Noname manuscript No.
(will be inserted by the editor)

Automated Planning for Business Process Management

Andrea Marrella

Received: date / Accepted: date

Abstract Business Process Management (BPM) is a central element of today's organizations. Over the years its main focus has been the support of business processes (BPs) in highly controlled domains. However - in the current era of Big Data and Internet-of-Things - several real-world domains are becoming cyber-physical (e.g., consider the shift from traditional manufacturing to Industry 4.0), characterized by ever-changing requirements, unpredictable environments and increasing amounts of data and events that influence the enactment of BPs. In such unconstrained settings, BPM professionals lack the needed knowledge to model all possible BP variants/contingencies at the outset. Consequently, BPM systems must increase their level of automation to provide the reactivity and flexibility necessary for process management.

On the other hand, the Artificial Intelligence (AI) community has concentrated its efforts on investigating dynamic domains that involve active control of computational entities and physical devices (e.g., robots, software agents, etc.). In this context, Automated Planning, which is one of the oldest areas in AI, is conceived as a model-based approach to synthesize autonomous behaviours in automated way from a model.

In this paper, we discuss how automated planning techniques can be leveraged to enable new levels of automation and support for solving concrete problems in the BPM field that were previously tackled with hard-coded solutions. To this aim, we first propose a methodology that shows how a researcher/practitioner should approach the task of encoding a concrete problem as an appropriate planning problem. Then, we discuss the required steps to integrate the planning technology in BPM environments. Finally, we show some concrete examples of the successful application of planning techniques to the different stages of the BPM life cycle.

Keywords Business Process Management · Automated Planning in AI · Process Management Systems · Process Adaptation · Process Mining

A. Marrella
Sapienza Università di Roma
E-mail: marrella@diag.uniroma1.it

1 Introduction

Business Process Management (BPM) is a central element of today's organizations due to its potential for increased productivity and saving costs. For this reason, BPM research has focused on overseeing how work is performed in an organization by managing and optimising its business processes (BPs) [107,3]. In order to support the design, automation, execution and monitoring of BPs, a dedicated generation of information systems, called Process Management Systems (PMSs), has become increasingly popular over the last decade [26,27].

From its origins, the BPM philosophy has been strongly influenced by the *principles of scientific management* by Frederick Taylor [32], and is built on the idea that there always exists an underlying fixed BP that can be used to automate the work and *executed like a program* from a PMS [26,107]. The required steps that have made this idea a reality consist of: *(i)* performing an up-front effort to identify a BP (i.e., a structured abstraction of a real workflow) that can be executed many times; *(ii)* formalizing it in a *process model* that captures the ways in which tasks are carried out to accomplish a business objective, often with the help of an explicit control flow expressed through a suitable graph-

ical notation, such as the standard BPMN[1]; and *(iii) automating the BP routing* with a PMS that properly assigns tasks to process participants (e.g., softwares or humans), so that several *instances* of the same BP can be run many times repeatedly. Since there may be some variation in the handling of individual instances, one can customize *in advance* a BP model by implementing specific *variants* of the BP itself, e.g., by adding or deleting process branches according to some customization options that determine how to handle all possible *predictable situations* that may happen at run-time [7, 53]. For routine BPs enacted in highly controlled business domains (e.g., financial and accounting domains), this approach works perfectly [107,27].

In the current era of Big Data and Internet-of-Things (IoT), we are witnessing the transformation of traditional working domains (e.g., manufacturing, healthcare, emergency management, etc.) into new challenging cyber-physical domains (e.g., Industry 4.0 [55,57], Health 2.0 [106,16], smart emergency management [49,61,71], etc.). Such domains are characterized by the presence of heterogeneous Information Technology (IT) components (e.g., robots, machines, sensors, etc.) that, on the one hand, perform complex tasks in the "physical" real world to achieve a common goal, and on the other hand monitor in detail the evolution of the concrete BP instances being executed, by producing huge amount of data and events that may dramatically influence the proper enactment of BPs [82, 100].

Under such dynamic conditions, the requirements imposed by the BPM philosophy, i.e., of having BPs fully *specified a priori* and that are executed as *rigid plans of actions* by a PMS, are too restrictive [94]:

- in domains that are to a large extent unclear and/or unpredictable, accurate BP modeling can not always be completed prior to execution. This is mainly due to lack of domain knowledge and complexity of tasks combination and branch conditions, which would require the use of a large and complex set of rules to determine in advance how to handle all possible situations that may happen at run-time [28];
- even for well-defined BPs, the combination and sequence of tasks may vary from instance to instance, due to changes in the execution context (such as user preferences and business rules), or changes in the environment (such as unplanned exceptions or exogenous events). In such cases, BP instances should be adapted case by case (e.g., by adding, removing or generating an alternative sequence of tasks) by exploiting information gathered at run-time. However, it is considered impossible to determine at design-time all potential adaptations that may be needed at run-time [94].

To tackle the above issues, BPM *is in need of techniques that go beyond hard-coded solutions* that put all the burden on IT professionals, which often lack the needed knowledge to model all possible contingencies at the outset, or this knowledge can become obsolete as BP instances are executed and evolve, by making useless their initial effort. Therefore, there are compelling reasons to introduce *intelligent techniques* that act *autonomously* to provide the reactivity and flexibility necessary for process management [22]. This matter is very actual, and was recently discussed in the keynote talks given by Rick Hull (BPM 2016 [48]) and Miguel Valdes (BPM 2017 [105]) at the major scientific international conference of Business Process Management.

On the other side, the challenge of building computational entities and physical devices (e.g., robots, software agents, etc.) capable of *autonomous behaviour* under dynamic conditions is at the center of the Artificial Intelligence (AI) research from its origins. At the the core of this challenge lies the *action selection problem*, often referred as the *problem of selecting the action to do next*. Traditional hard-coded solutions require to consider every option available at every instant in time based on the current context and pre-scripted plans to compute just one next action. Consequently, they are usually biased and tend to constrain their search in some way. For AI researchers, the question of action selection is: *what is the best way to constrain this search?* To answer this question, the AI community has tackled the action selection problem through two different approaches [36], one based on *learning* and the other based on *modeling*.

In the *learning-based* approach, the controller that prescribes the action to do next is *learned from the experience*. Learning methods, if properly trained on representative datasets, have the greatest promise and potential, as they are able to discover and eventually interpret meaningful patterns for a given task in order to help make more efficient decisions. For example, learning techniques were recently applied in BPM (see [64]) for predicting future states or properties of ongoing executions of a BP. However, a learned solution is usually a "black box", i.e., there is not a clear understanding of how and why it has been chosen. Consequently, the ability to explain why a learned solution has failed and fix a reported quality bug may become a complex task.

Conversely, in the *model-based* approach the controller in charge of action selection is *derived automatically* from a model that expresses the dynamics of the domain of interest, the actions and goal conditions. The

[1] The Business Process Modeling Notation (BPMN) is the ISO standard (ISO/IEC 19510:2013) for modeling BPs.

key point is that all *models are conceived to be general*, i.e., they are not bound to specific domains or problems. The price for generality is *computational*: The problem of solving a model is computationally intractable in the worst case, even for the simplest models [36].

While we acknowledge that both the *learning* and *model-based* approaches to action selection exhibit different merits and limitations, in this paper we focus on a specific model-based approach called *Automated Planning*. Automated planning is the branch of AI that concerns the automated synthesis of autonomous behaviours (in the form of strategies or action sequences) for specific classes of mathematical models represented in compact form. In recent years, the automated planning community has developed a plethora of *planning systems* (also known as *planners*) that embed very effective (i.e., scale up to large problems) domain-independent heuristics, which have been employed to solve collections of challenging problems from several Computer Science domains.

In this paper, which extends our previous work [66] in several directions[2], we discuss how automated planning techniques can be leveraged for solving real-world problems in BPM that were previously tackled with hard-coded solutions by enabling new levels of automation and support for BPs. To this aim, we first investigate under which conditions planning is feasible for tackling concrete problems in the BPM field, and how people familiar with BPM can encode their process knowledge in a planning problem. Then, we discuss the required steps to integrate and exploit the planning technology in PMSs. Finally, we show some concrete examples of the successful application of planning techniques to the different stages of the BPM life cycle.

Overall, the major purpose of this contribution is to show how the synergy between automated planning techniques and BPM can allow the realization of intelligent solutions that effectively tackle relevant challenges from the BPM domains.

To be more specific, while in Section 2 we introduce some preliminary notions on automated planning necessary to understand the rest of the paper, in Section 3 we propose a methodology that shows how a researcher/practitioner should approach the task of encoding a concrete problem as an appropriate planning problem. Then, in Section 4, we discuss the required steps to concretely integrate the planning technology in PMSs, and in Section 5 we show how instances of some well-known problems from the BPM literature (such as process modeling, process adaptation of running BPs and conformance checking) can be represented as planning problems for which planners can find a correct solution in a finite amount of time. Finally, in Section 6 we discuss related work on the use of planning both in the BPM field and in other BPM-related Computer Science domains, and in Section 7 we conclude the paper by providing a critical discussion about the general applicability of planning techniques in BPM and tracing future work.

[2] For readability purposes, the details of the additional contributions with respect to our previous work [66] are explained in Section 7.

2 Automated Planning

Automated planning addresses the problem of generating autonomous behaviours (i.e., *plans*) from a *model* that describes how *actions* work in a *domain* of interest, what is the *initial state* of the world and the *goal state* to be achieved. To this aim, automated planning operates on *explicit representations* of states and actions that are expressed in compact form through dedicated *languages*. Such representations can be "navigated" exploiting a collection of different *search algorithms*, whose main purpose is to find a "path" consisting of actions that allow to achieve the goal state starting from the initial state.

In a nutshell, as shown in Fig. 1, automated planning is made up of three "ingredients": *models*, *languages* and *algorithms*. In the following sections, we introduce such ingredients, which are required to automatically "cook" a plan.

2.1 Planning Models and Algorithms

There exist several forms of planning models in the AI literature, which result from the application of some orthogonal dimensions [34]: uncertainty in the initial state (fully or partially known) and in the actions dynamics (deterministic or not), the type of feedback (full, partial or no state feedback) and whether uncertainty is represented by sets of states or probability distributions.

The simplest form of planning where actions are deterministic, the initial state is known and plans are action sequences computed in advance is called *Classical Planning*. According to [36], the formal model underlying classical planning can be described as: $\mathsf{M} = \langle S, s_0, S_G, A, f, c \rangle$, where:

- S is a finite and discrete set of states;
- $s_0 \in S$ is the known initial state;
- $S_G \subseteq S$ is the non-empty set of goal states;
- $A(s) \in A$ is the set of actions in A that are applicable in each state $s \in S$;

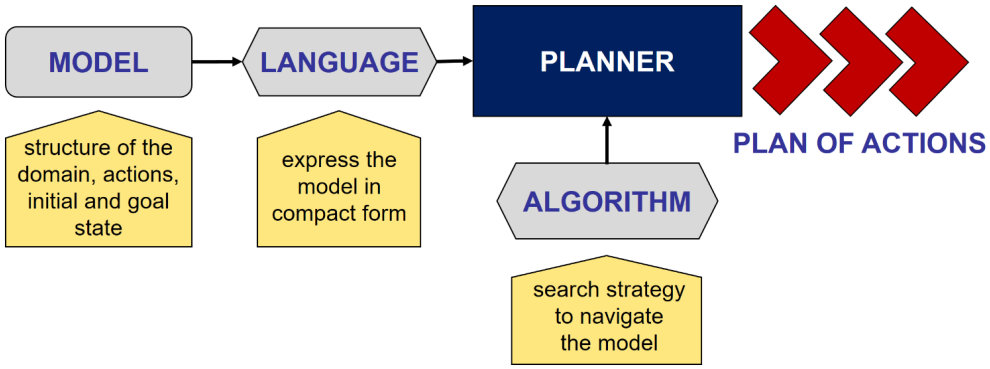

Fig. 1 A planner takes a compact representation of a planning problem over a certain class of models and automatically produce a plan.

- $f(a, s)$ is a deterministic transition function where $s' = f(a, s)$ is the state that follows s after doing action $a \in A(s)$;
- $c(a, s)$ is a positive action cost for doing action a in the state s.

For the classical planning model, a *plan* is a sequence of applicable actions $a_0, \ldots, a_n \in A$ that generates a state sequence $s_0, \ldots, s_{n+1} \in S$, where s_0 is the initial state and s_{n+1} is a goal state. The cost of the plan is the sum of the action costs $c(a_i, s_i)$, with $i = 0, \ldots, n$. A plan is *optimal* if it has minimum cost; conversely, a *satisfying* plan does not prove any guarantee other than the correctness of the solution.

For classical planning, according to [15], the general problem of coming up with a plan is PSPACE-complete. Many planning problems, however, involve more expressive features that are not part of this basic model, with a consequent increase of the complexity to find a plan. For example, if we consider also non-deterministic actions, when the features characterizing a state are fully observable, the problem to generate a plan is EXPTIME-complete (in the description of the state), or - even worse - 2EXPTIME-complete in case of partial observability, cf. [96].

In this paper, we do not aim at providing a complete description of all possible planning models/languages/algorithms available in the literature[3], but we aim at showing how a planning-based approach can be concretely applied to BPM. For this reason, we will focus on the *classical planning model*, which is sufficiently expressive to tackle several complex challenges of the BPM field (cf. Section 5). In addition, we notice that, even if the classical approach of solving planning problems can be too restrictive for environments in which information completeness can not be guaranteed, it is often possible to solve non-classical planning problems using classical planners by means of well-defined transformations [35].

Despite its complexity, the field of classical planning has experienced spectacular advances (in terms of scalability) in the last 20 years, leading to a variety of concrete *state-of-the-art planners* that are able to feasibly compute plans with thousands of actions over huge search spaces for real-world problems containing hundreds of propositions. Such progresses have been possible because state-of-the-art planners employ powerful *heuristic functions* that are automatically derived by the specific problem and allow to *intelligently* drive the search towards the goal. These efforts have led to development of a sort of "*science of search algorithms and heuristics*" for planning, which has allowed researchers to confine the inherent complexity of planning within a set of hard instances, while keeping most cases of practical interest efficiently solvable [63]. Among the most relevant state-of-the-art planners, we report the planners FF [45], Fast-Downward [42], LAMA [95], LPG-td [38] and Probe [62].

2.2 Planning domain Definition Language (PDDL)

The previous section suggests that the representation of a planning problem - actions, states and goals - should make it possible for planning algorithms to take advantage of the logical structure of the problem. The effort of the planning community was to find a language expressive enough to describe a wide variety of problems, but restrictive enough to allow efficient algorithms to operate over it and researchers to exchange benchmark problems and compare results [97]. In 1998, the various planning formalisms used in AI have been systematized within a standard syntax called Planning Domain Definition Language, or PDDL [79].

PDDL is a de-facto standard to formulate a compact representation of a *planning problem* $\mathcal{P}_{\mathcal{R}} = \langle I, G, \mathcal{P}_{\mathcal{D}} \rangle$,

[3] A tutorial introduction to planning models and algorithms can be found in [36].

where I is the description of the initial state of the world, G is the desired goal state, and $\mathcal{P}_\mathcal{D}$ is the planning domain.

A planning domain $\mathcal{P}_\mathcal{D}$ is built from a set of *propositions* describing the *state* of the world in which planning takes place (a state is characterized by the set of propositions that are true) and a set of *actions* A that can be executed in the domain. An *action schema* $a \in A$ is of the form $a = \langle Par_a, Pre_a, Eff_a, Cost_a \rangle$, where Par_a is the list of *input parameters* for a, Pre_a defines the *preconditions* under which a can be executed, Eff_a specifies the *effects* of a on the state of the world and $Cost_a$ is the *cost* of executing a.

Both preconditions and effects are stated in terms of the *propositions* in $\mathcal{P}_\mathcal{D}$, which can be represented through boolean *predicates*. PDDL allows for advanced propositional operator declarations including *negated preconditions* and *universal/existential quantification* of objects. The effects of actions can be also *conditional* (when-effects) and can include special *numeric fluents* to express the actions' cost. PDDL includes also the ability of defining *domain objects* and of *typing* the parameters that appear in actions, predicates and numeric fluents. In a state, only actions whose preconditions are fulfilled can be executed. The values of propositions in $\mathcal{P}_\mathcal{D}$ can change as result of the execution of actions, which, in turn, lead $\mathcal{P}_\mathcal{D}$ to a new state.

The above specification of a planning problem $\mathcal{P}_\mathcal{R}$ allows to fully accommodate the classical planning model. A *planner* that takes in input $\mathcal{P}_\mathcal{R}$ is said to be *domain-independent*, since it is able to automatically produce a *plan* P (i.e., a controller that specifies which actions are required to transform the initial state I into a state satisfying the goal G), without knowing what the actions and domain stand for.

It is worth to notice that, over the years, the planning community has developed several variants and extensions of PDDL to enable the representation of the different planning models available in the literature (e.g., to capture durative and nondeterministic actions, multi-valued variables, partial observability, etc.). Among them, in this paper we focus on the classical fragment of PDDL 2.1 [30], which is considered to be the most radical evolution compared to the original version of the language and is nowadays supported by the majority of state-of-the-art domain-independent planners [4].

Example 1 We show now how the *reachability analysis* of procedural BPs, which is a well-known problem in the BPM community [2], can be easily expressed in PDDL and solved using state-of-the-art planners. Despite many notations (yet presenting an ambiguous semantics) have been introduced to represent procedural BPs, such as BPMN, EPC, YAWL or UML Activity Diagrams [27], the reachability analysis of BPs is usually performed by converting BP models into Petri nets [25], which have a clear semantics and are proven to be sufficiently adequate to model crucial aspects of BPs [1, 24].

A Petri net $N = (P, T, F)$ is a directed graph with a set P of nodes called places and a set T of transitions [83]. The nodes are connected via directed arcs $F \subseteq (P \times T) \cup (T \times P)$. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles. Fig. 2(a) illustrates an example of Petri net. Given a transition $t \in T$, ${}^\bullet t$ is used to indicate the set of *input places* of t, which are the places p with a directed arc from p to t (i.e., such that $(p, t) \in F$). For example, ${}^\bullet a = \{start\}$. Similarly, $t^\bullet$ indicates the set of *output places*, namely the places p with a direct arc from t to p. For example, $a^\bullet = \{p1, p2\}$. At any time, a place can contain zero or more *tokens*, drawn as black dots. The state of a Petri net, a.k.a. *marking*, is determined by the number of tokens in places. Therefore, a marking m is a function $m : P \to \mathbb{N}$.

In any run of a Petri net, the number of tokens in places may change, i.e., the Petri net marking. A transition t is enabled at a marking m iff each input place contains at least one token, i.e., $\forall\ p \in {}^\bullet t$, $m(p) > 0$. A transition t can fire at a marking m iff it is enabled. As result of firing a transition t, one token is "consumed" from each input place and one is "produced" in each output place. Specifically, firing a transition t at marking m leads to a new marking m'. This is denoted as $m \xrightarrow{t} m'$.

When Petri nets are used to represent BPs, transitions are associated with BP activities, more specifically to activity labels, and markings indicate the process state [2]. Executions of BPs have a start and end. Therefore, Petri nets need to be associated with an initial m_0 and final marking m_n.

Based on the foregoing, the reachability problem of a Petri net can be defined as follow: Given a Petri net N with an initial marking m_0 and a target marking m_n, find a sequence of transition firing $\sigma = \langle t_1, \ldots, t_n \rangle \in T^*$ that leads from m_0 to m_n, such that $m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} m_n$. If such a firing sequence σ exists, m_n is said to be *reachable* from m_0 (this is denoted as $m_0 \xrightarrow{\sigma} m_n$).

In Fig. 2(a) it is shown an instance of this problem where, in the initial marking, *start* is the only place that contains a token. The goal is to find a sequence of transitions firing such that the final marking is achieved,

[4] cf. `http://icaps-conference.org/index.php/main/competitions`

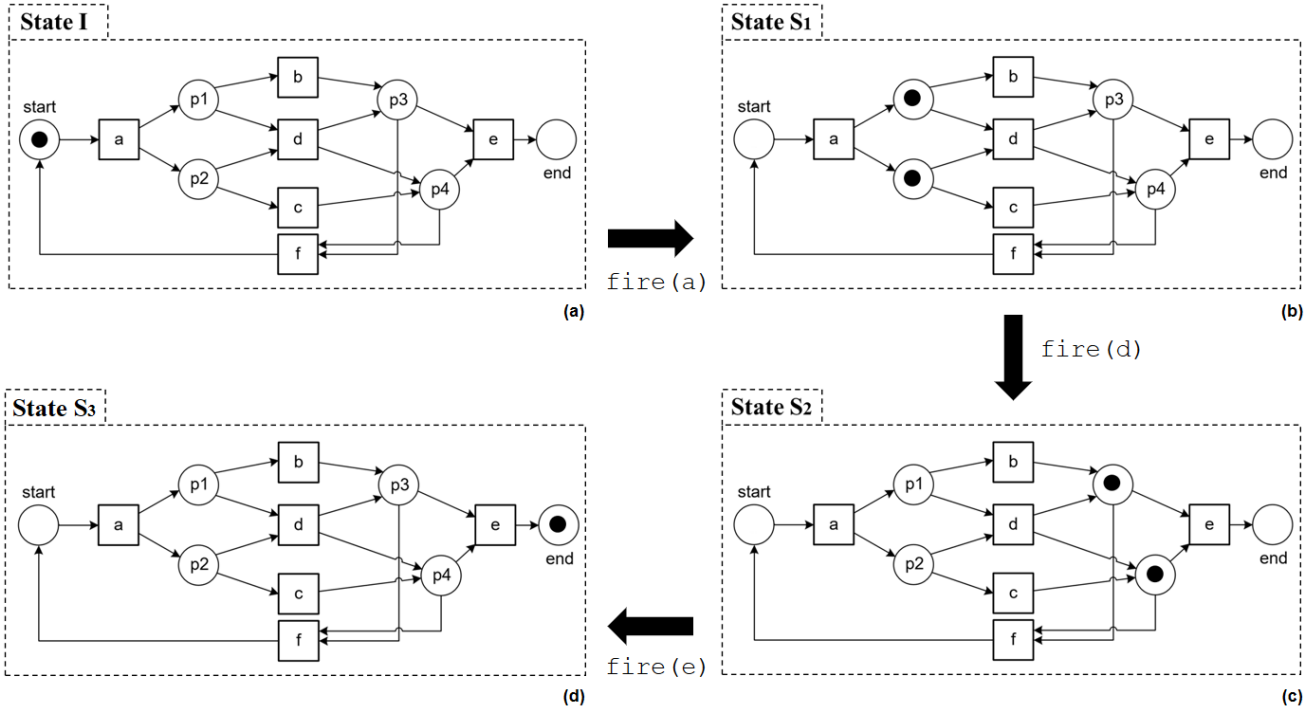

Fig. 2 Reachability analysis of business processes through Petri nets.

i.e., when there is one token in place *end* and no tokens in the other places (cf. Fig. 2(d)).

The problem can be easily expressed in PDDL [97]. From a technical point of view, the description of a planning problem $\mathcal{P}_{\mathcal{R}} = \langle I, G, \mathcal{P}_{\mathcal{D}} \rangle$ is organized in two separate files describing, respectively, the planning domain $\mathcal{P}_{\mathcal{D}}$ and the specific problem instance to be solved, represented through the initial state I and the goal G. Notice also that in PDDL parameters are distinguished by a '?' character at front, and the dash '-' is used to assign types to the parameters.

```
(define (domain petri-net)

 (:objects place transition)

 (:predicates (token ?p - place)
              (input_place ?t - transition
                           ?p - place)
              (output_place ?t - transition
                            ?p - place))

 (:action fire
  :parameters (?t - transition)
  :precondition (forall (?p - place)
                        (and (input_place ?t ?p)
                             (token ?p)))
  :effect (and (forall (?p - place)
                       (when (input_place ?t ?p)
                             (not (token ?p))))
               (forall (?p - place)
                       (when (output_place ?t ?p)
                             (token ?p)))
               (increase (total_cost) 1))
 )
)
```

In the planning domain $\mathcal{P}_{\mathcal{D}}$, the domain objects are the *places* and *transitions* of the Petri net. To capture all possible markings of the Petri net and their evolution after transitions firing, we define three boolean predicates and one numeric fluent in $\mathcal{P}_{\mathcal{D}}$, as follows:

`(token ?p - place)`. It holds if p contains a token in the currently reached marking.
`(input_place ?t - transition ?p - place)`. It holds iff p is an input place of t.
`(output_place ?t - transition ?p - place)`. It holds iff p is an output place of t.
`(total_cost)`. This numeric fluent keeps track of the cost of the plan found.

A single planning action `fire` is required to make the Petri net firing. The precondition of the action is that each place $p \in {}^\bullet t$ is enabled, i.e., it contains (at least) a token. The effect is to change the marking according to the firing rules, i.e., by consuming one token from input places of t and producing one token in any of the output places of t. It is worthy observing how the firing of a transition makes `total_cost` of the plan increases of a unitary value.

```
(define (problem pr1) (:domain petri-net)

 (:objects start p1 p2 p3 p4 end - place
           a b c d e f - transition)
```

```
 (:init
   (token start) (= (total_cost 0))
   (input_place a start) (output_place a p1)
   (output_place a p2) (input_place b p1)
   (output_place b p3) (input_place c p2)
   (output_place c p4) (input_place d p1)
   (input_place d p2) (output_place d p3)
   (output_place d p4) (input_place e p3)
   (input_place e p4) (output_place e end)
   (input_place f p3) (input_place f p4)
   (output_place f start)
 )

 (:goal (and (token end) (not (token start))
             (not (token p1)) (not (token p2))
             (not (token p3)) (not (token p4)))
 )
 (:metric minimize (total_cost))
)
```

Concerning the specific problem instance to be solved, we first declare the concrete domain objects involved in the planning problem, i.e., all places and transitions of the Petri net in Fig. 2. Then, the initial state I encodes the topology of the Petri net and the initial marking, i.e., it specifies instances to the predicates `input_place`, `output_place` and `token`. The goal state G reflects the reaching of the final marking in Fig. 2(d). Readers should notice that, if the purpose is to minimize the total cost of the plan, the planning problem also contains the following specification: `(:metric minimize (total-cost))`.

Fig. 2 shows a possible solution to the problem, which consists of first firing transition a (state S_1, cf. Fig. 2(b)), then on firing transition d (state S_2, cf. Fig. 2(c)), and finally on firing transition e (state S_3, cf. Fig. 2(d)). Since S_3 is a state satisfying the goal condition of the planning problem, the solution found is a valid *plan*. Furthermore, since we assumed that the cost of any `fire` action is equal to 1 (i.e., the cost of the plan will correspond to its length), then the plan found is *optimal*, as it contains the minimum number of planning actions to solve the reachability problem. □

For the sake of simplicity, in the previous example we have assumed a Petri net to be 1-bounded, also known as *safe*. A Petri net is safe if in any reachable marking from the initial marking, no place ever contains more than 1 token. Safeness is not a large limitation, as the behavior allowed by real BPs can be often represented as safe Petri nets [2]. However, with a simple modification of the encoding, we could extend the reasoning also to non-safe Petri nets.

Coming back to the complexity of classical planning, it is worth to notice that computing optimal plans is harder than computing satisfying plans, as the former involves a universal claim about the space of all plans [36]. Even if the problem of computing a plan is intractable in the worst case [15], yet currently large classical planning problems can be solved very quickly.

3 A Methodology to Build Planning Problems

The planning paradigm (in particular in its classical setting) provides a valuable set of theoretical and practical tools to tackle several potential challenges addressed by the BPM community. Of course, to exploit the benefits of the planning paradigm, it is necessary to reformulate such challenges in the form of planning problems in PDDL. This requires a *fundamental shift* in how one thinks about the specification of a problem. The "traditional" way consists of building a formal/semi-formal structure of the problem, codifying it in a suitable programming language and defining a dedicated algorithm that reasons over such a structure to find a possible solution to the problem, i.e., the focus is on "*how*" to tackle the problem. Conversely, if one aims at leveraging on the planning paradigm, the focus moves on "*what*" information is required to represent the problem in PDDL.

As a matter of fact, research into classical planning has for decades concentrated on theoretical issues [78], e.g., on the performances of search algorithms [39], the computational complexity of plan generation [15, 8], the expressiveness of the classical model [108] and the realization of frameworks for planning engines [42]. This has created a gap between the research area of theoretical (but often unrealistic) planning on the one hand, and real-world planning on the other. Whereas the main assumption of domain-independent planning should be the logical separation between the planning problem and the planning engine, the preferred trend has always been the encoding of planning problem representations in a "planner-friendly" way, mostly trying to accommodate "what planners can handle". As a consequence, even if several knowledge engineering (KE) methods exist to help developing and encoding the planning problems in PDDL, they had relatively little attention in the planning literature. In addition, as investigated by the work [101], a major issue of current KE approaches is that - for being utilized - they require a strong expertise in specific AI-oriented languages, such as PDDL, OWL, OCL, etc. If we look at BPM users, it is evident that this requirement might negatively affect the use of such approaches, since BPM users may not have the required expertise in languages that are not widely known in the BPM community.

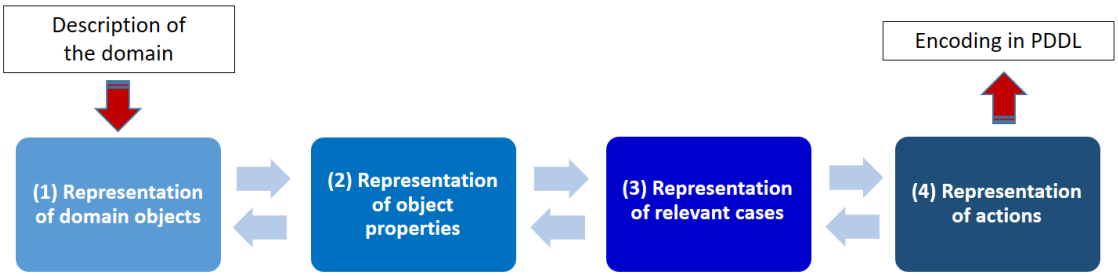

Fig. 3 A four-steps methodology to reformulate problems as planning problems.

For this reason, in this paper we have developed a concise four-steps methodology (cf. Fig. 3) that shows how a researcher/practitioner in BPM should approach the task of encoding a real-world problem as an appropriate planning problem. Instead of modeling a problem directly in the PDDL language, which can be complex for non-AI experts, the proposed methodology employs a simple semi-formal language for modeling planning problems and points out that the role of the designer is to reshape a concrete problem into a *object-centric way* rather than in an *action-centric way*, by identifying and specifying, in this exact order: *(i)* the domain objects and their potential hierarchy; *(ii)* the properties and relations involving domain objects; *(iii)* a number of relevant cases (i.e., initial and goal states, defined over instances of domain objects) to be potentially handled; *(iv)* a set of actions, together with their preconditions and effects, whose execution allows to modify the properties and relationships of domain objects. These four steps, which will be discussed in the following sections, can be iterated several times in order to refine the descriptions of objects/properties/cases/actions until a satisfactory representation has been obtained.

The proposed methodology, even if never comprehensively described before, has been successfully applied and evaluated against real BPM users in our previous work [69], where the target was to demonstrate that modeling the contextual knowledge of BPs executed in dynamic and knowledge-intensive contexts using a planning-oriented approach (leveraging our methodology), and then generating the procedural BP model through a state-of-the-art planner, can be more effective than directly modeling the BP in BPMN.

Being able to complete the four steps of the methodology allows to state that *a particular problem can be encoded as a planning problem and potentially tackled using planning technologies*. Notice that our methodology does not aim at acting as a tutorial for directly encoding a problem in PDDL, where more details can be added to the problem, such as the cost of executing single actions, etc. Conversely, its aim is to enable researchers/practitioners to understand *if and under which conditions planning is feasible for tackling a real-world problem* in BPM. In fact, as shown in Fig. 3, only after having enacted all the four steps of the methodology, the user has acquired the right understanding of the problem to decide if it is worth (or not) to concretely encode it in PDDL.

3.1 Representation of domain objects

The starting point for enacting our methodology is the availability of a description of the real-world problem of interest. Usually, such descriptions are provided in natural language and/or diagrammatic form, and outline the kind of problem to be solved, its main features and the ways it can be brought about. For example, in the case of the reachability analysis of BPs, a possible description of the problem is the one provided at the beginning of Example 1.

Of course, problem descriptions are usually not structured and discuss issues at different level of granularity. The main challenge is to extract the portion of reality that is being encoded as a planning problem at an appropriate level of detail. For example, in the reachability analysis problem, we are interested in *plans that involve the firing of transitions considering input and output places of such transitions, so that, starting from an initial marking, the intermediate marking of places can change until a target marking is achieved*. A useful strategy at this stage is to formulate example problems and descriptions of their possible solutions, also in a diagrammatic way, such as the sequence of firing shown in Fig. 2.

Our methodology is *object-centric*, i.e., it drives a user to represent a problem focusing on the relevant objects that are manipulated in the range of the problem itself. Thus, starting from a description of the problem, a user should first identify a list of *domain objects*, which represent logical containers of purely domain information in the problem domain space. In short, a domain object groups several (object) instances of interest that share common properties and behaviours.

For example, in the reachability analysis problem, we can define two domain objects, PLACE and TRANSITION, and as many object instances as are the places and transitions of interest. Sometimes it is also possible to build domain objects in a hierarchic way, by collecting objects together into a super-object, e.g., the object NODE can be potentially defined as a super-object of PLACE and TRANSITION.

3.2 Representation of properties and relationships of domain objects

Once selected the domain objects of interest, the second step consists of identifying the properties and relationships that involve domain objects and that are relevant for solving the problem under consideration, i.e., the idea is that *every problem instance that a planner will be asked to solve should be describable in terms of these properties and relationships.*

We can express them as a set of ground boolean *atomic terms* $v_1[y_1], v_2[y_2], ..., v_m[y_m] \in \mathsf{V}$ that range over a set of tuples (i.e., unordered sets of zero or more attributes) $y_1, y_2, \ldots y_m$ of *domain objects*. Argument types of an atomic term (taken from the set of domain objects previously defined) represent the finite domains over which the atomic term is interpreted.

Considering our example, for the domain objects PLACE and TRANSITION, the following atomic terms are suggested: INPUT_PLACE(T - TRANSITION, P - PLACE) and OUTPUT_PLACE(T - TRANSITION, P - PLACE) for expressing the fact that a place is in the set of input/output places of a transition, and TOKEN(P - PLACE) to specify that a place actually contains a token.

At this stage, we can make a binary distinction between two types of atomic terms, those whose truth value may change during the course of planning are called *dynamic*, while the remaining atomic terms are called *static*. In our example, atomic terms INPUT_PLACE and OUTPUT_PLACE are static, since they are used to define the structure of the Petri net of interest, which never changes in the range of a specific problem instance. On the other hand, atomic term TOKEN(P - PLACE) is dynamic, as it can change after the enactment of a firing action.

Such a distinction is important, because the set of dynamic atomic terms before and after performing any action manipulating their value can be seen as two different *world states* S_i and S_{i+1}. A state S_i is a complete assignment of values to dynamic atomic terms in V and is the result of i actions performed so far. Therefore, atomic terms in V may be thought of as "properties" of the world whose values may vary across states.

3.3 Representation of relevant cases

Modeling a planning problem involves representing how a plan pursues its goals. The goal may vary depending on the specific *case* C to be handled. A case C reflects an instantiation of the set of atomic terms V with a starting state INIT_C and a goal state GOAL_C. Both states are represented as conjunctions of atomic terms.

In INIT_C, one can instantiate only the static and dynamic atomic terms necessary for representing what is known about the initial state, i.e., the atomic terms that are initially true. On the other hand, GOAL_C is represented as a conjunction of those dynamic atomic terms we want to make true or false (in this case, the operator $\neg$ can be used) after the execution of a plan. Notice that a case contains also the instantiation of domain objects $\text{DOM\_OBJ}_\mathsf{C}$ that are deemed relevant for the case itself.

For example, in the specific instance of the reachability analysis problem of Fig.2, the case C can be defined as follows:

CASE C –>
$\text{DOM\_OBJ}_\mathsf{C}$:
–> PLACE : $\{start, p1, p2, p3, p4, end\}$
–> TRANSITION : $\{a, b, c, d, e, f\}$
INIT_C:
–> TOKEN(*start*), INPUT_PLACE(*a,start*),
–> OUTPUT_PLACE*(a,p1)*, OUTPUT_PLACE(*a,p2*),
–> INPUT_PLACE(*b,p1*), OUTPUT_PLACE*(b,p3)*,
–> INPUT_PLACE(*c,p2*), OUTPUT_PLACE(*c,p4*),
–> INPUT_PLACE(*d,p1*), INPUT_PLACE(*d,p2*),
–> OUTPUT_PLACE(*d,p3*), OUTPUT_PLACE(*d,p4*),
–> INPUT_PLACE(*e,p3*), INPUT_PLACE(*e,p4*),
–> OUTPUT_PLACE(*e,end*), INPUT_PLACE(*f,p3*),
–> INPUT_PLACE(*f,p4*), OUTPUT_PLACE(*f,start*)
GOAL_C:
–> TOKEN(*end*), ¬TOKEN(*start*),
–> ¬TOKEN(*p1*), ¬TOKEN(*p2*),
–> ¬TOKEN(*p3*), ¬TOKEN(*p4*)

Of course, in a real world problem there is a wide range of cases to handle. Consequently, the designer should identify a certain number of relevant cases that can be potentially represented with the domain objects and atomic terms previously specified. If this is not the case, according to Fig. 3, the designer can try to adapt the existing domain objects and atomic terms in order accommodate all the relevant cases to be solved.

3.4 Representation of actions

Once domain objects, atomic terms and a number of relevant cases have been identified, it is the required

to define the ways in which the world states (i.e., complete assignments of values to dynamic atomic terms) may change. To this aim, a set of individual *actions* $a_1, \ldots a_n \in \mathsf{A}$, described as single steps that consume input data and produce output data, must be specified.

The selection of actions should be performed by having in mind that an action specification defines what must be done in order to transform the current state of the world S_i to a new state S_{i+1}. For this reason, starting from the informal description of the problem of interest, actions can be identified as those operators whose enactment changes the value of (at least) a dynamic atomic term. In this direction, a good strategy to identify and build proper action specifications is to verify that:

- the action can be executed in some reachable world state;
- the action always affects a non-empty set of dynamic atomic terms;
- there exist at least an action that can be executed in the starting state INIT_c of the case c under consideration;
- any atomic term in the goal state GOAL_c of the case c under consideration should be affected by at least one action.

To be more concrete, each action is annotated with *preconditions* and *effects*. Preconditions are logical constraints defined as a *conjunction of static and dynamic atomic terms*, and they must be satisfied before the action is applied. As a consequence, an action $a_i \in \mathsf{A}$ can only be performed in a given world state S_i if that state satisfies the preconditions P_i of that activity, i.e., if $\mathsf{P}_i \subseteq \mathsf{S}_i$.

Each action has also a set of *effects* that establish the outcome of an action after its execution by changing the current world state S_i into a new world state S_{i+1}. Effects can be of two kinds. *Necessary effects* always change the value of some dynamic atomic term when the action is executed. *Conditional effects* are associated to a *condition* consisting of a conjunction of atomic terms; if such a condition holds when the action is executed, then the effect is enacted.

We can describe the effects of an action a_i as a set of atomic terms R_i that a_i removes from S_i, and another set of atomic terms L_i that a_i adds to S_i so that the overall result is the new state S_{i+1}. R_i is called the *remove-list* of a_i, whereas L_i is called the *add-list* of a_i. These entities are illustrated in Fig. 4. For their definition, we have taken inspiration from STRIPS, which is the simplest and oldest classical planning language originally developed by Fikes and Nillson in 1971 [29] for drastically reducing the expressiveness of a planning problem.

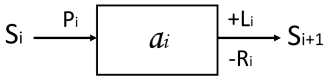

Fig. 4 The anatomy of an action in our methodology.

For example, in the reachability analysis problem, we can build the action FIRE for executing enabled transitions by considering that:

- it requires one *parameter* for representing the specific transition to be fired (i.e., TRANSITION(T)). The action can be specified as: FIRE(T - TRANSITION);
- this action can only be executed if any input place PLACE(P) of TRANSITION(T) contains at least a token, i.e., both atomic terms INPUT_PLACE(T,P) and TOKEN(P) hold. These are the *action's preconditions*;
- this action provides two conditional effects for (respectively) removing tokens from the input places of TRANSITION(T) and adding tokens to the output places of TRANSITION(T). This means that a new state S_{i+1} is produced by removing from S the atomic term TOKEN(P) (i.e., the *remove-list*) if INPUT_PLACE(T,P) holds, and adding the atomic term TOKEN(P) to S (that is the *add-list*) if OUTPUT_PLACE(T,P) holds.

The specification of the action FIRE can be compactly described as follows:

FIRE(T - TRANSITION):
P_{fire}: FORALL(P - PLACE) {INPUT_PLACE(T,P), TOKEN(P)}
R_{fire}: FORALL (P - PLACE) {IF(INPUT_PLACE(T,P)) –> TOKEN(P)}
L_{fire}: FORALL (P - PLACE) {IF(OUTPUT_PLACE(T,P) –> TOKEN(P)}

Note that universal/existential quantification of objects in preconditions and effects is allowed to assert that a property or relation, expressed with an atomic term, holds for all instances of a domain object. Then, while the definition of actions without preconditions that are always executable in any reachable state is possible, the designer should avoid to specify actions that provide no effects, since their execution would not contribute to the progress of the states.

In addition, coming back to the previous discussion on how to wisely selecting proper actions, the designer should consider to specify: *(i)* at least an action that is executable in the starting state, in a way that its preconditions are satisfied by the instances of (some) atomic terms in the starting state; *(ii)* a number of actions whose (combined) execution allows to achieve

all the instances of the atomic terms in the goal state. Concerning this last point, we notice that it is not required that any action affects all the atomic terms in the goal state, but it would be desirable that for any atomic term in the goal state there exists at least one action that manipulates it.

These are the minimum requirements that allow to build a candidate plan that is executable from the starting state and that is able to potentially achieve the goal state. In our example of reachability analysis, the action FIRE(T - TRANSITION) satisfies alone all the above requirements. Of course, a designer can provide more actions manipulating some atomic term, in a way that the effects of one action can match the preconditions of another action, allowing the actions to be connected into a sequence.

The proper completion of the four steps of the methodology guarantees that the problem of interest can be encoded as a classical planning problem in PDDL 2.1 and that is potentially solvable using state-of-the-art planning technologies. Note that the target of the methodology is not of driving a user to encode a planning problem directly in PDDL. For this task, several KE approaches from the planning community already exist in the research literature, as discussed at the beginning of this section. Conversely, our methodology aims at enabling researchers/practitioners to understand if and under which conditions planning is feasible for tackling a real-world problem from the BPM community.

4 Integrating Planning Technology with PMSs

In the previous section, we have proposed a four-steps methodology that drives a researcher/practitioner in reformulating a real-world problem into a planning problem. The next step is to encode the problem in PDDL and investigate how to concretely exploit state-of-the-art planning technologies to tackle challenges addressed by the BPM community. This requires to establish a tight collaboration between a BPM execution environment, i.e., a PMS, and a state-of-the-art planner.

Despite the variety of functionalities that a PMS can offer, the core features of such a software system reside in the modeling, automation, execution and monitoring of BPs. In the left-hand part of Fig. 5, the main components of a PMS (as identified in [26]) are shown, namely:

- the *process modeling tool*, which offers functionalities to create, modify, annotate (with business rules and additional data), and store/retrieve (into/from a dedicated repository) BP models;
- the *execution engine*, which uses a BP model to determine the logical/temporal order in which the activities of a BP have to be executed, create executable process instances and distribute work items to qualified and authorized BP participants in order to enact a BP from start to end;
- the *worklist handler*, which is the component through which BP participants are offered work items and commit to these;
- the *external services*, which can be invoked by the execution engine to delegate the enactment of some BP activities to external software applications across the organization;
- the *administration and monitoring tools*, which are required to deal with all operational matters of a PMS and to monitor the performance and the progress of the running BPs. In addition, such tools allow to record the execution of a BP step-by-step in the form of execution logs, which consist of traces of execution-related events.

On the other hand, a state-of-the-art planner can be seen as a black box, i.e., *no expertise of the internal working of the planner* is required to build a plan. The right-hand part of Fig. 5 shows the conceptual architecture of a state-of-the-art planner. Central to the planner is the *solver*, which is fed by a PDDL planning domain and a PDDL planning problem to produce a plan that achieves a certain goal starting from a (given) initial state.

In order to integrate a state-of-the art planner with a PMS, an additional component is required, namely the *Synchronization* component. The Synchronization component acts as unique entry point for incoming commands from the PMS to the planner; it enables communication and enforces synchronization between them. It includes three main components:

- the *Domain Builder* and the *Problem Builder* components, which build, respectively, a PDDL planning domain and a PDDL planning problem starting from a description of the domain objects, relationships between them, actions and a case (cf. sections 2 and 3);
- the *Plan Builder* component, which - on the one hand - is used to properly configure the planner with specific instructions to customize the plan synthesis (e.g., by selecting the specific search algorithm for building a plan), and - on the other hand - to translate a (synthesized) plan in a format that is interpretable by the PMS.

State-of-the-art planners can be invoked as services that are external to the PMS. Of course, which exact

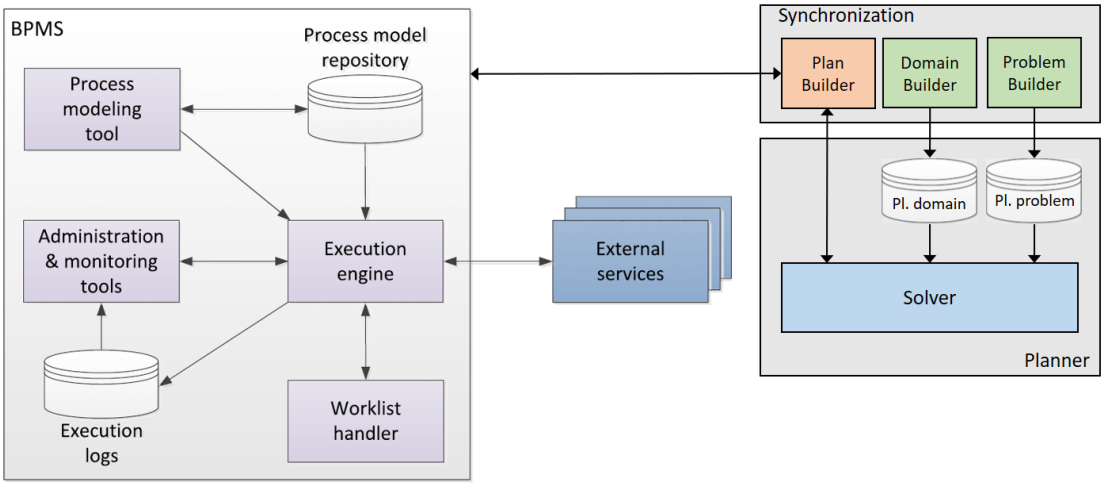

Fig. 5 The architecture of a PMS (left-hand part, original figure in [26]) and its integration with state-of-the-art planning technology.

PMS component interacts with a planner depends on the specific problem to be tackled. For example,

- if the planner is used to generate new process models, the interaction most likely involves the process modeling tool and/or the repository of BP models, e.g., see [69];
- if the planner is used to synthesize recovery procedures, a connection between the planner and the execution engine of the PMS is needed to retrieve information on the state of the faulty process to be adapted, e.g., see [73,74];
- if the planner is used to compute trace alignment in conformance checking, information about process models and event logs must be retrieved from the repository of BP models and the repository of execution logs, e.g., see [18,60].

The effort to integrate planning technology with PMSs does not go beyond installing the planner and developing the Synchronization component in a way that it enables to build (on-demand) the planning problem formulation, invoke the planner and interpret the plan returned by the planner in a format understandable by the PMS.

Notice that, since PDDL is independent from the specific planning system employed, *one can seamlessly integrate with the PMS different state-of-the-art planners compliant with PDDL*, as the problem formulation remains unchanged when moving from one planner to the other. This means that by converting a BPM problem into a planning problem, one can switch from a planner to another and from a searching algorithm to another with no or very limited effort.

5 What Automated Planning has done for BPM

In this section, we show how instances of some well-known problems from the BPM literature can be represented as planning problems for which planners can find a correct solution in a finite amount of time. Specifically, a number of research works exist on the use of planning techniques in the context of BPM, covering the various stages of the process life cycle (cf. Fig. 6).

For the *design-time* phase, existing literature has focused on exploiting planning to automatically generate: *(i)* candidate procedural BP models, describing which activities can be executed next in order to achieve some business goals starting from a complete description of the BP domain; and *(ii)* declarative models represented as execution constraints that the BP has to satisfy at run-time, thus providing more flexibility during BP execution. Some research works also exist that use planning to deal with problems for the *run-time* phase, e.g., to monitor and adapt running procedural BPs to cope with anomalous situations. Finally, for the *diagnosis phase*, the literature reports some works that use planning to perform conformance checking against procedural and declarative BP models. In the following sections we discuss in the detail how the use of planning has contributed to tackle the above research challenges from BPM literature.

5.1 Planning for the Automated Generation of Process Models

Process modeling is the first and most important step in the BPM life cycle, which intends to provide a high-level

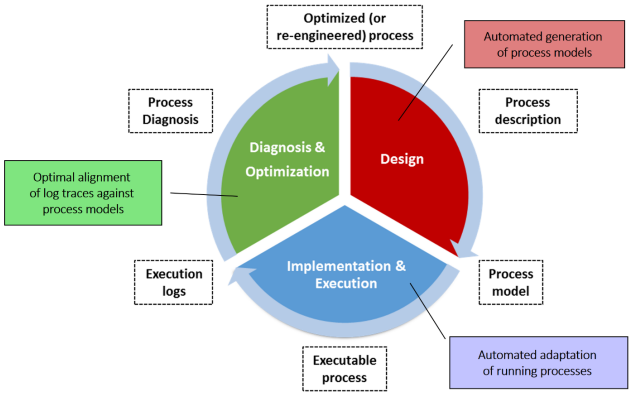

Fig. 6 Planning-based contributions in the various stages of the BPM life-cycle.

specification of a business process that is independent from implementation and serves as a basis for process automation and verification. Traditional process models are usually well-structured, i.e., they reflect highly repeatable and predictable routine work with low flexibility requirements. All possible options and decisions that can be made during process enactment are statically pre-defined at design-time and embedded in the control-flow of the process.

Challenge. Current BPM technology is generally based on rigid BP models making its application difficult in dynamic and possibly evolving domains, where pre-specifying the entire BP model is not always possible. This problem can be mitigated through specific approaches to process variability [53], which allow one to customize a base BP model by implementing specific variants of the BP itself. However, this activity is time-consuming and error-prone when more flexible BPs are to be modeled, due to their context-dependent nature that make it difficult to specify all the potential tasks interactions and BP variants in advance. A second solution consists of using declarative languages to model BPs. Instead of describing in a procedural way the order of activities to be performed, declarative languages define constraints between BP activities that must not be violated during BP execution. If, on the one hand, these languages allow for a high degree of flexibility, on the other hand, this freedom leads to understandability issues, since providing a clear representation of all possible BP executions becomes too complex for humans as the number of constraints increases on the BP model. Consequently, *the presence of mechanisms that facilitate the design phase of flexible BPs by allowing the automated generation of their underlying models is highly desirable* [22].

Application of Planning. The work [99] presents the basic idea behind the use of planning techniques for the *automated generation of a procedural process model*. Process activities can be represented as planning actions together with their preconditions and effects stating what contextual data may constrain the process execution or may be affected after an activity completion. The planning domain is therefore enriched with a set of propositions that characterize the contextual data describing the process domain. Given an initial description of the process domain, the target is to automatically obtain a plan (i.e., a process model and its control-flow) that consists of process activities contextually selected and ordered in a way to satisfy some business goals.

In the research literature, there are four main approaches that use planning on the basis of the general schema outlined above. In [91], the authors exploit a planner for modeling processes in SHAMASH, a knowledge-based system for the design of business processes. The planner, which is fed with a semantic representation of the process knowledge, produces a parallel plan of activities that is translated into SHAMASH and presented graphically to the user. The obtained plan proposes the scheduling of parallel activities that may handle time and resource constraints. Notice that the emphasis here is on supporting processes for which one has complete knowledge.

The work [28] is based on learning activities as planning operators and feeding them to a planner that generates the process model. An interesting result concerns the possibility of producing process models even though the activities may not be accurately described. In such cases, the authors use a best-effort planner that is always able to create a plan, even though the plan may be incorrect. By refining the preconditions and effects of planning actions, the planner will be able to produce several candidate plans, and after a finite number of refinements, the best candidate plan (i.e., the one with the lowest number of unsatisfied preconditions) is translated into a process model.

In the SEMPA approach [43], process actions are semantically described by specifying their input/output parameters with respect to an ontology implemented in OWL. Starting from such a knowledge, planning is used to derive an action state graph (ASG) consisting of those actions whose execution leads to a given goal from a pre-specified initial state. Then, a process model represented as an UML activity diagram is extracted from the ASG by identifying the required control flow for the process. Interestingly, the planning algorithm implemented in SEMPA provides the ability to build the ASG in presence of initial state uncertainty and with different conflicting goals.

The works [67,69] refer to a technique based on partial-order planning algorithms and declarative specifications of process activities in PDDL for synthesizing a library of process templates to be enacted in contextual scenarios. The resulting templates guarantee that concurrent activities of a process template are effectively independent one from another (i.e., they cannot affect the same data) and are reusable in a variety of partially specified contextual environments. A key characteristic of this approach is the role of contextual data acting as a driver for process templates generation.

Whereas the above works focused on generating procedural BP models, the work [56] proposes an approach that leverages on planning via model checking [40] to synthesize multiple candidate plans (in form of sequences of activities) satisfying a set of declarative constraints defined on BP activities at design-time. The target here is to provide a human modeler with a priori experience of how a declarative BP model may possibly evolve at run-time.

5.2 Planning for Process Adaptation

Process Adaptation is the ability of a process to react to exceptional circumstances (that may or may not be foreseen) and to adapt/modify its structure accordingly [94]. Exceptions can be either anticipated or unanticipated. An anticipated exception can be planned at design-time and incorporated into the process model, i.e., a (human) process designer can provide an exception handler which is invoked during run-time to cope with the exception. Conversely, unanticipated exceptions generally refer to situations, unplanned at design-time, that may emerge at run-time and can be detected by monitoring discrepancies between the real-world processes and their computerized representation.

Challenge. In many dynamic application domains (e.g., emergency management, smart manufacturing, etc.), the number of possible anticipated exceptions is often too large, and traditional manual implementation of exception handlers at design-time is not feasible for process designers, who have to anticipate all potential problems and ways to overcome them in advance. Furthermore, anticipated exceptions cover only partially relevant situations, as in such scenarios many unanticipated exceptional circumstances may arise during process execution. *The management of processes in dynamic domains requires that BPM environments provide real-time monitoring and automated adaptation features during process execution, in order to achieve the overall objectives of the processes still preserving their structure by minimising any human intervention* [22].

Application of Planning. The first work dealing with this research challenge is [50]. It discusses how planning can be interleaved with process execution to suggest compensation procedures or the re-execution of activities if some anticipated failure arises. The work [33] presents an approach for enabling automated process instance change in case of activity failures occurring at run-time that lead to a process goal violation. The approach relies on a partial-order planner for the generation of a new complete process model that complies with the process goal. The generated model is substituted at run-time to the original process that included the failed task.

The above works use planning to tackle anticipated exceptions or to completely redefine the process model when some activity failure arises. However, in dynamic domains, it would be desirable to adapt a running process by modifying only those parts of the process that need to be changed/adapted and keeps other parts stable, by avoiding to revolutionize the work list of activities assigned to the process participants [22].

In this direction, the SmartPM approach and system [73,74] provides a planning-based mechanism that requires no predefined handler to build on-the-fly the *recovery procedure* required to adapt a running process instance. Specifically, adaptation in SmartPM can be seen as reducing the gap between the *expected reality*, i.e., the (idealized) model of reality that reflects the intended outcome of the task execution, and the *physical reality*, i.e., the real world with the actual values of conditions and outcomes. A recovery procedure is needed during process execution if the two realities are different from each other. A misalignment of the two realities often stems from errors in the tasks outcomes (e.g., incorrect data values) or is the result of exogenous events coming from the environment. If the gap between the expected and physical realities is such that the process instance cannot progress, the SmartPM system invokes an external planner to build a recovery procedure as a plan, which can thereby resolve exceptions that were not designed into the original process. This technique is challenged in an emergency management domain where mobile technologies and environmental sensors are employed to execute BP activities. Notice that a similar framework to tackle process adaptation through planning is also adopted in the research works [70,72,76].

In SmartPM, the problem of automatically synthesizing a recovery procedure is encoded as a classical planning problem in PDDL. The planning domain consists of propositions that characterize the contextual data describing the process domain. Planning actions

are built from a repository of process activities annotated with preconditions and effects expressed over the process domain. Then, the initial state reflects the physical reality at the time of the failure, while the goal state corresponds to the expected reality. A classical planner (in this specific case, the LPG-td planner [38]) fed with such inputs searches for a plan that may turn the physical reality into the expected reality by adapting the faulty process instance. In [75], the effectiveness of SmartPM is also demonstrated by adapting a BP coming from the smart manufacturing domain.

A similar adaptation strategy is applied in [14], which proposes a goal-driven approach for service-based applications to adapt business processes to run-time context changes. Process models include service annotations describing how services contribute to the intended goal. Contextual properties are modeled as state transition systems capturing possible values and possible evolutions in the case of precondition violations or external events. Process and context changes that prevent goal achievement are managed through an adaptation mechanism based on service composition via planning.

Finally, the work [10] proposes a runtime mechanism that uses dependency scopes for identifying critical parts of the processes whose correct execution depends on some shared variables and intervention processes for solving potential inconsistencies between data. Intervention processes are automatically synthesised through a planner based on Constraint Satisfaction Problem (CSP) techniques. While closely related to SmartPM, this work requires specification of a (domain-dependent) adaptation policy, based on volatile variables and when changes to them become relevant.

5.3 Planning for Conformance Checking

Within the discipline of process mining, conformance checking is the problem of verifying whether the observed behavior stored in an event log is compliant with the process model that encodes how the process is allowed to be executed to ensure that norms and regulations are not violated. The notion of alignment [4] provides a robust approach to conformance checking, which makes it possible to exactly pinpoint the deviations causing nonconformity with a high degree of detail. An alignment between a recorded process execution (log trace) and a process model is a pairwise matching between activities recorded in the log (events) and activities allowed by the model (process activities).

Challenge. In general, a large number of possible alignments exist between a process model and a log trace, since there may exist manifold explanations why a trace is not conforming. It is clear that one is interested in finding the most probable explanation, i.e., one of the alignments with the least expensive deviations (i.e., optimal alignments), according to some function assigning costs to deviations. The existing techniques to compute optimal alignments against procedural [5] and declarative [59] process models provide ad-hoc implementations of the A* algorithm. The fact is that when process models and event logs are of considerable size the existing approaches do not scale efficiently due to their ad-hoc nature and they are unable to accomplish the alignment task. *In the era of Big Data, scalable approaches to process mining are desperately necessary, as also advocated by the IEEE Task Force in Process Mining [4].*

Application of planning. In case of procedural models represented as Petri Nets, the work [60] proposes an approach and a tool to encode the original algorithm for trace alignment [5] as a planning problem in PDDL. Specifically, starting from a Petri net N and an event log L to be aligned, for each log trace $\sigma_L \in L$ it is built:

- a planning domain P_D, which encodes the propositions needed to capture the structure of N and to monitor the evolution of its marking, and three classes of planning actions that represent "alignment" moves: synchronous moves (associated with no cost), model moves and log moves;
- a planning problem P_R, which includes a number of constants required to properly ground all the domain propositions in P_D; in this case, constants will correspond to the place and transition instances involved in N.

Then, the initial state of P_R is defined to capture the exact structure of the specific log trace σ_L of interest and the initial marking of N, and the goal condition is encoded to represent the fact that N is in the final marking and σ_L has been completely analyzed. At this point, for any trace of the event log, an external planner is invoked to synthesize a plan to reach the final goal from the initial state, i.e., a sequence of alignment moves (each of which is a planning action) that establish an optimal alignment between σ_L and N.

In the work [58], the authors report on a planning-based technique that extends what was proposed in [60], by removing the total-ordering assumption of trace events and tackling the issue of aligning partially-ordered traces - i.e., having a coarse granularity for the event timestamps - to Petri nets.

Relatively close to [60] is the work [23] where authors use planners to recover the missing recording of events in log traces. The concept of missing event recordings is very similar to model moves in [60]. How-

ever, in [23] it is assumed that all executions are compliant with the model and, hence, every event that is present in the incomplete log trace is assumed to be correct. In other words, they do not foresee log moves.

In case of declarative process models, where relationships among process activities are implicitly defined through logical constraints expressed in the well-known LTL_f (Linear Temporal Logic on finite traces) formalism, the work [18] leverages on planning techniques to search for optimal alignments. A planning domain is encoded to capture the structure of the finite state automata (augmented with special transitions for adding/removing activities to/from a log trace) corresponding to the individual LTL_f constraints that compose the declarative model. The same can be done for the specific trace to be aligned, which is represented as a simple automaton that consists of a sequence of states. In addition, the definition of specific domain propositions allows to monitor the evolution of any automaton. At this point, the initial state of the planning problem is encoded to capture the exact structure of the trace automaton and of every constraint automaton. This includes the specification of all the existing transitions that connect two different states of the automata. The current state and the accepting states of any trace/constraint automaton are identified as well. Then, the goal condition is defined as the conjunction of the accepting states of the trace automaton and of all the accepting states of the constraint automata. At this point, a planner is invoked with such inputs to synthesize a plan that establishes an optimal alignment between the declarative model and the log trace of interest.

A previous work tackling the same issue of aligning trace logs to declarative process models using classical planning is presented in [19]. However, differently from [18], in [19] the authors need to determine, for each log trace, a bound on the maximum number of instances of each process activity needed to align the trace. However, such a bound is not minimal, i.e., more activity instances than those needed for the alignment may be incorporated in the planning problem. This may dramatically increase the search space.

Notably, both the works [60] and [18] report on results of experiments conducted with several planners fed with combinations of real-life and synthetic event logs and processes. The results show that, when process models and event-log traces are of considerable size, their planning-based approach outperforms the existing approaches based on ad-hoc implementations of the A* algorithm [5,59] even by several orders of magnitude, and they are always able to properly complete the alignment task (while the existing approaches often run out of memory).

Finally, under the umbrella of the approaches developed for the diagnosis stage of the BPM life-cycle, the work [65] shows how classical planning can be used to efficiently solve the problem of checking compliance requirements expressed in terms of LTL_f rules, by pointing at the BP activities in a BPMN model where compliance is breached.

6 Related Work

The AI community has been involved with research on BPM for several decades, and some works have investigated the role that automated planning technologies can play in the construction of PMSs that manage complex BPs, while remaining robust, reactive, and adaptive in the face of both environmental and tasking changes. One of the first works dealing with this research challenge is [84], which describes how techniques from the planning community could be leveraged for synthesizing new BPs and repairing previously defined BPs that are no longer suitable for a given situation. The work [9] discusses at high level how the use of an intelligent assistant based on planning techniques may suggest compensation procedures or the re-execution of activities if some anticipated failure arises during the process execution. In [50] the authors describe how planning can be interleaved with process execution and plan refinement, and investigates plan patching and plan repair as means to enhance flexibility and responsiveness. Similarly, the approach presented in [92] highlights the improvements that a legacy workflow application can gain by incorporating planning techniques into its day-to-day operation.

To the best of our knowledge, since 2002 there is no other published work that has proposed a general methodology for the application of automated planning techniques to BPM; several works exist, but they are targeted to the application of planning techniques for tackling specific challenges in the area of BPM, as thoroughly detailed in Section 5.

Another research area (that is highly related to BPM) where the use of planning contributed to tackle complex challenges is the one of web service composition [11,89,90,6,20]. Among the most interesting works in this area, only few of them employ domain-independent planners to generate compositions of atomic web services [80,102,86,52,51]. In [88], planning technologies are used to obtain service composition at the knowledge level [77]. In [46] a customization of the FF (Fast Forward) planner [45] is used to construct service orchestrations from atomic IT entities de-

scribed in a planning-like manner, while the works [81, 103] employ situation calculus combined with planning techniques to perform service composition.

As opposed to the works that view services as atomic planning operators, there is a significant amount of research [104, 12, 13] that consider services as stateful, i.e., with a well-defined related behavioural description derived from a BPEL specification. In such works the requirements of the desired composite service are expressed in a temporal logic-like language, and planning techniques based on model-checking are used for service composition.

Finally, it is worth to mention that also in the area of Ubiquitous Computing there is a massive usage of planning technologies for dealing with the evolution and the uncertainty of dynamic environments. Interested readers can refer to the work [37] for a detailed and up to date literature review on this topic.

7 Discussion and Conclusion

We are at the beginning of a profound transformation of BPM due to the recent advances in AI research [48]. In this context, we have shown how Automated Planning can offer a mature paradigm to introduce autonomous behaviour in BPM for tackling complex challenges in a theoretically grounded and domain-independent way.

In this direction, we have first described how a researcher/practitioner should approach the task of encoding a real-world problem as an appropriate planning problem and under which conditions planning is feasible for processes. Then, we have investigated how to concretely integrate the planning technology with PMSs. Finally, we have shown how instances of some well-known problems from the BPM literature can be represented as planning problems for which planners can find a correct solution in a finite amount of time.

One of the motivations of using automated planning techniques in the context of BPM is to realize methods and technologies that are more general, flexible, and oriented to the reasoning task in dynamic contexts. Over the last years, several BP-oriented declarative modeling notations, such as Declare [87], Condition Response Graphs (DCR) [44], Guard-Stage-Milestone (GSM) [47] and Case Management Model and Notation (CMMN) [85], have been proposed to tackle this same objective. Instead of explicitly model the flow of the interactions among BP activities, Declare and DRC graphs rely (respectively) on a set of temporally extended constraints and on binary relations supporting decomposition for implicitly specifying the allowed behavior of a BP. On the other hand, GSM and CMMN provide a declarative rule-based framework for specifying artifact-centric BP models.

The above notations share with automated planning the declarative nature of the language, which is used not to model the exact flow of the problem, but to establish desired results, i.e., specifying what they want to happen but not how it should happen. The major difference lies in the *generality* of the language used. On the one hand, declarative notations for BPs can be employed to specify BP models in a declarative way through a set of constraints or artifacts. However, this means that they are bound to a specific problem (i.e., the definition of BP models) and their interpretation is well constrained in this domain. Conversely, planning models encoded in PDDL are *general*, in the sense that a planner can be fed with the description of any planning problem in PDDL (as defined in Section 2) without knowing what the actions and domain stand for, and for any such description it can synthesize a plan achieving the goal.

Leveraging the planning paradigm (in particular in its classical setting) may lead to several advantages:

- Planning models are *general* (see the discussion above). This generality is intimately tied to the notion of intelligence which requires the ability to deal with new problems [36]. *This means that planners can potentially solve any BPM problem that can be converted into a planning problem in PDDL.*
- Planning models are *human-comprehensible*, as the PDDL language allows to describe the planning domain and problem of interest in a high-level terminology, which is *readily accessible and understandable by IT professionals.*
- The *standardized representation* of a planning model in PDDL allows to *exploit a large repertoire of planners and searching algorithms* with very limited effort; i.e., one can seamlessly update to the recent versions of the best performing automated planners, with evident advantages in term of versatility and customization.
- Planning models, if encoded with the classical approach, constitute implicit representations of *finite state controllers*, and can be thus *queried by standard verification techniques*, such as Model Checking.
- BPM environments can invoke planners as *external services.* Therefore, *no expertise of the internal working of the planners* is required to build a plan.
- Planning systems employ search algorithms driven by intelligent heuristics that allow to scale up efficiently to large problems.

On the other hand, although Planning (in particular in its classical setting) embeds properties desirable for BPM, it imposes some restrictions for addressing more

expressive problems, including preferences and nondeterministic action effects. Furthermore, planning models require that actions are completely specified in term of I/O data elements, preconditions, and effects, and that the execution context can be captured as part of the planning domain. These aspects can frame the scope of applicability of the planning paradigm to BPM, and an interesting future work would consist of finding relevant counter-examples that may show the boundaries of such an applicability.

In the AI literature, there exist more advanced forms of non-classical planning models that can potentially mitigate the above restrictions and used to tackle further challenging tasks from the BPM literature. For example:

- Markov Decision Processes (MDPs) [21] and Partially observable MDPs (POMDPs) [93] planning generalize the model underlying classical planning by allowing actions with stochastic effects and fully observable states (in case of MDPs) or partially observable states (in case of POMDPs). Such planning models could be employed to simulate and monitor the executions of case-oriented BP models (e.g., CMMN), where the effect of the activities is not always completely predictable at design-time, and may emerge gradually at run-time on a case-by-case basis. In addition, the stochastic nature of MDPs planning provides a natural framework for predicting features of BPs in the context of research in BP prediction [64,31].
- Hierarchical Task Planning, or HTN planning, is focused on the definition of general strategies for solving problems rather than in representing and solving the problems themselves [39]. The main feature of HTN planning is that the dependency among planning actions can be given in the form of hierarchically structured networks. As suggested by [41,68], this feature makes HTN planning particularly suitable to tackle the challenge of automatically synthesizing at run-time, i.e., when it becomes clear what needs to be done at a specific point in the BP, the "content" (in form of sub-processes of different granularity) of those BP activities that are underspecified at design-time [98].
- Temporal planning [17] deals with durative actions and actions that may overlap in time, i.e., which can be taken simultaneously. Duration of actions may vary, and they may have complex interdependencies that determine which combinations are possible. These features represent a good basis to investigate trace alignment of procedural and declarative BPs when the compliance with temporal patterns, such as the ones introduced in [54], must be satisfied in addition to the traditional control-flow oriented constraints.

Of course, the above discussion is not exhaustive, since several non-classical variants of the classical planning models exist (cf. [36]), and their usefulness in the BPM context is yet to be demonstrated. Nonetheless, the use of classical planning techniques in the BPM context is still most widespread, thanks to the the robustness and maturity of the existing state-of-the-art planning systems, which allow to solve efficiently several complex real-world BPM challenges. In addition, it is often possible to solve non-classical planning problems using classical planners by means of well-defined transformations [35].

In this direction, as a further future work, we aim at developing a rigorous methodology to acquire relevant literature on the use of classical and non-classical planning for BPM and derive a common evaluation framework to systematically review and classify the existing methods.

This paper extends previous work in [66] including new elements such as: *(i)* a case study describing how a practical problem from the BPM domain can be encoded in PDDL and solved using planning techniques; *(ii)* a methodology that shows how approaching the task of encoding an appropriate planning problem; *(iii)* a discussion about the integration of the planning technology in PMSs; *(iv)* a new related work section. Furthermore, all the other sections have been refined and enhanced to present the material more thoroughly.

References

1. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of circuits, systems, and computers **8**(01), 21–66 (1998)
2. van der Aalst, W.M.P.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: Business Process Management, pp. 161–183. Springer (2000)
3. van der Aalst, W.M.P.: Business Process Management: A Comprehensive Survey. ISRN Software Engineering (2013). DOI 10.1155/2013/507984
4. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
5. Adriansyah, A., van Dongen, B.F., Zannone, N.: Controlling Break-the-Glass Through Alignment. In: SOCIALCOM'13. IEEE Computer Society (2013). DOI 10.1109/SocialCom.2013.91
6. Agarwal, V., Chafle, G., Dasgupta, K., Karnik, N.M., Kumar, A., Mittal, S., Srivastava, B.: Synthy: A system for end to end composition of web services. Journal on Web Semantics: Science, Services and Agents on the World Wide Web **3**(4), 311–339 (2005). DOI 10.1016/j.websem.2005.09.002

7. Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: VIVACE: A framework for the systematic evaluation of variability support in process-aware information systems. Information and Software Technology **57**(Supplement C), 248 – 276 (2015). DOI 10.1016/j.infsof.2014.05.009
8. Baral, C., Kreinovich, V., Trejo, R.: Computational complexity of planning and approximate planning in the presence of incompleteness. Artificial Intelligence **122**(1-2), 241–267 (2000)
9. Beckstein, C., Klausner, J.: A Meta Level Architecture for Workflow Management. Journal of Integrated Design and Process Science **3**(1) (1999)
10. van Beest, N.R., Kaldeli, E., Bulanov, P., Wortmann, J.C., Lazovik, A.: Automated runtime repair of business processes. Information Systems **39** (2014). DOI 10.1016/j.is.2013.07.003
11. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Composition of E-services That Export Their Behavior. In: Proceedings of the First International Conference on Service-Oriented Computing, ICSOC 2003, pp. 43–58. Springer (2003). DOI 10.1007/978-3-540-24593-3_4
12. Bertoli, P., Pistore, M., Traverso, P.: Automated Web Service Composition by On-the-Fly Belief Space Search. In: 26th Int. Conference on Automated Planning and Scheduling, ICAPS'16, pp. 358–361 (2006)
13. Bertoli, P., Pistore, M., Traverso, P.: Automated composition of web services via planning in asynchronous domains. Artificial Intelligence **174**(3), 316 – 361 (2010). DOI https://doi.org/10.1016/j.artint.2009.12.002
14. Bucchiarone, A., Pistore, M., Raik, H., Kazhamiakin, R.: Adaptation of service-based business processes by context-aware replanning. In: IEEE 4th Int. Conf. on Service-Oriented Computing and Applications (SOCA'11). IEEE (2011). DOI 10.1109/SOCA.2011.6166209
15. Bylander, T.: The computational complexity of propositional strips planning. Artificial Intelligence **69**(1-2), 165–204 (1994)
16. Cossu, F., Marrella, A., Mecella, M., Russo, A., Bertazzoni, G., Suppa, M., Grasso, F.: Improving Operational Support in Hospital Wards through Vocal Interfaces and Process-Awareness. In: 25th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2012). IEEE (2012). DOI 10.1109/CBMS.2012.6266329
17. Cushing, W., Weld, D.S., Kambhampati, S., Mausam, K.T., Talamadupula, K.: Evaluating Temporal Planning Domains. In: 17th Int. Conference on Automated Planning and Scheduling (ICAPS-07), pp. 105–112 (2007)
18. De Giacomo, G., Maggi, F.M., Marrella, A., Patrizi, F.: On the Disruptive Effectiveness of Automated Planning for LTLf-Based Trace Alignment. In: Thirty-First AAAI Conf. on Artificial Intelligence (AAAI-17). AAAI Press (2017). URL `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14652`
19. De Giacomo, G., Maggi, F.M., Marrella, A., Sardiña, S.: Computing Trace Alignment against Declarative Process Models through Planning. In: 26th Int. Conference on Automated Planning and Scheduling (ICAPS 2016), pp. 367–375 (2016). URL `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13094`
20. De Giacomo, G., Mecella, M., Patrizi, F.: Automated Service Composition Based on Behaviors: The Roman Model. In: Web Services Foundations, pp. 189–214. Springer (2014). DOI 10.1007/978-1-4614-7518-7_8
21. Dean, T.L., Kaelbling, L.P., Kirman, J., Nicholson, A.E.: Planning with deadlines in stochastic domains. In: Eleventh National Conference on Artificial Intelligence (AAAI-93), vol. 93, pp. 574–579 (1993)
22. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. J. Data Semantics **4**(1) (2015). DOI 10.1007/s13740-014-0038-4. URL `https://doi.org/10.1007/s13740-014-0038-4`
23. Di Francescomarino, C., Ghidini, C., Tessaris, S., Sandoval, I.V.: Completing Workflow Traces Using Action Languages. In: 27th Int. Conference of Advanced Information Systems Engineering (CAiSE'15). Springer (2015). DOI 10.1007/978-3-319-19069-3_20
24. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and analysis of BPMN process models using Petri nets. Queensland University of Techn., Tech. Rep (2007)
25. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
26. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.: Process-Aware Information Systems: Bridging People and Software through Process Technology, 1st edn. John Wiley & Sons (2005). DOI 10.1002/0471741442
27. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 1st edn. Springer-Verlag Berlin Heidelberg (2013). DOI 10.1007/978-3-642-33143-5
28. Ferreira, H., Ferreira, D.: An Integrated Life Cycle for Workflow Management Based on Learning and Planning. Int. J. Coop. Inf. Systems **15** (2006). DOI 10.1142/S0218843006001463
29. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence **2**(3-4), 189–208 (1971)
30. Fox, M., Long, D.: Pddl2. 1: An extension to pddl for expressing temporal planning domains. Journal of Artificial Intelligence Research **20**, 61–124 (2003)
31. Francescomarino, C.D., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: Which one suits me best? CoRR **abs/1804.02422** (2018). URL `http://arxiv.org/abs/1804.02422`
32. Frederick, T.: The principles of scientific management. USA: Harper & Brothers (1911)
33. Gajewski, M., Meyer, H., Momotko, M., Schuschel, H., Weske, M.: Dynamic Failure Recovery of Generated Workflows. In: 16th Int. Conference on Database and Expert Systems Applications (DEXA'05). IEEE Computer Society Press (2005). DOI 10.1109/DEXA.2005.78
34. Geffner, H.: Computational models of planning. Wiley Int. Reviews: Cog. Sc. **4**(4) (2013). DOI 10.1002/wcs.1233
35. Geffner, H.: Non-classical Planning with a Classical Planner: The Power of Transformations. In: 14th European Conf. on Logics in AI (JELIA) (2014). DOI 10.1007/978-3-319-11558-0_3. URL `https://doi.org/10.1007/978-3-319-11558-0\_3`
36. Geffner, H., Bonet, B.: A Concise Introduction to Models and Methods for Automated Planning. Morgan & Claypool Publishers (2013). DOI 10.2200/S00513ED1V01Y201306AIM022)
37. Georgievski, I., Aiello, M.: Automated planning for ubiquitous computing. ACM Computing Surveys (CSUR) **49**(4), 63 (2017)
38. Gerevini, A., Saetti, A., Serina, I., Toninelli, P.: LPG-TD: A fully automated planner for PDDL2. 2 domains.

In: 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04) International Planning Competition abstracts (2004)
39. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice (2004)
40. Giunchiglia, F., Traverso, P.: Planning as model checking. In: European Conference on Planning, pp. 1–20. Springer (1999)
41. Gonzlez-Ferrer, A., Fernndez-Olivares, J., Castillo, L.: From business process models to hierarchical task network planning domains. The Knowledge Engineering Review **28**(2), 175193 (2013). DOI 10.1017/S0269888912000410
42. Helmert, M.: The Fast Downward Planning System. Journal of Artificial Intelligence Research **26** (2006)
43. Henneberger, M., Heinrich, B., Lautenbacher, F., Bauer, B.: Semantic-Based Planning of Process Models. In: Multikonferenz Wirtschaftsinformatik (2008)
44. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Nested dynamic condition response graphs. In: International Conference on Fundamentals of Software Engineering, pp. 343–350. Springer (2011)
45. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14**, 253–302 (2001)
46. Hoffmann, J., Weber, I., Kraft, F.: SAP speaks PDDL. In: 24th National Conference of the American Association for Artificial Intelligence (AAAI'10) (2010)
47. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: On the Move to Meaningful Internet Systems: Confederated Int'l Conferences: CoopIS, DOA-SVI, and ODBASE, *LNCS*, vol. 5332, pp. 1152–1163. Springer (2008)
48. Hull, R., Motahari Nezhad, H.R.: Rethinking BPM in a Cognitive World: Transforming How We Learn and Perform Business Processes. In: 14th Int. Conference on Business Process Management (BPM'16). Springer (2016). DOI 10.1007/978-3-319-45348-4_1
49. Humayoun, S.R., Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Bortenschlager, M., Steinmann, R.: The WORKPAD User Interface and Methodology: Developing Smart and Effective Mobile Applications for Emergency Operators. In: 5th Int. Conf. on Universal Access in Human-Computer Interaction (UAHCI 2009), pp. 343–352. Springer Berlin Heidelberg (2009). DOI 10.1007/978-3-642-02713-0_36. URL `https://doi.org/10.1007/978-3-642-02713-0_36`
50. Jarvis, P., et al.: Exploiting AI Technologies to Realise Adaptive Workflow Systems. In: Proceedings of the AAAI Workshop on Agent-Based Systems in the Business Context (1999)
51. Kaldeli, E., Lazovik, A., Aiello, M.: Domain-independent planning for services in uncertain and dynamic environments. Artificial Intelligence **236**, 30–64 (2016)
52. Klusch, M., Gerber, A.: Fast composition planning of owl-s services and application. In: 4th European Conference on Web Services (ECOWS'06), pp. 181–190. IEEE (2006)
53. La Rosa, M., van der Aalst, W.M.P., Dumas, M., Milani, F.P.: Business Process Variability Modeling: A Survey. ACM Comput. Surv. **50**(1), 2:1–2:45 (2017). DOI 10.1145/3041957
54. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requirements Engineering **19**(2), 113–141 (2014)
55. Lasi, H., Fettke, P., Kemper, H.G., Feld, T., Hoffmann, M.: Industry 4.0. Business & Information Systems Engineering **6**(4), 239–242 (2014). DOI 10.1007/s11576-014-0424-4
56. Laurent, Y., Bendraou, R., Baarir, S., Gervais, M.P.: Planning for declarative processes. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 1126–1133. ACM (2014)
57. Lee, J., Bagheri, B., Kao, H.A.: A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manufacturing Letters **3**, 18–23 (2015)
58. de Leoni, M., Lanciano, G., Marrella, A.: Aligning Partially-Ordered Process-Execution Traces and Models Using Automated Planning. In: 28th Int. Conference on Automated Planning and Scheduling (ICAPS 2018), pp. 321–329 (2018). URL `https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17739`
59. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: Aligning Event Logs and Declarative Process Models for Conformance Checking. In: 10th Int. Conference on Business Process Management (BPM'12). Springer (2012)
60. de Leoni, M., Marrella, A.: Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. Expert Systems with Applications **82** (2017). DOI 10.1016/j.eswa.2017.03.047. URL `http://dx.doi.org/10.1016/j.eswa.2017.03.047`
61. de Leoni, M., Marrella, A., Russo, A.: Process-aware Information Systems for Emergency Management. In: Int. Workshop on Emergency Management through Service Oriented Architectures (EMSOA) co-located with the ServiceWave 2010 Conference, pp. 50–58. Springer Berlin Heidelberg (2010). DOI 10.1007/978-3-642-22760-8_5. URL `https://doi.org/10.1007/978-3-642-22760-8_5`
62. Lipovetzky, N., Geffner, H.: Searching for Plans with Carefully Designed Probes. In: 21st International Conference on Automated Planning and Scheduling (ICAPS '11), pp. 154–161 (2011)
63. Lipovetzky, N., Geffner, H.: Best-First Width Search: Exploration and Exploitation in Classical Planning. In: Thirty-First AAAI Conf. on Artificial Intelligence (AAAI-17), pp. 3590–3596 (2017)
64. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive Monitoring of Business Processes. In: 26th Int. Conference of Advanced Information Systems Engineering (CAiSE'14). Springer International Publishing (2014). DOI 10.1007/978-3-319-07881-6_31
65. Maggi, F.M., Marrella, A., Capezzuto, G., Cervantes, A.A.: Explaining Non-Compliance of Business Process Models through Automated Planning. In: Proceedings of the 16th International Conference on Service-Oriented Computing ICSOC, 2018, Hanghzou, Zhejang, China, November 12-15, 2018
66. Marrella, A.: What Automated Planning Can Do for Business Process Management. In: Business Process Management Workshops, pp. 7–19. Springer International Publishing, Cham (2018). DOI 10.1007/978-3-319-74030-0_1
67. Marrella, A., Lesperance, Y.: Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms. In: 14th Int. Conference on Business Process Modeling, Development and Support (BPMDS'13), pp. 277–291. Springer Berlin Heidelberg (2013). DOI 10.1007/978-3-642-38484-4_20. URL `https://doi.org/10.1007/978-3-642-38484-4_20`

68. Marrella, A., Lespérance, Y.: Towards a goal-oriented framework for the automatic synthesis of underspecified activities in dynamic processes. In: IEEE 6th Int. Conf. on Service-Oriented Computing and Applications (SOCA'13), pp. 361–365. IEEE (2013). DOI 10.1109/SOCA.2013.43. URL https://doi.org/10.1109/SOCA.2013.43
69. Marrella, A., Lesperance, Y.: A planning approach to the automated synthesis of template-based process models. Service Oriented Computing and Applications (2017). DOI 10.1007/s11761-017-0215-z. URL https://doi.org/10.1007/s11761-017-0215-z
70. Marrella, A., Mecella, M.: Continuous Planning for Solving Business Process Adaptivity. In: 12th Int. Conference on Business Process Modeling, Development and Support (BPMDS'11), pp. 118–132. Springer Berlin Heidelberg (2011). DOI 10.1007/978-3-642-21759-3_9. URL https://doi.org/10.1007/978-3-642-21759-3_9
71. Marrella, A., Mecella, M., Russo, A.: Collaboration On-the-field : Suggestions and Beyond. In: 8th Int. Conference on Information Systems for Crisis Response and Management (ISCRAM 2011) (2011)
72. Marrella, A., Mecella, M., Russo, A.: Featuring Automatic Adaptivity through Workflow Enactment and Planning. In: 7th Int. Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2011). IEEE (2011). DOI 10.4108/icst.collaboratecom.2011.247096. URL http://ieeexplore.ieee.org/abstract/document/6144823/
73. Marrella, A., Mecella, M., Sardina, S.: SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning. In: 14th Int. Conference on Principles of Knowledge Representation and Reasoning (KR'14). AAAI Press (2014). URL https://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7991
74. Marrella, A., Mecella, M., Sardina, S.: Intelligent Process Adaptation in the SmartPM System. ACM Trans. Intell. Syst. Technol. **8**(2) (2016). DOI 10.1145/2948071. URL http://doi.acm.org/10.1145/2948071
75. Marrella, A., Mecella, M., Sardiña, S.: Supporting adaptiveness of cyber-physical processes through action-based formalisms. AI Communications **31**(1), 47–74 (2018). DOI 10.3233/AIC-170748. URL http://dx.doi.org/10.3233/AIC-170748
76. Marrella, A., Russo, A., Mecella, M.: Planlets: Automatically Recovering Dynamic Processes in YAWL. In: 20th Int. Conference on Cooperative Information Systems (CoopIS 2012), On the Move to Meaningful Internet Systems. Springer Berlin Heidelberg (2012). DOI 10.1007/978-3-642-33606-5_17. URL http://dx.doi.org/10.1007/978-3-642-33606-5_17
77. Martínez, E., Lespérance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04). Workshop on Planning and Scheduling for Web and Grid Services, pp. 62–69 (2004)
78. McCluskey, T.L., Porteous, J.M.: Engineering and compiling planning domain models to promote validity and efficiency. Artificial Intelligence **95**(1), 1–65 (1997)
79. McDermott, D., et al.: PDDL—The Planning Domain Definition Language. Tech. Rep. DCS TR-1165, Yale Center for Computational Vision and Control (1998)
80. McDermott, D.V.: Estimated-Regression Planning for Interactions with Web Services. In: AIPS, vol. 2, pp. 204–211 (2002)
81. McIlraith, S., Son, T.C.: Adapting golog for composition of semantic web services. The Eighth Int. Conference on Principles of Knowledge Representation and Reasoning (KR'02) **2**, 482–493 (2002)
82. Meyer, S., Ruppen, A., Magerkurth, C.: Internet of things-aware process modeling: integrating iot devices as business process resources. In: International conference on advanced information systems engineering, pp. 84–98. Springer (2013). DOI 10.1007/978-3-642-38709-8_6
83. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
84. Myers, K.L., Berry, P.M.: Workflow Management Systems: An AI Perspective. AIC-SRI report (1998)
85. Object Management Group: Case Management Model and Notation, Version 1.0 (2014)
86. Peer, J.: A pop-based replanning agent for automatic web service composition. In: European Semantic Web Conference, pp. 47–61. Springer (2005)
87. Pesic, M., Schonenberg, H., van der Aalst, W.: DECLARE: Full Support for Loosely-Structured Processes. In: 11th IEEE Int. Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 287–300 (2007)
88. Petrick, R.P., Bacchus, F.: Extending the Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In: 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), pp. 2–11 (2004)
89. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated Synthesis of Composite BPEL4WS Web Services. In: IEEE Int. Conference on Web Services (ICWS'05). IEEE Computer Society (2005)
90. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of executable web service compositions from BPEL4WS processes. In: 14th Int. World Wide Web Conference (WWW'05). ACM (2005)
91. R-Moreno, M.D., Borrajo, D., Cesta, A., Oddi, A.: Integrating Planning and Scheduling in Workflow Domains. Exp. Syst. with App.: An Int. J. **33**(2) (2007)
92. R-Moreno, M.D., Kearney, P.: Integrating AI Planning Techniques with Workflow Management System. Knowledge-Based Systems **15**(5-6) (2002). DOI 10.1016/S0950-7051(01)00167-8
93. Ramırez, M., Geffner, H.: Goal recognition over pomdps: Inferring the intention of a pomdp agent. In: Twenty-second Int. Joint Conference on Artificial Intelligence (IJCAI'11), pp. 2009–2014 (2011)
94. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012). DOI 10.1007/978-3-642-30409-5
95. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research **39**, 127–177 (2010)
96. Rintanen, J.: Complexity of Planning with Partial Observability. In: 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), pp. 345–354 (2004)
97. Russell, S.J., Norvig, P., Canny, J.F., Malik, J.M., Edwards, D.D.: Artificial intelligence: a modern approach, vol. 2. Prentice hall Upper Saddle River (2003)
98. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.: Process flexibility: A survey of contemporary approaches. In: Advances in enterprise engineering I, pp. 16–30. Springer (2008)
99. Schuschel, H., Weske, M.: Triggering Replanning in an Integrated Workflow Planning and Enactment System. In: ADBIS'04 (2004)

100. Seiger, R., Keller, C., Niebling, F., Schlegel, T.: Modelling complex and flexible processes for smart cyber-physical environments. Journal of Computational Science (2014). DOI 10.1016/j.jocs.2014.07.001
101. Shah, M., Chrpa, L., Jimoh, F., Kitchin, D., McCluskey, T., Parkinson, S., Vallati, M.: Knowledge engineering tools in planning: State-of-the-art and future challenges. Knowledge Engineering for Planning and Scheduling **53** (2013)
102. Sheshagiri, M., DesJardins, M., Finin, T.: A planner for composing services described in DAML-S. In: 13th Int. Conference on Automated Planning and Scheduling (ICAPS-03). Workshop on planning for web services (2003)
103. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web service composition via generic procedures and customizing user preferences. In: International Semantic Web Conference, pp. 597–611. Springer (2006)
104. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: International Semantic Web Conference, pp. 380–394. Springer (2004)
105. Valdes, M.: Intelligent continuous improvement, when BPM meets AI (2017)
106. Van De Belt, T.H., Engelen, L., Berben, S., Schoonhoven, L.: Definition of Health 2.0 and Medicine 2.0: a systematic review. Journal of medical Internet research **12**(2) (2010). DOI 10.2196/jmir.1350
107. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 2nd edn. Springer Science & Business Media (2012). DOI 10.1007/978-3-642-28616-2
108. Wilkins, D.E.: Practical planning: extending the classical AI planning paradigm. Elsevier (2014)