INTEGRATED PLANNING AND SCHEDULING CAPABILITIES TO SUPPORT SPACE ROBOTICS

Amedeo Cesta, Riccardo De Benedictis, Andrea Orlandini, Alessandro Umbrico, and Giulio Bernardi

CNR – Consiglio Nazionale delle Ricerche, ISTC
I-00185 Rome, Italy
⟨name.surname⟩@istc.cnr.it

ABSTRACT

This paper focuses on the ESA APSI software platform for developing planning and scheduling applications for space missions. In particular, it describes work performed by CNR-ISTC developing new APSI-compliant features that address aspects considered as "improvable" in the current APSI distribution. The APSI version used within the GOAC project has been considered as the starting state for the new work. In particular, two aspects are described: (a) two new planning systems able to integrate causal and resource reasoning (called EPSL and J-TRE); (b) the KEEN knowledge engineering environment that allows an integrated use of various research facilities we have developed over time. Some examples are shown taken from the GOAC space robotics domain.

Key words: Planning; Scheduling; Space Robotics.

1. INTRODUCTION

Timeline-based planning has been quite successful in a number of space applications [24, 21, 8]. A lot of effort has been dedicated to build software development environments, like EUROPA [1], ASPEN [15], and APSI-TRF [7], that facilitate the further synthesis of applications. Also ESA concurs in this area of planning and scheduling (P&S) by promoting the development of APSI [12], its use inside the robotic software GOAC [6], its more general use in space automation [28]. Additionally the agency participates in the coordination effort among space agencies for standardizing timeline concepts for operations [16].

In this scenario, which is the role for research centers? The one of proposing innovation, synthesizing new ideas, identifying new paths for broadening the use of timeline-based approach creating more robust user-oriented tools. In this regard, this paper reports some recent directions taken at the CNR-ISTC. After a period of active participation in ESA initiatives like APSI (20216/06/F/VS) and GOAC (TRP/T313/006MM), we present here some recent APSI-compliant research work. In particular we discuss in Section 3 how we are synthesizing a new generation of planning systems able to integrate causal and resource reasoning. Two of these systems, named EPSL [14] and J-TRE [17], are introduced here and their main innovative features discussed. Additionally we discuss the KEEN system [3] for knowledge engineering support for P&S with timelines, discussing recent features of domain refinement support integrated with validation and verification (Section 4). In Section 5, we demonstrate the new capabilities in the GOAC scenario of space robotics.

2. THE APSI-TRF FRAMEWORK AND THE APSI-PLANNER-2

The modeling assumption underlying the timeline-based approach, see [25], is inspired by the classical Control Theory: the problem is modeled by identifying a set of relevant *components* whose temporal evolutions need to be controlled to obtain a desired behavior. Components are primitive entities for knowledge modeling, and represent logical or physical subsystems whose properties may vary in time. 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.

The APSI-TRF [7] is a software development framework for planning and scheduling systems which supports the development effort by providing a library of basic planning and scheduling domain independent solvers and a uniform representation of the solution database. It allows to represent several planning and scheduling concepts in the form of timelines. Indeed, components such as multi-valued state variables [25] and resources like those commonly used in constraint-based schedulers [13] provide enough modeling power for a good set of planning and scheduling needs.

The APSI project has produced the APSI-TRF [1] architecture, a layered software system that acts as a facilitator for building planning and scheduling systems. As shown

[1] TRF stands for Timeline-based Representation Framework.

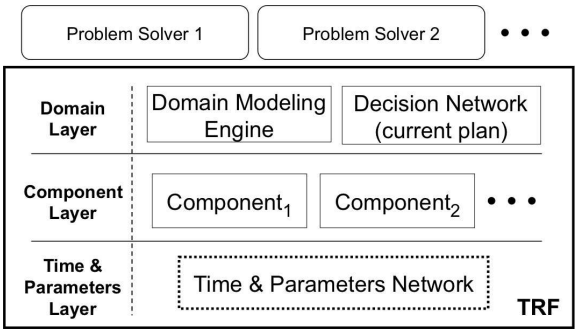

Figure 1. APSI-TRF *architecture.*

in Figure 1, the architecture is organized in several layers, according to the abstraction level of ingredient they contribute:

- The *Time and Parameter Layer* is the lowest layer of the APSI-TRF architecture and it is responsible of managing temporal and parameter information. In particular, temporal information are managed in shape of *Temporal Constraint Network*s (TCNs) and the current implementation is based on the *Simple Temporal Problem* [18]. This layer implements the choice of representing the plan as a flexible temporal network.
- The *Component Layer* is the point of expansion of the framework in which a *component* is a module encapsulating the logic for representing a time-varying domain feature. In a given problem, the associated timeline is generated after a set of applied decisions that satisfy the consistency of the computed component (according to a set of given rules);
- The *Domain Layer* is responsible for managing planning decisions and relations among them (essentially it guarantees the plan representation). Such plan representation is called *Decision Network* and provides a unified vision of the current solution (partial plan). In addition, a *Domain Modeling Engine* is responsible for the interaction with the user though the Domain Description Language that defines feature of the domain of interest including the so called *synchronizations*, that provide a unified mechanism for expressing causal constraints among domain components (constraints among component decisions).

It is worth noting that the APSI-TRF is not a planner *per se* but a development environment for planners and schedulers. In fact, as shown in Figure 1, to create a complete application it is necessary to insert a further module (a Problem Solver) on top of the domain representation in the Domain Layer. Such additional module is responsible for either driving a generic search or implementing a specific constructive heuristic for solving the problems at hand.

The APSI-Planner-2 (AP2) [19], is an APSI-compliant version of the Open Multi-component Planner and Scheduler (OMPS) [20]. AP2 has been built on-top of the APSI-TRF to model functionalities for the GOAC project and, in particular, uses an extension of the OMPS Domain Definition Language and Problem Description Language (the timeline equivalent of analogous files in classical planning) to describe the relevant constraints of the planning domain.

The AP2 planner (like its predecessor OMPS) generates time flexible plans, that is a set of timelines where the temporal transitions between values are not fixed by the planner but temporally related with a set of constraints. Transitions have a minimum and a maximum planned time (within which the transition is legal and compliant with the domain theory) as well as temporal relations with other transitions on the same timeline and/or on other timelines.

The improvements of the AP2 planner, with respect to the OMPS planner, were necessary for enabling continuous planning in a dynamically evolving solution database. In fact, in GOAC, the solution database is concurrently (and continuously) modified by an executive layer based on T-REX [29] to synchronize the status of the database with the status of the real world, and by the planner to plan the (foreseen) status of the world in the future.

3. TWO EMERGING TIMELINE-BASED SYSTEMS: EPSL AND J-TRE

As said in the previous section, APSI-TRF is designed as a software environments suitable for generating planning applications. So far, the environment has been mainly exploited in order to develop P&S applications rather than pursuing research work on specific issues. We are currently working to achieve a better balance between the two tasks. In particular, we are investigating two problems whose solutions are partial in planning with timelines: (a) the problem of search control (b) the intermix of causal and resource reasoning.

The current realm is that although AP2 and OMPS planners capture elements that are very relevant for applications, they both pursue a quite specific organization of the search strongly dependent from the domain specific information that ended up in performance limitations while applying them to different planning domains. Thus, one goal we are currently pursuing is the creation of timeline-based domain independent planners. Indeed, timeline-based planning is mostly based on the notion of partial order planning [31] and has almost neglected advantages in classical planning triggered from the use of GRAPHPLAN and/or modern heuristic search [4, 5], almost because of the lack of explicit representation of states in the state-transition model. Also the APSI-TRF architecture relies on a clear distinction between temporal reasoning and other forms of constraint reasoning.

A second aspect worth being considered is the capability of performing resource reasoning. One of the advantages of the timeline-based planning is to integrate in a smooth way scheduling algorithms derived from CSP heuristic approach, like [13] in which their Precedence Constraint Posting (PCP) works on the resource reasoning with “overconsumptions over time”. Such resource

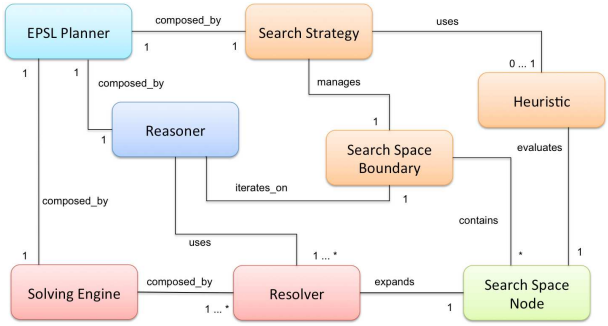

Figure 2. EPSL *concepts.*

reasoning approach has been first integrated in the special purpose search of OMPS while the AP2 planner was just focusing on temporal/causal timeline reasoning (because of the emphasis on planning and execution). Indeed, a lot of possibilities for research are still left in the direction of using PCP-like reasoning in domain independent search algorithms on timelines.

The two above issues are addressed in the two planner described in the remaining of this section.

3.1. The Extensible Planning and Scheduling Library: EPSL

Analyzing the deployment phase of the AP2 planner during the GOAC project, it was clear that a further effort would be beneficial to better support the solver development within the APSI framework and, in particular, to develop the capability to synthesize general-purpose domain independent solvers. This is mainly due to a lack of flexibility in the solving structure. Indeed, AP2 makes use of a set of specific solvers built on-top of the TRF. However integration of these solvers within AP2 solving process is limited in flexibility and it is tightly connected to the features of the addressed problem (namely the features of the GOAC domain). In this context, it is not possible to easily introduce specific knowledge (such as *heuristics*) that can help search procedure, or new solvers in order to increase the solving power of the planner. Any change on AP2 elements has an impact on their structure and requires a significant development effort.

The main goal of the Extensible Planning and Scheduling Library (EPSL) [14] is to provide a planning environment in which it is possible to easily define new timeline-based planners, customized for the particular problem to address. The EPSL key point is the "planner-interpretation" which defines a timeline-based planner as the composition of several independent modules combined together according to a set of interfaces defined within the environment (see Figure 2).

EPSL, defines a planner as the tuple $\langle \mathcal{P}, \mathcal{S}, \mathcal{H}, \mathcal{E} \rangle$. $\mathcal{P}$ is the problem to solve. $\mathcal{S}$ is the *strategy* used to manage the search space fringe. The strategy $\mathcal{S}$ can be choosen among several built-in options (A*, DFS, BFS, etc). $\mathcal{H}$ is the *heuristc* exploited to analyze the current plan π and to extract the "more relevant" (according to $\mathcal{H}$) *flaw* to be solved. A *flaw* represents either a *goal* or a *threat* that must be solved in order to find a valid solution plan. $\mathcal{E}$ is the *resolver engine* encapsulating the reasoning capabilities of the EPSL framework. $\mathcal{E}$ is composed by several solving algorithms, called *resolvers*, each of which is used to solve a particular type of flaw during any plan refinement (see [14] for further details).

Algorithm 1 solve$(\mathcal{P}, \mathcal{S}, \mathcal{H}, \mathcal{E})$

```
1:  root ← MakeRootNode(P)
2:  fringe ← Enqueue(root, ∅, S)
3:  π ← GetPlan(Dequeue(fringe))
4:  while HasFlaws(π) do
5:      flaws ← IdentifyFlaws(π)
6:      flaw ← SelectOneFlaw(flaws, H)
7:      for resv ← GetResolvers(E) do
8:          if CanHandle(resv, flaw) then
9:              nodes ← Handle(flaw, resv)
10:             fringe ← Enqueue(nodes, fringe, S)
11:     end for
12:     if fringe = ∅ and not(flaws = ∅) then
13:         return Failure
14:     π ← GetPlan(Dequeue(fringe))
15: end while
16: return π
```

The "abstract" solving procedure of the generic *EPSL-planner* is summarized in Algorithm 1. The planner initializes the search tree on a problem $\mathcal{P}$ (rows 1-2) and defines the initial partial plan π (row 3). While the current plan π contains flaws, the planner tries to refine it until a solution is found (rows 4-15). At each solving step, the flaws present in the current plan π are identified (row 5) and the more relevant is selected according to the *heuristic* $\mathcal{H}$ (row 6). Then, the plan π is refined by applying all the *resolvers* $\in$ *engine* $\mathcal{E}$ that can handle the selected flaw (rows 7-8) and successors nodes are enqueued into the search space fringe according to the *strategy* $\mathcal{S}$ (row 9-10). Namely, the strategy $\mathcal{S}$ defines the node evaluation criteria used in combination with the defined heuristic function $\mathcal{H}$. If all the open planning branches have been investigated and still unsolved flaws are present in π (row 12), then the planning procedure generates a failure (row 13), i.e., a solution plan has not been found. Otherwise, a new current plan is extracted from the fringe (row 14) and the procedure iterate again (going back to row 4). Finally, once the current plan π contains no unsolved flaws, then π represents a solution plan and it is provided as a result.

EPSL provides an enhanced framework for developing applications in which designers may focus on a single aspect of the solving process (i.e., a particular heuristic function or a new resolver in order to add reasoning capabilities to the library) without taking care of all details related to timeline-based planner implementation. In this regard, [14] presents promising experimental results demonstrating the EPSL capabilities. Here, EPSL features have been further developed, with respect to those part of AP2, with the introduction of resource reasoning capabilities. Indeed, EPSL provides now the functionalities (i.e., suitable *resolvers*) to reason about *Renewable Resource* components in addition to the *State Variable* components.

3.2. The Timeline Reasoning Environment for Java: J-TRE

The J-TRE environment [17] tries to go further in the simplification of concepts putting aside the APSI-TRF concept of "component" and merging it with the concept of timeline. From a planning perspective, *timeline*s are considered in J-TRE as a mere collection of tokens (a.k.a. component decisions in APSI-TRF). Within this definition, a timeline represents both a reasoning concept and a solution element. The values that can be accommodated on a timeline as well as the behavior assumed by the planner when a new token is added to a timeline depends on the nature of the timeline itself and, in some cases, on the modeled domain. J-TRE allows the utilization of several families of timelines which provide different modeling ability, such as multi-valued state variables [25] as well as renewable and consumable resources like those commonly used in constraint-based scheduling [22].

It is worth noting that, while APSI-TRF can represent numeric resources, neither OMPS nor AP2 can reason about them in a tightly integrated way causing often excessive thrashing in the search procedure. In particular, we can see that a tighter integration of planning and scheduling techniques allows earlier failure detection improving performances in finding a solution.

By calling *flaw* every possible inconsistency of the current plan, the role of the J-TRE planner is reduced to the identification and the resolution of each flaw in the plan. The planning process proceeds until a consistent plan is found, i.e., the propagation of the solving constraints succeeds and all flaws are eliminated. The general solving strategy broadly entails: (i) identifying a set of flaws, (ii) selecting a flaw according to a *selection strategy*, and (iii) solving it by using a resolution strategy (or *resolver*). If no resolution strategy can be applied or constraint propagation fails, the search backtracks and tries something else.

There can basically be four kinds of flaws: (i) *goal flaws* arise for each token in the problem (problem goal) and when a new token is added to a state variable in a synchronization requirement (subgoaling), (ii) *disjunction flaws* and (iii) *preference flaws* respectively arise when a disjunction or a preference statement is found while enforcing the domain rules (expressed in the DDL dialect used in J-TRE), and (iv) Minimal Critical Sets (*MCSs*) [13] arise when inconsistencies (a.k.a. contention peaks) are detected on some timeline like, for example, different values overlapping on a state variable, reusable resource oversubscriptions, consumable resources overproductions or overconsumption, etc...

Algorithm 2 describes in pseudocode a nondeterministic version of the whole solving procedure. *flaws* denotes the set of all the flaws in the partial plan π provided by procedures $Goals(\pi)$, $Disjunctions(\pi)$, $Preferences(\pi)$ and $MCSs(\pi)$. While the first three procedures can be efficiently implemented with an *agenda* structure, $MCSs(\pi)$ requires some sort of timeline extraction procedure similar to [13] or envelope computation like in [26]. ϕ is a particular flaw in this set. $resolvers$ denotes the set of all possible ways to resolve the current flaw ϕ in the current partial plan π and is given by procedure $Resolve$. The resolver ρ is an element of this set. Finally, π' is the new partial plan obtained by refining π according to resolver ρ through the procedure $Refine$.

Algorithm 2 Solve (π)

$flaws \leftarrow Goals(\pi) \cup Disjunctions(\pi) \cup Preferences(\pi) \cup MCSs(\pi)$
if $flaws = \emptyset$ **then return** π
select any flaw $\phi \in flaws$
$resolvers \leftarrow Resolve(\phi, \pi)$
if $resolvers = \emptyset$ **then return** $failure$
nondeterministically choose a resolver $\rho \in resolvers$
$\pi' \leftarrow Refine(\rho, \pi)$
return Solve (π')

The resolution procedure follows a pure meta-CSP approach allowing seamless integration between planning and scheduling concepts. In particular, flaws of the planning problem constitute the variables of the meta-CSP while resolvers represent their allowed values. The real challenge is than reduced to finding the "right" flaw and ordering its associated resolvers in the "right" way. A standard variable ordering criterion consists in preferring the *most constrained variable*, that is the variable most difficult to instantiate or, equivalently, the variable that is most likely to lead to an infeasible state. The idea, here, is to select the flaw that has the lowest number of resolvers. Two special cases may arise: (i) if the flaw cannot be resolved by any resolver the heuristic selects it allowing early detection of impossibility and saving a great deal of work; (ii) if the flaw can be resolved in only one way (this is equivalent to a deduction or propagation rule), the flaw should be selected because the decision is unavoidable and could provide additional explicit constraints on other flaws still to be solved. For what concerns value selection (in our case resolver ordering), it is quite common to choose the *least constraining value*, that is the value that leaves open as many values as possible for any remaining unassigned variables. While for MCSs and for goal unifications an ordering similar to the min-slack heuristic proposed in [30] seems to work quite well, there is currently no effective ordering heuristic for other kinds of resolvers.

It is worth noting that while the choice of the resolver is a *nondeterministic* step (i.e., it may be required to backtrack on this choice), the selection of a flaw is a *deterministic* step (i.e., there is no reason to backtrack on this choice) as all flaws need to be solved before or later in order to reach a solution plan. The order in which flaws are processed is very important for the efficiency of the procedure but is unimportant for its soundness and completeness [2].

[2] For the sake of completeness, this rule does not apply in case of domains containing consumable resources (not considered in this paper) although the problem can be easily worked around, for example, by ignoring MCSs generated by these resources until the agenda contains no flaws.

3.3. Extending the APSI-TRF planners portfolio

As stated in Section 2, different problem solvers may be developed using the APSI-TRF Domain Layer (see Figure 1) as a shared representation interface. In this work, we take advantage of such capability considering two additional problem solvers and integrating them within the APSI-TRF: (i) EPSL, representing a straightforward example of APSI-compliant problem solver; (ii) J-TRE, endowed with an intermediate structure that enables compliancy with APSI-TRF.

The integration schema is shown in Figure 3. The upper part of the figure shows the standard schema of using APSI-TRF. The domain and problem models are encoded as Domain Definition Language (DDL) and Problem Definition Language (PDL) input files. Then, both DDL and PDL files are parsed and managed by the (component-based) Domain Modeling Engine and a Current Plan (i.e., the initial planning problem) is created to be manipulated by the Problem Solver. Indeed, the current plan is the specialized data structure called Decision Network in APSI-TRF. Then, a generic problem solver, e.g., AP2, applies a solving procedure until the Current Plan satisfies all the planning goals (or fails in finding a solution plan).

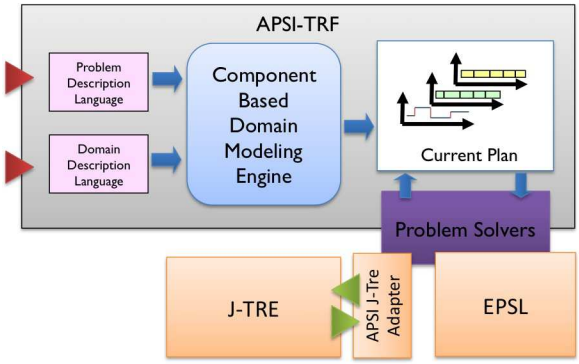

Figure 3. EPSL *and* J-TRE *as* APSI-TRF *solvers.*

Then, similarly to AP2, Figure 3 depicts how EPSL and J-TRE can interact with the Current Plan according to two different modalities: (a) as EPSL has been developed on top of the APSI-TRF modeling interface, it has direct access to the Decision Network; (b) while J-TRE has been developed as a completely independent planning system and, thus, the connection is implemented through an APSI-J-TRE adapter (i.e., an automatic software module) that performs pre- and post-processing of information from the Decision Network. That is, on one hand, the adapter translates the domain and problem information in suitable internal J-TRE structures. On the other hand, the solution generated by J-TRE are encoded as a APSI-TRF Decision Network. In this way, it is possible to extend the portfolio of available planners within the APSI-TRF even with planning systems developed in an independent manner but, obviously, sharing the same planning concepts.

4. KEEN: KNOWLEDGE ENGINEERING FOR TIMELINE-BASED FRAMEWORKS

The use of multiple solvers within the APSI-TRF introduces the issue of extending the framework with suitable knowledge engineering capabilities to support users while working with the system as well as to enhance its usability. The above is one of the motivations that lead us to develop a Knowledge Engineering ENvironment (KEEN) [3] built around the APSI-TRF.

The more general perspective we are pursuing with KEEN is the one of integrating classical knowledge engineering features connected to support for domain definition, domain refinement, etc. with services of automated Validation and Verification (V&V) techniques as those surveyed in [10]. That paper shows the possible role of V&V techniques in domain validation, planner validation, plan verification etc. A further motivation for that work comes from GOAC project [6] that pushed us to better investigate problems of robustness of plans at execution time. In particular, working on the representation of flexible temporal plans, that is the key feature of a timeline-based representation, we have obtained results related to checking the dynamic controllability property [9, 11] as well as to automatically generate robust plan controllers [27]. In those works, the problem of verifying flexible plans has been addressed considering an abstract plan view as a set of timelines with formal tools like model checkers. Then, the flexible plan verification problem has been translated in a model checking problem with Timed Game Automata (TGA), exploiting UPPAAL-TIGA [2] as verification tool. The goal pursued with KEEN synthesis is to obtain an integrated environment where all these results can be situated in a rational tool design and their use facilitated by the software environment.

In [3], we describe the features of the KEEN environment for Domain Definition and Visualization as well as some V&V tools at work applied to a running example, i.e., the GOAC domain, where we have accumulated quite an amount of basic knowledge. Here, the use of multiple solvers is introduced, thus providing each of them with a set of editing and visualization functionalities as well as V&V services that aim at supporting the design and development phase of P&S applications. The use of the KEEN system enables the possibility to provide also J-TRE and EPSL with the same Knowledge Engineering (KE) functionalities exploiting the shared representation layer of the APSI-TRF (see Fig. 3). Thus, every APSI-compliant solver (i.e., AP2, EPSL and J-TRE) takes advantage of a unique environment capable of supporting design and development phases.

More in detail, in KEEN, the APSI-TRF is surrounded by a set of active services that give support during the KE phase. Indeed in our view the knowledge engineering phase is interpreted in a very broad sense. For example, we also have a Plan Execution block (see Fig. 4) that contains a Dispatch Service to send actual commands to a controlled system and an Execution Feedback module that allows to receive the telemetry from an actual plan execution environment. The idea pursued is that you can

connect the KEEN to an accurate simulator of the real environment, to a real physical system (e.g., a robot) and have functionalities to monitor with visual tools also the execution phase. In Figure 4, the KEEN system is depicted as the composition of "classical tools" you may expect in a environments and by V&V services.

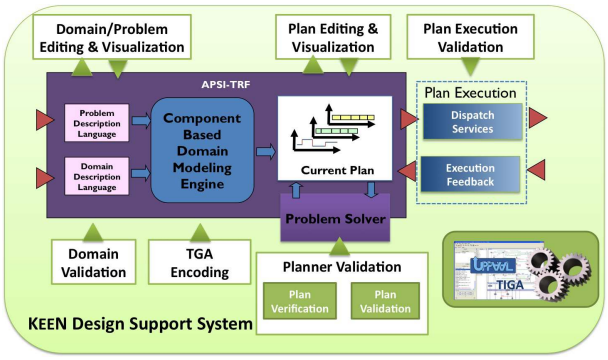

*Figure 4. The KnowledgE ENgineering (*KEEN*) Design Support System.*

In particular, the *Domain Editing and Visualization* module provides initial solution for a user interaction functionality for creating planning domain models. In this respect, we have developed an *Eclipse*[3] plugin that provides a graphical interface to model, visualize and analyze the P&S domains. Additionally, plans can be generated indifferently by means of either AP2, EPSL or J-TRE in a continuous loop of usage. The V&V services, comes from work described in papers like [10, 9, 11, 27]. They are all based on the use of Timed Game Automata, exploiting UPPAAL-TIGA [2]. As a consequence their entry point is the *TGA Encoding* module that implements a translation from P&S specification to TGA. The encoding method is the one presented in [9] allowing to share the same formal results presented in [11]. The other services rely on that encoding. The *Domain Validation* module is to support the model building activity providing a tool to assess the quality of the P&S models. The *Planner Validation* module is deputed to assess the P&S solver(s) with respect to system requirements. It is worth specifying how two sub-modules are needed: *Plan Verification* to verify the correctness of solution plans and *Plan Validation* to evaluate their goodness. Then, a *Plan Execution Validation and Controller Synthesis* module is to check whether the solution plans proposed by the solver(s) are suitable for actual execution as well as to generate robust plan controllers. To implement the modules functionalities, verification tasks are performed by means of UPPAAL-TIGA. Such a tool extends UPPAAL [23] providing a toolbox for the specification, simulation, and verification of real-time games. As a result, UPPAAL-TIGA is an additional core engine for KEEN.

5. APPLICATION TO THE GOAC DOMAIN

This section presents an initial evaluation of the different planners. As a reference we consider the GOAC domain [6] based on a robotic platform responsible for the

[3]http://www.eclipse.org

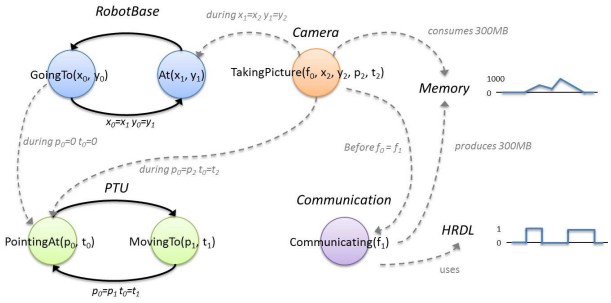

Figure 5. State variables and resources describing the robotic platform and the orbiter visibility.

movements, a payload represented by two stereo-cameras mounted on a Pan-Tilt Unit (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. As the Orbiter may be not visible for some periods during the mission, 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.

During operation, the rover should follow some causal rules to maintain safe and effective configurations. Namely, the following conditions must hold during the overall mission: (a) while the robot is moving, the PTU must be in a safe position[4]; (b) the robotic platform can take a picture only if the robot is still in one of the requested locations while the PTU is pointing to the related direction and if an adequate amount of on board memory is available to store the picture; (c) once a picture has been taken, the rover has to communicate the picture to the base station; (d) while communicating the rover has to stay still and the memory is released of the amount of transmitted file; (e) while communicating, the orbiter needs to be visible.

A possible mission action sequence is the following: navigate 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 next requested location. Once all the locations have been visited and all the pictures have been communicated, the mission is considered successfully completed.

Figure 5 offers a detail of the values that can be assumed by these state variables and the legal value transitions in accordance with the mission requirements and the robot physics. 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 $MovingTo(pan, tilt)$ value. The camera either takes a picture of a given object in a position $\langle x, y \rangle$ with the PTU in $\langle pan, tilt \rangle$ ($TakingPicture(file - id, x, y, pan, tilt)$) or is idle. Similarly, the communication facility can be either operative and dumping a given file ($Communicating(file - id)$) or idle. The consumable resource *Memory* represents memory consump-

[4]A safe PTU position is assumed to be $(pan, tilt) = (0, 0)$.

tion in time. Additionally, one external resource, the *HRDL*, represents contingent events, i.e., the communication channel availability.

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. 5).

In the actual domain model, an additional state variable is considered: the *Mission Timeline*. Such state variable is used just to model the reception from the external facilities of high level mission goals, i.e., $TakePicture(pic, x, y, pan, tilt)$ and $At(x, y)$ to model, respectively, the goal of taking a picture with a particular position/PTU setting and just moving the rover to a certain position. These goals are set on the Camera and RobotBase timeline as actual planning goals.

Evaluation Setup. Different GOAC problems are 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 on two set of problems: (a) **Set A** - *problems with state variables and no resources*. Problems can be solved by the three planners (AP2, EPSL, J-TRE). Comparisons are given in Figure 6, plots (a-d); (b) **Set B** - *problems with state variables and resources*. Problems can be solved by the the two new planners (EPSL, J-TRE). Comparisons are given in Figure 6, plots (e-h). Notice that J-TRE uses a randomized algorithm hence we report two results: a generic run (plot label J-TRE), the best result out of ten runs (J-TRE*). Problems in the current sets are solvable, hence we plot the time in msec needed to obtain the solution.

Results. A first comment concerns AP2: the planner is tailored for solving GOAC problems hence is performance is consistently good, but some scalability problems appear in the 4-time-window set. We observe that EPSL has a stable performance on both Set A and Set B. It solves the complete pool of problems with a natural increase of time for larger instances. Interesting is also the behavior of J-TRE: it is able to solve the problem very quickly if we plot the best over ten runs, but clearly we have to consider the cumulative time needed. The plot of the generic run (label J-TRE) is interesting. The planner solves all the instances, showing a less regular behavior with respect to EPSL due to the effects of randomization. Overall, it is worth underscoring that both EPSL and J-TRE solve problems with resources copying well with the scalability required by Set B (that is in line with the average problem complexity required in GOAC scenario tests).

6. CONCLUSIONS

This paper describes some recent research directions pursued around the ESA APSI platform. The goal is to demonstrate that improvements of the platform are possible and continuous investments of the agency on the platform are worth being pursued. It is worth saying that the improvements we are describing have a research goal but the link to the platform demonstrates that they can easily turned in real use for the ESA missions as shown by the GOAC example that are considering the real models developed for the project.

In the paper we have demonstrated once more the capability of APSI to interface multiple solvers. In [19] we have seen an example of two solvers (the AP2 and the APSI-Reactor) that combined together achieve the functional goal of planning and executing. Here we demonstrate an alternative use with J-TRE and EPSL that can be used for generating alternative solutions for the same problem, paving the way to create in the future a platform for on-ground analysis of plans that can be tailored for different missions.

The paper also shows some additional features of the KEEN system aimed at supporting domain definition and in general timeline-based reasoning. The lack of an environment for supporting domain modeling is a well known of APSI and we are aware that the Agency is pursuing other initiatives for addressing the problem. Our effort is focused on building an environment for integration of domain modeling and validation and verification tools as described in [3]. Using the environment we have compared different planners on a defined set of GOAC problem and obtained the quantitative results described before.

Acknowledgments. Authors competence in the GOAC domain comes from their previous participation in the TRP/T313/006MM. Their research is currently supported by PST internal funds and by CNR under the GECKO Project (Progetto Bandiera “La Fabbrica del Futuro”).

REFERENCES

[1] J. Barreiro, M. Boyce, M. Do, J. Frank, M. Iatauro, T. Kichkaylo, P. Morris, J. Ong, E. Remolina, T. Smith, and D. Smith. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *ICKEPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling*, 2012.

[2] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. UPPAAL-TIGA: Time for playing games! In *Proc. of CAV-07*, number 4590 in LNCS, pages 121–125. Springer, 2007.

[3] G. Bernardi, A. Cesta, A. Orlandini, and A. Finzi. A Knowledge Engineering Environment for P&S with Timelines. In *Proc. of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2013)*, 2013.

[4] A. Blum and M. L. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.

[5] B. Bonet and H. Geffner. Planning as Heuristic Search. *Artificial Intelligence*, 129:5–33, 2001.

[6] A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and

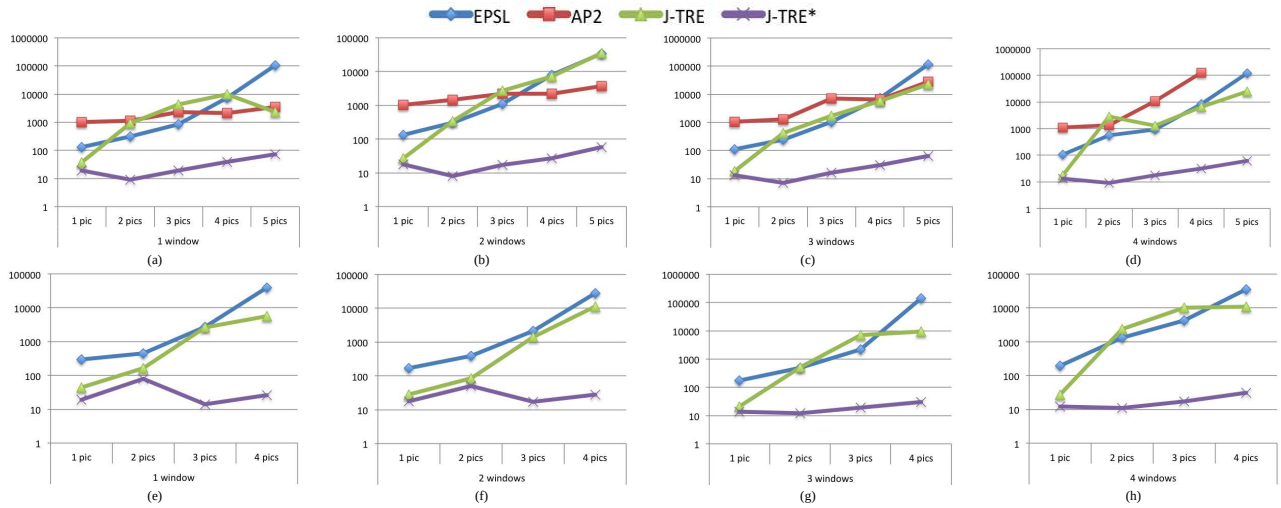

Figure 6. Testing results and comparison among EPSL, AP2 *and* J-TRE *on the* GOAC *problem with: (a) one communication window; (b) two communication windows; (c) three communication windows; (d) four communication windows. Timings are in milliseconds. Problem Set A: plots (a-d). Problem Set B: plots (e-h).*

M. van Winnendael. A Goal-oriented Autonomous Controller for Space Exploration. In *Proc. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, Noordwijk, the Netherlands, April 2011.

[7] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proc. of the 21th Conf. on Innovative Applications of Artificial Intelligence*, 2009.

[8] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella. An Innovative Product for Space Mission Planning: An A Posteriori Evaluation. In *ICAPS-07*, pages 57–64, 2007.

[9] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Analyzing Flexible Timeline Plan. In *ECAI 2010. Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215. IOS Press, 2010.

[10] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Validation and Verification Issues in a Timeline-Based Planning System. *Knowledge Engineering Review*, 25(3):299–318, 2010.

[11] A. Cesta, A. Finzi, S. Fratini, A. Orlandini, and E. Tronci. Flexible plan verification: Feasibility results. *Fundamenta Informaticae*, 107:111–137, 2011.

[12] A. Cesta, S. Fratini, A. Donati, H. Oliveira, and N. Policella. Rapid Prototyping of Planning & Scheduling Tools. In *SMC-IT 2009. Third IEEE International Conference on Space Mission Challenges for Information Technology*. IEEE Press, 2009.

[13] A. Cesta, A. Oddi, and S. F. Smith. A Constraint-based Method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–136, 2002.

[14] A. Cesta, A. Orlandini, and A. Umbrico. Toward a general purpose software environment for timeline-based planning. In *Proc. of the 20th RCRA International Workshop on "Experimental Evaluation of Algorithms for solving problems with combinatorial explosion"*, 2013.

[15] S. Chien, D. Tran, G. Rabideau, S. Schaffer, D. Mandl, and S. Frye. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th Int. Conf. on Automated Planning and Scheduling*, 2010.

[16] S. A. Chien, M. Johnston, J. Frank, M. Giuliano, A. Kavelaars, C. Lenzen, and N. Policella. A Generalized Timeline Representation, Services, and Interface for Automating Space Mission Operations. In *SpaceOps*, 2012.

[17] R. De Benedictis and A. Cesta. Timeline Planning in the J-TRE Environment. In J. Felipe and A. Fred, editors, *Agents and Artificial Intelligence. ICAART 2012 Revised Selected Papers*, volume 358 of *Communications in Computer and Information Science*, page 218233. Springer Berlin Heidelberg, 2013.

[18] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[19] S. Fratini, A. Cesta, A. Orlandini, R. Rasconi, and R. De Benedictis. APSI-based Deliberation in Goal Oriented Autonomous Controllers. In *Proc. of 11th ESA Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2011.

[20] S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

[21] A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on AI Planning and Scheduling*, 2000.

[22] P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artificial Intelligence*, 143:151–188, February 2003.

[23] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[24] N. Muscettola. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling*. Morgan Kauffmann, 1994.

[25] N. Muscettola. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling*. Morgan Kauffmann, 1994.

[26] N. Muscettola. Incremental Maximum Flows for Fast Envelope Computation. In *ICAPS*, pages 260–269, 2004.

[27] A. Orlandini, A. Finzi, A. Cesta, and S. Fratini. Tga-based controllers for flexible plan execution. In *KI 2011: Advances in Artificial Intelligence, 34th Annual German Conference on AI.*, volume 7006 of *Lecture Notes in Computer Science*, pages 233–245. Springer, 2011.

[28] N. Policella, H. Oliveira, and E. Benzi. Planning Spacecraft Activities: An Automated Approach. In *ICAPS-13. Proc. 23th Int. Conf. on Automated Planning and Scheduling*, Rome, Italy, June 2013.

[29] K. Rajan, F. Py, and J. Barreiro. Towards Deliberative Control in Marine Robotics. In M. Seto, editor, *Marine Robot Autonomy*. Springer Verlag, December 2012.

[30] S. F. Smith and C.-C. Cheng. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, pages 139–144. AAAI Press, 1993.

[31] D. S. Weld. An Introduction to Least Commitment Planning. *AI Magazine*, 15(4):27–61, 1994.