Steps in Assessing a Timeline-Based Planner

Alessandro Umbrico[1], Amedeo Cesta[2], Marta Cialdea Mayer[1], and Andrea Orlandini[2(✉)]

[1] Dipartimento di Ingegneria, Università degli Studi Roma TRE, Rome, Italy
[2] Consiglio Nazionale delle Ricerche, Istituto di Scienze e Tecnologie della Cognizione, Rome, Italy
andrea.orlandini@istc.cnr.it

Abstract. The "timeline-based" is a particular paradigm of temporal planning that has been successfully applied in many real-world scenarios. Different timeline-based planning systems have been developed, each using its own planning specification language and solving techniques. An analysis of the differences between such kind systems has not been addressed yet. In previous work we have developed EPSL a planning tool successfully applied in real-world manufacturing scenarios. During subsequent projects our tool achieved a level of stability and a relative maturity. In this paper we start addressing the problem of comparison with other timeline-based planners and presents an analysis that concerns the EUROPA2 framework which can be considered the *de-facto* standard for timeline-based planning. In the present work we analyze the modeling and solving capabilities of the two frameworks. This phase of our study identifies differences and discusses strengths and weaknesses when solving the same problem.

Keywords: Timeline-based planning · Planning and Scheduling · Constraint-based planning

1 Introduction

Timeline-based planning is an approach to temporal planning research which has been successfully applied to real-world problems [1–4] where time constitutes a crucial factor to deploy effective planning applications. The main feature of the approach stems in the capacity of modeling and dealing with temporal constraints and in the capability of integrating planning and scheduling (P&S) in a unified solving approach. Indeed, a lot of the reasons for its success stay in the modeling capability of the systems that support such applications. Several applications are supported by various timeline-based general purpose architectures, some of the most known are EUROPA2 [5], IXTET [6], ASPEN [7] and APSI [8]. Despite the practical success there is not a uniform shared view of what timeline-based planning is. Thus each existing framework applies its own *interpretation* of the planning approach. In contrast to action-based planning, theoretical aspects of timeline-based planning were not investigated until very

G. Adorni et al. (Eds.): AI*IA 2016, LNAI 10037, pp. 508–522, 2016.
DOI: 10.1007/978-3-319-49130-1_37

recently. A formal description of the problem has been proposed in [9], while a formalization in terms of *flexible* timelines appeared in [10], later extended in [11] to account also for plan controllability issues. Meanwhile, the connection between timelines and Timed Game Automata has been investigated for the purpose of plan verification [12,13] and robust plan execution [14]. Also, initial steps for a complexity-theoretic characterization of the planning problem has been recently proposed in [15].

As a counterpart of the formal work presented in [11], we have developed a general purpose timeline-based planning architecture, called EPSL (Extensible Planning and Scheduling Library), proposing a hierarchy-based approach for modeling and solving timeline-based problems [16]. Such system has been successfully tested during subsequent research projects to support a manufacturing plant (see [4,17,18]) and an industrial robotics scenario [19]. Through the use in these projects the EPSL tool has achieved both a level of stability and a relative maturity.

In this paper, we start addressing the problem of comparing EPSL with other timeline-based planners and present an analysis that concerns EUROPA2, a planning framework developed at NASA [5] which, given also the wide spectrum of missions that have used it, can be considered a *de-facto* state of the art for timeline-based planning systems. The goal of the paper is to provide the reader with an initial report about the differences between the two frameworks taking into account their modeling and solving capabilities. Namely, rather than focusing on a comparison of performances, we aim at understanding the different features of the frameworks in order to highlight weak and strength points of the approach we are pursuing. A general interesting result is that the development of EUROPA2 seems to have led the NASA's framework to assimilate some features of "classical" PDDL-like approach to planning. Conversely, EPSL framework maintains a modeling and solving approach inspired by the original idea of timeline-based planning as introduced in [1].

Plan of the Paper. The next section of the paper provides a brief description of the timeline-based approach also introducing a general formal framework. Then, we describe the features of the EPSL planning tool and the EUROPA2 framework. We also introduce a ROVER planning domain exploited as the case study for the comparison. The following section goes into details for the comparison considering the planning domain models, the solving approaches and some features of the generated plans. Finally, the paper ends with some conclusions concerning the overall work and future developments.

2 Timeline-Based Planning

The modeling assumption underlying the timeline-based planning approach is inspired by classical Control Theory: the problem is modeled by identifying a set of *relevant features* whose behavior over time is to be controlled in order to obtain a desired set of goals. In this respect, the domain features under control

are modeled as a set of temporal functions whose values have to be decided over a temporal horizon. Such functions are synthesized during problem solving by posting planning decisions. The evolution of a single temporal feature over a time horizon is called the *timeline* of that feature. The temporal behavior of an element of the domain is described in shape of a timeline, which is a sequence of values (states or actions) the related element of the domain can assume over time. Thus the set of the timelines of a domain (called the "timeline-based plan") describes the behavior of the overall system. A planner synthesizes timeline-based plans by posting temporal constraints between states or actions according to some domain rules that model the physical and logical constraints of the system and its elements.

2.1 Planning with Flexible Timelines

Despite the practical success of timeline-based approach to solve real world problems, a shared view and a well-defined formalization of the main planning concepts were missing (but see pointers in the introduction). For the purpose of this paper, we refer to the generic timeline-based planning framework presented by Cialdea Mayer et al. [11]. A timeline-based planning domain is composed by a set of features to be controlled over time. These features are modeled by means of *multi-valued state variables* that specify causal and temporal constraints characterizing the allowed temporal behavior of such domain features. A state variable describes the set of values $v \in V$ the related feature may assume over time with their flexible duration. For each value $v_i \in V$ the state variable describes also the set of values $v_j \in V$ (where $i \neq j$) that are allowed to follow v_i and the related *controllability property*. If a value $v \in V$ is tagged as *controllable* then the system can decide the actual duration of the value. If a value $v \in V$ is tagged as *uncontrollable* instead, the system cannot decide the duration of the value, the value is under the control of the *environment*. The behavior of state variables may be further restricted by means of *synchronization rules* that allow to specify temporal constraints between different values. Namely, while state variables specify *local* rules for the single features of the domain, synchronizations represent *global* rules specifying how different features of the domain must behave together. A *planning domain* is composed by a set of state variables and a set of synchronization rules. Specifically, there are two types of state variable in a planning domain. The *planned variables* that model the domain features the system can control (or partially control). The *external variables* that model domain features completely outside the control of the system. External state variables model features of the *environment* the system cannot control but that must care about in order to successfully carry out activities.

Planning with timelines usually entails considering sequence of valued intervals and time flexibility is taken into account by requiring that the durations of valued intervals, called *tokens*, range within given bounds. In this regard, a flexible plan represents a whole set of non-flexible timelines, whose tokens respect the (flexible) duration constraints. However a set of flexible timelines do not convey enough information to represent a flexible plan. The representation of a flexible

plan must include also information about the relations that must hold between tokens in order to satisfy the synchronization rules of the planning domain. A flexible plan Π over the horizon H is defined by a set of flexible timelines FTL and a set of temporal relations R representing a possible choice to satisfy the synchronization rules. Given the concepts above, a *planning problem* is defined by a temporal horizon H, a planning domain D, a planning goal G which specifies a set of tokens and constraints to satisfy and the *observations* O which completely describes the flexible timelines for all the external variables of the domain. Consequently a flexible plan Π is a solution for a planning problem if it satisfies the planning goal and if it does not make any hypothesis on the behavior of the external variables (i.e. the plan does not change the *observation* of the problem).

Moreover, given a solution plan Π, it is important to check the controllability properties of Π in order to verify the executability of the plan. The controllability problem aims at verifying if there exists a way to execute a plan according to the possible (temporal) evolutions of the environment [20]. Namely controllability properties are to define a set of decisions (i.e., a feasible temporal allocation of all plan's controllable events/intervals - *schedule*) that guarantee the execution of a plan according to the known possible evolutions of the environment (i.e., uncontrollable events/intervals of a plan). Different controllability properties have been introduced differing for the assumptions made on the known evolutions of the environment [20]. The most relevant property is the *dynamic controllability* of a plan which entails that a *dynamic execution strategy* exists to dynamically decide the schedule of the controllable intervals/events of a plan by reasoning on the perceived evolution of the environment. From a planning point of view, it is also important to check that partial plans maintain such property overall the solving process.

2.2 EPSL: A General-Purpose P&S Framework

The Extensible Planning and Scheduling Library (EPSL) [16] is the result of a research effort started after the analysis of different timeline-based systems (e.g., [5,6,21]) as well as some previous experiences in deploying timeline-based solvers for real world domains [22]. EPSL relies on the APSI framework [8] which provides the modeling capabilities to represent timeline-based domain. In particular, EPSL extends the APSI framework in order to comply with the semantics proposed in [11]. EPSL provides a modular software library which allows users to easily define timeline-based planners by specifying strategies and heuristics to be applied in a specific application.

The key point of EPSL modularity is the *planner interpretation*. A planner is a compound element whose solving process is affected by the specific set of *modules* applied. Indeed, a EPSL-based planner implements a plan refinement search by combining several modules responsible for managing different aspects of the search: (i) a *search strategy* for managing the fringe of the search space; (ii) a set of *resolvers* for detecting and solving different types of flaws on the plan; (iii) a *selection heuristics* for analyzing the flaws detected on the current plan

and selecting the *best* flaw to solve for plan refinement. Thus the actual behavior of the solving process of a EPSL-based planner is determined by the particular strategy, the heuristics and the resolvers set in the configuration of the P&S application. In this regard, the set of resolvers determines the expressivity of the framework. Adding new resolvers allows the framework to manage new types of feature of the domain (e.g., different type of resources) and detect/solve a wider range of flaws of a plan. Heuristics affect the performances of the solving process by encapsulating a *flaw-selection criteria* which guides the plan-refinement procedure to select the most *promising* flaw to solve at each iteration. Finally, the strategies may affect the *qualities* of the generated plans by encapsulating plan evaluation criteria that estimate some desired properties of the (partial) plans (e.g. plan cost).

Several EPSL-based solvers have been deployed in the real-world scenarios mentioned before [4,18,19]. Specifically, we have applied a hierarchy-based modeling and solving approach which identifies two types of state variables (in addition to the external ones described in the formalization) [16]. The *primitive variables* model the set of *low-level tasks* that can be directly executed by the system to control. The *functional variables* model the *complex tasks* that can be performed by combining the available primitive ones. Namely functional variables abstract the behavior of the system by modeling the functional capabilities it can perform. Synchronization rules define a *hierarchical task decomposition* which decomposes the values of functional variables in terms of temporal constraints between values of primitive variables (complex domains may have several *functional layers* between the top of the hierarchy and the primitive layer). The resulting hierarchical structure of the domain is then exploited by the solving process by means of a domain independent heuristics which allows to improve the performance of the framework as shown in [16]. During its solving process, EPSL also performs a *pseudo-controllability* check of partial plans as a necessary but not sufficient requirement for guaranteeing dynamic controllability.

The modular architecture of EPSL allows to easily extend the solving capabilities of the framework by adding new modules (e.g. heuristics or resolvers). Thus EPSL provides an enhanced framework for developing applications in which designers may focus on single aspects of the solving process without dealing with all the details related to timeline-based planner implementation.

2.3 The EUROPA2 Framework

With this paper, we start addressing the problem of comparing EPSL with other timeline-based planners. The natural starting point is to focus on the EUROPA2 framework which can be considered the *de-facto* standard for timeline-based planning. EUROPA2 is one of the most known timeline-based tools in the literature, its most known incarnation has been in the DS1 mission [2], attempts of formalization are given in [23,24], the more recent, public domain version is described in [5].

Similarly to EPSL, EUROPA2 models a planning domain by identifying a set of features to control over time. The modeling approach in EUROPA2 relies on an

object-oriented language, the *New Domain Description Language* – NDDL. An *object* models a specific feature of the domain. The behavior of an object can be described by means of *predicates* and/or *actions* that represent respectively states or operations the related feature can assume or perform over time. However not all objects behave in the same way. Indeed some types of objects may have specific rules that constrain their temporal evolutions. A *timeline* is a particular type of object whose temporal behavior is constrained to be a sequence of not overlapping values (i.e. predicates) over time.

EUROPA2 allows to model also *renewable resources* and *consumable resources* in addition to *general objects* and *timeline objects*. Renewable resources represent *shared* features of the domain with a limited capacity that can be *consumed* over time (e.g., a pool of workers in a manufacturing environment, or a communication channel). Namely no *production* of the resource is needed because the amount of resource consumed by an activity is restored as soon as the activity ends. Consumable resources represent *shared* features of the domain with a limited capacity that can be either *consumed* or *produced* over time (e.g., a battery). In this case *production* activities are needed in order to restore the capacity of the resource after *consumptions*. Objects of the domain declare the predicates or actions they can assume over time. The temporal behaviors of domain objects are constrained by means of *compatibilities*. Broadly speaking a compatibility represents a general rule composed by the *head* which represents the predicate or action the rule applies to, and a *body* which specifies a set of predicates/values of other objects together with a set of temporal and parameter constraints that must hold between the *head* of the rule and the target of the constraint.

Given a domain specification, a planning problem is composed by a *temporal horizon* and an *initial configuration*. The initial configuration describes the initial (partial) plan in terms of a set of predicates on the timelines and a set of goals that can be either actions to perform or predicates to achieve (i.e., a desired final state). Note that the term *timeline* is used to refer either to the domain objects or the temporal evolutions of all the objects of the domain. EUROPA2 applies an action-based modeling approach where *general objects* specify actions to constrain the predicates of the timelines of the domain. Thus given a set of domain objects and compatibilities, the solving process generates the temporal behaviors of the timelines according to the compatibilities of the planning domain and the specified goals. The EUROPA2 solving process relies on a constraint-based engine to encode a partial-plan refinement procedure. As described in [25], the planner implements a plan refinement search which starts from the initial configuration and incrementally refines the related plan by adding and ordering predicates or actions to the timelines until a final consistent configuration is found. Namely the refinement process consists of detecting and solving *flaws* on the current partial plan. EUROPA2 is able to manage three types of flaw during the solving process: (i) *open condition flaws* represent (sub)goals generated during the planning process; (ii) *ordering flaws* represent overlapping predicates of a timeline; (iii) *unbound variable flaws* represent variables of the underlying CSP engine that must be instantiated.

A feature of Europa2 solver worth being mentioned is that a planning goal can be either an action to perform or a desired *state* to reach. In the latter case the solving process applies all the actions defined in the model to *support* the desired state. Namely the solver acts like in PDDL planning where operators are applied to preconditions according to the desired effects.

3 Comparing Epsl and Europa2

In order to assess the actual maturity of Epsl, here an initial comparison with Europa2 is performed evaluating different aspects. Indeed, the objective is to compare the two timeline-based frameworks by taking into account aspects concerning the modeling approach, the expressiveness, the solving capabilities and the features of the generated plans. In this regard, the comparison between Epsl and Europa2 entails two steps: first, an analysis of different modeling capabilities considering the features of the planning domain models exploited by the systems; then, a comparison of the differences in the solving approaches shown by the two planners. To this aim, we consider a well known planning domain, i.e., a Rover planning domain, as a reference domain to analyze the different modeling features as well as to discuss the main differences on the planning processes also considering planning problems of growing complexity.

The Rover planning domain has been extracted from the scenario described on the Europa2's web site concerning an autonomous exploration rover[1]. This scenario represents a well known application context in AI [26,27]. Specifically, a rover is a robotic device endowed with a wheeled base to explore the environment, an instrument to sample rocks and collect scientific data that can be communicated back to Earth. The domain consists of a rover which must navigate between known points of interest and collect scientific data by means of payload instruments (e.g., camera) and communicate such data to Earth. Usually, some requirements are to be satisfied during the execution of a mission in order to successfully carry out tasks and guarantee the operational requirements of the rover. In the Rover domain, the mission plans must satisfy the following requirements: (i) the instruments of the rover must be set in a safe position (i.e., *stowed*) while the rover is moving; (ii) the rover must be still at a requested location and place the instrument accordingly in order to take a sample of the target (e.g., a rock); (iii) the rover must not move when communicating data to Earth.

3.1 Comparing Modeling Capabilities

Although following the same planning approach, Apsi and Europa2 use different ways for modeling the domain features. They rely on two different domain specification languages to model planning domains, i.e., DDL and Nddl respectively. And, given the Rover planning domain, two different models are then

[1] https://github.com/nasa/europa/tree/master/examples/Rover.

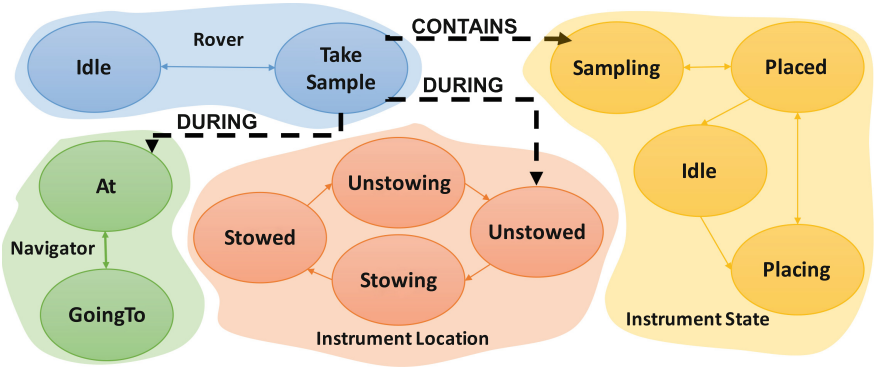

Fig. 1. EPSL model of the ROVER planning domain

analyzed according to the features of the specific languages. A preliminary analysis of the modeling capabilities suggests some main relevant differences concerning the *expressiveness* of the two frameworks. Indeed, EUROPA2 allows to model a wider range of domain features than EPSL. EUROPA2 can model *consumable resources* while EPSL cannot. For instance, EPSL is not able to model the *battery* of the ROVER planning domain. On the other hand, EUROPA2 does not allow to model *uncontrollable* features in a planning domain while EPSL allows to specify *external variables* to model features of the environment to *monitor*. Namely, EPSL can model a *visibility window* with a ground station on Earth in order to allow the rover control system to plan scientific data communication tasks within given time periods. For comparison purposes, a revised version of the ROVER scenario is defined in order to obtain planning models of *equivalent complexity* for the two planners. Thus, the original ROVER planning domain has been *simplified* by not considering *battery management* and *communication activities*[2].

A model of the ROVER *domain in* EPSL. EPSL allows to define planning domain by defining a set of state variables to be controlled over time and a set of synchronization rules to coordinate their temporal behaviors. Figure 1 shows the set of state variables defined to model the ROVER domain. The figure shows also temporal constraints entailed by the synchronization rules of the domain that allow to satisfy the goals (i.e., take samples). In general, EPSL allows to follow a *hierarchical approach* to domain modeling which starts by identifying a set of *primitive variables* that model the primitive/atomic tasks the system may directly execute. Then, *functional variables* are defined to model complex tasks, called *functions*, the system can perform over time by composing the primitive ones. Namely, functions represent complex tasks that cannot be directly

[2] Examples of the DDL and NDDL planning specification files for the ROVER domain considered in this paper can be found at the following link: http://tinyurl.com/TLRoverDomains-zip.

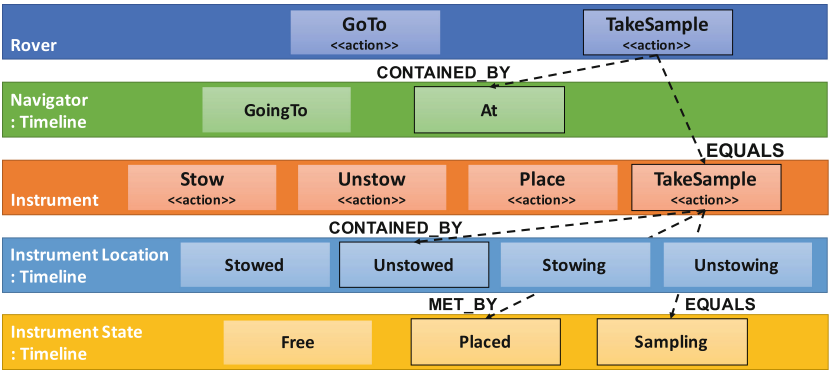

Fig. 2. EUROPA2-based model of the ROVER planning domain

performed by a single component of the system. Rather functions entail a coordination among system's (internal) devices (i.e., the primitive variables of the planning domain). Synchronization rules describe the *hierarchical decomposition* of the agent's functions in terms of *primitive tasks* the system's devices can directly handle. Complex domains may require different hierarchical levels. Then, a *hierarchical decomposition* may involve both primitive tasks and other functions. Thus, the EPSL modeling approach uses synchronizations to perform hierarchical task decomposition similarly to classical HTN planning.

In Fig. 1, the *Navigator* is a *primitive variable* which models the navigation facility of the rover. The *At* and *GoingTo* values represent that the rover can either be still at a known location or moving towards another location. Similarly, the *Instrument State* is a *primitive variable* which models the operating state of the rover's instrument. The *Rover* component of the domain is a *functional variable* which models the set of functions the rover may perform. The associated *TakeSample* value represents a high-level task (i.e., a function) the rover must perform by coordinating the behaviors of *Navigation*, *Instrument Location* and *Instrument State* variables. A dedicated synchronization rule specifies the set of temporal constraints that must hold to perform the *TakeSample* function (see the black dotted arrows in Fig. 1). Then, a consistent (temporal) behavior for taking a sample (i.e., *Sampling* value of *Instrument State* variable) requires that the rover is located at the target's location (i.e., *At*), the instrument is active (i.e., *Unstowed* value of *Instrument Location* variable).

A model of the ROVER *domain in* EUROPA2. Figure 2 depicts the EUROPA2 model generated for the ROVER planning domain. EUROPA2 uses an object-oriented modeling language to represent the features of a planning domain. The *objects* of the domain are described in terms of *predicates* and *actions*. Predicates represent the states that objects can assume over time. Actions represents the operations that objects can perform over time. There are two types of objects that compose a planning domain. *Timeline Objects* model the features of a domain that may change over time, e.g. the physical position of the rover

or the state of an instrument. *Objects* model the set of *actions* that can be performed to change the state of one or a set of Timeline Objects, e.g. the action for moving the rover from an initial position to a destination position. Then, for each action, *compatibilities* specifies the set of constraints that allow to build the plan. Namely, compatibilities specify the constraints affecting the temporal evolutions of one (or more) Timeline Object(s), e.g. a compatibility for a move action specifies that the rover must be at the initial position *before* the action start, and that must be at the destination position *after* the execution of the action.

Considering the model in Fig. 2, the instrument facility is modeled by means of two timelines, i.e., the *InstrumentState* and the *InstrumentLocation* timelines. They model the set of states and positions the instrument may assume over time. The *Instrument* object provides actions for controlling the device. The predicates of the *InstrumentState* timeline model the "operational status" of the device over time. The predicate *Placed* means that the instrument is placed on target. The predicate *Sampling* means that the instrument is sampling a particular target. Similarly the predicates of *InstrumentLocation* timeline model the position of the device over time. The predicate *Stowed* means that the instrument is stowed and it cannot perform sampling operations. The predicate *Unstowed* means that the instrument is ready to use. The actions of *Instrument* object model operations needed to properly manage the device. The action *Unstow* represents the operation which allows to "activate" the device by changing the position of the instrument from *Stowed* to *Unstowed*. Similarly the action *TakeSample* represents the operation which allows the device to actually take a sample of a desired target. The black dotted arrows in Fig. 2 represent some of the temporal constraints required by the compatibilities defined on the corresponding actions of the domain. In this regard it is important to point out that actions have *conditions* and *effects* that must hold to apply and *execute* the action. *Effects* represent predicates that the execution of the action adds to the plan. *Conditions* represent predicates that must be part of the plan in order to "execute" the action. This is an important aspect to take into account while modeling planning problems with EUROPA2. Indeed, an action-based planning perspective is actually pursued while solving problems (see next section for further considerations). That is, the EUROPA2 planner checks conditions and effects of actions in order to find a suitable sequence of actions that allow to build timelines and satisfy the desired goals.

Also, some differences can be noted between the formalization of EUROPA2 given in [24] and its actual implementation. These differences mainly concern the compatibility specification and their expressiveness w.r.t. synchronizations of EPSL. Indeed, from a theoretical point of view, compatibilities are less expressive than synchronizations because they do not allow to specify constraints between tokens of the rule's body. Namely, a compatibility can only specify constraints between the head of the rule and tokens in the rule's body. Moreover, unlike EPSL, EUROPA2 does not use quantified Allen's temporal relations. However, the "concrete implementation" of the framework overcomes all these (theoretical)

limitations by exploiting the underlying CSP engine. Indeed, it is possible to explicitly constrain compatibilities' tokens by specifying CSP's linear constraints between the tokens' temporal variables (i.e., tokens time point and duration variables). For instance, it is possible to specify a *before* temporal constraint between two tokens in the body of a compatibility by adding a linear constraint between the end time of the first token and the start time of the second token (e.g. $endTime(a) < startTime(b)$ where a and b are tokens declared into the body of a compatibility).

3.2 Comparing Solving Capabilities and Generated Plans

Both EPSL and EUROPA2 apply a plan refinement procedure which starts with an initial partial plan and some goals to plan for. The solving process iteratively refines the plan by solving *flaws* until a complete and valid plan is found [16,25]. However there are some relevant differences worth to be underscored (also related to the different modeling approaches). Given the ROVER planning models described in the previous section, we have defined several problem instances by considering an increasing number of planning goals (i.e., the number of targets to sample) to compare the solving capabilities of EPSL and EUROPA2. On these problems, the collected results show that EPSL performs rather better than EUROPA2 in terms of deliberation time. Obviously, this is not sufficient to support any general claim about the actual effectiveness of the planning systems. Providing a complete analysis entails to refer a set of benchmark domains with multiple problem instances for timeline-based planning and, to the best of our knowledge, such benchmark is still missing. Thus, a thorough comparison of the solving performances provided by the two frameworks is kept outside the scope of this work and left as future work. In this paper, the main objective of the experiments is to check the suitability of the defined models and assess the features of the plans generated by the two frameworks. Therefore, the most important result elicited from the comparison concerns, again, the different interpretation of timeline-based planning in EUROPA2 and EPSL frameworks. Indeed, despite they share the same conceptual origin (see [1]), they have developed two different ways of handling timeline-based problems.

The experimental campaign shows a first difference between EPSL and EUROPA2 concerning the *interpretation* of a solution plan. Namely, EPSL generates plans providing a set of timelines as a continuous sequences of (temporally) ordered tokens and a set of temporal constraints that relate their start and end times, while EUROPA2 interprets timelines as "discrete" sequence of ordered values. Namely, timelines may contain *gaps* according to EUROPA2's interpretation. The actual presence of gaps on the plans generated by EUROPA2 depends from the planning model specification provided as input. The user is then responsible for specifying a set of compatibilities that avoid gaps in the solution plans. On the contrary, the responsibility of filling gaps in EPSL is on the planning algorithm. Then, EPSL exploits state variables of the planning domain to guarantee consistent behaviors of the domain features. And every time a gap (i.e., a temporal interval with no value) is detected between two tokens, the EPSL solving

algorithm checks the state variable specification to extract the allowed transition between the values involved. This interpretation relies on the assumption that a gap represent an *uncertainty* about the actual behavior of the feature because it could assume any value in the related *unbounded* temporal intervals. A key point of this aspect is that the planner is not only responsible for applying the constraints specified by the user but it is also responsible for ensuring consistent transitions between the values according to state variable specification. In EUROPA2 such a behavior must be achieved by specifying a set of actions and related compatibilities that model the possible transitions of the objects. Such an operation is not always simple and, thus, the correctness of the generated plans strongly relies on the expertise of the user actually modeling the planning domain.

An interesting feature of the EUROPA2 modeling approach is the use of general objects to model actions of the domain. Objects can be seen as *relaxed* timelines where tokens are allowed to overlap in time. Thus, EUROPA2 planner can generate plans with parallel actions if they do not violate the related compatibilities' constraints. On the contrary, EPSL relies on state variables to model functional variables and the related tasks. Thus, *complex tasks* cannot overlap in time even if the related synchronization rules would allow parallelism. Let us suppose, for example, that the rover of the planning domain is endowed with two instruments and that the rover must sample two targets at the same location. In such a scenario, the rover should be able to perform the two *TakeSample* tasks in parallel. EUROPA2 models the *TakeSample* as an action of an object. Thus, the planner can generate plans where the rover performs the two planning goals (i.e., two *TakeSample* actions) in parallel by allowing the related tokens to overlap. Conversely, EPSL models *TakeSample* as a value of a functional variable (i.e., a function of the rover) and the related tokens of the timeline are not allowed to overlap. Thus, the planner can only generate plans where the two goals are in sequence.

3.3 Easy of Use

As discussed above, a main general comment is that EUROPA2 seems to have been influenced by "classical" PDDL-like planning techniques that lead the framework move towards an action-based approach rather than a "behavior-based" approach to planning with timelines. This mainly affect the modeling approach of EUROPA2 framework. The user must be aware of the solving process of the framework defining a suitable set of actions (and compatibilities) that allow the planner to build a valid plan. As a consequence, the user is supposed to completely specify how the planner can build the timelines of the plan. Conversely EPSL approach is compliant with the original idea of timeline-based planning where the focus is on the temporal behavior of the domain features to control. Indeed, state variables model the features of the domain by describing how they can *autonomously* evolve over time. Synchronizations allow the user to constrain these possible evolutions (i.e., temporal behaviors) to coordinate components and realize some complex operations. Thus the user is supposed to "simply" declare the values the domain components must assume over time in order to realize the desired (complex) behavior. The EPSL solving process is

then responsible for building the timelines according to the requirements of the domain and the desired goals.

As an example, in the ROVER domain, the EUROPA2 user is supposed to provide a complete specification while the EPSL user can simply declare that the rover must be at the target's location *during* sampling operations in order to successfully perform a *TakeSample* task. Then, in EPSL it is not necessary to declare the "rule" which allows the rover to reach the target's locations. Such a rule is encoded by the *Navigator* state variable. And the EPSL planner is responsible for checking whether the rover must move or not towards the desired location, and building the related timeline accordingly. In other words, as envisioned in the original idea of timeline-based planning [1], the crucial point is that users of the planning framework are not supposed to be aware of the internal functioning of the particular planning algorithm/technique adopted to deploy effective P&S applications. Users can be an expert of the particular application domain without being *forced* to know the details of the solving mechanism of the planner. This is a very important feature of a planning framework and it represent a *long-term* goal we are pursuing in order to design a general purpose tool which can be easily exploited by *end-users*. Thus most of the EUROPA2 modeling and solving capabilities rely on the expertise of the user and, as a consequence, a deep knowledge of the solving mechanism of the EUROPA2 framework is requested in order to develop effective P&S applications. Consequently we may argue that EUROPA2 approach to timeline-based planning seems to be harder to apply than the EPSL approach. However it is important to point out that EUROPA2 gives to the (skilled) users a total control of the plan generation process.

4 Conclusions

In this paper we have summarized some recent results in the development of a general-purpose timeline-based planning framework, called EPSL. Then we have described the approach followed to compare EPSL with EUROPA2, the most known timeline-based planning framework in the literature. We have analyzed the modeling and solving capabilities of the frameworks by taking into account a ROVER domain, which represents a "classical" planning domain extracted from a real-world application scenario. Despite EPSL and EUROPA2 share the same origin of timeline-based planning [1], the evaluation has pointed out some relevant differences between the two frameworks. Indeed, we have found significant differences in terms of both the modeling capabilities and the solving approach. After the assessment, we can conclude that the most relevant difference between the frameworks concerns the *usage* and the *level-of-expertise* needed to develop effective P&S applications. In our opinion, EPSL unlike EUROPA2, does not require a deep understanding of the solving process to design P&S applications. Thus EPSL seems to be easier to use than EUROPA2 for *end-users* that may have a deep knowledge of the specific application domain but not a good background in AI planning and the related planner applied. However the assessment has also pointed out some deficiencies in EPSL that we are going to address in the next future to improve the framework (e.g., introducing *consumable resources* and

taking into account *functional variables* concurrency issue). This paper provides an initial report that aims at starting an evaluation of the features and capabilities of the EPSL framework we are developing. Future work will extend the evaluation by addressing performance features with other timeline-based planning systems and taking into account also the "new generation" of planning frameworks like CHIMP [28] and FAPE [29].

References

1. Muscettola, N.: HSTS: Integrating planning and scheduling. In: Zweben, M., Fox, M.S. (eds.) Intelligent Scheduling. Morgan Kauffmann (1994)
2. Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in interplanetary space: theory and practice. In: Proceedings of the Fifth International Conference on AI Planning and Scheduling. AIPS-00 (2000)
3. Py, F., Rajan, K., McGann, C.: A systematic agent framework for situated autonomous systems. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. AAMAS-10 (2010)
4. Borgo, S., Cesta, A., Orlandini, A., Umbrico, A.: A planning-based architecture for a reconfigurable manufacturing system. In: The 26th International Conference on Automated Planning and Scheduling (ICAPS) (2016)
5. Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., Smith, D.: EUROPA: a platform for AI planning, scheduling, constraint programming, and optimization. In: The 4th International Competition on Knowledge Engineering for Planning and Scheduling. ICKEPS 2012 (2012)
6. Ghallab, M., Laruelle, H.: Representation and control in IxTeT, a temporal planner. In: 2nd International Conference on Artificial Intelligence Planning and Scheduling (AIPS), pp. 61–67 (1994)
7. Chien, S., Rabideau, G., Knight, R., Sherwood, R., Engelhardt, B., Mutz, D., Estlin, T., Smith, B., Fisher, F., Barrett, T., Stebbins, G., Tran, D.: ASPEN - automated planning and scheduling for space mission operations. In: Proceedings of Space Ops 2000 (2000)
8. Cesta, A., Fratini, S.: The timeline representation framework as a planning and scheduling software development environment. In: Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group. PlanSIG-08, Edinburgh, 11–12 December 2008
9. Cimatti, A., Micheli, A., Roveri, M.: Timelines with temporal uncertainty. In: 27th AAAI Conference on Artificial Intelligence (AAAI) (2013)
10. Cialdea Mayer, M., Orlandini, A., Umbrico, A.: A formal account of planning with flexible timelines. In: The 21st International Symposium on Temporal Representation and Reasoning (TIME), pp. 37–46. IEEE (2014)
11. Mayer, Cialdea: M., Orlandini, A., Umbrico, A.: Planning and execution with flexible timelines: a formal account. Acta Informatica **53**(6), 649–680 (2016)
12. Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Flexible timeline-based plan verification. In: Mertsching, B., Hund, M., Aziz, Z. (eds.) KI 2009. LNCS (LNAI), vol. 5803, pp. 49–56. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04617-9_7
13. Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Analyzing flexible timeline plan. In: Proceedings of the 19th European Conference on Artificial Intelligence. ECAI 2010, vol. 215. IOS Press (2010)

14. Cialdea Mayer, M., Orlandini, A.: An executable semantics of flexible plans in terms of timed game automata. In: The 22nd International Symposium on Temporal Representation and Reasoning (TIME). IEEE (2015)
15. Gigante, N., Montanari, A., Cialdea Mayer, M., Orlandini, A.: Timelines are expressive enough to capture action-based temporal planning. In: The 23rd International Symposium on Temporal Representation and Reasoning (TIME). IEEE (2016)
16. Umbrico, A., Orlandini, A., Mayer, M.C.: Enriching a temporal planner with resources and a hierarchy-based heuristic. In: Gavanelli, M., Lamma, E., Riguzzi, F. (eds.) AI*IA 2015. LNCS (LNAI), vol. 9336, pp. 410–423. Springer, Heidelberg (2015). doi:10.1007/978-3-319-24309-2_31
17. Carpanzano, E., Cesta, A., Orlandini, A., Rasconi, R., Suriano, M., Umbrico, A., Valente, A.: Design and implementation of a distributed part routing algorithm for reconfigurable transportation systems. Int. J. Comput. Integr. Manuf. (2015) http://www.tandfonline.com/doi/full/10.1080/0951192X.2015.1067911
18. Borgo, S., Cesta, A., Orlandini, A., Rasconi, R., Suriano, M., Umbrico, A.: Towards a cooperative knowledge-based control architecture for a reconfigurable manufacturing plant. In: 19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE (2014)
19. Cesta, A., Orlandini, A., Bernardi, G., Umbrico, A.: Towards a planning-based framework for symbiotic human-robot collaboration. In: 21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE (2016)
20. Morris, P.H., Muscettola, N., Vidal, T.: Dynamic control of plans with temporal uncertainty. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 494–502 (2001)
21. Fratini, S., Pecora, F., Cesta, A.: Unifying planning and scheduling as timelines in a component-based perspective. Arch. Control Sci. **18**(2), 231–271 (2008)
22. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: MrSPOCK: steps in developing an end-to-end space application. Comput. Intell. **27**(1), 83–102 (2011)
23. Frank, J., Jonsson, A.: Constraint based attribute and interval planning. J. Constraints **8**(4), 339–364 (2003)
24. Bernardini, S.: Constraint-based temporal planning: issues in domain modelling and search control. PhD thesis, Università degli Studi di Trento (2008)
25. Bernardini, S., Smith, D.E.: Towards search control via dependency graphs in Europa2. In: ICAPS Workshop on Heuristics for Domain Independent Planning (HDIP) (2009)
26. Bresina, J.L., Jónsson, A.K., Morris, P.H., Rajan, K.: Activity planning for the mars exploration rovers. In: International Conference on Automated Planning and Scheduling (ICAPS), pp. 40–49 (2005)
27. Fratini, S., Cesta, A., De Benidictis, R., Orlandini, A., Rasconi, R.: APSI-based deliberation in Goal Oriented Autonomous Controllers. In: 11th Symposium on Advanced Space Technologies in Robotics and Automation. ASTRA-11 (2011)
28. Stock, S., Mansouri, M., Pecora, F., Hertzberg, J.: Online task merging with a hierarchical hybrid task planner for mobile service robots. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6459–6464, September 2015
29. Dvorák, F., Barták, R., Bit-Monnot, A., Ingrand, F., Ghallab, M.: Planning and acting with temporal and hierarchical decomposition models. In: 2014 IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 115–121, November 2014