing remains relatively limited. A policy strategy is proposed by the authors in [39] where an MDP-based self-adaptive control model for Automated Guided Vehicles (AGVs) is proposed to facilitate navigation efficiently and safely. Even not adhering to one of the proposed strategies, other works employing MDPs are explored in the literature. For instance, in [40], authors utilize MDPs to explore the optimal control policy for real-time allocation of multiple workforce units. In [41], authors leverage an MDP to determine the optimal and cost-effective maintenance decisions based on the circumstances found during a single diesel engine examination. Additionally, aiming at creating a policy for sequential decision-making to maximize expected gains while adhering to predetermined constraints, authors in [42] present a hierarchical MDP approach for adaptive multi-scale prognostics and health management within smart manufacturing systems. In these instances, employing MDPs is particularly suitable for the manufacturing domain and other non-deterministic domains as it facilitates making the best choice under varying circumstances.

Several approaches have also been proposed in the operational research area to solve scheduling, allocation and task assignment problems in smart manufacturing [43]–[45]. These approaches include static methods such as linear programming techniques, dynamic methods such as genetic algorithms, and AI-based methods based on machine (ML) and reinforcement learning (RL) combined with heuristics and optimization methods. In this context, problem model implementation is a delicate task that must consider the complexities of the existing relations and behavior of the studied environment. Additionally, the problem model, which is formulated through mathematical formalisms, can often be challenging to interpret. In our work, we instead leverage logic-based approaches which offer a more human-readable representation, facilitating better understanding.

IV. Case Study

In this section, we present the case study of a supply chain producing integrated circuits, commonly known as chips, referred to as *ChipChain*[1]. The process involves multiple actors, including suppliers, machines, operators, and software, all dedicated to producing CPUs (central processing units). Chip production is a precise and accurate procedure comprising several crucial steps. It begins with the acquisition

[1]Cf. https://www.screen.co.jp/spe/en/process, https://www.asml.com/en/news/stories/2021/semiconductor-manufacturing-process-steps and https://www.amd.com/en/technologies/introduction-to-semiconductors.

of essential elements and raw materials, such as chip design blueprints, silicon, impurities, resistance, and chemicals. The mask template is then created to define the pattern of the silicon wafer. Subsequently, photolithography operation is employed to transfer this intricate pattern onto the wafer, requiring precise alignment and exposure to ensure accuracy. The process further incorporates ion implantation, a critical step involving the introduction of specific dopants to modify the electrical properties of the wafer. This step is crucial for establishing regions with varying conductivity for the formation of transistors and the altering of semiconductor material behavior. Rigorous functionality tests and quality checks follow to ensure the standards and reliability of the produced chips. The fabricated wafer is then carefully diced into individual chips. Finally, the chips undergo packaging, a step designed to provide not only protection but also effective cooling solutions, ensuring high performance and durability.

The manufacturing process is primarily based in the United States (US), given that a significant portion of leading chip suppliers worldwide are situated there[2]. Furthermore, we have identified the states involved in the procurement phase and treated them as individual entities within the supply chain. Additionally, we have established the cost associated with each operation by gauging the distance between the US and the respective states involved. For instance, acquiring the chip design from an organization in the United Kingdom incurs a cost of 6.8. Concerning the manufacturing process itself, since all manufacturing actors are situated within a single factory in the US, the cost of operations therein is standardized to one. However, we consider scenarios where multiple copies of the same actor undertake identical operations, in which case their cost is set to a value greater than one.

The proposed case study is used in the following sections to present three approaches orchestrating the manufacturing actors adaptively and resiliently, with the final goal of chip production. Specifically, we describe *(i)* an instance planning approach employed in a setting based on deterministic services and a loosely specified target (see Section V), *(ii)* a policy-based approach applied to services exhibiting stochastic behaviors and a fully specified target (see Section VI-A), and *(iii)* another policy-based approach employed with services with stochastic behaviors and a loosely specified target (see Section VI-B).

The three approaches presented in the following cover two out of the three categories introduced in Section III. In particular, we are not introducing any instance repair approach. The reason is as it follows: in the instance repair approaches process models and involved resources (i.e., actors) are predefined and available data are only used once something odd happens, conversely in real world scenarios it is fundamental to use information as soon as it is available, triggering adaptivity since the very beginning of a process instance. Instance planning and policy-based approaches are based on this latter, more realistic, intuition.

[2]Cf. https://macropolo.org/digital-projects/supply-chain/ai-chips/ai-chips-supply-chain-mapping.

V. Instance planning

By harnessing automated planning techniques, one can seamlessly coordinate the supply chain to meet precise manufacturing objectives while adhering to anticipated Key Performance Indicators (KPIs) [16]. Automated planning refers to the automated synthesis of plans derived from a model that outlines the operational dynamics of the environment in a concise mathematical style. Its integration with industrial manufacturing enables the automatic adjustment, enhancement, and coordination of the tasks required to meet output targets.

Definition V.1 (Classical planning)**.** Classical planning deals with the selection of actions to achieve goals under conditions where the initial situation is fully understood and actions yield deterministic outcomes [46]. In essence, a classical planning problem can be conceptualized as akin to a path-finding problem within a directed graph. Here, nodes symbolize states, and edges represent actions that transition the state from the source node to the target node. A plan, denoting the sequence of actions necessary to transform the initial state into a goal state, can thus be represented as a path originating from the initial node in the graph and extending to a node whose state aligns with one of the specified goal states of the problem.

A *classical planning problem* [47] is formally described as (X, I, γ, O), where X is the set of state variables, I is the description of the initial state of the system (i.e., an evaluation over X), γ is a formula over X representing the goal condition, and O is a list of operations (or actions). An action $o \in O$ is defined as a tuple $o = \langle \chi, e \rangle$, where χ is the *precondition* and e is the *effect*. Both precondition and effect are conjunctions of literals (positive or negative atomic sentences) over X. χ defines the states in which o can be executed, i.e., o is applicable in a state s if and only if $s \models \chi$; and e defines the result of executing o. For each effect e and state s, $[e_s]$ denotes the set of state variables whose value is modified when executing the action. The *successor state* of s with respect to the action o, is the state s' such that $s' \models [e_s]$ and $s'(v) = s(v)$ for all state variables v not mentioned in $[e_s]$. Solving a planning problem means automatically finding a sequence of actions that, when applied to the initial state, leads to a state s such that $s \models \gamma$ (s *satisfies* γ).

The state space of a problem may be huge. Throughout the years, numerous heuristics and algorithms have emerged and integrated into *planners* (planning systems) to efficiently seek solutions. Planners receive the problem model presented in the format of *domain* and *problem* descriptions using a standardized interface language. This division of descriptions makes it easier to distinguish intuitively between parts that define the specific problem (such as available objects, initial state, and objective) and elements that are inherent in each unique problem within the domain (like types, predicates, functions, and goal). The Planning Domain Definition Language (PDDL) [48] is the standard language used by classical planners. It allows the definition of the planning problem covering and extending STRIPS (Stanford Research Institute Problem Solver [49]), where the variables are boolean and specify whether a proposition about the world holds in a given state. Different extensions and versions of PDDL have been

developed to increase expressivity and describe more realistic problems, e.g., problems with time and number elements. However, classical planners ground the actions defined in PDDL, transforming predicates, variables, and constants into a propositional representation like STRIPS.

We describe a method for orchestrating services using classical planning to achieve a production goal and adjust to unexpected events. The involved manufacturing actors behave deterministically. That is, given an actor $A_i = (X_i, s_i, s_{0i}, O_i)$ with $o_i = \langle PRE_{o_i}, POST_{o_i}(s_i)\rangle$ (cf. Definition II.1), the multifunction $POST_{o_i}(s_i)$ leads to exactly one next state s_i' with probability $p_{s_i'} = 1$. This means that the effects of executing the actions lead to at most one successor state. A classical planning problem for a given manufacturing goal γ and a set of n deterministic manufacturing actors $\{A_1, \dots A_n\}$ can be derived in a straightforward way as (X, I, γ, O), where $X = \bigcup_{i=0}^{n} X_i$, $I = \{s_{01}, \dots, s_{0n}\}$ and $O = \bigcup_{i=0}^{n} O_i$.

The Industrial API of each actor A_i provides information, modeled in a PDDL-like fashion, related both to the actions O_i runnable by the specific actor (cf. Section IV) and the internal properties s_i indicating their current status, which might change over time. Each action contains specifics of the parameters, requirements, and effects (both positive and negative) necessary to perform such an action. In our modeling, we consider the notion of actor's objects, i.e., entities without actions, representing passive objects having nothing but properties to be satisfied to reach the production goal. In our case study, among the involved actors, we include raw materials and design. The other actors in the case study are inventories, machines, and humans, who, unlike passive items, specify the activities they are capable of performing.

The information retrieved by the Industrial API is combined with the production goal γ. All of this is translated into a classical planning problem. The domain description (or simply domain) contains the list of the *types* of actors involved in the problem (e.g., Material, Service[3]) and their inheritance. Additionally, it describes the *predicates* that apply either to specific types of objects or to all objects and are either true or false at any point in the plan. An example of a predicate included in our case study is *Taken(d)*, meaning that the design *d* has been taken. Finally, the domain contains the list of actions with the preconditions, effects, and the cost of execution which is represented as a *function* (similar to a predicate) named *total-cost*.

The list of actions contained in the domain corresponds to the operations offered by the available actors. In our case study, since we have multiple copies of similar actors performing actions having the same effects, we are forced to distinguish the actions' names, hence having different operations related to an action. For example, three operations that take a design (i.e., *TakeDesignCN*, *TakeDesignUK*, and *TakeDesignUS*) could be present if three actors providing the design are involved. Below, we present the grounded action schema of the *TakeDesignCN* action[4].

```
Action(TakeDesignCN(srv, d),
  PRECOND: Provides(srv, takingDesign) ∧ ¬Taken(d) ∧
           Status(srv, available)
  EFFECT: Taken(d) ∧ Increase(total-cost, 11))
```

It defines the parameters (i.e., a service *srv* and a design *d*), the preconditions (i.e., the service *srv* must be able to take a design *and* the design *d* must not already be taken *and* the service *srv* need to be available) and the effects of the action (i.e., the design *d* is taken *and* the cost of the plan is increased by 11).

The problem description (or simply problem) contains the *objects* involved in the problem. The term objects has a more generic meaning here, referring to all the possible arguments of the predicates. In our case study, the problem contains the list of the actor identifiers and the *types*, e.g., *design - Design*, *takingDesign - Capability*, *designerchina - Service_des_ch*. Furthermore, an *init* section is present that defines which predicates are true at the beginning of the problem, i.e., the initial state I. The data contributing to the initial state is collected from the Industrial API of the actors, including information on the state variables X and the machine operating status, which, in our case, can assume values from the set $\{$*available, broken*$\}$. Finally, the problem specifies the *goal* of the process. The goal of *ChipChain* is *Packed(wafer)* since packaging is the last operation of chip manufacturing.

The *plan* produced by the planner comprises a sequence of actions aimed at achieving the final goal while minimizing *total-cost*. These outcomes precisely entail the effective orchestration (synthesis) of the digital services available to the actors to fulfill a production objective. It's crucial to emphasize that prior to executing an action (via the designated Industrial API), the current status of the relevant actor is verified. If the status is *available*, the action request is sent. Conversely, if the status is *broken*, a re-evaluation of the plan commences to ensure the attainment of the final goal. As outlined in Section III, the proposed method employs an instance planning approach, generating a new plan based on the current environmental state when one or more actors are in a broken state.

VI. Policy-based Synthesis through MDPs

The actors involved in industrial manufacturing are commonly entities that can exhibit stochastic behaviors. Manufacturing actors, indeed, often present behaviors where, depending on some probability, they act as either working or broken actors. Consequently, orchestrating the manufacturing process becomes a probabilistic problem. This problem can be addressed by leveraging Markov Decision Processes (MDPs) which can model the probabilities and costs related to the execution of an action. In particular, differently from Section V, the Industrial APIs of the actors are mapped to a set of MDPs.

Definition VI.1 (Markov Decision Processes)**.** An MDP [50] $\mathcal{M} = \langle \Sigma, O, P, R, \lambda\rangle$ is a discrete-time stochastic control process consisting of *(i)* a set Σ of states, *(ii)* a set O of

[3]We refer to all non-passive actors, such as machines and inventories, as being of the type `Service`. This contrasts with the concept of digital services in the Industrial API of an actor.

[4]The complete domain expressed using PDDL can be found at https://github.com/iaiamomo/TSC-planning-experiments/tree/main/res_pddl.

actions, *(iii)* a transition function $P : \Sigma \times O \to \mathbb{P}(\Sigma)$ that returns for every state σ and action o a distribution over the next state, *(iv)* a reward function $R : \Sigma \times O \to \mathbb{R}$ that specifies the reward (or the operation cost) when transitioning from state σ to state σ' by applying action o, and *(v)* a discount factor $\lambda \in (0, 1)$. The discount factor λ indicates the importance of future rewards. Values of λ close to 0 specify an interest in immediate rewards, while values of λ close to 1 indicate an interest for future rewards. A solution to an MDP is represented by a function known as a *policy*, which maps each state to an action, potentially influenced by past states and actions [51]. The *value* of a policy ρ at a given state σ, denoted as $v_\rho(\sigma)$, represents the expected sum of rewards obtained when initiating from state σ and choosing actions according to policy ρ. The policy value $v_\rho(\sigma)$ could be discounted by the factor λ. Every MDP has an optimal policy ρ^*, i.e., a policy that maximizes $v_\rho(\sigma)$. Typically, the MDP is assumed to start in an initial state σ_0 and policy optimality is evaluated with respect to it, i.e., $v_\rho(\sigma_0)$. In the realm of finding an optimal policy of an MDP, notable techniques include *value iteration* and *policy iteration* [51].

We describe an MDP-based approach that orchestrates manufacturing actors by finding a policy that accomplishes a manufacturing goal. We consider the manufacturing actors that behave in a stochastic manner, i.e., $A_i = (X_i, s_i, s_{0i}, O_{A_i})$ with $o_i = \langle PRE_{o_i}, POST_{o_i}(s_i)\rangle$ (cf. Definition II.1), where the next state s_i' and the reward $r_{s_i'}$ of the effects $POST_{o_i}$ depend on the probability $p_{s_i'}$. We identify each actor A_i as an MDP $\mathcal{M}_i = \langle \Sigma_i, O_i, P_i, R_i, \lambda_i\rangle$. In particular, $\Sigma_i = \{s_j : \forall o_j \in O_{A_i}, s_j \models PRE_{o_i}\}$, $O_i = O_{A_i}$, $P_i = \{p_{s_i'} : \forall o_i \in O_{A_i}, p_{s_i'} \in POST_{o_i}(s_i) \wedge s_i \models PRE_{o_i}\}$ and $R_i = \{r_{s_i'} : \forall o_i \in O_{A_i}, r_{s_i'} \in POST_{o_i}(s_i) \wedge s_i \models PRE_{o_i}\}$. The resulting translation is not straightforward since, on the one hand, the MDPs rely on the concept of states, while, on the other hand, the formalization of actors proposed in Section II is described in terms of their actions, which here contributes to the definition of the MDPs representing the actors.

We distinguish different types of actors involved in the *ChipChain* case study, i.e., *humans*, *warehouses*, *generic breakable actors* and *complex breakable actors*, which differ in the number of states they are composed of. The human, depicting a human worker, has one state, is less efficient and does not break. For instance, we model as humans the actors performing the *testing* and *packaging* operations. Like humans, the warehouse has one state and permits the retrieval of raw materials and objects provided along the supply chain. The generic and complex breakable actors have three and five states respectively. The former represents machines that may break while executing an operation, while the latter represents machines having configuration and verification phases to incur breakages less frequently. An important aspect the breakable actors expose is that both can be repaired in case of failures.

We characterize the MDPs actors as *stochastic services* [52]. A *stochastic service*, also known simply as a service, is defined by the tuple $\tilde{\mathcal{S}} = \langle \Sigma_s, \sigma_{s0}, F_s, O_s, P_s, R_s\rangle$, where: *(i)* Σ_s is the finite set of service states, *(ii)* $\sigma_{s0} \in \Sigma$ is the initial state, *(iii)* $F_s \subseteq \Sigma_s$ is the set of final states, *(iv)* O_s is the finite

Fig. 2: The MDP of the *silicon warehouse* actor.

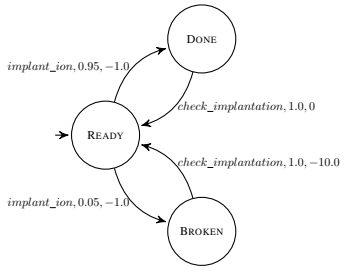


Fig. 3: The MDP of the *ion implanter* machine.

set of services actions, *(v)* $P_s : \Sigma_s \times O_s \to Prob(\Sigma_s)$ is the transition function, and *(vi)* $R_s : \Sigma_s \times O_s \to \mathbb{R}$ is the reward function. The stochastic service can be conceptualized as an MDP, facilitating a flexible modeling approach for real manufacturing actors. This approach allows for the definition of various states, including conditions of unavailability (such as a malfunctioning machine) along with their associated probabilities. Rewards can also be used to model different aspects like the loss of quality over time or the cost of execution. Stochastic services are continuously monitored via the Industrial API to retrieve information. Figure 2 provides the model of the *silicon warehouse*. It has a single state and a self-loop deterministic transition triggered by the $pick\_silicon$ action which has a cost of -1.0. Notably, the same structure is used for all human and warehouse actors.

Figure 3 illustrates the *ion implanter* machine depicted as a generic breakable actor. Initially, the machine resides in the READY state, ready to undertake an action OP_i (e.g., $implant\_ion$). This action may result in the machine transitioning to the BROKEN state with a probability p_i^b, or to the DONE state with a probability $1 - p_i^b$. No matter of the outcome, executing OP_i incurs a certain cost $c_i < 0$. Following the execution of OP_i, the actor performs the action CHECKOP_i (e.g., $check\_implantation$) to prepare itself for subsequent operations and to trigger repairs in case it ends up in the BROKEN state. In the latter scenario, $c_i^r < 0$ denotes the repair cost for actor i. Actors involved in executing generic operations, designed as generic breakable actors, are characterized in this manner.

A more autonomous and comprehensive machine can be represented as a *complex breakable* actor, as depicted in Figure 4. Initially, the actor is in the AVAILABLE state, where it awaits commands. Upon receiving the CONFIG[DEV] command, it transitions to the CONFIGURATION state for setup and warming up. Subsequently, upon execution of the CHECKED[DEV] action, if the configuration fails (with a probability p_i^u), the actor moves to the BROKEN state; otherwise (with a probability $1 - p_i^u$), it proceeds to the READY state, where a series of operations (OP) can be performed. In certain instances, executing an action OP_i may lead the actor (with a probability p_i^b) to the BROKEN state, incurring

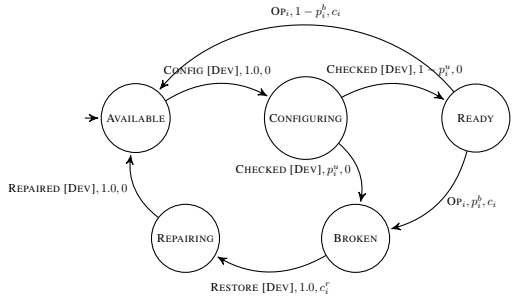


Fig. 4: The (template) MDP of complex breakable actors.

a cost $c_i < 0$. Regardless of the cause, when the actor ends up in the BROKEN state, it must undergo a RESTORE[DEV] operation (incurring a repair cost $c_i^r < 0$) to transition to the REPAIRING state, followed by a REPAIRED[DEV] operation that returns it to the AVAILABLE state, making it available for further use. Challenging operations involve actors modeled as complex breakable services.

A set of actors (stochastic services) define a community of services and we use the concept of *stochastic system service* [52] $\tilde{\mathcal{Z}} = \langle \Sigma_z, \sigma_{z0}, F_z, O_z, P_z, R_z \rangle$ to model it. A stochastic system service is defined as follows: *(i)* $\Sigma_z = \Sigma_1 \times \cdots \times \Sigma_n$ is the set of the states of all the involved services, *(ii)* $\sigma_{z_0} = (\sigma_{10}, \ldots, \sigma_{n0})$ is the set of current states of each service, *(iii)* $F_z = \{(\sigma_1, \ldots, \sigma_n) \mid \sigma_i \in F_i, 1 \leq i \leq n\}$ is the set of the final states of the services, *(iv)* $O_z = O_i \times \{1, \ldots n\}$ is the set of pairs (o, i) formed by a shared action o and the index i of the service that executes it, *(v)* $P_z(\boldsymbol{\sigma'} \mid \boldsymbol{\sigma}, (o, i)) = P(\sigma_i' \mid \sigma_i, o)$, for $\boldsymbol{\sigma} = (\sigma_1 \ldots \sigma_n)$, $\boldsymbol{\sigma'} = (\sigma_1' \ldots \sigma_n')$ and $o \in O_i(\sigma_i)$, with $\sigma_i \in \Sigma_i$ and $\sigma_j = \sigma_j'$ for $j \neq i$, represent the transition function, and *(vi)* $R_z(\boldsymbol{\sigma}, (o, i)) = R_i(\sigma_i, o)$ for $\boldsymbol{\sigma} \in \Sigma_z$, $o \in O_i(\sigma_i)$ depict the reward function.

In the following sections, we propose two techniques relying on the above-mentioned concepts: *(i)* Section VI-A describes a solution based on a fully specified manufacturing goal, and *(ii)* Section VI-B solves the problem in the presence of a loosely specified manufacturing goal.

A. Fully specified target process

A manufacturing process can usually be represented using imperative process model notations such as Petri Nets and BPMN [53]. Such notations explicitly specify all possible behaviors, prescribing the execution flow in its entirety. In this section, we show how to synthesize a policy when the target to realize is a fully specified process. In particular, as demonstrated in prior work [52], [54], the manufacturing goal will be modeled as an additional service, referred to as the *target service* [52], [54], to be achieved by combining the actions of actor services.

Formally, the *target service* is specified by the tuple $\mathcal{T} = \langle \Sigma_t, \sigma_{t0}, F_t, O_t, \delta_t, P_t, R_t \rangle$, where *(i)* Σ_t represents the finite set of target states, *(ii)* $\sigma_{t0} \in \Sigma$ denotes the initial state, *(iii)* $F_t \subseteq \Sigma$ indicates the set of the target final states, *(iv)* O_t denotes the finite set of target actions, *(v)* $\delta_t : \Sigma_t \times O_t \to \Sigma_t$ signifies the deterministic and partial transition function, *(vi)* $P_t : \Sigma_t \to \pi(O_t) \cup \emptyset$ represents the action distribution function, typically uniform distribution, indicating which actions can be executed in each state, and *(vii)* $R_t : \Sigma_t \times O_t \to \mathbb{R}$ denotes the reward function. This definition characterizes the target service as a specific instance of an MDP. Unlike stochastic services, the target service is primarily deterministic in nature. This concept serves the functional purpose within our formal framework, with $P_t \in \{0, 1\}$ and R_t remaining constant.

It is worth noting that the set $O_t \subseteq O_z$, as the target may ignore specific actions that are instead needed for composing services to operate. These latter are called *auxiliary actions*.

Figure 5 depicts the target service of the *ChipChain* case study (due to space constraints, some steps are omitted). The target service represents the deterministic sequence of actions related to the process. It is evident the intuition of the "fully specified" concept: the entire set of actions and their order of execution are specified, including the auxiliary ones, e.g., CONFIG[CREATOR], CHECKED[CREATOR], $check\_test\_smart$ for the relative services characterized by that property.

The proposed approach's solution is based on finding an optimal policy for the *composition MDP*. Given the specification of the stochastic system service and the target service, the composition MDP is computed as $\tilde{\mathcal{M}}(\tilde{\mathcal{Z}}, \tilde{T}) = \langle S_{\tilde{\mathcal{M}}}, O_{\tilde{\mathcal{M}}}, T_{\tilde{\mathcal{M}}}, R_{\tilde{\mathcal{M}}}, \lambda \rangle$, where: *(i)* $S_{\tilde{\mathcal{M}}} = \Sigma_{\tilde{\mathcal{Z}}} \times \Sigma_{\tilde{\mathcal{T}}} \times O \cup \{s_{\mathcal{M}0}\}$ is the set of states which, in this case, contains also the next actions; *(ii)* $O_{\tilde{\mathcal{M}}} = \{o_{\mathcal{M}0}, 1, \ldots, n\}$ is the set of actions that are the selections of a specific service that execute an action (specified in the state); *(iii)* the transition function is defined as $T_{\tilde{\mathcal{M}}}(s_{\mathcal{M}0}, o_{\mathcal{M}0}, (\sigma_{z0}, \sigma_{t0}, o)) = P_t(\sigma_{t0}, o)$, $T_{\tilde{\mathcal{M}}}((\sigma_z, \sigma_t, o), i, (\sigma_z', \sigma_t', o')) = P_t(\sigma_t', o') \cdot P_z(\sigma_z' \mid \sigma_z, \langle o, i \rangle)$ if $P_z(\sigma_z' \mid \sigma_z, \langle o, i \rangle) > 0$ and $\sigma_t \xrightarrow{o} \sigma_t'$ and 0 otherwise, it takes into account the probability of transitioning to the stochastic system service successor state σ_z' from σ_z doing the action $\langle o, i \rangle$; *(iv)* $R_{\tilde{\mathcal{M}}}((\sigma_z, \sigma_t, o), i) = R_t(\sigma_t, o) + R_z(\sigma_z, \langle o, i \rangle)$ if $(o, i) \in O(\sigma_z)$ and 0 otherwise is the reward function which includes also the reward observed from doing a system action $\langle o, i \rangle$ in σ_z, and sums it to the reward signal coming from the target; and *(v)* λ is the discount factor.

The objective is to derive a plan (policy) from the composition MDP that delineates a sequence of actions assigned to actors (services) for execution. The primary aim is to consider breaking probabilities and action and repair costs to devise a plan where the overall expected sum of costs is minimized (i.e., an optimal policy). The optimal policy illustrates a comprehensive orchestration of the actors (and their digital services) involved in the problem. The digital services provide information about breaking probabilities and costs; therefore, determining beforehand which actor a certain action should be assigned to is not straightforward. The computation of the optimal policy employs the *value iteration* algorithm [51].

B. Loosely specified target process

In the following, rather than utilizing a structured formalism as demonstrated in Section VI-A, we employ DECLARE, a flexible process model that describes the process through the definition of constraints [55]. Declarative process models enable the description of dynamic environments where processes

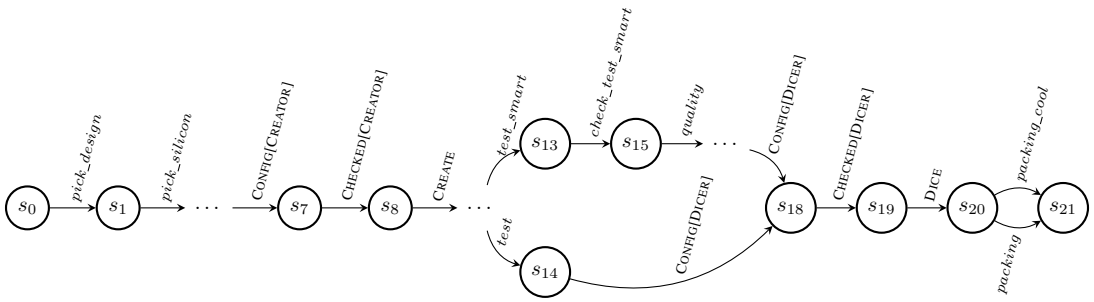


Fig. 5: The fully specified target service of *ChipChain*

are flexible and subject to changes. The proposed technique is an extension of the approach presented in Section VI-A, with a notable distinction in the representation of the target specification for the manufacturing process.

Definition VI.2 (Linear Temporal Logic & DECLARE)**.** LTL_f is a variant of Linear-time Temporal Logic (LTL) interpreted over finite, instead of infinite, traces [56]. Given a set $\mathcal{P}$ of atomic propositions, LTL_f formulas φ are defined by $\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi$, where a denotes an atomic proposition in $\mathcal{P}$, $\bigcirc$ is the next operator, and $\mathcal{U}$ is the until operator. We use abbreviations for other Boolean connectives, as well as the following: eventually as $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$; always as $\Box\varphi \equiv \neg\Diamond\neg\varphi$; weak next as $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$ (on finite traces, $\neg\bigcirc\varphi$ is not equivalent to $\bigcirc\neg\varphi$); and weak until as $\varphi_1\,\mathcal{W}\,\varphi_2 \equiv (\varphi_1\,\mathcal{U}\,\varphi_2 \vee \Box\varphi_1)$ (φ_1 holds until φ_2 or forever). LTL_f formulas are interpreted over finite traces, denoted as $a = a_0 \ldots a_{l-1}$, where a_i at instant i represents a propositional interpretation from the alphabet $2^{\mathcal{P}}$, and l is the length of the trace. An LTL_f formula φ can be converted into a deterministic finite automaton (DFA) $\mathcal{A}_\varphi = \langle \mathcal{P}, Q, q_0, F, \delta \rangle$ where *(i)* $\mathcal{P}$ represents the alphabet, *(ii)* Q denotes a finite set of states, *(iii)* q_0 signifies the initial state, *(iv)* $F \subseteq Q$ indicates the set of accepting states and *(v)* $\delta : Q \times \mathcal{P} \to Q$ represents the transition function. The alphabet of the DFA corresponds to the set of traces satisfying the formula φ.

DECLARE is a language and framework designed for the declarative constraint-based modeling of processes [57]. It revolves around a set $\mathcal{P}$ of propositions representing atomic tasks, which serve as the fundamental units of work within the process. In a DECLARE model, denoted as $\mathcal{C}$, LTL_f constraints over $\mathcal{P}$ are employed to define and constrain the permissible execution traces. DECLARE operates under the assumption that at any given point in time, precisely one task is executed. This assumption is captured implicitly by the following LTL_f formula, referred to as the DECLARE *assumption*: $\xi_{\mathcal{P}} = \Box(\bigvee_{a\in\mathcal{P}} a) \wedge \Box(\bigwedge_{a,b\in\mathcal{P},a\neq b} a \to \neg b)$. Among all possible LTL_f constraints, some specific *patterns* have been singled out as particularly meaningful for processes, i.e., *existence*, *choice*, *relation*, and *negation* constraints. These patterns play a crucial role in defining the constraints within a DECLARE model.

As proposed in [58], the process specification is defined as an LTL_f formula φ over the set of propositions $\mathcal{P}$, delineating the allowed traces for the process. The collection of finite traces satisfying the specification φ alongside the DECLARE assumption $\xi_{\mathcal{P}}$ can be encapsulated by a single deterministic process DFA $\mathcal{A}_\varphi$. This is achieved through the following steps: *(i)* generating the corresponding nondeterministic finite automaton (NFA), involving an exponential computational step, *(ii)* converting the NFA into a DFA, also entailing an exponential computational step [59], and *(iii)* trimming the resulting DFA by eliminating any state from which no final state is reachable, which is a polynomial computational step. The resulting DFA serves as a process, meaning that at each step, contingent solely upon the history (i.e., the current state), it delineates the set of actions that are legal and ultimately lead to a final state. This assumes fairness of execution, disallowing the process from indefinitely remaining in a loop.

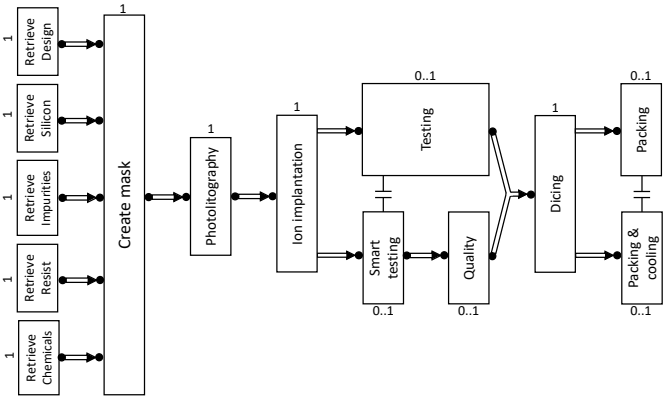


Fig. 6: The supply chain manufacturing process represented using DECLARE.

Figure 6 depicts the DECLARE model of the *ChipChain* case study. The procurement phase involves the acquisition of five materials. These are retrieved in any order and then eventually the mask is defined (*alternate succession* constraints). After the mask is created, photolithography and ion implantation operations are performed (*alternate succession* constraints), and at most one (*alternate precedence* and *not coexistence* constraints) between testing and the branch smart testing and quality check is carried out. Subsequently, the dicing operation yields single chips (*precedence* branched constraint) that are then packed into either standard or with cooling pack (*alternate precedence* and *not coexistence* constraints).

We define the manufacturing actors as stochastic services capable of performing the process actions in $\mathcal{P}$. To enhance our model, we allow services to carry out a wider range of

actions, denoted as $\mathcal{P}'$ s.t. $\mathcal{P} \subseteq \mathcal{P}'$, specific to the factory model aiming to implement the manufacturing process.

In this case, the composition MDP is defined as a function of the stochastic system service $\tilde{\mathcal{Z}}$ and the DFA $\mathcal{A}_{\varphi}$ derived from the LTL_f formula φ. It is represented as $\mathcal{M}(\tilde{\mathcal{Z}}, \mathcal{A}_{\varphi}) = \langle S_{\mathcal{M}}, O_{\mathcal{M}}, T_{\mathcal{M}}, R_{\mathcal{M}}, \lambda \rangle$, where: *(i)* the set of states $S_{\mathcal{M}} = \Sigma_z \times Q$ is the product of the states of the system service and the DFA states; *(ii)* $O_{\mathcal{M}}$ represents the set of actions and comprises the product between the DFA action and the service acting; *(iii)* the transition function is defined as follows $T_{\mathcal{M}}((\boldsymbol{\sigma}, q), \langle o, i \rangle, (\boldsymbol{\sigma'}, q')) = P_z(\boldsymbol{\sigma'} \mid \boldsymbol{\sigma}, \langle o, i \rangle)$ if $\langle o, i \rangle \in O_z(\boldsymbol{\sigma}) \wedge \Big((o \in \mathcal{P} \wedge q \xrightarrow{o} q') \vee q' = q \Big)$ and 0 otherwise, here, $P_z(\boldsymbol{\sigma'} \mid \boldsymbol{\sigma}, \langle o, i \rangle)$ represents the probability of transitioning to the system successor state $\boldsymbol{\sigma'}$ from $\boldsymbol{\sigma}$ by performing the system action $\langle o, i \rangle$ and q' denotes the successor state of q in $\mathcal{A}_{\varphi}$ after reading o if it is a process action ($o \in \mathcal{P}$), otherwise, $q' = q$ indicating the system remains in the same state; *(iv)* $R_{\mathcal{M}}((\boldsymbol{\sigma}, q), \langle o, i \rangle, (\boldsymbol{\sigma'}, q'))$ is the reward function that models the process specification φ, it is either equal to 1 if $q' \in F$, or equal to $R_z(\sigma_i, o, \sigma_i')$ if $\langle a, i \rangle \in \hat{O}_z(\boldsymbol{\sigma})$, or 0; particularly the reward function returns 1 if the automaton component of the state is an accepting state $q' \in F$; *(v)* λ is the discount factor.

An optimal policy of the composition MDP that minimizes the overall expected sum of costs can be computed as in Section VI by employing the *value iteration* algorithm. The solution induces an orchestration that coincides with the exact solution if a composition exists. Otherwise, it provides an approximate solution that maximizes the expected discounted sum of values of user requests that can be serviced.

VII. Experiments and Discussion

As stated in Section I, the primary disadvantage of automated synthesis techniques is that their performance quickly degrades as the size of the problem (i.e., number of actors, actions available, complexity of the goal) increases. As resilience requires a quick reaction to changes, the time required to compute a new plan or policy for the supply chain is of utmost importance. As, to the best of our knowledge, no standard dataset exists in this area, we will perform experiments by adjusting the complexity of the use case presented in Section IV.

This section provides experimental results on the case study described in Section IV. Moreover, we provide further experiments on other two different case studies in the provided repositories[6].

A. Experimental setup

We conducted experiments to compare the performance and demonstrate the usage of *(i)* the instance planing-based approach based on classical planning (cf. Section V), *(ii)* the policy-based approach based on MDPs with fully specified target (cf. Section VI-A), and *(iii)* the policy-based approach based on MDPs with loosely specified target (cf. Section VI-B). The classical planning approach is implemented by leveraging the *Fast Downward* planner [60], which supports PDDL 2.2 with cost actions [61]. The MDP-based approaches are implemented using the `MDP-DP-RL` library[5]. These methods were tested using the *ChipChain* case study. The experiments were run on a computer with an Intel Core Ultra 7-155U (1.70GHz) processor with 32 GB RAM. Execution time (in seconds) and memory usage (in MiB) were measured over four scenarios differing in the number of available actors (services) capable of performing a particular action. The scenarios include: *xsmall case* – the simplest and basic supply chain case, containing exactly one service for each action, resulting in a total of 14 services; *small*, *medium*, and *large cases* – these cases include additional copies for the services, resulting in a total of 21, 28, and 35 services, respectively. These cases more effectively represent supply chains where multiple copies of actors (likely with different quality and status properties) cooperate toward a common goal. The presence of extra copies results in more complex scenarios where the controller needs to consider characteristics such as costs, rewards, and probabilities of the available actors. The experiment source codes are available at the provided repositories[6].

B. Discussing the experimental results

Figure 7 presents the results of the experiments. It is evident that depending on the approach used, performance values vary greatly.

The planning approach exhibits time and memory consumption that remain relatively stable and effective as the number of services increases. The planning solver applies well-known heuristics to derive a solution efficiently. However, this approach does not account for the stochasticity of the manufacturing environment, such as the probability of a machine encountering a failure situation. Indeed, this is the main motivation driving the other approaches addressed in the paper, which sacrifices performance - even if still realistic as demonstrated in this section for greater representativeness and realism of the modeling.

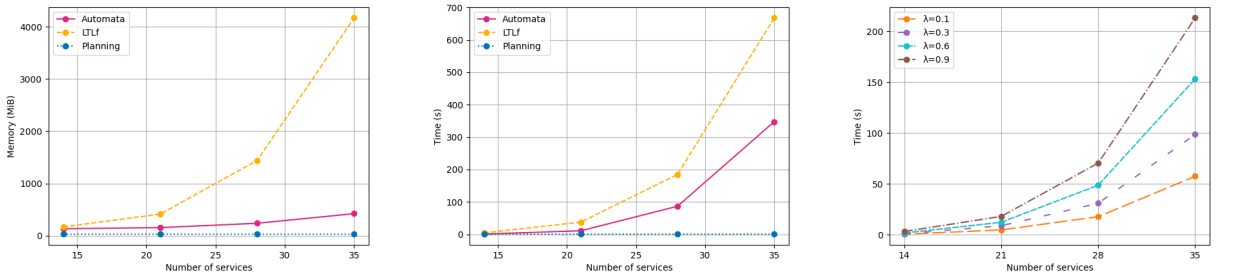
The resources needed by stochastic policy approaches show an exponential increase with the growing number of services. This escalation can be attributed to the composition MDP's formulation, which necessitates a cartesian product operation involving both the target and system service sets. While the target remains static and clearly defined, the system service's states may increase with the inclusion of additional services, consequently elevating memory usage and execution duration for both the composition MDP and policy computation.

Moreover, the stochastic constraint-based policy approach proves to be more resource-intensive in terms of time and memory compared to the stochastic policy approach. This substantial variance stems from the nature of the target service, particularly when expressed in LTL_f, where it comprises a series of logical constraints among actions. Converting such constraints into a DFA can lead to an explosion of states. Additionally, the presence of *auxiliary actions* — actions

[5]https://github.com/coverdrive/MDP-DP-RL

[6]The instance planning approach source code is available at https://github.com/iaiamomo/TSC-planning-experiments, the stochastic policy approaches source code is available at https://github.com/iaiamomo/TSC-mdp-experiments. The same links also report the results of the experiments described in the main text, to demonstrate the full repeatability of the approaches, and the results of the experiments run on two other case studies.

This article has been accepted for publication in IEEE Transactions on Services Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSC.2024.3495521



(a) Memory usage (b) Execution time (c) Policy execution time (LTL_f target)

Fig. 7: Experimental results

indirectly linked to the process but necessary for the services to execute specific actions — exerts influence on the transition function and the overall state count.

An important observation concerns the time and memory consumption of the stochastic constraint-based policy. It was found that policy computation contributes significantly to the increase in execution time. While one might expect that once the composition MDP is available, the policy can be directly derived, these results vary greatly depending on the lambda λ value. Figure 7b displays the overall results (composition MDP and policy calculation) with $\lambda = 0.9$; this value is experimentally adopted as the one allowing the solver to calculate the complete solution. A high value of lambda is commonly used to generate more strategic and optimal policies. Additional experiments were performed varying lambda over the four cases. Results of such experiments are depicted in Figure 7c. The effects of decreasing λ are observed in a reduction of time, suggesting that having a low lambda prioritizes short-term results.

The experimental results illustrate the practical applicability of automated synthesis techniques, not with the aim of proving the superiority of any one approach over the others, but rather to demonstrate that all approaches can indeed be employed in real manufacturing scenarios.

The comparison, in terms of performance, between planning and MDPs may seem clearly in favor of the former. Nonetheless, the employment of MDP-based approaches (whether with fully or loosely specified targets) and the resulting additional computational cost are sometimes unavoidable because of the specific requirements of the application scenario. The selected tool must indeed enable the designer to naturally represent the assets' behaviors and the target process. In order to enforce this concept, let us discuss the possible characteristics of assets (services) and how they fit with the presented approaches:

- **Unpredictability.** This is common in certain types of industries, particularly those dependent on semiconductors and, consequently, affected by geopolitical tensions. In such cases, the additional computational cost is unavoidable as the number of nodes and especially connections in MDPs increases. Conversely, if MDPs are used to model completely predictable services, the size of the diagrams is reduced, and the performance becomes comparable to that of planning-based approaches, not significantly influencing the performance. Noticeably, in this paper, we do not address cases where probabilities exist but cannot be estimated (e.g., the likelihood of lightning striking a working machine).
- **Modeling costs.** While the cost of many actions can be considered unitary (i.e., the execution of any action incurs a similar cost), some actions can be significantly more expensive (e.g., repairing a device). The cost (or reward) of an action is naturally modeled by MDPs, but it can also be incorporated into classical planning. However, when costs are factored into planning, the computational cost of finding a solution increases exponentially.
- **Sequential dependencies between actions.** In many cases, assets naturally undergo sequential working phases. While these phases can be modeled in classical planning by introducing appropriate predicates, representing them as state machines is much more intuitive for the user. Once again, in MDPs, the computational challenge does not arise from the number of states, but from the number of connections between states, which increases due to the presence of probabilities.
- **Dependency of decisions on data.** In this case, using MDPs would require an increase in the number of states (exponential in the number of considered variables), making it more challenging for the designer to model the asset. In this case, using planning (specifically planning languages like PDDL) simplifies the designer's work more than it impacts performance.
- **Structure of the manufacturing goal.** If the goal of manufacturing is expressed in terms of data conditions, modeling the problem is much simpler using planning or when the process is modeled declaratively in the case of MDPs. However, planning is less suited for scenarios where temporal constraints between actions are imposed, as modeling these constraints is not intuitive, requiring the introduction of auxiliary predicates. In such cases, it is easier to use MDPs, whether with fully or loosely specified targets. Here again, the crucial factor to be considered is the effort required by the designer, as the computation cost of planning increases exponentially anyway with the number of predicates and objects.

From the previous considerations, it is clear that the comparison proposed in the paper is aimed more at demonstrating

how both approaches scale computationally as the problem size increases. However, the modeling features utilized in each case, which impact the computational cost, are quite different. If both approaches were evaluated under similar settings, their computational costs would be comparable. Notably, the additional features available in MDPs are also supported by planners that support Probabilistic PDDL (PPDDL) [62]. It has been demonstrated that PPDDL problems can be represented and solved using MDPs, leading to comparable performance among the three evaluated approaches. However, if MDPs are used while only exploiting some of their features (e.g., making all actions deterministic), their computational cost approaches that of planning. As discussed then, the key consideration is not the computational cost itself, which is unavoidable in certain cases, but rather the modeling effort and cognitive distance required by designers when modeling the scenarios.

To analyze the trade-off between the expressive power of the modeling approaches and the effort required for practitioners to apply them, we conducted a user study with eight practitioners. We provided a questionnaire containing several scenarios and asked them to model these scenarios. We measured the time each practitioner spent on the modeling exercises and then conducted interviews with them.

The results indicated that MDP-based modeling was more expressive, enabling a detailed representation of manufacturing assets when needed. However, it was also more time-consuming and required more effort from practitioners. Additionally, some experts struggled to produce the correct version of the model, highlighting its higher complexity.

In contrast, planning-based modeling was simpler and faster for all practitioners but could only be adopted in certain scenarios. Every practitioner was able to model the services correctly within a shorter time frame, suggesting that this method, while less expressive, is more accessible. However, the limitation is that not all scenarios can be modeled using this approach.

These findings illustrate that while MDPs offer more powerful representations, they require greater effort to implement. In contrast, planning-based modeling should be utilized when control flow is not a critical factor. For the sake of brevity, the details and results of the user study are included in the repository link.[7]

C. The AIDA tool

All the proposed approaches are also available as part of the AIDA tool[8] (Adaptive InDustrial APIs) [63]. The tool implements a platform to manage the digital services of the manufacturing actors, called Industrial API platform. The Industrial API relates to the concept of Asset Administration Shell (AAS), which facilitates the digitization of physical things (assets) for virtual representation, turning an object into an Industry 4.0 component [64]–[66]. The digital services of each Industrial API are implemented as REST services, however, any other synchronous/blocking-invocations-based standard for Web services can be used. The REST services can wrap already available endpoints of actors or realize new endpoints for new actors. Information through the various actors in the factories is made available by available technologies including OPC-UA, EtherCAT or CANopen [67].

AIDA includes a *design GUI* for the final user, e.g., a production manager, allowing to specify both the manufacturing goal and actors. The former is described using the *target description language (tdl)* format, while the latter employs the *service description language (sdl)* format. Essentially, both formats are realized as JSON files, offering a standardized language for defining the problem specification. The controller (described below) imports the resulting `.sdl` files, which are used in the Industrial API platform for deploying the services, and the `.tdl` file, which is used to synthesize the plan.

VIII. Concluding remarks

In this paper, we proposed a service-oriented approach to model the manufacturing actors involved in manufacturing processes, identifying each actor through the set of digital services exposed via the Industrial API. This enables the retrieval of important information such as operating status and available actions. We leveraged this modeling approach to explore and showcase the utilization of automated reasoning techniques in smart manufacturing, with the objective of enhancing adaptivity and resilience within supply chains. Through a use case centered around chip manufacturing, we applied various techniques and thoroughly assessed the experimental outcomes. The techniques selected were representative of the categories defined in Section III.

In certain cases, it may be necessary to model constraints between actors. For example, we may need to specify that if actor A is used, then actor B cannot be used due to commercial or political issues. At the current stage, we do not directly support this kind of rules. However, they could be modeled using PDDL conditional effects in the approach based on automated planning, although this may impact computation costs. In the case of the MDP-based approach, these rules are more challenging to model as the representation is hierarchical and the target process is not aware of the specific actors employed. One option in this case would be to include constraints when computing the composition MDP.

Additionally, the presented case study and conducted experiments assume that the state space is discrete. While continuous variables can be discretized, this often results in an explosion of the number of states, making the solution harder to compute. In the case of automated planners, while continuous variables are supported by the latest versions of PDDL, few planners support them. In the case of MDPs, discretization must be done manually.

Regarding the employed approaches, we mainly focused on AI synthesis. Classical numerical optimization techniques can also be applied; however, these techniques, even if potentially fast, are challenging in terms of modeling. Usually, a set of equations forms an optimization problem, which is more complex to edit and validate than the formalisms used here.

Furthermore, in terms of AI, in this paper, we focused on symbolic AI instead of machine or deep learning. In

[7] https://github.com/iaiamomo/TSC-planning-experiments.

[8] Source code of AIDA can be found at https://github.com/DIAG-Sapienza-BPM-Smart-Spaces/AIDA

particular, a policy-based approach could be implemented via Reinforcement Learning (RL) that, interestingly, can be mathematically modeled through MDPs. With respect to synthesis, learning requires a minor modeling effort relying instead on the availability of a huge amount of training data, which can be a challenging requirement to fulfill in manufacturing environments, especially concerning negative samples. While techniques exist to make RL more sample efficient, they usually result on a major modeling effort [68]. The problem of data availability can be solved through simulation techniques, that could in turn be addressed through synthesis techniques. Finally, provided a standard way to model actors in the supply chain, *transfer learning* can be used to alleviate the problem of data availability even more.

REFERENCES

[1] A. Kusiak, "Smart manufacturing," in *Springer Handbook of Automation*. Springer, 2023, pp. 973–985.

[2] M. Pellicciari, A. O. Andrisano, F. Leali, and A. Vergnano, "Engineering method for adaptive manufacturing systems design," *IJIDeM*, 2009.

[3] Y. Cohen, H. Naseraldin, A. Chaudhuri, and F. Pilati, "Assembly systems in Industry 4.0 era: a road map to understand Assembly 4.0," *The International Journal of Advanced Manufacturing Technology*, 2019.

[4] M. Onori, P. Neves, H. Akillioglu, A. Maffei, A. Hofmann, and N. Siltala, "Dealing with the unpredictable: An evolvable robotic assembly cell," in *Proceedings of the 4th International Conference on Changeable, Agile, Reconfigurable and Virtual production*. Springer, 2012.

[5] R. Schmitt, E. Permin, J. Kerkhoff, M. Plutz, and M. G. Böckmann, "Enhancing resiliency in production facilities through cyber physical systems," *Industrial Internet of Things: Cybermanufacturing Systems*, pp. 287–313, 2017.

[6] O. Alharbi, "Industry 4.0 operators: core knowledge and skills," *Advances in Science, Technology and Engineering Systems Journal*, 2020.

[7] G. F. Frederico, "Towards a supply chain 4.0 on the post-COVID-19 pandemic: a conceptual and strategic discussion for more resilient supply chains," *Rajagiri Management Journal*, vol. 15, no. 2, pp. 94–104, 2021.

[8] A. Villar, S. Paladini, and O. Buckley, "Towards supply chain 5.0: redesigning supply chains as resilient, sustainable, and human-centric systems in a post-pandemic world," in *Operations Research Forum*, vol. 4, no. 3. Springer, 2023, p. 60.

[9] M. Christopher, *Logistics and supply chain management*. Pearson Uk, 2022.

[10] B. Yang, S. Wang, S. Li, and F. Bi, "Digital Thread-Driven Proactive and Reactive Service Composition for Cloud Manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 3, pp. 2952–2962, 2023.

[11] H. L. Lee *et al.*, "The triple-A supply chain," *Harvard business review*, vol. 82, no. 10, pp. 102–113, 2004.

[12] D. S. Fowler, G. Epiphaniou, M. D. Higgins, and C. Maple, "Aspects of resilience for smart manufacturing systems," *Strategic Change*, 2023.

[13] N. Bicocchi, G. Cabri, F. Mandreoli, and M. Mecella, "Dynamic digital factories for agile supply chains: An architectural approach," *Journal of Industrial Information Integration*, vol. 15, pp. 111–121, 2019.

[14] H. Stadtler, "Supply chain management: An overview," *Supply chain management and advanced planning: Concepts, models, software, and case studies*, pp. 3–28, 2015.

[15] E. Sawyerr and C. Harrison, "Developing resilient supply chains: lessons from high-reliability organisations," *Supply Chain Management: An International Journal*, vol. 25, no. 1, pp. 77–100, 2020.

[16] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella, and F. Sapio, "A conceptual architecture and model for smart manufacturing relying on service-based digital twins," in *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 2019, pp. 229–236.

[17] Z. Liu, P. Sampaio, G. Pishchulov, N. Mehandjiev, S. Cisneros-Cabrera, A. Schirrmann, F. Jiru, and N. Bnouhanna, "The architectural design and implementation of a digital platform for Industry 4.0 SME collaboration," *Computers in Industry*, vol. 138, p. 103623, 2022.

[18] H. Han and S. Trimi, "Towards a data science platform for improving SME collaboration through Industry 4.0 technologies," *Technological Forecasting and Social Change*, vol. 174, p. 121242, 2022.

[19] S. Mokaram, J. M. Aitken, U. Martinez-Hernandez, I. Eimontaite, D. Cameron, J. Rolph, I. Gwilt, O. McAree, and J. Law, "A ROS-integrated API for the KUKA LBR iiwa collaborative robot," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 859–15 864, 2017.

[20] N. Mitro, K. Argyri, L. Pavlopoulos, D. Kosyvas, L. Karagiannidis, M. Kostovasili, F. Misichroni, E. Ouzounoglou, and A. Amditis, "AI-Enabled Smart Wristband Providing Real-Time Vital Signs and Stress Monitoring," *Sensors*, 2023.

[21] M. Achouch, M. Dimitrova, K. Ziane, S. Sattarpanah Karganroudi, R. Dhouib, H. Ibrahim, and M. Adda, "On predictive maintenance in industry 4.0: Overview, models, and challenges," *Applied Sciences*, vol. 12, no. 16, p. 8081, 2022.

[22] M. Grieves, "Digital twin: manufacturing excellence through virtual factory replication," *White paper*, vol. 1, pp. 1–7, 2014.

[23] E. Alnazer and I. Georgievski, "Understanding Real-World AI Planning Domains: A Conceptual Framework," *arXiv preprint arXiv:2307.04701*, 2023.

[24] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi, "Automatic Service Composition and Synthesis: the Roman Model." *IEEE Data Eng. Bull.*, vol. 31, no. 3, pp. 18–22, 2008.

[25] A. L. Lemos, F. Daniel, and B. Benatallah, "Web service composition: a survey of techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–41, 2015.

[26] M. Reichert and B. Weber, *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer, 2012.

[27] F. Monti, J. G. Mathew, F. Leotta, A. Koschmider, and M. Mecella, "On the Application of Process Management and Process Mining to Industry 4.0," *Softw. Syst. Model.*, 2024.

[28] M. Dumas, F. Fournier, L. Limonad, A. Marrella, M. Montali, J.-R. Rehse, R. Accorsi, D. Calvanese, G. De Giacomo, D. Fahland *et al.*, "AI-augmented business process management systems: a research manifesto," *ACM Transactions on Management Information Systems*, vol. 14, no. 1, pp. 1–19, 2023.

[29] F. Monti, L. Silo, F. Leotta, and M. Mecella, "Services in Smart Manufacturing: Comparing Automated Reasoning Techniques for Composition and Orchestration," in *Symposium and Summer School on Service-Oriented Computing*. Springer, 2023, pp. 69–83.

[30] ——, "On the Suitability of AI for Service-based Adaptive Supply Chains in Smart Manufacturing," in *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, 2023, pp. 704–706.

[31] A. Marrella, "Automated planning for business process management," *Journal on data semantics*, vol. 8, no. 2, pp. 79–98, 2019.

[32] L. Braccesi, M. Monsignori, and P. Nesi, "Monitoring and optimizing industrial production processes," in *Proceedings. Ninth IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 2004.

[33] J. Blythe, "Decision-theoretic planning," *AI magazine*, 1999.

[34] A. Marrella, M. Mecella, and S. Sardina, "Intelligent process adaptation in the SmartPM system," *ACM Trans. on Intell. Systems and Technology*, vol. 8, no. 2, pp. 1–43, 2016.

[35] L. Malburg, M. Hoffmann, and R. Bergmann, "Applying MAPE-K control loops for adaptive workflow management in smart factories," *Journal of Intelligent Information Systems*, pp. 1–29, 2023.

[36] D. Ciolek, N. D'Ippolito, A. Pozanco, and S. Sardiña, "Multi-tier automated planning for adaptive behavior," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 66–74.

[37] B. Wally, J. Vyskočil, P. Novák, C. Huemer, R. Šindelář, P. Kadera, A. Mazak-Huemer, and M. Wimmer, "Leveraging iterative plan refinement for reactive smart manufacturing systems," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 230–243, 2020.

[38] S.-O. Bezrucav, N. Mandischer, and B. Corves, "Artificial Intelligence Task Planning of Cooperating Low-Cost Mobile Manipulators: A Case Study on a Fully Autonomous Manufacturing Application," *Procedia Computer Science*, vol. 217, pp. 306–315, 2023.

[39] H. Hu, X. Jia, K. Liu, and B. Sun, "Self-adaptive traffic control model with Behavior Trees and Reinforcement Learning for AGV in Industry 4.0," *IEEE Transactions on Industrial Informatics*, 2021.

[40] J. Tu and L. Zhang, "An MDP-based Method for Dynamic Workforce Allocation in Bernoulli Serial Production Lines," in *2023 IEEE 19th International Conference on Automation Science and Engineering*, 2023.

[41] S. V. Amari, L. McLaughlin, and H. Pham, "Cost-effective condition-based maintenance using Markov decision processes," in *RAMS'06. Annual Reliability and Maintainability Symposium, 2006.* IEEE, 2006.

[42] B. Y. Choo, S. C. Adams, B. A. Weiss, J. A. Marvel, and P. A. Beling, "Adaptive multi-scale prognostics and health management for smart manufacturing systems," *International journal of prognostics and health management*, vol. 7, 2016.

[43] A. Prashar, G. L. Tortorella, and F. S. Fogliatto, "Production scheduling in Industry 4.0: Morphological analysis of the literature and future research agenda," *Journal of Manufacturing Systems*, 2022.

[44] K. Bakon, T. Holczinger, Z. Süle, S. Jaskó, and J. Abonyi, "Scheduling under uncertainty for Industry 4.0 and 5.0," *IEEE Access*, 2022.

[45] R. Lo Bianco, R. Dijkman, W. Nuijten, and W. van Jaarsveld, "Action-Evolution Petri Nets: A Framework for Modeling and Solving Dynamic Task Assignment Problems," in *International Conference on Business Process Management*. Springer, 2023, pp. 216–231.

[46] H. Geffner, "Computational models of planning," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 4, no. 4, pp. 341–356, 2013.

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

[48] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson *et al.*, "Pddl— the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[49] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, 1971.

[50] M. L. Puterman, *Markov Decision Processes*, 1994.

[51] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2018.

[52] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, and L. Silo, "Digital twins composition in smart manufacturing via markov decision processes," *Computers in Industry*, vol. 149, p. 103916, 2023.

[53] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2013.

[54] R. I. Brafman, G. De Giacomo, M. Mecella, and S. Sardina, "Service Composition in Stochastic Settings," in *Conference of the Italian Association for Artificial Intelligence*. Springer, 2017, pp. 159–171.

[55] W. M. van Der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science-Research and Development*, vol. 23, pp. 99–113, 2009.

[56] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 2013.

[57] M. Pesic, H. Schonenberg, and W. M. Van der Aalst, "Declare: Full support for loosely-structured processes," in *11th IEEE international enterprise distributed object computing conference (EDOC 2007)*.

[58] G. De Giacomo, M. Favorito, and L. Silo, "Composition of Stochastic Services for LTL Goal Specifications," in *International Symposium on Foundations of Information and Knowledge Systems*. Springer, 2024.

[59] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM journal of research and development*, 1959.

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

[61] M. Fox and D. Long, "PDDL2. 1: An extension to PDDL for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[62] H. L. Younes and M. L. Littman, "Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects," *Techn. Rep. CMU-CS-04-162*, vol. 2, p. 99, 2004.

[63] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, F. Monti, and L. Silo, "AIDA: A Tool for Resiliency in Smart Manufacturing," in *International Conference on Advanced Information Systems Engineering*. Springer, 2023, pp. 112–120.

[64] Platform Industrie 4.0, "Details of the Asset Administration Shell - Part 1," https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html, online; accessed July 2024.

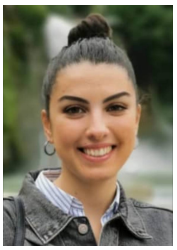
[65] T. Bangemann, C. Bauer, H. Bedenbender, M. Diesner, U. Epple, F. Elmas, J. Friedrich, T. Goldschmidt, F. Göbe, S. Grüner *et al.*, "Industrie 4.0-Technical Assets: Basic terminology concepts life cycles and administration models," *VDI/VDE and ZVEI*, 2016.

[66] W. Ochoa, F. Larrinaga, and A. Pérez, "Architecture for managing AAS-based business processes," *Procedia Computer Science*, vol. 217, pp. 217–226, 2023.

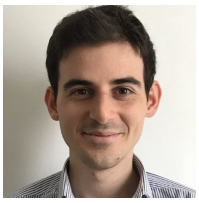
[67] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[68] Y. Yu, "Towards sample efficient reinforcement learning." in *IJCAI*, 2018, pp. 5739–5743.

Flavia Monti is a PhD student in Engineering in Computer Science at the Dipartimento di Ingegneria informatica, automatica e gestionale of Sapienza Università di Roma, where she got the MSc in Engineering in Computer Science in 2021. Her research interests focus on Industry 4.0 and smart manufacturing, particularly the integration of computer vision, machine learning, and artificial intelligence to improve production quality, reduce costs, increase machinery uptime, and achieve zero-defect manufacturing.



Luciana Silo is a PhD student in Artificial Intelligence (National Program) at Sapienza Università di Roma, where she got the MSc in Engineering in Computer Science in 2021. Currently, she is also working as a software engineer at Camera dei Deputati in Rome. Her research activity focuses on the study of Artificial Intelligence and Service Composition techniques applied to industrial processes, in the realm of smart manufacturing and Industry 4.0.



Marco Favorito got a MSc in Engineering in Computer Science at Sapienza in 2018. He then pursued a PhD under the supervision of Giuseppe De Giacomo, working on automata-theoretic techniques for reasoning and learning temporal logic in the context of artificial intelligence. Since 2021, he has worked as a researcher in the Applied Research Team at the Bank of Italy.

Giuseppe De Giacomo is a Governing Body Fellow, Green Templeton College and a Professor of Computer Science, Department of Computer Science, University of Oxford. He was previously a professor at Sapienza University of Rome. His research activity has concerned theoretical, methodological, and practical aspects in different areas of artificial intelligence. He is a AAAI Fellow, ACM Fellow, and EurAI Fellow. He has received a European Research Council (ERC) Advanced Grant for the project WhiteMech (2019-2024).

Francesco Leotta got his PhD in Engineering in Computer Science at Sapienza in 2014, where he currently covers the position of associate professor. Since the beginning of his research activity, he addressed several challenges related to how users interact with a smart space and how the environment senses the users and reactively performs actions to meet user requirements. Besides this main research activity, his research interests cover advanced user interfaces, service-oriented architectures (SOA), and e-Government.

Massimo Mecella , PhD in Engineering in Computer Science, is a full professor at Sapienza, where he is conducting research in the fields of information systems engineering, software architectures, distributed middleware and service-oriented computing, mobile and pervasive computing, process management, data and process mining, big data analytics, advanced interfaces and human-computer interaction, focusing on smart applications, environments and communities. He was the General Chair of CAiSE 2019, BPM 2021, and ICSOC 2023 in Rome. He currently sits on the Steering Committees of the conference series CAiSE, ICSOC, SummerSOC, and previously in Intelligent Environments (IE) and Advanced Visual Interfaces (AVI).