Toward a general purpose software environment for timeline-based planning

Amedeo Cesta, Andrea Orlandini, and Alessandro Umbrico

CNR - Consiglio Nazionale delle Ricerche, ISTC, Rome, Italy

Abstract. Timeline-based Planning and Scheduling applications have been successfully deployed in various contexts. Often such applications use specific solving algorithms and cannot be easily applied for solving different kind of problems. An open research issue for such planning modeling is the one of creating a software infrastructure with a controllable search engine. This paper presents an attempt to synthesize such a software environment. The *Extensible Planning and Scheduling Library* (EPSL) evolves from the *Timeline Representation Framework* (APSI-TRF) a software environment supported by the European Space Agency. Goal of EPSL is to obtain a software architecture having the flexibility to focus on specific problem solving aspects. The paper is an initial report on this effort: it introduces the whole idea, then focuses on the definition of suitable heuristic functions, and presents experiments related to two domains generated by current applications.

1 Introduction

Timeline-based planning has been shown very effective for applications in real-world domains – see examples in space like [1,2,3]. On the side of these practical works several timeline-based planning and scheduling (P&S) environments have been defined with the goal of acting as a seed for facilitating the synthesis of new domain specific planners – see as examples EUROPA [4,5], ASPEN [6], APSI-TRF [7]. Some work has been dedicated to show similarities between timeline-based and classical planning [8,4] (features already operational in IxTeT [9]). Other recent work is dedicated to determine similarities between different approaches to achieve a synthesis useful for applications [10].

Indeed, an open problem is the one of importing within the timeline-based environments the capabilities for speeding up search that have been developed in the last ten years for PDDL-based planners. The most known environments for application development (EUROPA, ASPEN, and APSI-TRF) have limited abilities in this respect. The applications they contribute to realize are typically closely connected to the domains for which they have been made and are hard to adapt to other kind of problems. Some works on EUROPA have described the general search algorithm [11], or specifically have tried to integrate heuristic search features [5].

Our group has been working within the Advanced Planning and Scheduling Initiative (APSI-TRF) promoted by the European Space Agency. Goal of the initiative was the design and realization of a framework for mission planning development able to integrate AI-based planning modules, in order to improve the flexibility of the mission planning systems, to increase the automation of the planning process, and to generate plans more robust with respect to execution uncertainty and adaptable to change. Starting from the observation that a lot of effort is usually spent in designing, implementing and testing software components that could be reusable among different applications that share the underlying timeline assumption, the APSI-TRF Timeline Representation Framework (APSI-TRF) infrastructure for P&S has been synthesized as a first basic result [7] synthesizing a software library devoted to speed-up, simplify and increase the quality of P&S software design and deployment. Some works have described applications from the APSI framework [7,12,13] focusing on the support offered by the software platform.

Indeed not enough support exists in APSI-TRF to develop general-purpose domain independent solvers. As an example, the planning algorithm used in [14] was strongly influenced by the OMPS [15] experience, an example of predefined solving structure as a combination of macro-steps. Our goal is the one of building a new software environment on top of the APSI-TRF, preserving some interfaces with respect to the original tool, but overcoming the existing limitations in creating different solvers. The paper is organized as follows: we first give some basic information on the APSI-TRF framework, then introduce the *Extensible Planning and Scheduling Library* (EPSL) our current software effort, and finally report a set of experimental data that show the progress EPSL obtains with respect to the planner used in [13]. Some conclusions end the paper.

2 Timeline-based Planning with APSI-TRF

The main modeling assumption underlying the timeline-based approach [1] is inspired by the classical Control Theory: the problem is modeled by identifying a set of *relevant features* whose temporal evolutions need to be controlled to obtain a desired behavior. In this respect, the set of domain features under control are modeled as a set of temporal functions whose values have to be decided over a time 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[1] [2].

We consider multi-valued *state variables* representing time varying features as defined in [1,16]. As in classical control theory, the evolution of controlled features are described by some causal laws which determine legal temporal evolutions of timelines. For the state variables, such causal laws are encoded in a *Domain Theory* which determines the operational constraints of a given domain. Task of a planner is to find a sequence of control decisions that bring the variables into

[1] According to Wikipedia, a *timeline* is a way of displaying a list of events in chronological order. It is worth saying that this style of planning synthesizes a timeline for each dynamic feature to be controlled.

[2] In this paper we use the term "timeline-based planning" because recently it is more widely used, see for example [10]. Other authors prefer "constraint-based interval planning" [4] following a perspective more connected to the technical way of creating plans.

a final set of desired evolutions (i.e., the *Planning Goals*) always satisfying the domain specification.

2.1 The APSI-TRF

The APSI-TRF software framework supports the development effort by providing a library of basic planning and scheduling, domain independent solvers and a uniform representation of the solution database. Modeling risks are reduced because the use of the framework standardizes and simplifies the process of application deployment fostering a rapid and iterative prototyping cycle, involving directly the users to take into account their feedbacks during the application design. In [7,12,13], some works have described applications from the APSI framework focusing on the support offered by the software platform.

The APSI-TRF architecture. The APSI-TRF software consists of a layered architecture which is organized according to the abstraction level of the solving process. Broadly speaking, constraints are posted on the lower levels as a consequence of decisions taken on higher levels by analyzing variables in the underlying layers (for further details, the reader should refer to [7]).

The Time and Parameter Layer is the lowest layer of the APSI-TRF architecture and it is responsible of managing temporal and parameter information. It provides the functionalities for creating temporal and parameter elements, imposing constraints on them and querying the database to access information about events, temporal positions and parameter values. In particular, temporal information are managed in shape of *Temporal Constraint Network*s (TCNs). This layer is also endowed with propagation algorithms to maintain the consistency of the possible value assignments to time points. The current implementation is based on the *Simple Temporal Problem. The Component Layer* is the point of expansion of the APSI-TRF architecture. A component is a software module which encapsulates the logic for computing a timeline resulting from decisions, evaluating the consistency of the computed timeline with respect to a set of given rules and computing a set of temporal and/or parameter constraints or new decisions to solve (if possible) any flaw for the consistency of the computed timeline. *The Domain Layer* is responsible for managing decisions and relations among them. This information are managed through a particular data structure called *Decision Network*. This layer is responsible for providing *Domain Theory* management functions and generating synchronizations among components. The *Decision Network* provides an unified vision of the current solution while the *Domain Theory* synchronizations provide a unified means for expressing the constraints that the decisions must satisfy. A decision is a generic term to represent a *choice* with respect to the temporal evolution of the domain components and it constitutes the primitive operator to interact with them.

The APSI-TRF provides the timeline-based modeling primitives, to obtain a complete application that solves a particular problem, it is necessary to build a problem solver on top of the framework. The Open Multi-Component Planner & Scheduler (OMPS) [15], as implemented for the GOAC project [13] is an example of such solvers. It builds upon the APSI-TRF framework by selecting the domain components relevant to control and implements the solving engine able to find solution plans.

Limitations in APSI-TRF and OMPS. Analyzing the deployment phase of the OMPS planner during the GOAC project, it was clear that a further effort was required in order to better support the solver development within the APSI-TRF and, in particular, enabling the possibility to develop general-purpose domain independent solvers. This is mainly due to a lack of flexibility in the solving structure. In fact, it resulted that the solving process developed during the GOAC project has been strictly coupled to the domain for which it has been made. Indeed, an extensive analysis of APSI-TRF solvers showed a structure in which introducing different kind of search strategies or heuristics to adapt the solver to the characteristics of different problems requires a not negligible effort as well as an extensive knowledge of the internal technical functionalities of the framework. More in general, the need of meeting operational requirements in challenging domains (like for instance the space context) often leads to the use of highly efficient software modules to address specific sub-parts of the problem with ad-hoc solving algorithms, while the need of reducing modeling mistakes leads to the need of involving users as much as possible in all the steps of software development. And this issue became more evident while applying that APSI-TRF application in solving problems different from the one for which it was developed (an example is given in Section 4.3). In fact, as applications are designed and implemented to focus on very complex problem contexts, the lack of generality is a reasonable payoff. The EPSL has been developed to enhance the current APSI-TRF solving structure exploiting the same representation layers. In fact, the EPSL has been built relying on the APSI-TRF functionalities for managing and representing the domain timelines and on a set of operators (*Resolvers*) obtained by a decomposition of the OMPS solving processes.

3 The Extensible Planning and Scheduling Library

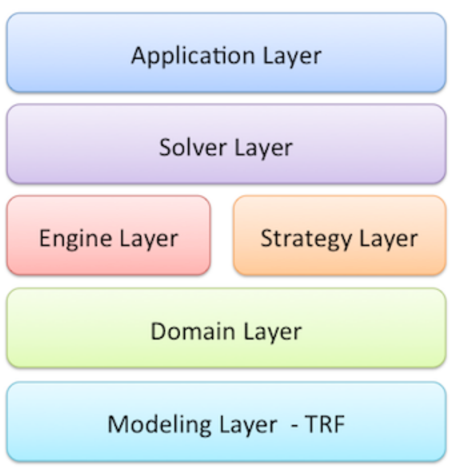


Fig. 1. EPSL key concepts

The main goal of the Extensible Planning and Scheduling Library (EPSL) is to provide a planning environment in which it is possible to easily define and evaluate different solving configurations in order to find the best one for the particular addressed problem. The key point for the EPSL is its planner interpretation. Indeed, as shown in Fig. 1, the system is defined as the composition of a set of *independent* building blocks which define the configuration of a planner. Each block is responsible to manage a particular aspect of the planning process and they rely on the APSI-TRF modeling layer.

Such approach allows a fast definition of different flexible planners in which it is possible to focus on a single module implementation, without taking care of aspects of the other planning processes. For instance, it is possible to focus on heuristic functions definition without considering other details on search algorithms or to compare several searching strategies by simply replacing the responsible module in the planner *configuration*. Moreover, EPSL provides the possibility to further customize the planner behavior by assign a set of weights to the engine operators in order to further guide the solving process. The main architectural elements of the EPSL framework are depicted in Fig. 1.

Engine Layer. This module implements the set of operators (*Resolver*s) used during the solving process. Each *Resolver* is used to process the search space nodes in order to refine the initial plan. EPSL provides a set of already implemented *Resolver*s obtained by a decomposition of the OMPS solving process. In particular, the main Resolvers are the following:
Decision Justification. It is the process step by which the planner can introduce new decisions into the plan. This phase is supported by the *Domain Theory* which models relations among domain components in order to manage their synchronizations. Therefore a decision can be added into the plan only when, according to the *Domain Theory*, the component behavior resulting from its imposition does not conflict with other problem constraints. In particular there are two techniques to justify a decision: (i) *Decision Expansion*, the decision is imposed on its domain component expanding the *Domain Theory*. In this case the justification process may involve the introduction of new decisions (*subgoals*) on other domain components; (ii) *Decision Unification*, the goal decision is merged with a compatible one in the plan (i.e. already justified). In this case, no decisions are imposed on domain timelines so the related component behavior is not changed and no further synchronization must be applied.
Decision Scheduling. It is the process step by which timeline decisions are scheduled in order to avoid inconsistencies. Indeed timeline decisions are activities representing "resource usage" which must be scheduled in order to achieve a consistent behaviour of the associated component (in EPSL a component is modelled as a binary resource).
Timeline Extraction. It is the process step by which plan decisions are allocated on timelines. Indeed, before the extraction step, plan decisions are "floating" so domain timelines have several possible behaviours. It uses the *Earliest Start Time* (EST) approach according to which each decision is allocated as soon as possible over its timeline.
Timeline Completion. It is the process step to complete (if needed) a timeline after the extraction step. Indeed, the extraction may leave unallocated timeline segments (*flaws*) that must be avoided because cause uncontrollable component behaviours. The completion phase is supported by *Consistency Features*, the component *Finite State Automata* (FSA) describing allowed component state transitions.

Strategy Layer. This is the module responsible to manage the search space boundary. A *Search Strategy* identifies a particular queuing policy of the search space nodes to be expanded and it may use an *Heuristic* function which defines a node status evaluation criteria to support the search process. Within EPSL, the heuristic implementation is defined as follows:

Definition 1. *Given that $G(n) = \{g_0, g_1, ..., g_k\}$ is the set of* goal*s of the problem status associated to the node n, the estimated cost solving the problem $h(n)$*

computed by the heuristic function is: $h(n) = \sum_{g \in G(n)} f_i(g)\ w_i, \forall i \in OP$ *where* OP *is the set of* primitive *operators of the solving process,* f_i *is the* $i_t h$ *solving step and* w_i *is the associated cost.*

It is important to point out that the EPSL architecture provides the possibility to focus on heuristic implementation in order to define and test more informed heuristic for enhancing the solving process. Similarly, the EPSL is endowed with a set of search strategies but, also, it is possible to implement and integrate additional search strategies in the planning environment. The list of basic search strategies are the following:
A^* Strategy. This search strategy compute the node value $f(n)$ as a function of the node generation cost $g(n)$ and the value $h(n)$ assigned by the heuristic function used. As usual, an admissible h function guarantees optimality and completeness properties of the search.
Probabilistic Strategy. This strategy consists of a *greedy* search in which the evaluating function $f(n) = P(n)$ represents a probabilistic distribution which expresses the probability of a node n to be the solution node, so $f(n) = P(n) \in (0, 1]$.

Definition 2. *The probabilistic function* $P(n)$ *is defined as:* $P(n) = \frac{1}{e^{h(n)}}$ *with*

$$\lim_{h(n)\to+\infty} \frac{1}{e^{h(n)}} = 0 \qquad \lim_{h(n)\to 0^+} \frac{1}{e^{h(n)}} = 1 \text{ with } h(n) \in [0, +\infty).$$

The strategy sorts boundary nodes according to their probability values in order to extract first the nodes with the highest probability to be the solution.
Depth-(Breadth-)First Search (DFS/BFS) Strategy. It implements a simple *Depth-(Breadth-)First Search* according to which no heuristic function is used for nodes evaluation. The boundary is managed by means of a *stack (queue)* which implements a *Last-In-First-Out (First-In-First-Out)* queuing policy.
Composite Strategy. It implements a (sort of) composition operator to integrate different search strategies in one composite strategy.

Solver Layer. This module implements the management for search algorithms and, more in general, the structure of the solving process. It provides a set of interfaces for managing and integrating in the planning environment new search algorithms on which new planning instances may be defined.

3.1 Main advantages in using EPSL

The EPSL design has been started in order to address the limitations discussed in Section 2.1. In fact, the APSI-TRF framework provides the possibility to effectively design applications relying on tailored solving structures but, in general, they can not be easily (and quickly) deployed in different contexts. The main reason is that APSI-TRF currently allows the design and development of applications relying on solvers strictly coupled with search strategy and heuristic information.

Then, as discussed above, EPSL provides an enhanced framework for developing applications in which designers may focus on a single aspect of the solving process. In particular, EPSL allows to focus on (i) operators design (i.e., adding new reasoning capabilities), (ii) search strategies (i.e., possibly implementing additional algorithms) and (iii) defining heuristic functions (i.e., identifying suitable control strategies for the specific problem). All the above features concur in creating a portfolio of operators, algorithms and heuristics that actually enable the possibility to combine them in many possible ways (not only the ones for which they have been designed), then, providing application designers with a really flexible and rich framework for P&S application development.

Finally, even though the EPSL is in an initial development stage, results collected after an experimental evaluation show that EPSL performances are comparable with OMPS in the domain for which such APSI-TRF application has been developed (i.e., the GOAC domain). On the other hand, a particular EPSL planner (whose configuration has been identified after a fast assessment on the considered domain) shows better performances of OMPS on a different planning domain elicited from a different space domain (see next section).

4 Current Empirical Results

In this section, we present an experimental evaluation comparing EPSL and OMPS performances while solving different problem instances in two different planning domains derived from real world scenarios: a robot control domain extracted from the GOAC project[3] [13] and a space facility management domain extracted from the ULISSE project[4]. Before discussing empirical results, a brief description of the problem domains and their timeline-based representations is provided. Then, experimental results are reported and discussed. All the experiments have been ran on a MacBook endowed with a Intel Core 2 Duo (2.26GHz) processor and 2GB RAM.

4.1 GOAC: A Robotic Domain

The Goal Oriented Autonomous Controller [13] is an ESA effort to create a common platform for robotic software development. In particular, the delivered GOAC architecture has integrated: (a) a timeline-based deliberative layer which integrates a planner based on the APSI Platform [7] and an executive a la T-REX [17]; (b) a functional layer which integrates $\mathrm{G}^{en}{}_{o}\mathrm{M}$ and BIP [18].

The GOAC Domain. The robotic domain considers a planetary rover equipped with a Pan-Tilt Unit (PTU), two stereo cameras (mounted on top of the PTU) and a communication facility. The rover is able to autonomously navigate the environment, move the PTU, take pictures and communicate images to a Remote Orbiter. A safe PTU position is assumed to be $(pan, tilt) = (0, 0)$. Finally, during the mission, the Orbiter may be not visible for some periods. Thus, the robotic platform can communicate only when the Orbiter is visible. The mission goal is a list of required pictures to be taken in different locations with an associated PTU configuration. A possible mission action sequence is the following: navigate

[3] Funded by the European Space Agency (ESA) TRP/T313/006MM.
[4] Partially funded by EU FP7 SPACE G.A. 218815.

to one of the requested locations, move the PTU pointing at the requested direction, take a picture, then, communicate the image to the orbiter during the next available visibility window, put back the PTU in the safe position and, finally, move to the following requested location. Once all the locations have been visited and all the pictures have been communicated, the mission is considered successfully completed. The rover must operate following some operative rules to maintain safe and effective configurations. Namely, the following conditions must hold during the overall mission: **(C1)** While the robot is moving the PTU must be in the safe position (pan and tilt at 0); **(C2)** The robotic platform can take a picture only if the robot is still in one of the requested locations while the PTU is pointing at the related direction; **(C3)** Once a picture has been taken, the rover has to communicate the picture to the base station; **(C4)** While communicating, the rover has to be still; **(C5)** While communicating, the orbiter has to be visible.

Timeline specification for the robotic domain. To obtain a timeline-based specification of our robotic domain, we consider two types of state variables: *Planned State Variables* to represent timelines whose values are decided by the planning agent, and *External State Variables* to represent timelines whose values over time can only be observed. Planned state variables are those representing time varying features like the temporal occurrence of navigation, PTU, camera and communication operations. We use four of such state variables, namely the *RobotBase*, *PTU*, *Camera* and *Communication*.

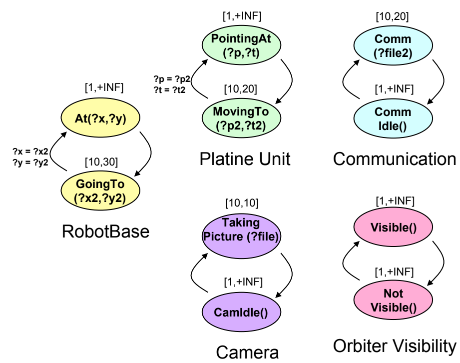


Fig. 2. *State variables describing the robotic platform and the orbiter visibility (durations are stated in seconds)*

In Fig. 2, we detail the values that can be assumed by these state variables, their durations and the legal value transitions in accordance with the mission requirements and the robot physics[5] Additionally, one external state variable represents contingent events, i.e., the communication opportunities. The *Orbiter Visibility* state variable maintains the visibility of the orbiter. The allowed values for this state variable is *Visible* or *Not-Visible* and are set as an external input. The robot can be in a position (*At(x,y)*) or moving towards a destination (*GoingTo(x,y)*). The PTU can assume a *PointingAt(pan,tilt)* value if pointing a certain direction, while, when moving, it assumes a *MovingTo(pan,tilt)*. The camera can take a picture of a given object in a position $\langle x, y \rangle$ with the PTU in $\langle pan, tilt \rangle$ and store it as a file in the on-board memory (*TakingPicture(file-id,x,y,pan,tilt)*) or be idle (*CamIdle()*). Similarly, the communication facility can be operative and dumping a given file (*Communicating(file-id)*) or be idle (*ComIdle()*).

[5] Note that variables (e.g., ?x) represents parameters with values in a finite set of symbols, used to compactly represent the allowed values for a given state variable.

Domain operational constraints are described by means of *synchronizations*. A synchronization models the existing temporal and causal constraints among the values taken by different timelines (i.e., patterns of legal occurrences of the operational states across the timelines).

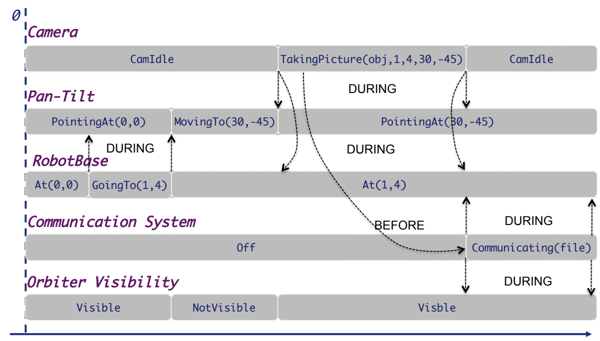


Fig. 3. *An example of timeline-based plan.*

Fig. 3 exemplifies the use of synchronizations implementing the operative rules (see Section 4.1) in our case study domain. The synchronizations depicted are: *GoingTo(x,y)* must occur during *PointingAt(0, 0)* (**C1**); *TakingPicture(pic, x, y, pan, tilt)* must occur during *At(x, y)* and *PointingAt(pan, tilt)* (**C2**); *TakingPicture(pic, x, y, pan, tilt)* must occur before *Communicating(pic)* (**C3**); *Communicating(file)* must occur during *At(x,y)* (**C4**); *Communicating(file)* must occur during *Visible* (**C5**).

In addition to those synchronization constraints, the timelines must respect transition constraints among values and durations for each value specified in the domain (see again Fig. 2).

4.2 Testing Results on the GOAC Domain

This section investigates the EPSL planners performance by using our robotic case study as a benchmark. For this purpose, we introduce different planning problem scenarios obtained by varying the problem complexity along the following dimensions: *plan length* by playing on both the number of pictures (from 1 to 5) to be taken and the plan horizon; *plan choices* by changing the number of communication opportunities (from 1 to 4 visibility windows). Notice that an increasing number of communication opportunities raises the complexity of the planning problem with a combinatorial effect. More in general, among all the generated problem instances, the ones with higher number of required pictures and higher number of visibility windows result as the hardest ones. In these scenarios, we analyzed the performance of the planners.

Exploiting the flexibility provided by EPSL, we have easily configured five different planners in order to test the framework capabilities and also to compare their performances with OMPS. Therefore, exploiting the strategies and heuristic functions described in Sec. 3, we have defined the following planners: *BFS Planner*, *A* Planner*, *Probabilistic Planner*, and *DFS Planner*. In addition, a more general planner, called *Chronological Backtracking Planner*, has been defined extending the *DFS Planner* using the *Probabilistic Strategy* described in Sec. 3.

In addition we defined a more general planner, called Chronological Backtracking Planner, which makes use of the Composite Strategy combining the

Probabilistic Strategy and the DFS Strategy. This strategy splits the solving process in two phases. During the first phase, the planner uses the *Probabilistic Strategy* to apply the justification decisions, then, the planner switch to the *DFS Strategy* in order to complete the plan and find a solution as soon as possible. Basically, this strategy provides a trade off between controlling the decisions to apply and completing the plan as quick as possible.

Moreover it is important to point out that EPSL features give the possibility to set several configuration parameters in order to affect planner choices during solution search. Therefore, Fig. 4 shows the results obtained by the best parameter configuration for each tested planner.

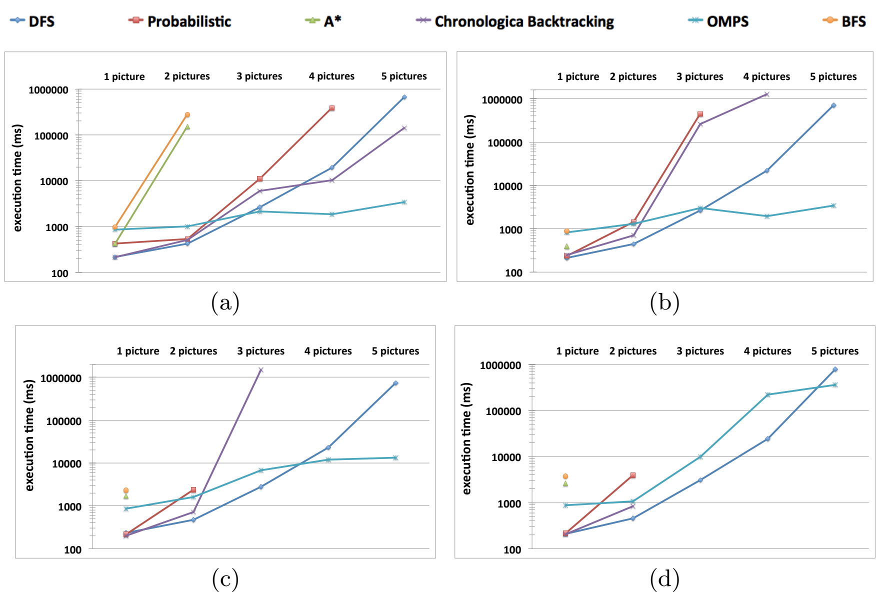


Fig. 4. EPSL testing results and comparison with OMPS on GOAC problem with: (a) one communication window; (b) two communication windows; (c) three communication windows; (d) four communication windows.

The resulting performances demonstrate how different planner configurations may lead to pretty different results. The *BFS Planner* is the worst among the defined EPSL planners as it is able to solve few problem instances. Indeed, using such strategy leads to expand a lot of nodes before a solution is found, possibly because the problem domain structure determines very deep solution nodes. So, in this case, the planner spends a lot of time in expanding nodes and in maintaining *Temporal Network* consistency without approaching the solution node. This is also confirmed by the fact that *A* Planner* has performances very close to those of the *BFS Planner*. This also allows us to argue that the exploited *heuristic* function is not informed enough.

On the other hand, the *DFS Planner* and OMPS results as the best planners (see again Fig. 4). In fact, both use a strategy that does not make an evaluation of boundary nodes during the search. They always make the same choices (decision expansion) during the search. Given the characteristics of the GOAC problem, this results as the best approach as they are able to manage in a more effective way the underlying *Temporal Network*, thus, reducing the *propagation* cost due to the maintenance of temporal constraint consistency. However, such planners do not provide any control on the solving process and, as we will see in the next section, this is not a good behavior for different kind of problems.

As a final remark, EPSL does not always offer performances better than OMPS. In particular, only *DFS Planner* is comparable to OMPS (even better in some cases). However, it is important to notice that the main goal of the system is to realize a flexible and extensible software library by which it is possible to easily define new planning instances or to adapt an already defined planner to a particular problem. Indeed, EPSL unlike OMPS gives the possibility to focus the efforts on specific aspects of solving process so by means of its functionalities it is possible, for instance, to improve solution search by providing a more informed heuristic as well as to improve the *Temporal Network* management by providing more efficient algorithms.

4.3 FSL: A Space Facility Management Domain

The "USOCs Knowledge Integration and dissemination for Space Science Experimentation" (ULISSE) is a project (funded by EU and indicated by REA as example of successful FP7 project in the Space field) whose objective is data valorization around the ISS experiments. Each USOC (User Support and Operation Centre) is responsible for a particular on-board facility that is to be operated to perform scientific experiments and to generate the related scientific data. Here, we report a simplified planning domain derived from the one described in [19] aiming at addressing a short-term planning problem in managing a particular ISS facility.

The Fluid Science Laboratory. The FSL is a ISS multi-user facility designed for the execution of experiments on fluid physics under microgravity conditions. The FSL is equipped with a number of optical instruments that allow to separately implement a wide variety of diagnostic techniques that can be combined together and each combination is called *optical mode*. Generally, an experiment execution consists of several runs, a run is a part of the experiment that uses a defined configuration and setting of the facility (as for example a specific optical mode). The data recorded by the experiments are sent in a real-time manner to the ground. Data is routed to the Columbus for communicating to the ground through a *High Rate* downlink channel mainly intended for high rate science data. The FSL is always in one defined status, among the following: Off, Standby, Configuration & Checkout and Nominal. A set of operations on FSL has been identified that require an initial status of FSL and may eventually lead to a transition to a new status of the facility. Each activity is characterized by several parameters, e.g., the team that operates the FSL, the duration of the activity, etc. A complete specification of the FSL activities is given in [19].

Each plan executed on the FSL has to comply with a set of operative constraints. These are related to general operational requirements: prior to performing scientific experiments and/or diagnostic tests correctly, the FSL must follow a precise sequence of steps in order to be fully operative: initially, it must be mechanically configured according to the experiment/test to be performed; subsequently, the operative rack has to be activated; finally, the set of diagnostics must be executed before the experiment can commence. At the end of each operative cycle, the previous operations must be planned to be executed in the reverse order. At the beginning of each operative period, an optical mode test has to be performed to check whether the mechanical configuration has been correctly completed (avoiding to perform experiments with optical targets wrongly set); the High Rate Data Link (HRDL) has to be allocated during each run that requires real time data transmission; a run cannot be interrupted and has to be executed continuously. Finally, other operative constraints are required for safety issues: during non operative periods, the status of both the FSL and the Rack must be off, FSL mechanical configuration and de-configuration activities have to be performed with both FSL Rack and FSL switched off.

Generally, the execution of an experiment consists of several runs; a run is a segment of the experiment that uses a defined configuration and setting of the facility (as for example a specific optical mode). The objective function for the problem is that all the planned experiments must be performed and the recorded activities should be safely downloaded on ground stations.

A timeline specification for the FSL domain. To obtain a timeline-based specification of the FSL domain a set of *multi-valued state variables* has been considered. Multi-valued state variables are those representing time varying feature for the temporal occurrence of mechanical configurations, rack activations as well as both FSL status and activities. In this regard, we consider four different state variables, i.e., *Mechanical Configuration*, *Rack*, *FSL* and *FSL_Activities*. The Figure 5 depicts a detailed view of the values that can be assumed by these state variables, their durations and the allowed value transitions in accordance with the operative constraints.

In general, before any operative period, the FSL requires a *Mechanical Configuration*. In particular, during not operative period no optical target is mounted on the FSL (*Deconfigured*) while, before starting an optical checkout, a suitable optical target is to be mounted to result mechanically configured (*Configured*). The *Rack* can assume a *Active* status when switched on while, when switched off, it assumes a *Not Active* value.

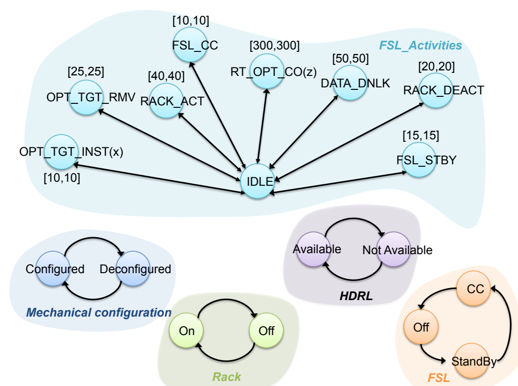


Fig. 5. Value transitions for state variables describing the FSL (temporal durations in minutes).

The FSL can assume different status as well as perform different activities. The *FSL* may be in one of the following status: *Off* while not operating, *StandBy* after initialization and in operative mode when ready for Control and Checkout (i.e., *CC*). The *FSL_Activities* represents the full set of activities that can be performed on-board. Namely, the FSL activities are the following: in-

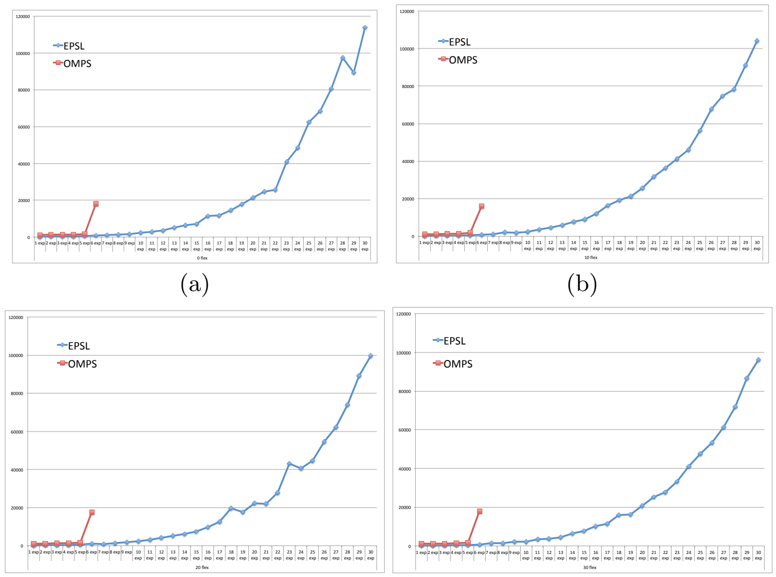
stallation or removal of an optical target (respectively, *OPT_TGT_INST(x)*[6] and *OPT_TGT_RMV*); activation and deactivation of the rack (*RACK_ACT* and *RACK_DEACT*); initialization and activation of the optical component (*FSL_STBY* and *FSL_CC*); finally, while running an optical checkout the FSL may assume the *RT_OPT_CO(x)*; finally, executing a downlink activity results in assuming the *DATA_DNLK* value. In Figure 5, we detail the values that can be assumed by these state variables, their durations and the allowed value transitions in accordance with the operative requirements.
In the FSL domain, the following compatibilities are considered (not shown in Figure 5 not to overload the representation): (1) *OPT_TGT_INST* values must occur DURING a *Deconfigured* value on the Mechanical Configuration variable; (2) *OPT_TGT_RMV* as well as *Active* and *RACK_ACT* must occur DURING a *Configured* value on the Mechanical Configuration state variable; (3) *RACK_DEACT*, *RACK_STBY* and *RACK_CC* must occur DURING a *Active* value on the Rack state variable; (4) *RT_OPT_CO* and *DATA_DNLK* must occur during a *CC* value on the FSL state variable. The first two compatibilities globally express the circumstance that all FSL activities must be performed within a Rack activation/deactivation cycle, and that such cycle must be performed once an optical target has been configured. The third compatibility enforces that the FSL is supposed to be initialized before being fully operative. Finally, constraint (4) enforces that the main FSL activities should be performed while in *CC*.

4.4 Testing Results on the FSL Domain

This section further investigates the performances of the EPSL in the FSL case study. Again, we consider different planning problem scenarios obtained by varying the problem complexity by varying: (1) *plan length*, i.e., both the number of optical check out (from 1 to 30) to be taken and the plan horizon; (2) *Plan Flexibility*, i.e., for each FSL activity, we set a minimal duration, but allow temporal flexibility on the activity termination, namely, the end of each activity has a tolerance ranging from 0 to 30 seconds. This temporal interval represents the degree of temporal flexibility/uncertainty that we introduce in the system. It is worth to underscore that, among all the generated problem instances, the ones with higher number of required pictures and higher temporal flexibility correspond to the hardest ones. In these scenarios, we analyzed the performance of the planners but, here, rather than testing all the possible EPSL planners as in the previous case study, we exploited the *Chronological Backtracking Planner* configuration already defined for the GOAC problem (see Sec. 4.2) taking advantage of the EPSL flexibility to quickly configure the more suitable planner to test the framework as well as to compare its performance with OMPS.

In this case, the EPSL Planner *dominates* OMPS (see Fig. 6) providing best performances in every configuration. In this domain, there are different kind of choices (decisions for unification) that must be applied in order to efficiently find a solution and, in this regard, OMPS can not be tune in order to change its solving behavior. Therefore, then OMPS applies the same solving approach (i.e., a fully DFS approach) as for the GOAC domain, while EPSL exploit the possibility to control the application of justification decisions during the search,

[6] The variable x represents one of the 86 available optical modes.



(a) (b) (c) (d)

Fig. 6. EPSL comparison with Omps (time in msecs.) on ULISSE problem with: (a) 0 seconds flex.; (b) 10 seconds flex.; (c) 20 seconds flex.; (d) 30 seconds flex..

thus, providing a search strategy that allows the planner to efficiently address the FSL problem. As discussed in Sec. 2.1, this is an expected behavior. In fact, Omps has been designed and developed to face a different planning domain and different problem instances. Thus, given its structure, it would require additional design efforts in order to address also the FSL management.

5 Conclusion

This paper has introduced EPSL a tool that following the APSI-TRF interfaces represents a step in developing an extensible and general-purpose planning environment. The system aims at providing the possibility to focus on specific aspects (e.g. heuristic or strategy definition) without taking care all details related to the timeline-based solving process. The evaluation presented in this paper shows how it can be used for synthesizing domain independent planners. Such planners have been evaluated with respect to two application domains against the pre-heating Omps planner as developed for the Goac project. We have shown how the EPSL-based planners are comparable with Omps on the Goac domain and outperform it on FSL. Somehow this represents the basic seed result for justifying our research. We will now continue our work comparing with other possible approaches (e.g., EUROPA) but also pursuing further enhancements of EPSL to integrate state of the art technology for domain-independent search control.

References

1. Muscettola, N.: HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed.: Intelligent Scheduling. Morgan Kauffmann (1994)

2. Jonsson, A., Morris, P., Muscettola, N., Rajan, K., Smith, B.: Planning in Interplanetary Space: Theory and Practice. In: AIPS-00. Proc. of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling. (2000) 177–186
3. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A., Policella, N.: An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In: ICAPS-07. (2007) 57–64
4. Frank, J., Jonsson, A.: Constraint Based Attribute and Interval Planning. Journal of Constraints **8(4)** (2003) 339–364
5. Bernardini, S.: Constraint Based Temporal Planning: Issues in Domain Modeling and Search Control. PhD thesis, Universita degli studi di Trento (2008)
6. 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: Proc. of SpaceOps 2000. (2000)
7. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: Developing an End-to-End Planning Application from a Timeline Representation Framework. In: IAAI. Proc. of the 21st Conference on Innovative Applications of Artificial Intelligence. (2009)
8. Smith, D., Frank, J., Jonsson, A.: Bridging the Gap Between Planning and Scheduling. Knowledge Engineering Review **15**(1) (2000) 47–83
9. Laborie, P., Ghallab, M.: IxTeT: an integrated approach for plan generation and scheduling. In: ETFA: Emerging Technologies and Factory Automation. (1995)
10. Chien, S.A., Johnston, M., Frank, J., Giuliano, M., Kavelaars, A., Lenzen, C., Policella, N.: A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations. In: SpaceOps. (2012)
11. Jónsson, A.K., Morris, P.H., Muscettola, N., Rajan, K., Smith, B.: Planning in interplanetary space: Theory and practice. In: ICAPS. (2000) 177–186
12. Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: MrSPOCK: Steps in Developing an End-to-End Space Application. Computational Intelligence **27**(1) (2011)
13. Ceballos, A., Bensalem, S., Cesta, A., de Silva, L., Fratini, S., Ingrand, F., Ocon, J., Orlandini, A., Py, F., Rajan, K., Rasconi, R., van Winnendael, M.: A Goal-Oriented Autonomous Controller for Space Exploration. In: ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation. (2011)
14. Fratini, S., Cesta, A., De Benidictis, R., Orlandini, A., Rasconi, R.: APSI-based deliberation in Goal Oriented Autonomous Controllers. In: ASTRA-11. 11th Symposium on Advanced Space Technologies in Robotics and Automation. (2011)
15. Fratini, S., Pecora, F., Cesta, A.: Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. Archives of Control Sciences **18**(2) (2008) 231–271
16. Cesta, A., Oddi, A.: DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains,. In Ghallab, M., Milani, A., eds.: New Directions in AI Planning. IOS Press: Amsterdam (1996)
17. Py, F., Rajan, K., McGann, C.: A Systematic Agent Framework for Situated Autonomous Systems. In: AAMAS-10. Proc. of the 9^{th} Int. Conf. on Autonomous Agents and Multiagent Systems. (2010)
18. Bensalem, S., de Silva, L., Gallien, M., Ingrand, F., Yan, R.: "Rock Solid" Software: A Verifiable and Correct-by-Construction Controller for Rover and Spacecraft Functional Levels. In: i-SAIRAS-10. Proc. of the 10^{th} Int. Symp. on Artificial Intelligence, Robotics and Automation in Space. (2010)
19. Carotenuto, L., Ceriello, A., Cesta, A., Benedictis, R.D., Orlandini, A., Rasconi, R.: Planning and Scheduling Services to Support Facility Management in the ISS. In: 63^{rd} International Astronautics Congress, IAC-12-B5.2.10., Naples, IT. (2012)