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

A Planning Approach to the Automated Synthesis of Template-Based Process Models

Andrea Marrella · Yves Lespérance

Received: date / Accepted: date

Abstract The design-time specification of flexible processes can be time-consuming and error-prone, due to the high number of tasks involved and their context-dependent nature. Such processes frequently suffer from potential interference among their constituents, since resources are usually shared by the process participants and it is difficult to foresee all the potential tasks interactions in advance. Concurrent tasks may not be independent from each other (e.g., they could operate on the same data at the same time), resulting in incorrect outcomes. To tackle these issues, we propose an approach for the automated synthesis of a library of template-based process models that achieve goals in dynamic and partially specified environments. The approach is based on a declarative problem definition and partial-order planning algorithms for template generation. The resulting templates guarantee sound concurrency in the execution of their activities and are reusable in a variety of partially specified contextual environments. As running example, a disaster response scenario is given. The approach is backed by a formal model and has been tested in experiments.

Keywords Flexible Processes, Template-based Process Models, Partial-Order Planning, Business Process Management

A. Marrella
Sapienza - Universitá di Roma, Italy
E-mail: marrella@diag.uniroma1.it

Y. Lespérance
York University, Toronto, Canada
E-mail: lesperan@cse.yorku.ca

1 Introduction

Over the last decade, organizations and companies have started adopting process management methodologies and tools, with the aim of increasing the level of automation support for their operational business processes. *Business Process Management* (BPM) has, therefore, become a leading research area in the broader field of information systems [2].

Conventional BPM solutions require us to pre-define a detailed *model* of the process which completely prescribes the process execution flow and captures every possible *process instance* to be executed at run-time through a *Process Management System* (PMS). These models, typically based on imperative process specification, allow for a complete predefined specification of process logic able to represent procedural process knowledge, capturing the tasks to be executed and their control-flow, as well as process-related data and organizational perspectives [24]. Examples of traditional business processes include insurance claim processing, order handling, administrative processes, etc.

In recent years, the maturity of process management methodologies and of PMSs has lead to the application of BPM approaches in new challenging *dynamic* and *knowledge-intensive* scenarios [69,18], such as domotics [33], healthcare [47], projects coordination [19] and emergency management [57]. In these working environments, most business functions involve collaborative features and semi-structured (or unstructured) procedures that do not have the same level of predictability as the routine structured work.

This has lead to the definition of a new class of *flexible processes* [69]. Processes are defined as "flexible" when people/agents carry them out with a fair degree of "uncertainty", where the uncertainty may depend on many factors, such as the high number of tasks to be represented, their unpredictable nature, or their dependency on the contextual scenario. In a flexible process, the sequence of tasks depends heavily on the specifics of the context (e.g., which resources are available and what particular options exist at the time), and it is often unpredictable how the process will unfold.

While there exist several approaches supporting the enactment and automated adaptation of flexible processes at run-time [69,29,52,54,57,55,9,56], the design-time specification of such processes can be difficult, time-consuming and error-prone. Flexible processes frequently suffer from potential interference among their constituents, since resources are usually shared by the process participants and it is difficult to foresee all the potential tasks interactions in advance. Moreover, the process designer often lacks the needed knowledge to anticipate and incorporate all potential alternatives into the process model at design-time, as well this knowledge can become obsolete as process instances are executed and the context evolves. As a consequence, flexible processes may include some underspecified activities whose exact definition is not yet known at design-time, and may not be known until the time that an instance of the process has started execution, due to their context-dependent nature. In the worst case, there is no pre-defined view of a flexible process.

To tackle this issue, in this paper we build upon our previous work [51] and present a planning-based approach to the automated synthesis of *template-based process models* starting from a representation of the contextual domain in which a flexible process is embedded in and from an extensive repertoire of tasks defined for such a context. We refer to the concept of *process templates* to emphasize that further refinements may be needed to actually implement a flexible process. A process template depicts the *best-practice procedure drawn up with whatever contextual information available at the time*; it describes a recommended control flow for the process that does not only work in a specific state of the world, but can be enacted in a range of states satisfying the context conditions.

Specifically, we advocate a modeling approach involving a declarative specification of process tasks. We annotate process tasks with *preconditions* and *effects* defined over the contextual data of the dynamic scenario, i.e., we consider tasks as single steps that consume input data and produce output data. Then, we propose to use *partial-order planning algorithms* (aka POP [81,62]) to dynamically generate process templates based on descriptions of the initial state of the world and of a goal condition to be achieved through the execution of the template. A state-of-the-art partial-order planner is fed with task descriptions, initial state and goal condition, and returns a process template that satisfies the specification. The use of POP algorithms and, in general, of automated planning techniques, guarantee some interesting properties in the construction of the template:

- *Contextual selection.* Tasks composing the template are contextually selected from a specific repository and partially ordered in a way consistent with the context conditions to ensure that the template's objectives are achieved.
- *Sound concurrency.* Concurrent activities of a process template are proven to be effectively *independent* one from another (i.e., concurrent tasks cannot affect the same data). This means more flexibility during process execution. At run-time, the most appropriate execution path can be selected from those allowed by the design-time process template definition, without the risk of interference between concurrent tasks.
- *Executability in partially specified environments.* Once synthesized, a template can be executed in several initial states, since it (usually) requires a fragment of the knowledge of the initial state to successfully achieve its objectives. We identify the *weakest preconditions* of process templates, and all the states satisfying such preconditions are good candidates for executing them.
- *Versatility.* Our approach allows to represent the problem of synthesizing a process template as a planning problem in the standard Planning Domain Definition Language [58] (PDDL [58]), which can be solved through off-the-shelf automated planners. Since PDDL is independent from the specific planning system employed, one can seamlessly integrate different state-of-the-art planners compliant with PDDL. The effort to integrate a different planner does not go beyond installing the new planner and loading the planning-problem formulation that is generated through our planning-based approach. The problem formulation remains unchanged when moving from one planner to the other.

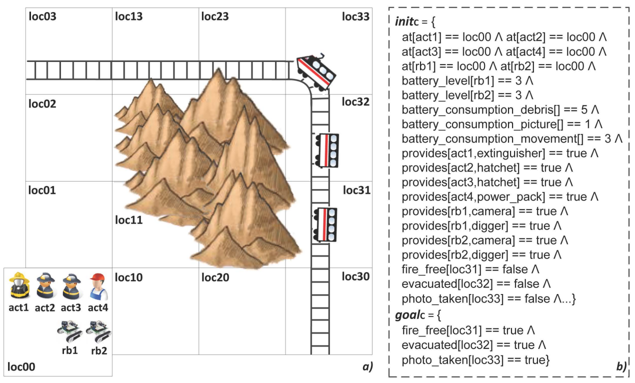
We exploit the idea behind POP of representing flexible plans that enable deferring decisions. Instead of committing prematurely to a complete, totally ordered sequence of actions, plans are represented as a partially ordered set, and only the required ordering decisions are recorded. A process template is generated on the basis of such a partially ordered set of activities, and we are able to identify what knowledge about the initial state is required for successful template execution. Moreover, we propose a methodology to build step-by-step a *library of process template specifications* and support efficient retrieval of appropriate templates from the library in partially specified environments.

The rest of the paper is organized as follows. Section 2 presents a running example describing a flexible process that comes from the emergency management domain. Section 3 introduces the required background and preliminaries for our work. Section 4 describes the basic ingredients for modeling the contextual properties of a dynamic environment, and introduces formally the concept of process template. Section 5 discusses in detail our approach to synthesize a library of process templates. A prototypical implementation of our approach is presented in Section 6, together with a set of experimental evaluation results that assess the practical applicability of our approach to design flexible processes. Section 7 discusses related work and Section 8 concludes by discussing benefits, limitations and future developments of the approach. Finally, an Appendix gives some technical details about the algorithms used for computing process templates.

2 Running Example

In the *emergency management* domain, teams of first responders act in disaster locations with the main purpose of achieving specific goals, including assisting potential victims and assessing and stabilizing the situation. First responders can benefit from the use of mobile devices and wireless communication technologies, as well as from the adoption of a process-oriented approach for team coordination. A response plan encoded as a business process and executed by a PMS deployed on mobile devices can help to coordinate the activities of first responders equipped with smartphones and supported by mobile networks. The use of business processes for coordinating first responders in disaster scenarios has been investigated in the European project WORKPAD[1] [12,39,40,11,53]. The following running example, which represents an excerpt of a real case study on emergency management investigated within the WORKPAD project, will be used to illustrate our approach to the synthesis of process templates.

Let us consider the disaster scenario described in Fig. 1(a). It concerns a train derailment and depicts a map of the area (as a 4x4 grid of locations) where the disaster happened. We suppose that the train is composed of a locomotive (located in *loc33*) and two coaches (located in *loc32* and *loc31* respectively). The goal of an incident response plan defined for such a context is to evacuate people from the coach located in



***init*c** = {
at[act1] == loc00 ∧ at[act2] == loc00 ∧
at[act3] == loc00 ∧ at[act4] == loc00 ∧
at[rb1] == loc00 ∧ at[rb2] == loc00 ∧
battery_level[rb1] == 3 ∧
battery_level[rb2] == 3 ∧
battery_consumption_debris[] == 5 ∧
battery_consumption_picture[] == 1 ∧
battery_consumption_movement[] == 3 ∧
provides[act1,extinguisher] == true ∧
provides[act2,hatchet] == true ∧
provides[act3,hatchet] == true ∧
provides[act4,power_pack] == true ∧
provides[rb1,camera] == true ∧
provides[rb1,digger] == true ∧
provides[rb2,camera] == true ∧
provides[rb2,digger] == true ∧
fire_free[loc31] == false ∧
evacuated[loc32] == false ∧
photo_taken[loc33] == false ∧...}
***goal*c** = {
fire_free[loc31] == true ∧
evacuated[loc32] == true ∧
photo_taken[loc33] == true}

Fig. 1: Area and context of the intervention.

loc32, to extinguish a fire in the coach in *loc31* and finally to take pictures for evaluating possible damages to the locomotive, located in *loc33*. Thus, a response team can be sent to the derailment scene. The team is composed of four first responders (in the remainder, we refer to them as *actors*) and two robots, initially located in *loc00*. We assume that actors are equipped with mobile devices (for picking up and executing tasks) and provide specific capabilities. For example, actor *act1* is able to extinguish fires, while *act2* and *act3* can evacuate people from train coaches. The two robots, instead, may take pictures and remove debris from specific locations. Each robot has a battery and each action consumes a given amount of battery charge. When the battery of a robot is discharged, actor *act4* can charge it. Fig. 1(b) summarizes the above.

The definition of an incident response plan as a business process involves a dynamically selected set of activities to be executed on the field by the first responders. Since the process may be different every time it is defined because it strictly depends on the actual contextual information (the positions of actors/robots, the battery level of robots, etc.), it is unrealistic to assume that the process designer can pre-define all the possible process models for dealing with this kind of intervention and environment (apparently simple). Moreover, if contextual data describing the environment are known, the synthesis of a process dealing with such an environment is not straightforward, as the correctness of the process is constrained by the values (or combination of values) of contextual data. A simple approach to solving our problem is to build a process as a sequence of activities, e.g., the sequence of actions shown in Fig. 2 (solid arrows represent ordering constraints between actions). The process in Fig. 2 instructs actor *act1* to reach *loc31* to extinguish fire. Then, after the battery of robot *rb2* has been recharged by *act4*, it can move in *loc32* for removing debris and the actor *act2*

[1] `http://www.diag.uniroma1.it/~workpad/`

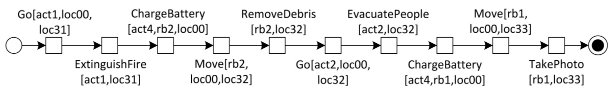


Fig. 2: A process dealing with the scenario of Fig. 1.

can start to evacuate people in that location. Finally, the battery of robot *rb1* is recharged as well, and it can move into *loc33* for taking pictures.

However, this solution is highly inefficient, as many actions are independent, and they could be executed concurrently to reduce intervention time; e.g., a robot could take pictures in parallel with the extinguishing of the fire in *loc31*. But, at the same time, a process designer may find it difficult to organize activities for concurrent execution, since each action, for its executability, depends on the values of contextual data (e.g., a robot needs enough battery charge for moving into a location and taking pictures or removing debris). Also dependencies between actions play a key role in the definition of the process model (e.g., in order to evacuate people at *loc32*, a robot must have removed the debris beforehand). Finally, a process designer tends to represent more contextual information than that strictly needed for defining a process. For example, the process in Fig. 2 does not involve actor *act3*, meaning that any information concerning *act3* (e.g., its capabilities, its location, etc.) is not required for synthesizing and executing the process.

3 Background

In this section, we present the required background and preliminaries for our work. Specifically, in Section 3.1 we discuss the main characteristics underlying a flexible process and describe some high-level requirements that have to be fulfilled for its design, while in Section 3.2 we introduce the basic notions on partial-order planning necessary to understand the rest of the paper

3.1 Characteristics and Requirements for Designing Flexible Processes

Over the last decade, BPM research has been expanded in many emerging directions, to support process mining [3], service-oriented computing [64], case management [75], cognitive computing [37], knowledge-intensive processes [18] and flexible processes [69].

The latter may be considered as specific cases of knowledge-intensive processes. According to [18], where they are called "dynamic processes", flexible processes are seen as "*fragments of larger unstructured processes [...] whose tasks need to be dynamically selected (or generated) at run-time*". In addition, according to [69], these processes are *implicitly described* in text or other forms of abstract procedures rather than explicitly modeled, i.e., they are often described directly by their process instances, by making the separation between process model and process instance largely blurred or non-existent.

In Section 2 we have presented an example of a flexible process coming from the emergency management domain. Starting from the experience gained in the area and lessons learned from the WORKPAD project and from the extensive analysis on flexible processes performed in [69, 18], in the following we present the main characteristics underlying a flexible process to be enacted in a dynamic real-world scenario:

– [C1] *Loosely structured behavior.* The set of possible activities may be known and predefined, but their execution ordering is not rigidly pre-definable as many possible execution alternatives are allowed.
– [C2] *Data- and Constraint-driven.* The main driver for the process progression is not (only) given by activity completions, but rather by the availability and evolution of data and knowledge objects (e.g., for representing contextual properties of a dynamic environment), which drive human decision making and directly influence the flow of process actions and events. Sometimes, data objects are used for defining constraints acting as eligibility criteria and preconditions for selecting the activity to be executed at run-time and the data sources to be exploited.
– [C3] *Unpredictable.* The exact activities and control flow in a flexible process depends on situation- and context-specific parameters, whose values may not be known a priori, may change during process execution, and may vary over different process instances. As a consequence, any detailed and fine-grained control-flow modeling attempt for flexible processes may become useless and inefficient.
– [C4] *Goal-oriented.* Usually, a flexible process evolves through a series of intermediate goals or milestones to be achieved. These goals may be known a priori and predefined, or, they may be gradually defined as a result of acquired knowledge and previously achieved goals. Each performed action and decision taken towards the achievement of a given goal has the effect of producing knowledge that will be exploited for supporting subsequent decisions and determining the next goals to be achieved and actions to be executed. Moreover, goals may be modified or even invalidated as a consequence of an event occurrence that has an impact on the process state and execution context.

The above characteristics are related to both the modeling and execution phases of a flexible process. However, in this paper, we mainly focus on the *modeling phase*. Specifically, we present an approach for the declarative modeling of contextual data and process tasks, and for automatically synthesizing a library of template-based process models ready to be enacted in contextual scenarios. Many of the aspects related to the execution of a flexible process are out of the scope of this paper, and there exists a vast research literature on the topic [69, 29, 52, 54, 57, 55, 9, 56].

To this end, starting from the above characteristics, we derive and define a set of 5 high-level requirements related to flexible process modeling:

- [R1] *Modeling contextual data.* A strong requirement for flexible processes is to *provide an information model* including all relevant data affecting the process and manipulated by it. Contextual data need be formalized and encoded in some form of *domain theory*, so as to define data objects to be considered as part of the process context and execution state. A process designer should also be allowed to *express conditions over process data*, if needed.
- [R2] *Representing data-driven activities.* A flexible process is characterized by activities whose enactment is related to the evolution of the information model. Such activities must be *enriched with declarative elements and constraints* (e.g., pre and post-conditions) defined on contextual data, stating what data may constrain the process execution or may be affected after an activity completion.
- [R3] *Providing synchronized access to shared data.* To prevent process tasks for possibly accessing and modifying the same data at the same time, it is required to provide some form of *synchronized access to shared data.*
- [R4] *Defining process goals.* The process designer should be able to *define process goals* on the basis of the contextual data defined for the process.
- [R5] *Modeling for reuse.* Even if a flexible process is often unpredictable, with the models of two process instances that may differ from one another, this does not exclude the possibility of predefining common fragments of the process models and *deriving process templates* to be reused, selected and adapted in a context-dependent way.

In Section 5, we show how our planning approach to synthesize process templates allows to properly address the above requirements. We recognize that there are other modeling approaches in the research literature that are able to (partially and fully) deal with the requirements: We investigate them in Section 7.

3.2 Partial-Order Planning

The *Automated Planning* field is a branch of Artificial Intelligence (AI) that aims to the realization of automated systems for the synthesis of organized sequences of real-world activities. Planning systems are problem-solving algorithms that operate on explicit representations of states and actions [62, 27]. The standard representation language for automated planners is known as the Planning Domain Definition Language (PDDL [58]); it allows us to formulate a problem PR through the description of the initial state of the world $init_{\mathsf{PR}}$ and of the desired goal condition $goal_{\mathsf{PR}}$. The domain PD of the planning problem mostly introduces relational predicates and a set of possible action definitions Ω. An action definition defines the conditions under which an action can be executed, called *preconditions*, and its *effects* on the state of the world. Each action $a \in \Omega$ has a preconditions list and an effects list to be applied on the state of the world, denoted respectively as Pre_a and Eff_a. A planner that works on such inputs generates a sequence of actions (the *plan*) that corresponds to a path from the initial state to a state meeting the goal condition.

In the literature, there exists a wide range of different planning techniques, that are characterized by the specific assumptions made. In this paper, we make use of *plan-space planning algorithms.* They differ from traditional *state-space planning algorithms*, that explore only strictly linear sequences of actions going from the initial state to the goal, by devising totally ordered plans. A plan space is an implicit directed graph whose nodes are *partially specified plans* and whose edges correspond to refinement operations intended to further complete a partial plan, i.e., to achieve an open goal or to remove a possible inconsistency. In order to demonstrate our approach, we focus on Partial-Order Planning (POP) algorithms [62, 81], a specific type of plan-space planning algorithms. POP algorithms take as input a planning problem defined in PDDL and search the space of partial plans without committing to a totally ordered sequence of actions. They usually work back from the goal, by adding actions to the plan to achieve each subgoal. A tutorial introduction to POP algorithms can be found in [81].

Basically, a *partial plan* is a tuple $\mathsf{P} = (A, O, CL)$, where $A \subseteq \Omega$ is a set of (ground) actions, O is a set of *ordering constraints* over A, and CL is a set of *causal links* over A. Ordering constraints O are of the form $a \prec b$, which is read as "a before b" and means that action a must be executed sometime before action b, but not necessarily immediately before. Causal links CL may be represented as $c \xrightarrow{p} d$, which is read as

"c achieves p for d" and means that p is an effect of action c and a precondition for action d. It also asserts that p must remain *true* from the time of action c to the time of action d. In other words, the plan may not be extended by adding a new action that conflicts with the causal link and makes p false between c and d. A precondition without a causal link requires further refinement to the plan to establish it, and is considered to be an *open condition* in the partial plan. Loosely speaking, the open conditions are preconditions of actions in the partial plan which have not yet been achieved in the current partial plan. More formally, an open condition is of the form (p, a), where $p \in Pre_a$ and $a \in A$, and there is no causal link $b \xrightarrow{p} a$ (where b is any action of the partial plan P).

A classical POP algorithm starts with a null partial plan P and keeps refining it until a solution plan is found. The null partial plan contains two dummy actions $a_0 \prec a_\infty$ where the preconditions of a_∞ correspond to the top level goals $goal_{\mathsf{PR}}$ of the problem, and the effects of a_0 correspond to the conditions in $init_{\mathsf{PR}}$. Intuitively, a refinement operation avoids adding to the partial plan any constraints that are not strictly needed for addressing the refinement objective. This is called the *least commitment principle* [81], and its advantage is that decisions about action ordering are postponed until a decision is forced; constraints are not added to a partial plan unless strictly needed, thus guaranteing flexibility in the execution of the plan and allowing actions to run concurrently. A *consistent* plan is defined as a plan with no cycles in the ordering constraints and no conflicts with the causal links. A consistent plan with no open conditions is a *solution* [81].

4 Process Templates

Our approach for the generation of a process template requires us to explicitly model the contextual knowledge in which the flexible process is embedded through some declarative rules (some pre-defined at design-time, some known just before the synthesis of the template) and logical constraints expressed in terms of task preconditions and effects. This information is given as input to an external partial-order planner that will be in charge of building a *process template*, i.e., a graph of activities reflecting the flexible process required for solving the specific contextual problem.

The synthesis of a flexible process requires a tight integration of process activities and contextual data in which the process is embedded in. The context is represented in the form of a *Domain Theory* D, that captures a set of tasks $t_i \in \mathsf{T}$ (with $i \in 1..n$) and supporting information, such as the people/agents that may be involved in performing the process (roles or participants), the data and so forth. Tasks are collected in a specific repository, and each task can be considered as a single step that consumes input data and produces output data. Data are represented through some ground 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 *data objects*, defined over some *data types*. In short, a data object depicts an entity of interest; for example, in our scenario we need to define data objects for representing participants (e.g., data type $Participant = \{act1, act2, act3, act4, rb1, rb2\}$), capabilities (e.g., data type $Capability = \{extinguisher, movement, \ldots hatchet\}$) and locations in the area (e.g., data type $Location = \{loc00, loc10, \ldots loc33\}$). Each tuple y_j may contain one or more data objects belonging to different data types. The domain $dom(v_j[y_j])$ over which a term is interpreted can be of various types:

- *Boolean*: $dom(v_j[y_j]) = \{true, false\}$;
- *Integer*: $dom(v_j[y_j]) = \mathbb{Z}$;
- *Functional*: the domain contains a fixed number of data objects of a designated type.

Terms can be used to express properties of domain objects (and relations over objects), and argument types of a term (taken from the set of data types previously defined) represent the finite domains over which the term is interpreted. In our example, we may need boolean terms for expressing the presence of a fire in a location (e.g., $fire\_free[loc : Location] = (bool : Boolean)$), integer terms for representing the battery charge level of each robot (e.g., $battery\_level[prt : Participant] \in \mathbb{Z}$) or functional terms for recording the position of each actor in the area (e.g., $at[prt : Participant] = (loc : Location)$). Moreover, since each task has to be assigned to a participant that provides all of the skills required for executing that task, there is the need to consider the participants "capabilities". This can be done through a boolean term $provides[prt : Participant, cap : Capability]$ that is *true* if the capability *cap* is provided by *prt* and *false* otherwise.

Each task is annotated with *preconditions* and *effects*. Preconditions are logical constraints defined as a conjunction of atomic terms, and they can be used to constrain the task assignment and must be satisfied before the task is applied, while effects establish the outcome of a task after its execution. Note that our approach treats each task as a "black box" and no assumption is made about its internal behavior (we consider the task execution as an instantaneous activity).

This is not a limitation, since it corresponds to the traditional way of modelling tasks in the BPM world at *design-time*, where nothing is said on the internal behaviour of a task (cf. [1,59,24]).

Definition 1 *A* task $t[x] \in \mathsf{T}$ *consists of:*

- *the name of the action involved in the enactment of the task (it often coincides with the task itself);*
- *a tuple of data objects x as input parameters;*
- *a set of preconditions Pre_t, represented as the conjunction of k atomic conditions defined over some specific terms, $Pre_t = \bigwedge_{l \in 1..k} pre_{t_l}$. Each pre_{t_l} can be represented as $\{v_j[y_j]$ **op expr**$\}$, where:*
  - *$v_j[y_j] \in \mathsf{V}$ is an atomic term, with $y_j \subseteq x$, i.e., admissible data objects for y_j need to be defined as task input parameters;*
  - *An **expr** can be a boolean value (if v_j is a boolean term); an input parameter identified by a data object (if v_j is a functional term); an integer number or an expression involving integer numbers and/or terms, combined with the arithmetic operators {+,-} (if v_j is a integer term);*
  - ***op** $\in \{<, >, ==, \leq, \geq\}$ is a relational operator. The condition **op** can be expressed as the equality (==) between boolean terms or functional terms and an admissible **expr**. On the contrary, if v_j is a integer term, it is possible to define the **op** condition as an expression that make use of relational binary comparison operators ($<, >, ==, \leq, \geq$) and involve integer numbers and/or integer terms in the **expr** field.*
- *a set of deterministic effects Eff_t, represented as the conjunction of h atomic conditions defined over some specific terms, $Eff_t = \bigwedge_{l \in 1..h} eff_{t_l}$. Each eff_{t_l} (with $l \in 1..h$) can be represented as $\{v_j[y_j]$ **op expr**$\}$, where:*
  - *$v_j[y_j] \in \mathsf{V}$ and **expr** are defined as for preconditions.*
  - ***op** $\in$ {=,+=,-=} is used for assigning (=) to a term a value consistent with the **expr** field or for incrementing (+ =) or decrementing (− =) an integer term by that value.*

Note that if no preconditions are specified, then the task is always executable. The use of classical partial-order planning techniques for synthesizing process templates imposes some limitation in the expressiveness of the language used for defining the domain theory D. Specifically, negative preconditions are not admitted (e.g., the use of the NOT operator is forbidden and all the atomic conditions that require to evaluate if a boolean term is equal to *false* will be ignored) and we assume that all effects are deterministic.

For example, let us consider the complete specification of the task *Go*:

```
<task>
  <name>Go</name>
  <parameters>
        <arg>actor - Participant</arg>
        <arg>from - Location</arg>
        <arg>to - Location</arg>
  </parameters>
  <precondition>at[actor] == from AND
                provides[actor,movement] == true
  </precondition>
  <effect>at[actor] = to</effect>
</task>
```

It involves two input parameters $from$ and *to* of type *Location*, representing the starting and arrival locations, and a parameter *actor* of type *Participant*, representing the first responder that will execute the task. An instance of *Go* can be executed only if *actor* is currently at the starting location $from$ and provides the required capabilities for executing the task. As a consequence of task execution, the actor moves from the starting to the arrival location, and this is reflected by assigning to the term *at*[*actor*] the value *to* in the effect.

Modeling a business process involves representing how a business pursues its objectives/goals. The goal may vary depending on the specific *Process Case* C to be handled. A case C reflects an instantiation of the domain theory D with a starting condition $init_\mathsf{C}$ and a goal condition $goal_\mathsf{C}$. Both conditions are conjunctions of atomic terms. We do not assume complete information about $init_\mathsf{C}$; this means we allow a process designer to instantiate only the atomic terms necessary for representing what is known about the initial state, i.e., $init_\mathsf{C} = \{v_1[y_1] == val_1 \wedge ... \wedge v_j[y_j] == val_j\}$, where val_j (with $j \in 1..m$) represents the j-th value assigned to the j-th atomic term. Fig. 1(b) shows a portion of $init_\mathsf{C}$ concerning the scenario depicted in Fig. 1(a). The goal is a condition represented as a conjunction of some specific terms we want to make true through the execution of the process. For example, in the scenario shown in Section 2, the goal has to be represented as : $goal_\mathsf{C} = \{fire\_free[loc31] == true \wedge evacuated[loc32] == true \wedge photo\_taken[loc33] == true\}$. The syntax of goal conditions is the same as for tasks preconditions.

A state is a complete assignment of values to atomic terms in V. Given a case C, an intermediate state $state_{\mathsf{C}_i}$ is the result of i tasks performed so far, and atomic terms in V may be thought of as "properties" of the world whose values may vary across states.

Definition 2 *A task t can be performed in a given $state_{\mathsf{C}_i}$ (and in this case we say that t is **executable***

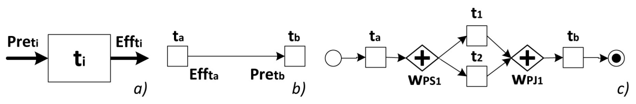


Fig. 3: Task anatomy (a), causality (b) and concurrency (c) in a process model.

in $state_{\mathsf{C}_i}$*) iff* $state_{\mathsf{C}_i} \vdash Pre_t$*, i.e.* $state_{\mathsf{C}_i}$ ***satisfies*** *the preconditions* Pre_t *for the task* t*.*

Moreover, if executed, the effects Eff_t of t modify some atomic terms in V and change $state_{\mathsf{C}_i}$ into a new state $state_{\mathsf{C}_{i+1}} = update(state_{\mathsf{C}_i}, Eff_t)$. The *update* function returns the new state obtained by applying effects Eff_t on the current state $state_{\mathsf{C}_i}$.

Starting from a domain theory D, a *Process Template* captures a partially ordered set of tasks, whose successful execution (i.e., without exceptions) leads from $init_{\mathsf{C}}$ to $goal_{\mathsf{C}}$. Formally, we define a template as a directed graph consisting of tasks, gateways, events and transitions between them.

Definition 3 *Given a domain theory* D*, a set of tasks* T *and a case* C*, a Process Template* PT *is a tuple (N,L) where:*

- $N = T \cup E \cup W$ *is a finite set of nodes, such that :*
  - T *is a set of tasks instances, i.e., occurrences of a specific task* $t \in \mathsf{T}$ *in the range of the process template;*
  - E *is a finite set of events, that consists of a single start event* $\bigcirc$ *and a single end event* $\odot$*;*
  - $W = W_{PS} \cup W_{PJ}$ *is a finite set of parallel gateways, represented in the control flow with the* $\diamond$ *shape with a "plus" marker inside.*
- $L = L_T \cup L_E \cup L_{W_{PS}} \cup L_{W_{PJ}}$ *is a finite set of transitions connecting events, task instances and gateways:*
  - $L_T : T \to (T \cup W_{PS} \cup W_{PJ} \cup \odot)$
  - $L_E : \bigcirc \to (T \cup W_{PS} \cup \odot)$
  - $L_{W_{PS}} : W_{PS} \to 2^T$
  - $L_{W_{PJ}} : W_{PJ} \to (T \cup W_{PS} \cup \odot)$

The constructs used for defining a template are essentially a subset of the BPMN notation [82], a graphical language designed to specify a process in a standardized way. Intuitively, an execution of the process starts at $\bigcirc$ and ends at $\odot$; a *task* is an atomic activity executed by the process, cf. Fig. 3(a); *parallel splits* W_{PS} open parallel parts of the process, whereas *parallel joins* W_{PJ} reunite parallel branches. *Transitions* are binary relations describing in which order the flow objects (tasks, events and gateways) have to be performed, and determine the *control flow* of the template. A transition $l \in L$ is represented as $p \to q$, where $(p, q) \in N$. This represents the fact that there is a transition from the flow object p to the flow object q. For $n \in N$, $IN(n)/OUT(n)$ denotes the set of incoming/outgoing transitions of n, with the following restrictions :

- Only one outgoing/incoming flow may be associated with $\bigcirc$ and $\odot$ respectively, i.e., $IN(\bigcirc) = 0$, $OUT(\bigcirc) = 1$, $IN(\odot) = 1$, $OUT(\odot) = 0$
- Each parallel split $w_{PS} \in W_{PS}$ accepts one incoming flow and more outgoing flows, i.e., $IN(W_{PS}) = 1$, $OUT(W_{PS}) > 1$
- Each parallel join $w_{PJ} \in W_{PJ}$ accepts more incoming flows and one outgoing flow, i.e., $IN(W_{PJ}) > 1$, $OUT(W_{PJ}) = 1$;
- Every task $t \in T$ is connected exactly to one incoming/outgoing flow, i.e., $IN(t) = 1$, $OUT(t) = 1$.

In Fig. 3(b) we have a relation of *causality* between tasks t_a and t_b, stating that t_a must take place before t_b happens as t_a achieves some of t_b's preconditions. An important feature provided by a process template is *concurrency*, i.e., several tasks can occur concurrently. In Fig. 3(c) an example of concurrency between t_1 and t_2 is shown. In order to represent two or more concurrent tasks in a template, the process designer makes use of the parallel gateways, that indicate points of the template in which tasks can be carried out concurrently. A parallel gateway may act as a *divergence element* (parallel split W_{PS}) or *convergent element* (parallel join W_{PJ}). As a point of divergence, the diamond shape is used when many tasks have to be carried out at the same time and in any order, which indicates that all transitions that exit this shape will be enabled together. As a point of convergence, the diamond shape is used to synchronize paths that exit a divergence element. This means that a process template is a *graph of tasks* (i.e., not a sequence) that imposes a *partial order* on their execution. A *linearization* of a process template is any linear ordering of the tasks that is consistent with the ordering constraints of the template itself [28]; i.e., a linearization of a partial order is a potential *execution path* of the template from the start event $\bigcirc$ to the end event $\odot$. For example, the template in Fig. 3(c) has two possible execution paths $r_1 = [\bigcirc; t_a; t_1; t_2; t_b; \odot]$ and $r_2 = [\bigcirc; t_a; t_2; t_1; t_b; \odot]$.

Definition 4 *Given a process template* PT *and an initial state* $state_{\mathsf{C}_0} \vdash init_{\mathsf{C}}$*, a state* $state_{\mathsf{C}_i}$ *is said to be* ***reachable*** *with respect to* PT *iff there exists an execution path* $r = [\bigcirc; t_1; t_2; ...t_k; \odot]$ *of* PT *and a task* t_i *(with* $i \in 1..k$*) such that* $state_{\mathsf{C}_i} = update(update(\ldots update(state_{\mathsf{C}_0}, Eff_{t_1}) \ldots, Eff_{t_{i-1}}), Eff_{t_i})$*.*

Therefore, $state_{\mathsf{C}_i}$ is a state that may be reached along some possible execution of PT in $init_{\mathsf{C}}$.

Definition 5 *A task t_1* ***affects*** *the execution of a task t_2, written $t_1 \triangleright t_2$, iff there exists a reachable state $state_{\mathsf{C}_i}$ of* PT *(for some initial state $state_{\mathsf{C}_0}$) such that:*

– $state_{\mathsf{C}_i} \vdash Pre_{t_2}$
– $update(state_{\mathsf{C}_i}, Eff_{t_1}) \nvdash Pre_{t_2}$

This means that Eff_{t_1} modify some terms in V that are required as preconditions for making t_2 executable in $state_{\mathsf{C}_i}$.

Definition 6 *Given a process template* PT*, a case* C *and an initial state $state_{\mathsf{C}_0} \vdash init_{\mathsf{C}}$, an execution path $r = [\bigcirc; t_1; t_2; ...t_k; \odot]$ (where $k = |T|$) of* PT *is said to be* ***executable*** *in* C *iff:*

– $state_{\mathsf{C}_0} \vdash Pre_{t_1}$
– *for* $1 \leq i \leq k-1$, $update(state_{\mathsf{C}_{i-1}}, Eff_{t_i}) \vdash Pre_{t_{i+1}}$
– $update(state_{\mathsf{C}_{k-1}}, Eff_{t_k}) = state_{\mathsf{C}_k} \vdash goal_{\mathsf{C}}$

Definition 7 *A process template* PT *is said to be* ***executable*** *in a case* C *iff any execution path of* PT *is executable in* C.

The concept of execution path of a template helps in defining formally the *independence* property between concurrent tasks:

Definition 8 *Given a process template* PT*, a task t_x is said to be* ***concurrent*** *with a task t_z iff there exist two execution paths r_1 and r_2 of* PT *such that $r_1 = [\bigcirc; t_1; t_2; ...; t_x; ...; t_z; ...; \odot]$ and $r_2 = [\bigcirc; t_1; t_2; ...; t_z; ...; t_x; ...; \odot]$.*

A more interesting feature of concurrent tasks concerns their independence property:

Definition 9 *Two concurrent tasks t_1 and t_2 are said to be* ***independent****, written $t_1 \parallel t_2$, iff $t_1 \not\triangleright t_2$ and $t_2 \not\triangleright t_1$; that is, t_1 does not affect t_2 and vice versa.*

5 An Approach to Synthesize a Library of Process Templates

This section describes the approach we developed to address the requirements defined in Section 3.1. Our approach is focussed on the development and use of a *library* of *process templates.* These are reusable process models that achieve specified goals of interest in any initial state that satisfies the template's required preconditions. Specifically, we focus on the use of a POP-based tool that can synthesize complex concurrent process models, while ensuring that concurrent tasks never interfere (cf. R3). The process designer's role is to specify the domain and context in which the template may be executed (cf. R1), that is a repository of atomic tasks annotated with preconditions and effects defined over the contextual data (cf. R2), as well as the case to be handled (cf. R4).

The selection of tasks during the development of the repository is performed by having in mind that a task specification defines what must be done in order to *transform inputs from the domain theory into outputs that will affect the domain theory.* Specifically, a process designer specifies a task in a way that: *(i)* it can be potentially executed in some reachable state of the domain theory (cf. Definition 4 in Section 4); *(ii)* it will affect a non-empty set of atomic terms of the domain theory. While the definition of tasks without preconditions that are always executable in any reachable state is possible, the designer should avoid to specify tasks that provide no effects, since their execution would not contribute to the achievement of the goal condition.

Our POP-based tool can then be used to synthesize some candidate process models for the template. If the tool fails to generate a process model or the generated processes are of insufficient quality (e.g., they are too time consuming, unreliable, or lack concurrency), the designer can refine the domain theory and case to obtain better solutions. Once a satisfactory template has been obtained, it is added to the library. The POP-based tool automatically identifies the required preconditions for the template to achieve its goal, meaning the template can be reused whenever a case that matches the template's preconditions arises (cf. R5).

In a flexible process domain, there is a wide range of cases/contexts to handle. New cases often arise and the requirements for the system frequently evolve. The designer maintains the template library over time, in order to have templates that handle effectively most the cases that arise. The library also stores the templates specifications, i.e., their process domains, goals, and initial conditions/cases. New cases are often variants of existing cases and the designer will be able to adapt existing domain and case specifications to generate templates for the new cases using the tool.

In the following, we first describe thoroughly an architecture and methodology for developing such a library-based approach, and then we describe some interesting properties satisfied by the approach.

5.1 The General Framework

Our approach to the definition of process templates (cf. Fig. 4) requires a fundamental shift in how one thinks about modeling business processes. Instead of defining a process model "by hand", the process designer has to address her/his efforts to specifying the domain theory

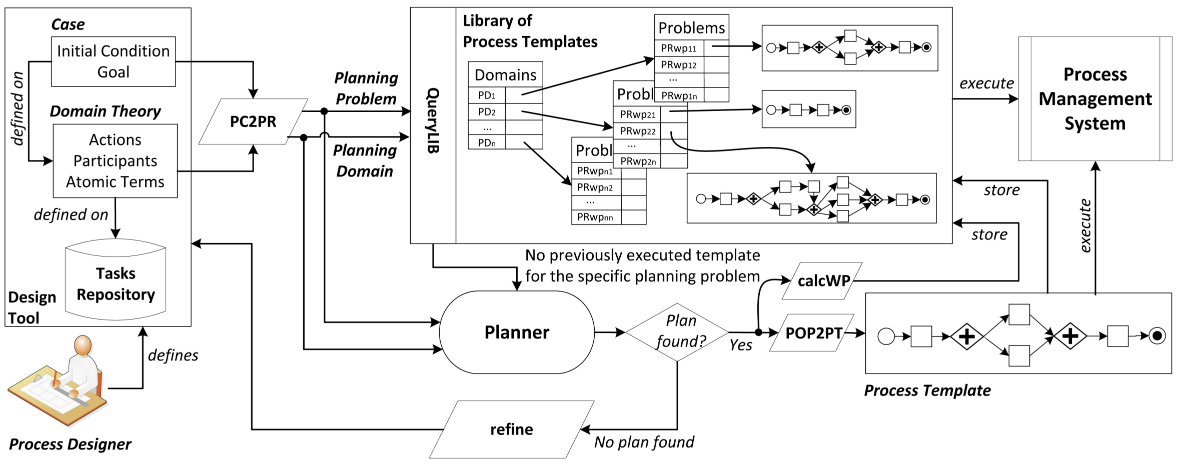


Fig. 4: Overview of the general approach.

D and the case C to be handled. In particular, s/he has to identify the starting condition $init_{\mathsf{C}}$, by instantiating only those atomic terms needed for depicting the context of interest. This means that $init_{\mathsf{C}}$ can be partially specified, i.e, not all terms need to be instantiated with some value. Also the goal condition $goal_{\mathsf{C}}$ is required, since it reflects the target state after having executed the template.

Example. *Let us consider the scenario depicted in Section 2, represented with a domain theory* D_1 *and a goal condition* $goal_{\mathsf{C}_1} = \{fire\_free[loc31]{==}\ true \wedge evacuated[loc32]{==}\ true \wedge photo\_taken[loc33]{==}\ true\}$. *Since the process designer may be interested in an emergency process that involves the fewest participants, s/he can start by modeling a starting condition* $init_{\mathsf{C}_1}$ *with information involving only actors act1 and act2 and the robot rb1, while terms involving act3, act4 and rb2 are not explicitly instantiated in* $init_{\mathsf{C}_1}$.

A specific module named PC2PR is in charge of converting the domain theory D and the case C just defined into the corresponding planning domain PD and planning problem PR specified in PDDL version 2.1[2] (cf. [26]). Basically, PC2PR implements a function:

$$f_{\mathsf{PC2PR}} : (\mathsf{D}, init_{\mathsf{C}}, goal_{\mathsf{C}}) \rightarrow (\mathsf{PD}, init_{\mathsf{PR}}, goal_{\mathsf{PR}}). \quad (1)$$

Since the use of classical partial-order algorithms for synthesizing the template requires the initial state of PR to be a complete state, we make the closed world assumption [72] and assume that every atomic term $v_j[y_j]$ that is not explicitly specified in $init_{\mathsf{C}}$ is assumed to be false (if $v_j[y_j]$ is a boolean term) or "not assigned" (if $v_j[y_j]$ is a integer or a functional term) in $init_{\mathsf{PR}}$. Technical details of the algorithm employed in PC2PR are shown in Appendix A.1.

At the heart of our approach lies a library of process templates built for specific planning domains and problems/cases. If library templates exist for the current values of PD and PR, we can retrieve an appropriate template and have it executed through an external PMS. However, if no template matches for the current values of PD and PR, we can invoke an external POP planner on these same inputs. The planner will try to synthesize a plan fulfilling the goal condition $goal_{\mathsf{PR}}$.

If the planner is unable to find a plan, this suggests that there are some missing elements in the definition of the domain theory D or in the case C. Hence, to address this, one can try to *refine* the case C and add information so that it becomes possible to generate a plan. There are many ways to strengthen a problem description, such as adding to the starting condition $init_{\mathsf{C}}$ some terms initially ignored (e.g., to specify the position of every participant), or adding new objects in D or new activities in T (e.g., if a task for extinguish fire is missing). Our approach assumes that one specifies the context step-by-step, and requires the process designer to contribute to the system.

Example. *If the planner is invoked with* $init_{\mathsf{PR}_1}$ *(devised by applying* f_{PC2PR} *on the triple* D_1, $init_{\mathsf{C}_1}$, $goal_{\mathsf{C}_1}$*), it will not be able to find any plan for the specific problem. This is because rb1 does not have enough battery charge for moving, taking pictures and removing debris. The designer can try to add new information to the problem description by instantiating in* $init_{\mathsf{C}_1}$ *all those atomic terms related to actor act4, the only one able to charge robot batteries, and devise a new starting condition* $init_{\mathsf{C}_2}$ *(and, consequently, a new initial planning state* $init_{\mathsf{PR}_2}$*). A planner invoked with* $init_{\mathsf{PR}_2}$

[2] PDDL 2.1 enables the representation of realistic planning domains, which include operators with universally quantified effects and numeric fluents. However, our formalism does not currently handle conditional effects nor negative preconditions.

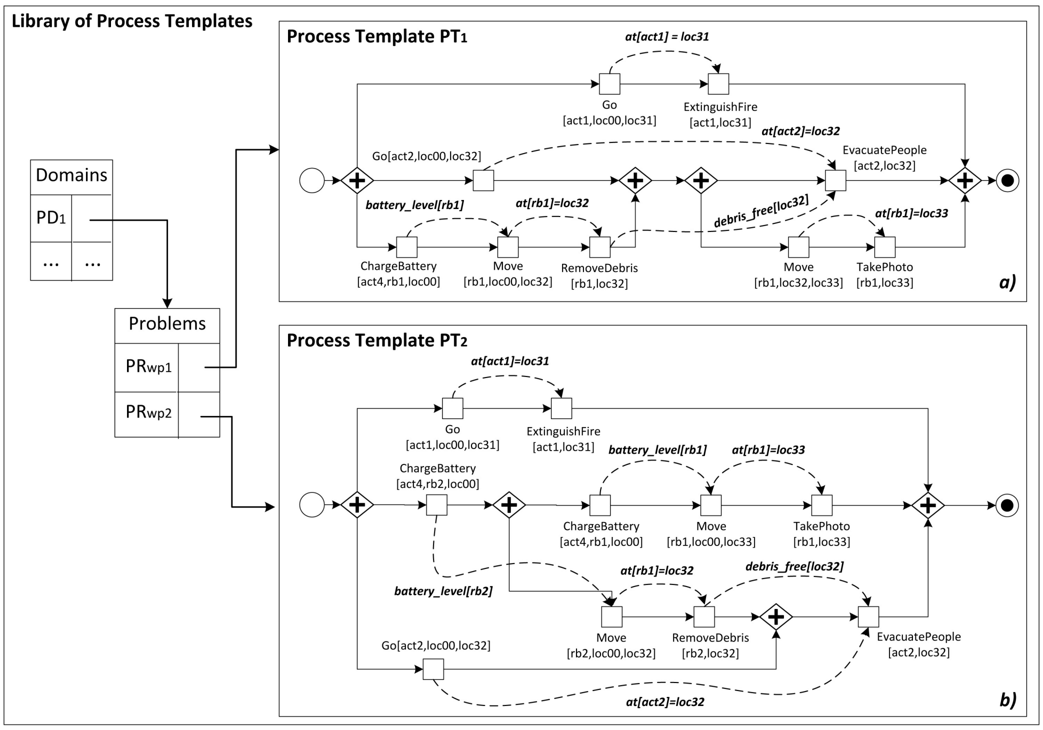


Fig. 5: Templates dealing with the scenario in Fig. 1.

is finally able to find a consistent plan P_1 *satisfying* $goal_{\mathsf{PR}_1}$.

When the POP planner is able to find a partially ordered plan P consistent with the actual contextual information, three further steps are required. First we need to translate the plan into a template PT that preserves the ordering constraints imposed by the plan. A **solution plan** is a three-tuple $\mathsf{P} = (A, O, CL)$ that specifies the causal relationships for the actions $a_i \in A$, but without specifying an exact order for executing them. Since the actions and the set of ordering constraints must be represented explicitly as nodes and transitions in the template, we developed a module POP2PT implementing a function:

$$f_{\mathtt{POP2PT}} : \mathsf{P} \rightarrow \mathsf{PT} \tag{2}$$

that takes as input P and converts it into a template PT. It works by first finding the immediate predecessors/successors of actions in the plan using the ordering constraints, and then constructing the desired plan template, inserting parallel splits (resp. join) gateways when an action has more than one immediate successor (resp. predecessor). A detailed description of the algorithm implemented in POP2PT is presented in Appendix A.2.

Example. *By applying* $f_{\mathtt{POP2PT}}$ *to* P_1*, we devise the template* PT_1 *in Fig. 5(a). Dashed arrows are causal links that imply an ordering constraint between pairs of tasks. For example, the ordering constraint between Go[act1,loc00,loc31] and ExtinguishFire[act1,loc31] is derived from the fact that Go has the effect at[act1]=loc31 that is needed by ExtinguishFire as precondition (i.e., act1 has to be located in loc31 for extinguish the fire in that location).*

Secondly, our approach *infers* the weakest preconditions w_{PT} about the initial state that are required for the template to achieve its goal. The module we use for inferring w_{PT} is called `calcWP` and works by analyzing the set of causal links CL computed by the POP planner, to see which logical facts f_k are involved in causal links that originate from the dummy start action a_0 and end in some $a_k \in A$. More formally:

$$\forall (cl_k, f_k, a_k) \; s.t. \; cl_k = (a_0 \xrightarrow{f_k} a_k) \in CL, then \; f_k \in w_{\mathsf{PT}}. \tag{3}$$

Observe that the effects of $a_0 \in A$ specify all atomic facts that are true in the initial state $init_{\mathsf{PR}}$. The initial facts that are actually required for the plan to be executable and achieve its goal are those that are involved in a causal link with another action in the plan, and we collect those in w_{PT} as specified in Equation 3 (the plan cannot depend on any negative facts as they cannot appear in either the goal or in action preconditions). Basically, w_{PT} is the conjunction of those facts strictly required for executing the plan P (and, consequently, the devised template PT), and is used for devising a new problem $\mathsf{PR}_{wp} = \{w_{\mathsf{PT}}, goal_{\mathsf{PR}_{wp}}\}$. We can then drop the closed world assumption. For any initial state that satisfies w_{PT}, the obtained process template PT will be executable and achieve the goal condition $goal_{\mathsf{PR}}$.

Example. *If we invoke* `calcWP` *on the causal links devised from* P_1 *(cf. the dashed arrows of template* PT_1 *in Fig. 5(a)), we may infer* w_{PT_1}*. They indicate that, for executing template* PT_1*, it is required to know the capabilities and the positions of act1, act2, act4 and rb1 (cf. the logical facts on top of the causal links, e.g.,* $at[act1] = loc31$*,* $at[act2] = loc32$*, etc.), the information about the battery level of rb1 (cf.* $batteryLevel[rb1]$*) and the status of the location loc31 (cf.* $debris\_free[loc32]$*). No other contextual information is required for a correct execution of the template; for example, any additional information about further actors and robots (their positions, battery level, etc.) can be neglected for the enactment of* PT_1*.*

Thirdly, after the process template PT has been synthesized starting from P, it can be stored in our library together with information about the planning domain PD and abstracted problem PR_{wp}. Specifically, for every different planning domain PD devised through our approach, there is a pointer to a list of different abstracted planning problems PR_{wp} used for obtaining consistent plans in previous executions of our tool, together with the devised process templates.

When a process designer defines a new domain theory D_{new} and a case C_{new}, the software module `QueryLIB` checks if the corresponding planning domain PD_{new} and problem PR_{new} (obtained by applying f_{PC2PR} to D_{new} and C_{new}) are already present in our library. If the library contains a planning domain PD_{lib} and an abstracted planning problem PR_{wp} (together with the associated template PT_{lib}) such that $\mathsf{PD}_{lib} \subseteq \mathsf{PD}_{new}$ and $goal_{\mathsf{PR}_{wp}} \vdash goal_{\mathsf{PR}_{new}}$ and with $init_{\mathsf{PR}_{new}} \vdash w_{\mathsf{PT}}$, then PT_{lib} is executable with respect to PR_{new} (and therefore with respect to C_{new}).

Since a library template PT_{lib} reflects an instantiation of a flexible process in a specific case (i.e., it is a single process instance ready to be enacted), there is the risk that other library templates defined on the same planning domain $\mathsf{PD}_{lib} \subseteq \mathsf{PD}_{new}$ and solving similar problems to PR_{new} are not considered for the selection. To tackle this issue, the `QueryLIB` module allows the process designer to "generalize" the search by setting some abstraction rules on specific data types defined in the domain theory. For example, if the process designer flags the data type *Participant* as "abstracted", the searching of an existing library template that satisfies PD_{new} and PR_{new} is performed in two steps. First, from each library template PT_{lib} indexed with $\mathsf{PD}_{lib} \subseteq \mathsf{PD}_{new}$, the `QueryLIB` module substitutes all the constants values representing the specific participants involved in task instances with as many variables (of type *Participant*) as the participants are. The same substitution is done for the participants included in the abstracted planning problem PR_{wp} associated to PT_{lib}. Then, during the search, the `QueryLIB` module applies a simple *substitution* mechanism of type {variable/data object} [73] for verifying if there exists a *valid binding* between the constant values representing the available participants in $init_{\mathsf{PR}_{new}}$ and the generic variables of type *Participant* in PT_{lib}. A valid binding is associated to a {variable/data object} binding list that applies to the abstracted planning problem PR_{wp} and satisfies the preconditions of task occurrences in PT_{lib}.

Example. *Let us consider the template* PT_1 *in Fig. 5(a), obtained with* PD_1 *and* PR_2*. Let us suppose now that the designer, on the same planning domain* PD_1*, has devised a new planning problem* PR_{2b} *(derived from a new case* C_{2b}*), with information about actor act3 rather than about actor act2. At a first glance,* PT_1 *does not match with the information contained in* $init_{\mathsf{PR}_{2b}}$*, since* PT_1 *requires the presence of actor act2 for executing the template. However, if the process designer decides to generalize the search task with respect to the data type Participant, every constant value of type Participant in* PT_1 *is substituted with a generic variable of type Participant. For example, the tasks Go[act2,loc00,loc32] and EvacuatePeople[act2,loc32] become Go[x,loc00,loc32] and EvacuatePeople[x,loc32], and the information that concern act2 in* PR_{wp} *are affected as well (e.g., provides[act2, hatchet] becomes provides[x,hatchet]). The* `QueryLIB` *module starts searching for a valid binding. The partial binding {..,x/act1,..} is not valid, since the term provides[act1,hatchet] is false in* $init_{\mathsf{PR}_{2b}}$*. A valid partial binding is instead {..,x/act3,..}, since the term provides[act3,hatchet] is true in* $init_{\mathsf{PR}_{2b}}$ *and the tasks Go[act3,loc00,loc32] and EvacuatePeople[act3,loc32] are executable in* PT_1 *(i.e., they satisfy the template's tasks preconditions).*

Despite the fact that a process template is executable "as is" in a specific contextual situation, when `QueryLIB` searches for an existing template in the library (i.e., for a valid binding with the current values of PD_{new} and PR_{new}), we can apply some abstraction rules that allow a library template to be used for generating several models matching the properties of different situations. This makes our templates reusable in a variety of different contexts, in which we don't have complete information about the initial state. At this point, the process designer may decide to execute through an external PMS the template PT_{lib} just found, or to refine D_{new} and C_{new} if PT_{lib} does not fit with the designer expectations.

Example. *Let us suppose that the template shown in Fig. 5(a) does not satisfy at all the process designer, since s/he could add one further robot rb2 to the scenario in order to increase the degree of parallelism in the tasks execution. It follows that a new starting condition* $init_{\mathsf{C}_3}$ *including also contextual information about rb2 can be defined. The associated initial planning state* $init_{\mathsf{PR}_3}$*, together with the original goal condition* $goal_{\mathsf{PR}_1}$ *and the planning domain* PD_1 *are first used for verifying if a matching synthesized template is already stored in the library. The library returns the template* PT_1 *shown in Fig. 5(a), since its weakest preconditions* w_{PT_1} *are satisfied by* $init_{\mathsf{PR}_3}$ *(i.e.,* $init_{\mathsf{PR}_3} \vdash w_{\mathsf{PT}_1}$*), and goal condition and planning domain are the same as before. Even if the template in Fig. 5(a) is executable with* $init_{\mathsf{PR}_3}$*, the designer may try to obtain another plan that would exploit the presence of the new robot rb2. This can be done by directly invoking the planner with the new information about the initial state. The planner can generate a new plan starting from* $init_{\mathsf{PR}_3}$*, and the associated template* PT_2 *is shown in Fig. 5(b).* PT_2 *requires the presence of one more robot (i.e., robot rb2) and more contextual information for being executed (so its weakest preconditions* w_{PT_2} *are stronger than* w_{PT_1}*), but it provides a higher degree of concurrency in the execution of its tasks. Notice also that the initial state associated to* PT_1 *is satisfied by the weakest preconditions* w_{PT_2}*. This means that* PT_1 *is executable in* w_{PT_2}*, and that the process designer can choose which template is best for her/his purposes: one with less concurrency in the tasks enactment but with the fewest participants (cf., Fig. 5(a)), or one with more concurrency but requiring more resources for being executed (cf., Fig. 5(b)).*

If several matching templates (for which the weakest preconditions hold and which reach the desired goal state) are retrieved from the library, the `QueryLIB` component categorizes them on the basis of some parameters useful for evaluating their quality. Specifically, each template is stored with information about *(i)* the number of different participants involved; *(ii)* the number of tasks in the control flow; *(iii)* the degree of parallelism provided (i.e., the maximum number of tasks that can be possibly executed concurrently at the same time) and *(iv)* the weakest preconditions w_{PT} associated to the template, that help to understand the amount of knowledge about the initial state that is required for executing the template itself. At this point, the process designer can choose which template provides the required quality on the basis of the current case to be dealt with. We do not currently discuss any mechanism to evaluate and quantify the *similarity* between different templates, but it is interesting to notice that the presence of annotated tasks and causal links describing the structure of a template allows potentially to convert a template in a *business process graph* [21] and to perform all the similarity metrics as described in [22] (i.e., node matching similarity, structural similarity and behavioral similarity).

5.2 Properties

A process template PT guarantees some interesting properties, such as the *executability* of the template with respect to the information available in the initial state, and the property of *sound concurrency*, meaning that concurrent activities of a template are proven to be independent from each other.

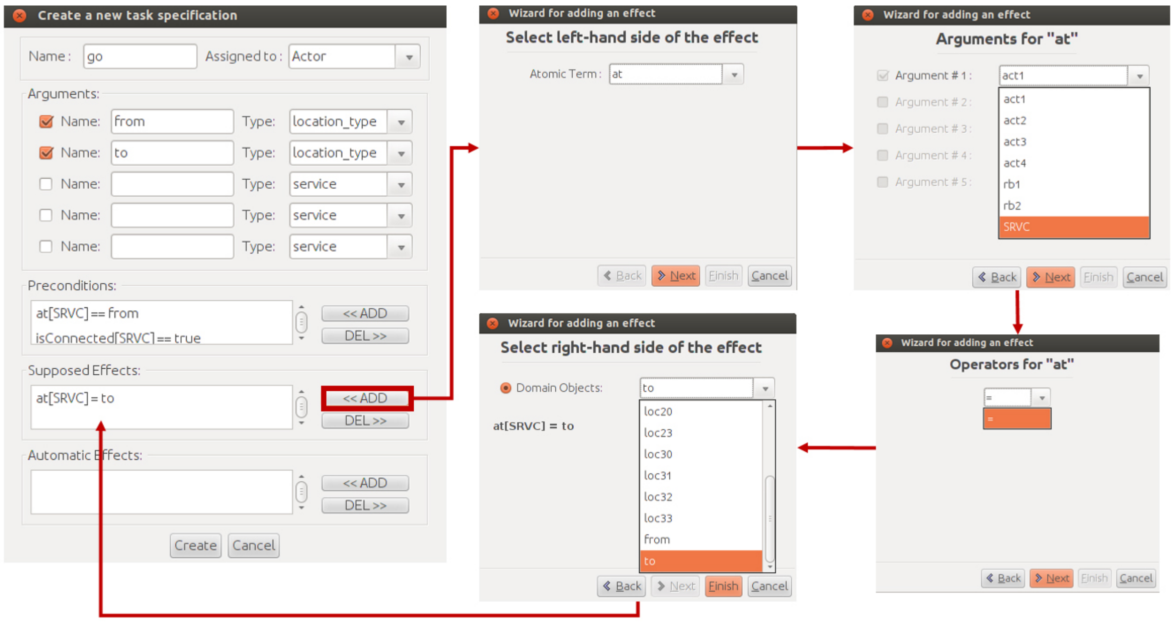
Theorem 1 *Given a solution plan* P*, a process template* PT *synthesized for* P *using our approach is* ***executable*** *for any process case* C *that satisfies the weakest preconditions* wp_{PT} *inferred from* P*.*

The proof is straightforward. By definition, a sound planner generates a *consistent* plan [81] that leads from an initial state to a goal. Since we represent the domain theory/case as a PDDL planning domain/problem, the planner synthesizes a plan (i.e., a process template) that is executable with respect to Definition 7.

A second property we can prove is *sound concurrency.* Even if in a process designed by following data and workflow patterns [23] the concurrent execution of two or more tasks should guarantee the consistency of data accessed by the concurrent tasks, in practice this is often not true. In fact, in complex environments there is not a clear correlation between a change in the context and corresponding process changes, making it difficult to design by hand a process where concurrent tasks are also independent. On the contrary, all concurrent tasks of a template built with our approach are proven to be *independent* from one another.

Theorem 2 *Given a process template* PT *synthesized with our approach, all concurrent tasks are* ***independent****.*

Proof By contradiction, let us suppose that a process template PT has two concurrent tasks t_1 and t_2 such that $t_1 \not\parallel t_2$. Hence, t_1 (or t_2) has some effect affecting the precondition of t_2 (or of t_1). This means that $t_1 \rhd t_2$ or $t_2 \rhd t_1$. Since PT has been synthesized as result of a POP planner, this dependency between t_1 and t_2 would


Create a new task specification

Name: go Assigned to: Actor

Arguments:

✓ Name:	from	Type:	location_type
✓ Name:	to	Type:	location_type
☐ Name:		Type:	service
☐ Name:		Type:	service
☐ Name:		Type:	service

Preconditions:
at[SRVC] == from
isConnected[SRVC] == true

Supposed Effects:
at[SRVC] = to

Automatic Effects:

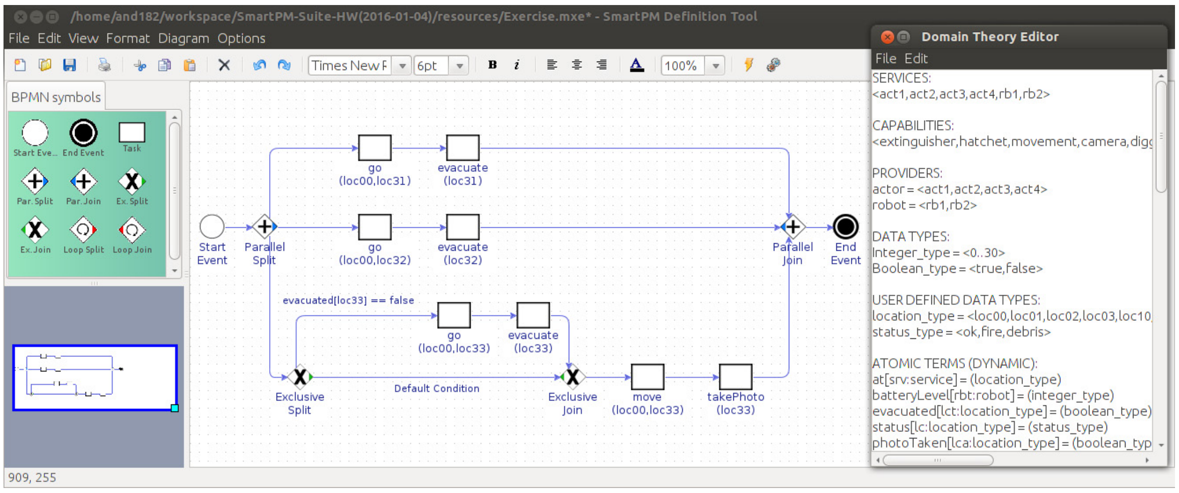
Wizard for adding an effect
Select left-hand side of the effect
Atomic Term: at

Wizard for adding an effect
Arguments for "at"
Argument # 1: act1
act1, act2, act3, act4, rb1, rb2, SRVC

Wizard for adding an effect
Operators for "at"
=

Wizard for adding an effect
Select right-hand side of the effect
Domain Objects: to
at[SRVC] = to
loc20, loc23, loc30, loc31, loc32, loc33, from, to

Fig. 6: The wizard-based editor to build tasks specifications.


/home/and182/workspace/SmartPM-Suite-HW(2016-01-04)/resources/Exercise.mxe* - SmartPM Definition Tool

Domain Theory Editor

SERVICES:
<act1,act2,act3,act4,rb1,rb2>

CAPABILITIES:
<extinguisher,hatchet,movement,camera,digg

PROVIDERS:
actor = <act1,act2,act3,act4>
robot = <rb1,rb2>

DATA TYPES:
Integer_type = <0..30>
Boolean_type = <true,false>

USER DEFINED DATA TYPES:
location_type = <loc00,loc01,loc02,loc03,loc10,
status_type = <ok,fire,debris>

ATOMIC TERMS (DYNAMIC):
at[srv:service] = (location_type)
batteryLevel[rbt:robot] = (integer_type)
evacuated[lct:location_type] = (boolean_type)
status[lc:location_type] = (status_type)
photoTaken[lca:location_type] = (boolean_typ

Fig. 7: The workspace required to design and visualize BPMN processes.

be represented with a causal link $t_1 \xrightarrow{e} t_2$ (or $t_2 \xrightarrow{e} t_1$), where e is an effect of task t_1 and a precondition for task t_2 (or vice-versa). This causal link requires an ordering between t_1 and t_2, meaning they need to be executed (and represented in the process template) in sequence. But this means that t_1 and t_2 are not concurrent tasks, by contradicting the original hypothesis. □

6 Validation

One of the main obstacles in applying AI techniques to real problems is the difficulty to model the domains. Usually, this requires that people that have developed the AI system carry out the modeling phase since the representation depends very much on a deep knowledge of the internal working of the AI tools.

To tackle the above issue, we have implemented our approach through a GUI-based tool consisting of several wizard-based editors that assist the process designer in the definition of the process knowledge (i.e., data objects, atomic terms, tasks with preconditions and effects, initial and goal condition, etc.), without the need of being expert of the internal working of the planning system used for synthesising process templates. In Fig. 6 we show one of the wizard-based editors provided by the tool, specifically the one to build a task specification through the definition of the single conditions composing the task preconditions and effects.

The tool has been developed as a standard Java application, using the Java SE 7 Platform, and the JGraphX open source graphical library[3], and has been built on top of the SmartPM adaptive PMS [55,56]. SmartPM provides a graphical editor to design the control flow of a business process using a relevant subset of the BPMN 2.0 notation; we customized such an editor to allow the visualization of the synthesized process templates (see Fig. 7).

In order to investigate the practical applicability of our approach, we performed two different experiments with the implemented tool. The first experiment consisted of an empirical evaluation with real users to assess whether our approach is actually usable in practice to design flexible processes (see Section 6.1). The second experiment was targeted to learn the time amount needed for synthesizing a partially ordered plan for some variants of our running example (see Section 6.2).

6.1 Empirical User Evaluation

To assess the usability of our approach against the complexity of realistic dynamic environments, we performed an empirical evaluation with 20 users. Users were selected from a group of Master students in Engineering in Computer Science at Sapienza University of Rome. Only students with a good experience in process modeling with BPMN were chosen.

After a preliminary training session to describe the usage of the tool, we provided the students with 3 different homeworks of growing complexity in 3 consecutive weeks (one homework per week). Each homework consisted of a general description of a realistic problem (with a specific objective to be fulfilled) taken from real-world application domains (specifically, smart manufacturing, emergency management and home cleaning) to be solved through the modeling of a BPMN process. In order to correctly tackle the homework problems, the control flow of each solution process required (at least) two, three and four parallel branches, respectively. Notice that the second homework was derived from our running example in Section 2, and its solution process is the one shown in the top part of Fig. 5.

We asked each student to provide two different solutions for each homework: A first solution obtained by simply modeling the process through BPMN (for this purpose, we made use of the BPMN editor provided by SmartPM, see Fig. 7), and a second solution by designing the domain theory, the task descriptions and the initial/goal conditions with our approach, which generates automatically a BPMN process. The target was to understand the actual usability of our approach to design flexible processes, in contrast to the traditional control-flow oriented way of modeling processes. The students were requested to complete the homework assigned to them in a specific week within the end of the week itself. Before the assignment of a new homework, we shown to the students the minimal solution to perform correctly the previous homework.

The results of the experiments are provided in Fig. 8. Collected data are organized in 3 diagrams related to the achievement of the objectives of the three homeworks. For any diagram, the x-axis indicates the specific approach used to solve the homework (either with our approach or by straightforwardly modeling the solution processes with BPMN), while the y-axis indicates the number of students that successfully complete the homework. Notice that for each homework we represent two bars for separating the students that performed correctly the whole homework from the students that were not able to complete it or to compute a correct solution.

The analysis of the performed experiments points out some interesting aspects. For example, let us consider the first diagram in Fig. 8, which shows the number of students that performed correctly the first homework. In this case, 19 out of 20 students were able to design the correct solution process by simply modeling it in BPMN, while less students (16 out of 20) obtained a correct solution with our approach. This result can be explained by the fact that the first homework (the simplest one) contained very precise information about the sequencing of the activities, with few activities that required to be executed concurrently (two parallel branches were sufficient to satisfy the solution).

In the second and third diagram of Fig. 8, which show the number of students that performed correctly the second and third homework (respectively), the trend drastically changes. The number of students that were able to provide a correct solution process with BPMN decreases, probably due to the presence of more contextual information (aimed at driving the design of the solution processes) in the description of the homeworks, and to the need of defining three and four parallel branches (respectively) in the BPMN solution process to achieve the homework objectives. Notice that even if the complexity of the second and third homework is comparable, the presence of (at least) four parallel branches in the solution process of the third homework makes very complex its modeling with BPMN (50% of students failed to provide a correct solution).

[3] `http://www.jgraph.com/`

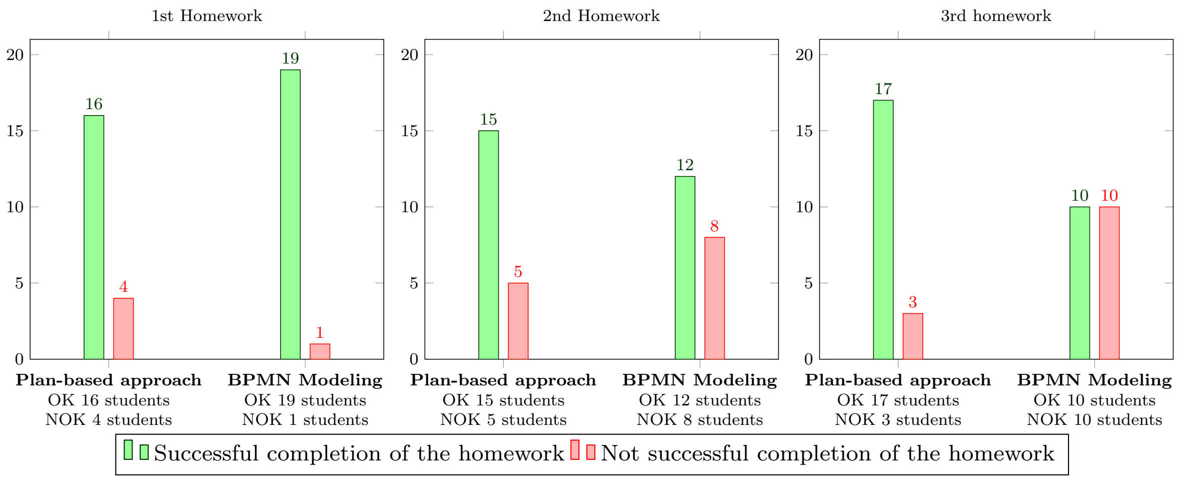


Fig. 8: Analysis of the practical usability of the planning-based tool.

Table 1: Time performances of POPF2.

Facts in $goal_{\mathsf{PR}}$	Knowledge in $init_{\mathsf{PR}}$	Time for a sub-opt. sol.
1	complete state	0.17
	No information about act1	0.15
	No information about act1 and act3	0.12
2	complete state	0.12
	No information about act3	0.10
	No information about act3 and rb1	0.08
3	complete state	0.13
	No information about act3	0.11
	No information about act3 and rb2	0.09
4	complete state	0.21
	No information about act3	0.20
	No information about act3 and rb1	0.10
5	complete state	0.17
	No information about act3	0.16
	No information about act3 and rb1	0.10
6	complete state	1.56
	No information about act3	1.19
	No information about act3 and act1	1.13

To sum up, the results obtained allow to observe a decreasing success rate when the solution process is modeled directly with BPMN and the complexity of the homework increases, in particular when there is an increase of parallel branches in the solution BPMN process. Conversely, the experiments depict an increase of the succession rate with our approach, with a “discontinuity” of this trend between the first and the second homework, probably caused by the first relevant increase of complexity. The results also suggests that students performing the homeworks were able to efficiently *learn* the usage of the tool without any knowledge of the planning system employed into the tool.

6.2 Experiments with a partial-order planner

To show the feasibility of our approach from the timing perspective, we ran some experiments and measured the time required for synthesizing a partially ordered plan for some variants of our running example described in Section 2.

We ran our tests using POPF2 [13] on an Intel U7300 1.30GHz, 4GB RAM machine. POPF2 is a temporal planner that handles PDDL 2.1 [26] and preserves the benefits of partial-order plan construction in terms of producing makespan-efficient, flexible plans. Search in POPF2 is based around the idea of expanding a partial-order plan in a forwards direction; steps added to the plan are ordered after a subset of those in the partial plan, rather than after every step in the plan.

The experimental setup was run on variants of our running example. We represented 7 planning actions in PD (corresponding to 7 different tasks stored in the tasks repository T), annotated with 7 relational predicates and 6 numeric fluents, in order to make the planner search space sufficiently challenging. Then, we defined 18 different planning problems of varying complexity by manipulating the number of facts in the goal. As well, we examined how irrelevant domain knowledge affects the performance of the planner. Starting from a planning problem PR with an initial state $init_{\mathsf{PR}}$ completely specified and with a goal condition $goal_{\mathsf{PR}}$ expressed as the conjunction of n facts, we manipulated the specification of the initial state $init_{\mathsf{PR}}$ to reduce the number of known facts. In our experiments, the number of facts in goal condition ranges from 1 single fact to a conjunction of 6 logical facts (that make the contextual problem harder). As shown in Table 1, for a given goal condition composed of n facts, our purpose was to measure the computation time needed for finding a sub-optimal solution for problems specified with initial states with a decreasing amount of knowledge. The column labeled as "Knowledge in $init_{\mathsf{PR}}$" makes explicit which information is removed from the initial state of the planning problem. For example, if we consider our running scenario, whose goal condition is composed of 3 facts and characterized by a complete specification of the initial state, the time needed for finding a solution plan is of 0.13 seconds. After removing from the initial state all the information concerning the actor *act3*, the time required for computing the plan decreases to 0.11 seconds. In general, for a given goal condition, removing "irrelevant information" from the initial state reduces the possibilities to select "wrong" planning actions (e.g., any action that involves *act3*) during the plan synthesis and, consequently, reduces the search space and the computation required for synthesizing the plan.

Concerning the fact that variant times observed in Table 1 are quite small, this depends by the specific case study tested, which is of medium size. With larger domain theories, since automated planning is known to be PSPACE-complete (cf. [27]), the variants would have been certainly larger. This emphasizes once more the importance of exploiting the weakest preconditions in the definition of the initial state to considerably limit the exploration of the search space.

We finally notice that a sub-optimal solution may include more actions than those strictly required for fulfilling the goal, and when the number of facts in a goal condition increases, the quality of the solution may decrease. This happens beacuse the current version of POPF2 (which is currently the best performing POP planner available in the literature) is able to build partial-order plans for which only the correctness of the solution is guaranteed (nothing can be really said on its optimality in terms of plan's length). However, the fact that our approach is able to generate standard PDDL files would allow us to integrate different planners for the generation of the templates. Obviously using planning we can also look for optimal plans, though this would be more costly. Nonetheless, based on the results of our experiments, we can conclude that POPF2 is feasible for the generation of medium-sized flexible process templates as used in practice.

7 Related Work

Process modeling is the first and most important step in the BPM lifecycle [50], which intends to provide a high-level specification of a business process that is independent from implementation and serves as a basis for process automation and verification. Traditional business process models are usually *well-structured*, i.e, they reflect highly repeatable and predictable routine work with low flexibility requirements [49]. All possible options and decisions that can be made during process enactment are statically pre-defined at design-time. A major assumption is that such processes, after having been modeled, can be repeatedly instantiated and executed in a predictable and controlled manner. The current leading commercial PMS products [14,77,41,60,83] and research prototypes [46,35] support the full business processes life-cycle by providing tools used to model, configure and enact business processes.

However, current BPM technology is generally based on rigid process models making its application difficult in highly dynamic and possibly evolving domains [69], where pre-specifying the entire process model is not possible. This problem can be mitigated through specific approaches to *process variability*, which allow to *customize* a process by implementing specific variants of the process in a way that a model of each variant can be derived by adding or deleting fragments according to a domain model or to some customization option [44,5]. Customization of process models can be made either at *design-time* or *run-time*.

Approaches that support customization decisions made at run-time are not concerned with maintaining a library of process models (and their variants) to be reused at run-time. On the contrary, such approaches rely on a *unitary process model* from which individual process instances may deviate at run-time on a case-by-case basis. Run-time customization has been studied as a separate topic in the literature, and is known as *process adaptation*. This is the case of two well-known adaptive PMSs, like ADEPT2 [29] and SmartPM [55, 56], which support the run-time adaptation of the process instances by dynamically adding/implementing process fragments that were not specified in the original process model. *In our work, as stated in Section 3.1 we made the choice to focus on design-time aspects of flexible processes. Nonetheless, we think that our approach can be complementary to process adaptation. In fact, the process templates that are automatically generated by our approach can be seen as candidate process models to be enacted with the existing adaptive PMSs.*

On the other hand, process models can be customized at *design-time* in order to capture families of process variants derived via transformations of the customizable process model of interest, e.g., by removal of behaviour from the underlying process model (*variability by restriction*) or by addition of dedicated process variants (*variability by extension*). There exist four different ways to approach the customization of a process model: *(i)* by defining specific *configurable nodes* within the process that capture variability through different customization options [71,45]; *(ii)* by *annotating process model elements* (control-flow nodes, resources, objects, etc.) with properties of the application domain [70]; in this way, all the model elements whose domain conditions are evaluated to false will be removed from the model; *(iii)* by specializing process model elements with specific *variation points* that provide several possible refinements to the process [31]; *(iv)* by applying specific *change operations* (e.g., insert, delete, move, replace, etc.) to restrict or extend the process model [32].

The fact is that the existing approaches to design-time process variability do not consider that flexible processes are incomplete "by nature". Basically, such approaches require to manually pre-define a "base model" explicitly marked with several "adjustment points" identified in one of the ways listed above. While this activity works perfectly in case of traditional business processes, it can be time-consuming and error-prone when flexible processes are to be modeled, due to their context-dependent nature that make difficult the specification of all the potential tasks interactions and process variants in advance.

Conversely, in our planning-based approach the process designer is not required to define in advance neither a base process model nor its individual variants. The designer specifies just the contextual knowledge (as a domain theory) in which the flexible process is embedded, a repository of tasks and some logical constraints expressed in terms of task preconditions and effects, which allow to make explicit the connection between tasks and contextual data. This information is then used to feed a POP planner that will automatically build a process template. At this point, by modifying the domain theory or by inserting/removing new/existing tasks in the repository, it is possible to obtain several process templates that can be considered as different process variants to achieve a goal condition. If compared with the traditional approaches to process variability, the novelty here is that such templates are synthesised automatically on a case-by-case basis, without the need of defining them manually; this feature makes our approach specifically tailored to the modeling of flexible processes.

As discussed in Section 3.1, flexible processes usually exhibit a *loosely structured* behavior and require process tasks to be *semantically annotated* with preconditions and effects defined over contextual data. While most approaches in the Semantic Business Process Management field (SBPM) mainly concentrate on the annotation of process models (cf. [76,10]), *without any automatic generation of the models, in this paper we focus on automatically synthesizing process templates by means of their semantic annotations.*

To this end, we focused on *declarative approaches* to business process management, as they offer a way of modeling processes that integrates a certain amount of flexibility into the process models [66]. The idea underlying declarative approaches consists of performing the modeling phase of processes by exploiting the concept of control-flow *constraints* for the specification of relationships among tasks, instead of strictly and rigidly defining the control-flow of process tasks using a procedural language. Recently, several *declarative approaches* for supporting loosely-structured processes have been proposed [65,80,30]. DECLARE [65] is a constraint-based PMS that uses a declarative language grounded in linear temporal logic (LTL [79]) for the development and execution of process models. Rather than using a procedural language for expressing the allowed sequences of tasks, it describes processes through the usage of constraints: the idea is that every task sequence can be performed, except when it does not respect them. Similarly, Alaska [80], a tool suite developed at the University of Innsbruck, allows to define task constraints that implicitly define possible execution alternatives by prohibiting undesired execution behavior. Finally, the work [30] presents a declarative approach that enables the modeling of dynamic sets of candidate activities from which a subset is automatically selected for execution. *While the above approaches allow to define constraints exclusively on process activities (i.e., the data perspective, which plays a relevant role for flexible processes, is neglected in the definition of task constraints), our approach allows a process designer to explicitly capture the contextual domain (i.e., the data perspective)* *in which the process is meant to run, and to define constraints between tasks and data.*

Processes may also be specified using the *data-centric* paradigm. In data-centric methodologies, the data perspective is predominant and captures domain-relevant object types, their attributes, their possible states and life-cycles, and their interrelations, which together form a complex data structure, i.e., an *information model*. This information model enables the identification and definition of the activities that rely on the object-related information and act on it, producing changes on attribute values, relations and object states. Notice that activities of a data-centric process, which do not have a pre-specified order, are triggered only when the required data is available, i.e., the evolution of the information model drives the process execution. There exist several approaches belonging to the data-centric paradigm: Case Handling [78], Artifact-centric process models [36], Object-aware process models [43], Data-Centric Dynamic Systems [6] and the OMG's emerging Case Management Modeling Notation standard (CMMN) [63]. All these approaches are potentially good candidates to deal with the requirements presented in Section 3.1. However, they require to manually anticipate in an explicit way all the interrelations between tasks and the information model (i.e., the contextual properties of the domain), and this could be a complex activity to be performed at design-time. This is particularly true in dynamic environments, where the process designer often lacks the needed knowledge to model every required constraint ahead of time.

On the contrary, the approach presented in this paper allows to declare in an implicit way (by means of tasks preconditions and effects) the constraints between tasks and data. The novelty is that the control flow constraints between tasks (i.e., causal links and ordering constraints) will be automatically and contextually generated by an external planner, together with the process template to be executed.

A further research area related to our approach is the one of *service discovery*. In the literature, there exists a wide range of solutions for the automated discovery and selection of services specified via a *service profile* and a *process model* [84]. A service profile describes the *signature* of a service in terms of its *input* and *output* parameters, and its execution semantics via *preconditions* and *effects*. Prominent approaches to represent such profiles are, for example, WSDL and SML. A service *process model* describes the operational behavior of a service in terms of its internal control and data flow. Such models are described, for example, in OWL-S, WSML and USDL.

Most of service discovery techniques rely on the matching of the semantic annotations of service profile (i.e., signature matching) and/or process model. Without any doubt, task specifications in our approach and service profiles are similar concepts. However, there exists a *relevant difference* in what our approach and service discovery techniques aim to discover. While service discovery approaches aim at locating and selecting *existing* services that are relevant for a given request, the main target of our approach is to *synthesize new non-existing process templates*. Therefore, we exploit the semantic annotations of tasks (preconditions and effects) included in a repository *not for discovering tasks*, but for *generating process templates that orchestrate such tasks in order to achieve a goal condition.*

Nonetheless, we think that service discovery techniques can provide a great value in the phase of retrieving process templates from our existing library. As discussed in Section 5, we currently provide a basic semantic binding mechanism for the selection of a suitable template. As a future work, it is our intention to consider service discovery techniques to improve the selection mechanism of process templates.

7.1 Plan-based Process Design

The AI community has been involved with research on process management for several decades, and AI technologies can play an important role in the construction of PMS engines that manage complex processes, while remaining robust, reactive, and adaptive in the face of both environmental and tasking changes [61].

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. For the *run-time* phase, existing literature works reports on the use of planners to allocate process activities to (human) resources [15], whereas works [52, 54, 57, 9, 55, 56] report on approaches to adapt the running process instances to cope with anomalous situations, including connection anomalies, exogenous events and task faults. The work [8] 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 [42] 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 [68] highlights the improvements that a legacy workflow application can gain by incorporating planning techniques into its day-to-day operation. In the context of process mining, the works [20,17,16,48] use planning techniques to recover and align events log traces against imperative and declarative process models.

Readers should observe that the entire aforementioned approaches use planning techniques for completely different purposes. Some research works also exist that use planning techniques to deal with problems for the design-time phase, and the closest to our approach are [74,67,25,34].

The work [74] presents the basic idea behind the use of planning techniques for generating a process schema, but no implementation seems to be provided, and the direct use of the PDDL language for specifying the domain theory requires a deep understanding of AI planning technology. In [67], the authors exploit the IPSS planner for modeling processes in SHAMASH [4], a knowledge-based system that uses a rule-based approach. To automate the process model generation, they first translate the semantic representation of SHAMASH into the IPSS language. Then, IPSS produces a parallel plan of activities that is finally translated back into SHAMASH and is presented graphically to the user. This work proposes the scheduling of parallel activities (that implicitly handle time and resource constraints), meta-modeling that deals with planning explicitly, and suggests that learning could be used for process optimization. However, in [74] and [67] the emphasis is on supporting processes for which one has complete knowledge, while for flexible processes some contextual information may not be available at the time of process model synthesis. *Conversely, our approach allows the specification of process templates in partially specified environments, i.e., such templates require a fragment of the knowledge of the initial state to successfully achieve their objectives. We identify the weakest preconditions of process templates, and all the states satisfying such preconditions are good candidates for executing them.*

The work [25] proposes a new life cycle for workflow management based on the continuous interplay between learning and planning. The approach 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. Unfortunately, as acknowledged in [25], the best plan found is often far from the correct solution. *According to Theorem 1, one of the strengths of our approach relies on its ability to generate process templates that are always executable for any case that satisfies the weakest preconditions inferred from the template itself.*

The SEMPA approach [34] introduces a semantic-based automatic planning of process models. In SEMPA, process actions are semantically described by specifying their input/output parameters with respect to an ontology implemented in OWL [7]. An action state graph (ASG) is automatically derived from the semantic information about input/output parameters. It includes those process actions that lead to the given goals from a pre-specified initial state. Then, a process model represented as an UML activity diagram is derived from the ASG by identifying the required control flow for the process. If compared with other planning approaches, the planning algorithm implemented in SEMPA for the derivation of the ASG provides some interesting characteristics, such as the ability to build the ASG in presence of initial state uncertainty and with different conflicting goals. *The problem here is that concurrent activities of the generated process are not proven to be independent from each other with respect to data. On the contrary, concurrent activities of a process template generated by our approach are proven to be effectively* independent *one from another (i.e., concurrent tasks cannot affect the same data)*

8 Concluding Remarks

In the last years, there was a growing trend of managing flexible processes in dynamic environments [69], mainly due to the increased availability of sensors disseminated in the world that allow to monitor the evolution of several real-world objects of interest and to support mobile workers that execute tasks in such dynamic settings. According to the recent keynote talk performed by Rick Hull at BPM 2016 Conference [38], the traditional BPM approaches are not tailored for the management of flexible processes, and advances in goal identification and planning are clearly needed, that is, enabling process designers to take advantage of knowledge that is relevant to a decision or task at hand, and ignore knowledge that is irrelevant.

This paper goes exactly in the above direction and addresses the problem of developing template-based process models for processes and application contexts which demand *flexibility* (e.g., processes for emergency management). To this end, we developed a technique based on Partial-Order Planning algorithms and declarative specifications of process tasks for synthesizing a library of process templates to be enacted in contextual scenarios. The resulting templates guarantee sound concurrency in the execution of their activities and are reusable in a variety of partially specified contextual environments. A key characteristic of our approach is the role of contextual data acting as a driver for process templates generation.

Since one of the main obstacles when dealing with a flexible process lies in the difficulty of interacting with and manipulating the knowledge embedded into the process, we have implemented a dedicated user-friendly graphical interface that helps the process designer in the definition/refinement of the domain theory/case and in the choice of the right template (i.e., one having the required quality, if several templates satisfy the current case) to be selected for the execution in a specific contextual situation. To this end, we have performed a complete validation of the approach by investigating its practical applicability for designing flexible processes.

Even if our work is focused on process modeling at design-time, we think that the proposed approach can be complementary to run-time approaches to process adaptation. In fact, the process templates that are automatically generated by our approach can be seen as candidate process models to be enacted with the existing adaptive PMSs, which in general rely on a unitary process model from which individual process instances may deviate at run-time on a case-by-case basis. In particular, we are currently working on integrating our approach with SmartPM [55,56], an adaptive PMS that accepts in input process models whose tasks specification and domain theory formalization is similar to the one proposed in our approach.

The fact that our approach relies on well-founded planning formalisms opens the door to very promising future works. First of all, since our approach is able to automatically encode planning problems in the standard and system-independent PDDL language, one can seamlessly update to the recent version of the best performing (or expressive) POP planner, with evident advantages in term of versatility and customization. In fact, the effort to integrate a different planner does not go beyond installing the new planner and loading the planning-problem formulation that is generated through our approach. The problem formulation remains unchanged when moving from one planner to the other. A second future direction for this work is to generate and support hierarchical process templates, with high-level templates achieving more general goals that can invoke simpler templates to achieve some of their subgoals. It seems that agent-technology can provide promising approaches and methods to address this challenge. We also plan to address some expressiveness limitations of our POP-based tool, such as expressing complex goal conditions (comprising, for example, the use of existential quantifiers, disjunctions and intermediate subgoals), handling preferences, representing conditional effects and negative preconditions (they are currently not supported by most POP planners, including POPF2) and supporting non functional-properties (like cost or time-constraints). A third interesting future work is to devise a set of design-time guidelines that may help the process designer in the selection of the tasks to be included in the repository and in the refinement of a process template. Finally, we are also working on the formalization of new quantitative metrics for evaluating process templates' quality.

This paper extends previous work in [51] in several directions by includes many new elements that were neglected in our previous work:

- a new section describing the characteristics of flexible processes and requirements for managing them. If, on the one hand, such requirements motivate the need of our work in the BPM context, on the other hand they have provided concrete pointers to the building of the tool underlying the proposed approach;
- a new section describing the tool's implementation and user interface;
- a more thorough description of how process templates are concretely retrieved from the library, an aspect that was neglected in our previous work;
- a completely new set of experiments to test the usability of the approach against real users, in comparison with traditional activity-centric approaches to BPM;
- the related work section has been partially rewritten and extended significantly;
- the description of flexible processes is more detailed and complete;
- all the details of the translation algorithms for interfacing with the planner; i.e., for translating a domain theory and a process case into a planning domain and a planning problem, and for converting a partially ordered plan into a process template. These algorithms, which are included in a dedicated Appendix, allow our approach to be replicated and concretely implemented by any researcher;
- all other sections of the paper have been edited and refined to present the material more thoroughly.

References

1. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing between Flexibility and Support. Computer Science - Research and Development **23**(2), 99–115 (2009)
2. van der Aalst, W.M.: Business Process Management: A Comprehensive Survey. ISRN Software Engineering **2013** (2013). URL `http://dx.doi.org/10.1155/2013/507984`
3. van der Aalst, W.M.: Process mining: data science in action. Springer (2016)
4. Aler, R., Borrajo, D., Camacho, D.: A Knowledge-based Approach for Business Process Reengineering, SHAMASH. Know.-Based Syst. **15**(8) (2002)
5. 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**, 248–276 (2015). URL `https://doi.org/10.1016/j.infsof.2014.05.009`
6. Bagheri Hariri, B., Calvanese, D., de Giacomo, G., Deutsch, A., Montali, M.: Verification of Relational Data-centric Dynamic Systems with External Services. In: Proceedings of the 32nd Symposium on Principles of Database Systems, PODS '13, pp. 163–174. ACM, New York, NY, USA (2013). DOI 10.1145/2463664.2465221. URL `http://doi.acm.org/10.1145/2463664.2465221`
7. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A., et al.: Owl web ontology language reference. W3C recommendation **10**, 2006–01 (2004)
8. Beckstein, C., Klausner, J.: A Meta Level Architecture for Workflow Management. Journal of Integrated Design and Process Science **3**(1), 15–26 (1999)
9. van Beest, N.R., Kaldeli, E., Bulanov, P., Wortmann, J.C., Lazovik, A.: Automated runtime repair of business processes. Information Systems **39**, 45–79 (2014). DOI 10.1016/j.is.2013.07.003
10. Brockmans, S., Ehrig, M., Koschmider, A., Oberweis, A., Studer, R.: Semantic alignment of business processes. In: ICEIS (3), pp. 191–196 (2006)
11. Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Russo, A., Steinmann, R., Bortenschlager, M.: Workpad: Process management and geo-collaboration help disaster response. IJISCRAM **3**(1) (2011)
12. Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Salvatore, B., Vetere, G., Dustdar, S., Juszczyk, L., Manzoor, A., Truong, H.L.: Pervasive software environments for supporting disaster responses. IEEE Internet Computing **12**(1) (2008)
13. Coles, A.J., Coles, A., Fox, M., Long, D.: Forward-Chaining Partial-Order Planning. In: ICAPS (2010)
14. Cosa GmbH: COSA BPM product description. Retrieved July 03, 2017, from: `http://www.cosa.nl/docs/EN/COSA%20BPM%205.7%20Productdescription_en_new_K.pdf` (2013)
15. Currie, K., Tate, A.: O-Plan: The Open Planning Architecture. Artificial Intelligence **52**(1), 49–86 (1991). URL `10.1016/0004-3702(91)90024-E`
16. 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 Conference on Artificial Intelligence, pp. 3555–3561 (2017). URL `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14652`
17. De Giacomo, G., Maggi, F.M., Marrella, A., Sardiña, S.: Computing Trace Alignment against Declarative Process Models through Planning. In: Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS), pp. 367–375 (2016). URL `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13094`
18. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches. Journal on Data Semantics **4**(1), 1–29 (2015). URL `http://dx.doi.org/10.1007/s13740-014-0038-4`
19. Di Ciccio, C., Mecella, M.: Mining Constraints for Artful Processes. In: 15th International Conference on Business Information Systems (BIS), pp. 11–23. Springer Berlin Heidelberg (2012). URL `https://doi.org/10.1007/978-3-642-30359-3_2`
20. Di Francescomarino, C., Ghidini, C., Tessaris, S., Sandoval, I.V.: Completing Workflow Traces Using Action Languages, pp. 314–330. Springer International Publishing (2015). URL `http://dx.doi.org/10.1007/978-3-319-19069-3_20`
21. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: Business Process Management, pp. 48–63. Springer (2009)
22. Dijkman, R., Dumas, M., Van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems **36**(2), 498–516 (2011)
23. Dumas, M., van der Aalst, W.M.: Process-aware information systems: bridging people and software through process technology. Wiley-Interscience (2005)
24. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Berlin Heidelberg (2013). URL `http://dx.doi.org/10.1007/978-3-642-33143-5`
25. Ferreira, H., Ferreira, D.: An Integrated Life Cycle for Workflow Management Based on Learning and Planning. Int. J. Coop. Inf. Systems **15** (2006)
26. Fox, M., Long, D.: PDDL2.1: an Extension to PDDL for Expressing Temporal Planning Domains. J. Artif. Int. Res. **20**(1) (2003)
27. Geffner, H., Bonet, B.: A Concise Introduction to Models and Methods for Automated Planning. Synthesis Lectures on Artificial Intelligence and Machine Learning **8**(1), 1–141 (2013). DOI 10.2200/S00513ED1V01Y201306AIM022
28. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem. Springer-Verlag (1996)
29. Goser, K., Jurisch, M., Acker, H., Kreher, U., Lauer, M., Rinderle-Ma, S., Reichert, M., Dadam, P.: Next-generation Process Management with ADEPT2. In: Demonstration Program of the 5th International Conference on Business Process Management (BPM) (2007)
30. Grambow, G., Oberhauser, R., Reichert, M.: Contextual generation of declarative workflows and their application to software engineering processes. International Journal on Advances in Intelligent Systems **4**(3 and 4), 158–179 (2012)
31. Gröner, G., Bosković, M., Silva Parreiras, F., Gasević, D.: Modeling and Validation of Business Process Families. Information Systems **38**(5), 709–726 (2013). URL `http://dx.doi.org/10.1016/j.is.2012.11.010`
32. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. Journal of Software: Evolution and Process **22**(6-7), 519–546 (2010). URL `http://dx.doi.org/10.1002/smr.491`

33. Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., Jansen, E.: The Gator Tech Smart House: A Programmable Pervasive Space. Computer **38**, 50–60 (2005). URL `http://dx.doi.org/10.1109/MC.2005.107`
34. Henneberger, M., Heinrich, B., Lautenbacher, F., Bauer, B.: Semantic-based planning of process models. In: Multikonferenz Wirtschaftsinformatik (2008)
35. ter Hofstede, A., van der Aalst, W., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2009). URL `https://doi.org/10.1007/978-3-642-03121-2`
36. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: On the Move to Meaningful Internet Systems: OTM 2008, *Lecture Notes in Computer Science*, vol. 5332, pp. 1152–1163. Springer Berlin Heidelberg (2008)
37. Hull, R., Motahari Nezhad, H.R.: Rethinking BPM in a cognitive world: Transforming how we learn and perform business processes. In: Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings, *Lecture Notes in Computer Science*, vol. 9850, pp. 3–19. Springer (2016). DOI 10.1007/978-3-319-45348-4_1
38. Hull, R., Motahari Nezhad, H.R.: Rethinking BPM in a Cognitive World: Transforming How We Learn and Perform Business Processes, pp. 3–19. Springer International Publishing (2016). URL `http://dx.doi.org/10.1007/978-3-319-45348-4_1`
39. Humayoun, S.R., Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Bortenschlager, M., Steinmann, R.: Designing mobile systems in highly dynamic scenarios. the workpad methodology. Springer's International Journal on Knowledge, Technology and Policy **22**(1) (2009)
40. 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: HCI (7), pp. 343–352 (2009)
41. IBM Inc.: An introduction to WebSphere Process Server and WebSphere Integration Developer. Retrieved July 03, 2017, from: `ftp://ftp.software.ibm.com/software/integration/wps/library/WSW14021-USEN-01.pdf` (2008)
42. Jarvis, P., Moore, J., , J.S., Macintosh, A., du Mont, A.C., Chung, P.: Exploiting AI Technologies to Realise Adaptive Workflow Systems. In: Proceedings of the AAAI Workshop on Agent-Based Systems in the Business Context (1999)
43. Künzle, V., Weber, B., Reichert, M.: Object-aware Business Processes: Fundamental Requirements and their Support in Existing Approaches. International Journal of Information System Modeling and Design (IJISMD) **2**(2), 19–46 (2011). URL `http://dbis.eprints.uni-ulm.de/721/`
44. La Rosa, M., van der Aalst, W.M., Dumas, M., Milani, F.P.: Business Process Variability Modeling: A Survey. ACM Computing Surveys (2013). URL `http://doi.acm.org/10.1145/3041957`
45. La Rosa, M., Dumas, M., Ter Hofstede, A.H., Mendling, J.: Configurable multi-perspective business process models. Information Systems **36**(2), 313–340 (2011). URL `https://doi.org/10.1016/j.is.2010.07.001`
46. Lanz, A., Kreher, U., Reichert, M., Dadam, P.: Enabling process support for advanced applications with the AristaFlow BPM Suite (2010)
47. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Data & Knowledge Engineering **61**, 39–58 (2007). URL `https://doi.org/10.1016/j.datak.2006.04.007`
48. de Leoni, M., Marrella, A.: Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. Expert Syst. Appl. **82**, 162–183 (2017). URL `https://doi.org/10.1016/j.eswa.2017.03.047`
49. Leymann, F., Roller, D.: Production workflow: concepts and techniques. Prentice Hall PTR (2000)
50. Lu, R., Sadiq, S.: A Survey of Comparative Business Process Modeling Approaches. In: 10th International Conference on Business Information Systems (BIS), pp. 82–94. Springer-Verlag (2007)
51. Marrella, A., Lesperance, Y.: Synthesizing a library of process templates through partial-order planning algorithms. In: S. Nurcan, H. Proper, P. Soffer, J. Krogstie, R. Schmidt, T. Halpin, I. Bider (eds.) Enterprise, Business-Process and Information Systems Modeling, *Lecture Notes in Business Information Processing*, vol. 147, pp. 277–291. Springer Berlin Heidelberg (2013). DOI 10.1007/978-3-642-38484-4_20. URL `http://dx.doi.org/10.1007/978-3-642-38484-4_20`
52. Marrella, A., Mecella, M.: Continuous Planning for Solving Business Process Adaptivity. In: 12th International Conference on Business-Process and Information Systems Modeling (BPMDS), pp. 118–132. Springer Berlin Heidelberg (2011). URL `http://dx.doi.org/10.1007/978-3-642-21759-3_9`
53. Marrella, A., Mecella, M., Russo, A.: Collaboration On-the-field: Suggestions and Beyond. In: 8th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2011) (2011)
54. Marrella, A., Mecella, M., Russo, A.: Featuring Automatic Adaptivity through Workflow Enactment and Planning. In: 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2011), pp. 372–381 (2011). URL `https://doi.org/10.4108/icst.collaboratecom.2011.247096`
55. Marrella, A., Mecella, M., Sardiña, S.: Smartpm: An adaptive process management system through situation calculus, indigolog, and classical planning. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014 (2014)
56. Marrella, A., Mecella, M., Sardiña, S.: Intelligent process adaptation in the smartpm system. ACM TIST **8**(2), 25:1–25:43 (2017). DOI 10.1145/2948071. URL `http://doi.acm.org/10.1145/2948071`
57. Marrella, A., Russo, A., Mecella, M.: Planlets: Automatically Recovering Dynamic Processes in YAWL. In: 20th International Conference on Cooperative Information Systems (CoopIS) - OTM Conferences (1), pp. 268–286. Springer Berlin Heidelberg (2012). URL `https://doi.org/10.1007/978-3-642-33606-5_17`
58. Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL - The Planning Domain Definition Language. Tech. rep., Yale Center for Computational Vision and Control (1998)
59. Meyer, A., Smirnov, S., Weske, M.: Data in Business Processes. Universitätsverlag Potsdam (2011)
60. M.Kinateder: SAP advanced workflow techniques. Retrieved July 03, 2017, from: `http://scn.sap.com/docs/DOC-3286.`
61. Myers, K.L., Berry, P.M.: Workflow Management Systems: An AI Perspective. AIC-SRI report (1998)
62. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)

63. OMG: Case Management Model and Notation, Version 1.0 (2014). URL `http://www.omg.org/spec/CMMN/1.0`
64. Papazoglou, M.P., Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. The VLDB JournalThe International Journal on Very Large Data Bases **16**(3), 389–415 (2007)
65. Pesic, M., Schonenberg, H., van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International, pp. 287–287. IEEE (2007)
66. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: Business Process Management Workshops, pp. 383–394. Springer (2012)
67. 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)
68. R-Moreno, M.D., Kearney, P.: Integrating AI Planning Techniques with Workflow Management System. Knowledge-Based Systems **15**(5-6) (2002). URL `https://doi.org/10.1016/S0950-7051(01)00167-8`
69. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems. Springer Berlin Heidelberg (2012). URL `http://dx.doi.org/10.1007/978-3-642-30409-5`
70. Reijers, H.A., Mans, R., van der Toorn, R.A.: Improved model management with aggregated business process models. Data & Knowledge Engineering **68**(2), 221–243 (2009). URL `https://doi.org/10.1016/j.datak.2008.09.004`
71. Reinhartz-Berger, I., Soffer, P., Sturm, A.: Extending the Adaptability of Reference Models. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **40**(5), 1045–1056 (2010). URL `https://doi.org/10.1109/TSMCA.2010.2044408`
72. Reiter, R.: On Closed World Data Bases. In: M. Ginsberg (ed.) Readings in Nonmonotonic Reasoning. Morgan Kaufmann Publishers Inc. (1987)
73. Russell, S.: Artificial intelligence: A modern approach, 2/E. Pearson Education India (2003)
74. Schuschel, H., Weske, M.: Triggering replanning in an integrated workflow planning and enactment system. In: ADBIS (2004)
75. Silver, B.: Case management: Addressing unique bpm requirements. Taming the Unpredictable: Real-World Adaptive Case Management pp. 1–12 (2009)
76. Thomas, O., Fellmann, M.: Semantic business process management: Ontology-based process modeling using event-driven process chains. IBIS **4**, 29–44 (2007)
77. Tibco Software Inc.: TIBCO iProcess Engine - Architecture Guide. Retrieved July 03, 2017, from: `https://docs.tibco.com/pub/iprocess-engine/11.1.0-september-2009/pdf/tib-iprocess-engine-architecture-guide.pdf` (2009)
78. van der Aalst, W.M., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data & Knowledge Engineering **53**(2), 129–162 (2005). DOI 10.1016/j.datak.2004.07.003
79. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for concurrency, pp. 238–266. Springer (1996)
80. Weber, B., Pinggera, J., Zugal, S., Wild, W.: Alaska simulator toolset for conducting controlled experiments on process flexibility. In: Information Systems Evolution, pp. 205–221. Springer (2011)
81. Weld, D.: An Introduction to Least Commitment Planning. AI mag. **15**(4) (1994)
82. White, S.A., Miers, D.: BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc. (2008)
83. Wongwatkit, C.: A development of order processing system: BPMN model. In: 14th International Conference on Advanced Communication Technology (ICACT), pp. 653–658 (2012)
84. Zunino, A., Campo, M.: A survey of approaches to web service discovery in service-oriented architectures. Innovations in Database Design, Web Applications, and Information Systems Management **107**(1) (2012). URL `https://doi.org/10.4018/jdm.2011010105`

A Translation Algorithms

This appendix is focussed primarily on presenting technical details concerning the translation algorithms introduced in Section 5.1. Specifically, in the following, we describe:

- the module `PC2PR` used for translating a domain theory D and a process case C into a planning domain PD and a planning problem PR;
- the module `POP2PT` used for converting a partially ordered plan P into a process template PT.

A.1 Representing Domain Theories and Process Cases in PDDL

To obtain a process template that handles a case C, a corresponding PDDL planning problem definition PR has to be specified. This can be done by mapping $init_{\mathsf{C}}$ to $init_{\mathsf{PR}}$ and $goal_{\mathsf{C}}$ to $goal_{\mathsf{PR}}$. The planning domain PD is built starting from the definition of ground atomic terms and data types as shown in Section 4, and by making explicit the *actions* associated with each annotated task $t \in \mathsf{T}$, together with their pre-conditions, effects and input parameters. Basically, the planning domain describes how predicates and functions change after an action's execution, and specifies the contextual properties constraining the execution of tasks stored in the tasks repository.

Our framework provides a software module named `PC2PR` in charge of performing such a translation, which makes use of PDDL version 2.1 (cf. [26]). In the following, we discuss how the domain theory D can be translated into a PDDL file representing the planning domain PD:

- the *name* and the *domain* of a data type correspond to an *object type* in the planning domain;
- boolean terms have a straightforward representation as *relational predicates* (templates for logical facts) in the planning domain;
- integer terms correspond to PDDL *numeric fluents*, and are used for modeling non-boolean resources (e.g., the battery charge level of a robot) in the planning domain;
- functional terms do not have a direct representation in PDDL 2.1, but may be represented as relational predicates. Since a functional term is a function $f : Object^n \rightarrow Object$ that maps tuples of objects with domain types D^n to objects with co-domain type U, it may be encoded in the planning domain as a relational predicate P of type (D^n, U).

  For example, let us consider the running example presented in Section 2 and its formalization with the domain theory D as shown in Section 4. The module `PC2PR` converts D into a PDDL planning domain PD defined as follows:

```
(define (domain Derailment)
...
(:types participant location capability)
(:predicates
    (at ?prt - participant ?loc - location)
    (provides ?prt - participant
              ?cap - capability)
...
)
(:functions
    (battery_level ?prt - participant)
    (battery_consumption_debris)
...
)
```

  The `:types` field is used to declare the relevant types of objects interacting in our contextual scenario. Specifically, three new types have been declared: `participant`, `location` and `capability`. The `:predicates` field consists of a list of declarations of relational predicates, reflecting the properties of the contextual scenario. Notice that in PDDL, variables are distinguished by an initial '?' character, for example `?prt` and `?loc` are two variables. The dash '−' is used to assign types to the variables. In the example above, `?prt` is defined to be of type `participant` and `?loc` is defined to be of type `location`. For example, the predicate `at` holds true if a participant `?prt` is situated in location `?loc`. The predicate `provides` is used for declaring if a capability `?cap` is provided by the participant `?prt`.
- a given task, together with the associated pre-conditions, effects and input parameters, is translated into a PDDL *action schema*. An action schema describes how the relational predicates and/or numeric fluents change after the action's execution. For example, given the following XML specification of the task $Go \in \mathsf{T}$ (with respect to the language provided in Section 4):

```
<task>
<name>Go</name>
<parameters>
    <arg>prt - Participant</arg>
    <arg>from - Location</arg>
    <arg>to - Location</arg>
</parameters>
<precondition>at[prt] == from AND
              provides[prt,movement] == true
</precondition>
<effect>at[prt] = to</effect>
</task>
```

  the module `PC2PR` produces the following PDDL representation:

```
(:action go
:parameters (?prt - participant
             ?from - location
             ?to - location)
:precondition (and (at ?prt ?from)
                   (provides ?prt movement))
:effect (and (forall (?loc - location)
                     (not (at ?prt ?loc)))
             (at ?prt ?to))
```

  This task can be executed only if the actor denoted by `?prt` is not currently located in the target location `?to` (and is located in her/his starting location `?from`) and is capable of moving into the area. The desired effect turns the value of the predicate `(at ?prt ?to)` to *true* and of `(at ?prt ?loc)` to $false$, where `?loc` is any location different from the destination.

The planning problem PR can be seen as an instance of the planning domain PD. It first declares the constant symbols in the initial state of the planning problem, then defines the initial state $init_{\mathsf{PR}}$ and finally specifies a goal condition $goal_{\mathsf{PR}}$ for the planning problem. Specifically, the module `PC2PR` converts the case C into a PDDL file representing the planning problem PR as follows:

- for each data type defined in the planning domain, all the possible object instances of that particular data type are explicitly instantiated as *constant symbols* in the initial state of the planning problem (e.g., the fact that *act1*, *act2*, *act3*, *act4*, *rb1* and *rb2* are *Participants*, and that *loc00*, ..., *loc33* are *Locations*);
- a representation of the *initial state* of the planning problem is generated; basically, the initial state of the planning problem $init_{\mathsf{PR}}$ is composed of a conjunction of relational predicates (representing functional and boolean terms in $init_{\mathsf{C}}$) and the initial value of each numeric fluent (e.g., the value of the battery charge level for each robot), corresponding to the values of integer terms in $init_{\mathsf{C}}$;
- the *goal condition* of the planning problem is a logical expression over facts, which partially specifies the state to be reached after the execution of the process template; it is a condition represented as a conjunction of relational predicates and numeric fluent atoms representing the specific boolean, functional and integer terms we want to make true through the correct execution of the process template (as defined in $goal_{\mathsf{C}}$).
  Notice that the description of the planning problem PR derived from the case C of our running example roughly corresponds to the contextual information available on the dynamic environment described in Fig. 1(b):

```
(define (problem EM1) (:domain Derailment)
(:objects
    act1 - participant
    ...
    rb2 - participant
    movement - capability
    camera - capability
    ...
    loc00 - location
    ...
    loc33 - location
)
(:init
    (at act1 loc00)
    ...
    (at rb2 loc00)
    (provides act1 movement)
    ...
    (provides rb2 battery)
    ...
    (= battery_level rb1 3)
)
(:goal
    (and (evacuated loc32)
    (fire_free loc31)
    (photo_taken loc33))
))
```

A.2 Translating a Partially Ordered Plan P into a Process Template PT

Once the plan P has been synthesized, it needs to be translated in a process template PT. As explained in Section 3.2, a **solution plan** is a three-tuple $\mathsf{P} = (A, O, CL)$, where A is the set of actions appearing in the plan, O and CL are respectively the set of ordering constraints and of causal links over A. Since the set of actions A composing the plan and the set of ordering constraints O over A must be explicitly expressed as nodes and transitions of the template's control flow (as well as their intrinsic ordering), we have implemented a module named POP2PT that takes as input a solution plan P and converts it into a process template PT.

We provide two algorithms - named respectively "$Find_{PREC/NEXT}$" (cf. Algorithm 1) and "$Build_{\mathsf{PT}}$" (cf. Algorithm 2) - to be executed sequentially for automatically computing a process template PT. For each planning action $a_i \in A$, Algorithm 1 is in charge of detecting which actions "directly" precede and follow a_i in the plan. This information will be crucial for instantiating the transitions between the flow objects of the process template. However, this knowledge is not directly available in O. In fact, an ordering constraint $a \prec b$ between two actions $a \in A$ and $b \in A$ indicates that a must be executed sometime before action b, but not necessarily immediately before. Algorithm 1, for each action $a_i \in A$, builds two sets containing the actions that immediately precede a_i (the set $PREC(a_i)$) and immediately follow a_i (the set $NEXT(a_i)$).

Definition 10 *Given an ordering constraint $(a \prec b) \in O$ between two actions $a \in A$ and $b \in A$, we say that* ***a directly precedes b*** *and* ***b directly follows a*** *iff no further action $c \in A$ exists such that $a \prec c$ and $c \prec b$.*

Basically, for each ordering constraint $o_k = (a_i \prec a_j) \in O$,[4] the algorithm $Find_{PREC/NEXT}$ works as follows:

- if there does not exist any planning action $a_h \neq a_j$ that precedes a_i - i.e., such that $(a_h \prec a_i) \in O$ - then the only predecessor of a_i is the dummy start action a_0. Therefore, a_0 is added to the set of predecessors of a_i (i.e., $PREC(a_i) = PREC(a_i) \cup \{a_0\}$) and a_i is added to the set of successors of a_0 (i.e., $NEXT(a_0) = NEXT(a_0) \cup \{a_i\}$).
- if there does not exist any planning action $a_h \neq a_i$ that follows a_j - i.e., such that $(a_j \prec a_h) \in O$ - then the only successor of a_j is the dummy end action a_∞. Therefore, a_j is added to the set of predecessors of a_∞ (i.e., $PREC(a_\infty) = PREC(a_\infty) \cup \{a_j\}$) and a_∞ is added to the set of successors of a_j (i.e., $NEXT(a_j) = NEXT(a_j) \cup \{a_\infty\}$).
- if there does not exist any planning action $a_h \neq a_i$ that precedes a_j - i.e., such that $(a_h \prec a_j) \in O$ - then a_i directly precedes a_j (and a_j directly follows a_i), meaning that $PREC(a_j) = PREC(a_j) \cup \{a_i\}$ and $NEXT(a_i) = NEXT(a_i) \cup \{a_j\}$.
- if there exists a planning action $a_h \neq a_i$ that precedes a_j - i.e., such that $(a_h \prec a_j) \in O$ - but there does not exist any finite sequence of actions $a_1, a_2, ..., a_n$ such that $(a_i \prec ... \prec a_1 \prec a_2 \prec ... \prec a_n \prec ... \prec a_h)$, then a_i directly precedes a_j (and a_j directly follows a_i), meaning that $PREC(a_j) = PREC(a_j) \cup \{a_i\}$ and $NEXT(a_i) = NEXT(a_i) \cup \{a_j\}$.

Starting from the two sets just computed, the algorithm $Build_{\mathsf{PT}}$ is in charge of building the process template PT, by instantiating the tasks, gateways, events and transitions between them. For every action $a \in A$, a corresponding task instance $t_a \in T$ (basically, a task instance is an occurrence of a specific task $t \in \mathsf{T}$) is generated (cf. the function *taskify* in Algorithm 2). Transitions between tasks depend on the number of predecessors/successors contained in $PREC(a)/NEXT(a)$. In particular, if an action $b \in A$ is such that $b \in PREC(a)$ or $b \in NEXT(a)$, it is clear that there must be some kind of transition between t_a and t_b in PT:

[4] Before the execution of the algorithm, all the ordering constraints involving the dummy start action a_0 and the dummy end action a_∞ are removed from O.

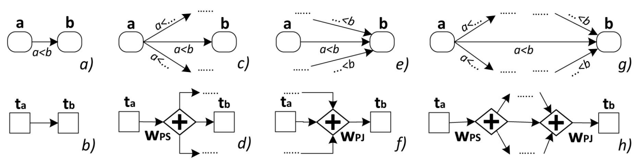


Fig. 9: Overview of the working of the Algorithm $Find_{PREC/NEXT}$.

- if b is the only successor of a, and a is the only predecessor of b (cf. Fig. 9(a)), then $(t_a \to t_b) \in L_T$ (cf. Fig. 9(b));
- if a is the only predecessor of b, but b is not the only successor of a (cf. Fig. 9(c)), then a parallel split $w_{PS} \in W_{PS}$ is needed between t_a and t_b. Hence $(t_a \to w_{PS}) \in L_T$ and $(w_{PS} \to t_b) \in L_{W_{PS}}$ (cf. Fig. 9(d));
- if b is the only successor of a, but a is not the only predecessor of b (cf. Fig. 9(e)), then a parallel join $w_{PJ} \in W_{PJ}$ is needed between t_a and t_b. Hence $(t_a \to w_{PJ}) \in L_T$ and $(w_{PJ} \to t_b) \in L_{W_{PJ}}$ (cf. Fig. 9(f));
- if t_a is not the only predecessor of t_b and t_b is not the only successor of t_a (cf. Fig. 9(g)), then a parallel split $w_{PS} \in W_{PS}$ and a parallel join $w_{PJ} \in W_{PJ}$ are needed between t_a and t_b. Hence $(t_a \to w_{PS}) \in L_T$, $(w_{PS} \to w_{PJ}) \in L_{W_{PS}}$ and $(w_{PJ} \to t_b) \in L_{W_{PJ}}$ (cf. Fig. 9(h)).

Finally, if an action $a \in A$ has no predecessors/successors (i.e., the set $PREC(a)/NEXT(a)$ is empty), this means that t_a must be connected with the start event $\bigcirc$/end event $\odot$.

A stand-alone software implementation of the module `POP2PT` is available for testing at: `https://goo.gl/z8uJ8S`.

Algorithm 1: $Find_{PREC/NEXT}$ - Find actions predecessors and successors

```
Data:
  – $A$ : the set of actions appearing in the final plan P,
    including dummy actions $a_0$ and $a_\infty$.
  – $O$ : the set containing the ordering constraints returned
    by the planner, in the form $o_k = (a_i \prec a_j)$, with
    $(a_i, a_j) \in A$. Ordering constraints that involve $a_0$ and
    $a_\infty$ are removed.
  – $NEXT(a_i)$ : a set containing the list of successors of the
    i-th action $a_i \in A$.
  – $PREC(a_i)$ : a set containing the list of predecessors of
    the i-th action $a_i \in A$.
  – $k$ : an integer number, used as counter for the ordering
    constraints.
  – $lenght(S)$ : returns the size of a set $S$.
  – $add(a, S)$ : inserts an action $a$ into a set $S$.
Result: for each $a_i \in A$, it returns $NEXT(a_i)$ and
        $PREC(a_i)$
Init:
    $k = 0$
    for each $a_i \in A$ do
        NEXT($a_i$) = $\emptyset$
        PREC($a_i$) = $\emptyset$
begin
    while $k < lenght(O)$ do
        take the k-th ordering constraint,
         $o_k = (a_i \prec a_j)$ from O
        start scanning the O set
        if $\nexists\ (a_h \in A) : a_h \neq a_j \wedge (a_h \prec a_i \in O)$ then
            $add(a_i, NEXT(a_0))$;
            $add(a_0, PREC(a_i))$;
        if $\nexists\ (a_h \in A) : a_h \neq a_i \wedge (a_j \prec a_h \in O)$ then
            $add(a_\infty, NEXT(a_j))$;
            $add(a_j, PREC(a_\infty))$;
        if $\nexists\ (a_h \in A) : a_h \neq a_i \wedge (a_h \prec a_j \in O)$ OR
         $\exists\ (a_h \in A) : a_h \neq a_i \wedge (a_h \prec a_j \in O)$ AND
         $\nexists$ any finite sequence of actions $a_1, a_2, ..., a_n$
         such that
         $(a_i \prec ... \prec a_1 \prec a_2 \prec ... \prec a_n \prec ... \prec a_h)$
         then
            $add(a_j, NEXT(a_i))$;
            $add(a_i, PREC(a_j))$;
        else
            do nothing, because it means that
             $a_i \prec a_h \prec a_j$. The ordering constraint
             $a_i \prec a_h$ will be considered in a future
             iteration of the algorithm.
        $k{+}{+}$
```

Algorithm 2: $Build_{\mathsf{PT}}$ - Build Process Template

Data:
- A : the set of actions appearing in the final plan, including a_0 and a_∞.
- $L = L_T \cup L_E \cup L_{W_{PS}} \cup L_{W_{PJ}}$ is a finite set of transitions connecting events, task instances and gateways. Initially it is empty.
- $NEXT(a_i)/PREC(a_i)$: a set with the list of successors/predecessors of a_i.
- $lenght(S)$: returns the size of a set S.
- $insert(x, S)$: given an element x and a set S, if $x \notin S$ inserts x into S.
- $taskify(a)$: given a planning action $a \in A$, it generates a corresponding task instance $t_a \in T$.

```
begin
    for each $a_i \in A$ do
        $t_{a_i} = taskify(a_i)$
        if $lenght(PREC(a_i)) > 1$ then
            for each $a_q \in PREC(a_i)$ do
                $t_{a_q} = taskify(a_q)$
                if $lenght(NEXT(a_q)) > 1$ then
                    $insert(\{w\_split_q \rightarrow w\_join_i\}, L_{W_{PS}})$
                else
                    $insert(\{t_{a_q} \rightarrow w\_join_i\}, L_T)$
            $insert(\{w\_join_i \rightarrow t_{a_i}\}, L_{W_{PJ}})$
        else
            if $lenght(NEXT(a_q)) > 1$ (let $a_q$ be the only
              predecessor of $a_i$) then
                if $lenght(PREC(a_q)) > 0$ then
                    $insert(\{w\_split_q \rightarrow t_{a_i}\}, L_{W_{PS}})$
                else
                    $insert(\{\bigcirc \rightarrow w\_split_q\}, L_E)$
                    $insert(\{w\_split_q \rightarrow t_{a_i}\}, L_{W_{PS}})$
            else
                if $lenght(PREC(a_q)) > 0$ then
                    $insert(\{t_{a_q} \rightarrow t_{a_i}\}, L_T)$
                else
                    $insert(\{\bigcirc \rightarrow t_{a_i}\}, L_E)$
        if $lenght(NEXT(a_i)) > 1$ then
            for each $a_j \in NEXT(a_i)$ do
                $t_{a_j} = taskify(a_j)$
                if $lenght(PREC(a_j)) > 1$ then
                    $insert(\{w\_split_q \rightarrow w\_join_i\}, L_{W_{PS}})$
                else
                    $insert(\{w\_split_i \rightarrow t_{a_j}\}, L_{W_{PS}})$
            $insert(\{t_{a_i} \rightarrow w\_split_i\}, L_T)$
        else
            if $lenght(PREC(a_j)) > 1$ (let $a_j$ be the only
              successor of $a_i$) then
                if $lenght(NEXT(a_j)) > 0$ then
                    $insert(\{t_{a_i} \rightarrow w\_join_j\}, L_T)$
                else
                    $insert(\{t_{a_i} \rightarrow w\_join_j\}, L_T)$
                    $insert(\{w\_join_j \rightarrow \odot\}, L_{W_{PJ}})$
            else
                if $lenght(NEXT(a_j)) > 0$ then
                    $insert(\{t_{a_i} \rightarrow t_{a_j}\}, L_T)$
                else
                    $insert(\{t_{a_i} \rightarrow \odot\}, L_T)$
```