QUIJOTEEXPRESS - A NOVEL APSI PLANNING SYSTEM FOR FUTURE SPACE ROBOTIC MISSIONS

Juan M. Delfa Victoria[1,2,3], **Simone Fratini**[2], **Nicola Policella**[2], **Yang Gao**[3], **and Oskar von Stryk**[1]

[1]*Technische Universität, Darmstadt, 64289 Darmstadt, Germany*
[2]*European Space Agency, 64293 Darmstadt, Germany*
[3]*University of Surrey, GU2 7XH Guildford, United Kingdom*

ABSTRACT

Planetary rovers require automated systems in order to overcome the difficulties of deep space missions. This paper presents APSI*, an evolution of the APSI framework and QuijoteExpress, its planner intended to improve the performance of previous planners, handle the uncertainty inherent to scenarios like Mars and facilitate the work of the operators using the system.

Key words: Timeline-based planning; Autonomy; Planetary rover; APSI; QuijoteExpress.

1. INTRODUCTION

Robotics represent a multidisciplinary area which has not yet been exploited due to the multiple scientific advancements it requires. Specially relevant for deep-space robotic missions is the lack of autonomous systems. It is wrongly understood that the technology is not yet ready as there has been in fact a number of successful examples in the recent history [19, 20, 18, 4, 3, 6]. It is the understanding of the authors that other reasons have prevented a faster integration of these technologies on space missions, being one of them the lack of confidence. The collaboration between human operators and autonomous systems use to be problematic as the user cannot express what he wants or do not understand/agree the output of the system. However, autonomy is not optional. Given the increasing complexity of future missions (the reader just need to compare Curiosity and MER Mars rovers), the advantages in terms of science return and costs are undeniable. As an example, MER would have never been so successful without the AUTONAV system [3]. While autonomous navigation is already well understood, new opportunities seem to open up to autonomy, being AEGIS [14] another good example.

This paper presents a novel planning formalism called HTLN inspired on two different but complementary planning theories: timeline planning (TLP) [19, 15, 17] and hierarchical task networks (HTN)[13]. We further present an implementation of this formalism called APSI*, an evolution of APSI framework[16] and a temporal, hierarchical, heuristic-driven and domain-independent planner called QuijoteExpress (QE) based this new framework.

Our motivation is to tackle some of the main problems found on space robotics. First, to improve APSI-based planner performance, which has been shown to be critical in some scenarios where on-board re-planning needs to be agile in order to react in time to unexpected events [20, 10] and could be relevant in future rover missions demanding higher agility such as MSR. Second, to be able to deal with problems that present uncertainty in any of its dimensions as expressed in the list bellow. This property is specially relevant for rovers, as mars represent an unpredictable environment for which the generation of robust plans is still a challenge. Third, to improve the interaction user/planner in two ways: increase the language expressiveness to create more realistic models and second to generate more understandable plans, easier to be verified and validated by visual inspection. Finally, we intend to design a system backwards compatible with previous APSI versions without affecting the previous goals.

We have focused our research on a planner capable of generating realistic plans for planetary rover missions. However, this fact represents and advantage rather than an impediment in order to use the planer in other type of missions, as planetary rovers (and other robotic domains) have a number of specific characteristics that make them very hard to be solved from a planning point of view:

- Uncertainty - Dynamic environment: The environment can spontaneously change its state due to external events.
- Uncertainty - Partial observability: Some aspects of the state of the world are unknown. It has three consequences: Planning based in a complete understanding of the world is not feasible, some of the assumptions considered during planning might be wrong and new relevant information for the plan might be discovered only during execution time.

- Uncertainty - Non determinism: Not possible to estimate with precision the outcome of the robot actions.
- Complexity: Space missions has increased exponentially its complexity [12, 2]. From the point of view of autonomy, engineers need the help of tools to create more understandable models and plans.
- Highly constrained: In real applications and specially in space robotics, constraints play a major roll both in the design of the models and during the plan generation.
- Restricted communications: Some scenarios do not allow continuous or real-time communications with the robot. For example, the round trip of a radio signal from Earth to Mars can take more than forty minutes.
- Low CPU performance: Space-oriented processors have much lower performance than those integrated in conventional computers.
- Failure recovery: The possibilities to recover a mission in case of a spacecraft failure are very limited. Therefore, safety and V&V play a major roll in space missions.

Along the paper, a mars rover scenario will be used to introduce examples and concepts. It consists on a single robot that can drive either in autonomous way to a target point, or directed with more precision in order to approach a target. The robot can also take pictures and send information back to Earth.

The rest of the paper is organized as follows. Section 2 presents the background in which HTLN is based. Section 3 introduces HTLN and APSI*, an extension of APSI that implements HTLN formalism. Section 4 presents the concept of sufficient planning, central to handle uncertainty. Section sec:qe introduces QuijoteExpress, a planner based on APSI* and initial results are given in Section 6. Finally Section 7 presents the conclusions and future work.

2. BACKGROUND

As mentioned in the introduction, HTLN[11] lies on three different sources: TLP, HTN and APSI.

Timeline planning and temporal planning in general allows to use time in the representation of actions and constraints. There are several examples of temporal planners that have been applied to space missions. HSTS [19], used for the short term scheduling of the Hubble Space Telescope, may be the first successful system. Europa2 [15], successor of HSTS, is currently used for several NASA missions including MAPGEN [4], the ground-based daily activity planning system for the Mars Exploration Rover (MER). ASPEN [9] is a hierarchical, timeline-based planner used in several space missions such as EO-1, Data-Chaser, Citizen Explorer, etc. Finally, IxTET was a pioneer about time and resources reasoning. It has been tested in different robotic platforms, even though it was never used in a real mission. APSI is a framework to develop timeline-based automated planning and scheduling solutions widely documented in several papers [7, 17]. A continuation of this project produced APSI-GOAC [5] which integrates dispatching and execution capabilities. Finally, a planner called AP^2 [8] was added to the framework.

HTN is one of the most used planning techniques in real-world applications, in part because it allows to effectively encode knowledge into domain-independent planners. It is based on the differentiation between primitive tasks that can be directly executed and compound tasks, which must be decomposed in primitive tasks. The organization of tasks in hierarchies helps to simplify the modeling, which is one of the major problems that engineers need to face to deploy automated tools and at the same time allows for more understandable plans for the human expert. Some relevant HTN planners used in real applications are SIPE-2 [22] and O-Plan [21].

3. HIERARCHICAL TIMELINE NETWORKS (HTLN)

Temporal planning and HTN are powerful techniques, but they have both some deficiencies that can be overcome by combining them in a new joint formalism that could satisfy the objectives described in section 1. While there is no "standard" timeline planning theory, HTN definition is broadly accepted. Nevertheless, in order to combine both theories into HTLN[11], it was required to adapt the existing terminology, both adding and redefining some concepts. It is important to emphasize that we have tried to be as compliant as possible with the underlying theories.

Even though a formal definition of HTLN is out of the scope of this paper, its main concepts (summarized in table 1) are briefly described in the following subsections.

Problem (P) and Decision Network (dn). TLP and HTN represent a *problem* as a partially supported *network* called *decision network* (dn) in APSI and *task network* in HTN. Both networks are represented as a graph $w = (N, E)$ that contains a set of $goal$'s (TLP) or $task$'s (HTN), that is the nodes of the graph and relations among them (the edges). Besides, a list of initial conditions (ic) are used to specify the initial state of the world. While in TLP ic's are directly encoded into the network, in HTN they are represented as a list of atoms passed together with the network to the planner.

In APSI a decision network is implemented as a hypergraph. However, this structure is not enough to represent hierarchical tasks. In order to represent how a compound task is decomposed in sub-tasks, a tree-structure seems

HTN	APSI	HTLN	Symbol
Primitive task	Component decision	Primitive decision	d^p
Compound or Nonprimitive task	-	Compound decision	d^c
Operator	Component Decision	Primitive decision	d^p
Method	-	Method	m
Task Network	Decision Network	Decision Network	dn
Decomposition	-	Decomposition	δ
Action	Instantiated component decision	Action	a
Transition function	Transition function	Transition function	γ

Table 1. HTLN nomenclature

to be optimal, but trees are not suitable for HTLN as we need to represent also relations between the sub-tasks. In consequence, a dn is represented in HTLN by a combination of these two data structures as a set of nested hypergraphs organized in layers, that is, as a graph (cyclic tree) of hypergraphs. A dn is now formed by a set of relations, decisions and other dn's representing compound tasks already decomposed, formally $dn = (N, E, DN)$. The dn that represents the problem is situated in the layer with index 0, its sub-goals (simple and compound) in layer 1 and so forth, being the maximum possible depth equivalent to the maximum length of all decomposition trees.

Primitive decision (d^p). The original implementation of APSI does not distinguish between primitive and compound tasks. However, in order to support HTN planning, we need to increase the expressiveness of the framework. Like in HTN, a primitive decision (d^p) cannot be further decomposed and is directly executable. It is represented in HTLN as a component decision (cd), that is a node of the dn. The set of all d^p's in a problem will be referred to as D^p.

Compound decision (d^c). Unlike HTN, a compound decision d^c in HTLN can be implemented in different ways. In case it has not yet been decomposed, it is represented as a node, that is a cd. Otherwise it is represented as a sub-hypergraph (dn) that will be added to dn_i. For the sake of simplicity, we will refer to the network selected to decompose a task as dn_{dec}. In order to distinguish whether a cd contains a d^p or a d^c, a flag $isPrimitive$ is added to each cd.

A dn in HTLN is a sub-type of cd, therefore dn_{dec} inherits the value, parameters and relations of d^c. In consequence, a planner based on HTLN can interpret a decomposed compound task such as d^c as a black box by using

Task type	Structure
d^p	$cd, isPrimitive(cd) = true$
d^c	cd or $dn, isPrimitive(cd/dn) = false$

Table 2. Task type versus structure

dn_{dec} as a cd. Due to this duality, for a given dn, every decomposed compound decisions such as $d \in DN$ is also included in the list of nodes $d \in N$ of dn. This property is useful in case the planner is not interested on the details about how d^c is achieved and just wants to make modifications that affect d^c as a whole. It can also interpret d^c as a sub-problem, that is, as the sub-hypergraph represented by dn_{dec}, which is helpful in case the planner wants to make modifications on the sub-tasks. Encapsulating tasks into compound tasks allows for a more rational modeling, closer to the organization of real systems, as well as more understandable outcomes by the planner. It also improves the planner performance, as it is possible now to affect groups of cd's (like dn_{dec}) applying just one resolver (section 3). The set of all d^c's is represented with the symbol D^c, where $D^p \cap D^c = \emptyset$ and $D^{all} = D^p \cup D^c$ is the set of all decisions.

Summarizing, in HTLN the structure used to store a task and the type of task are decoupled as shown in table 2.

Method (m). A *method* is used to express control-knowledge that tells the planner which areas of the search space should be explored by telling the only valid ways in which compound tasks can be decomposed. It is formally defined as $m = (name(m), task(m), network(m))$ where $name(m)$ is a unique name for the method, $task(m)$ a nonprimitive task and $network(m)$ is a task network whose tasks are called the *sub-tasks* of m. If there is a substitution σ such that $\sigma(t) = task(m)$, then m is *relevant* for t and the decomposition of t by m under σ is $\delta(t, m, \sigma) = network(m)$. Considering that the same task might happen in several nodes, σ is used to rename the different occurrences of the same task (changing the name of the parameters in $name(m)$) in order to be able to distinguish them.

The only difference between the definition of method for HTN and HTLN is that for the second, the task is represented as a cd and the network as a dn.

The set of methods M of a domain constitutes the domain-dependent information used by a domain-independent planner to limit the size of the search space. If every d^c in the domain can be decomposed to a level in which all its sub-tasks are d^p, the model is complete. Model completeness has important consequences, as a planner cannot generate an executable plan in case the model is incomplete.

The following example shows a formal description of two methods relevant for the compound task *drive*:

Each network is composed of a set of decisions N and

$name : m\_drive_1$;
$task : drive(p_{init}, p_{dest})$;
$network$:
$N = \{n_1 : autonav(p_{init}, p_i), n_2 : drive(p_i, p_{dest})\}$
$E = \{e_1 : f^{before}_{temp}(n_1, n_2)\}$

$name : m\_drive_2$;
$task : drive(p_{init}, p_{dest})$;
$network : \ N = \{n_1 : approach(p_{init}, p_{dest})\} \ \ E = \emptyset$

constraints E. In m_drive_1, the temporal constraint e_1 establishes that the different $autonav$'s will happen in sequence and implicitly that $approach$ will take place in the end.

Relations (rlt). A relation rlt represents an edge that bind together the decisions (nodes) affected by rlt. Temporal and parameter relations used in temporal planning, more specifically in APSI, are a superset of those traditionally used in HTN planning. In consequence, HTLN is based in APSI relations:

- Temporal ($f^{type_t}_{temp}(d_1, d_2)$): Usually represented as an binary Allen [1] temporal constraint between decisions where $type_t$ is a quantitative Allen relation and d_1, d_2 the decisions involved. For example: $drive\ before\ communication$.
- Parametric ($f^{type_p}_{param}(p_1, p_2, \ldots, p_m)$): N-ary boolean or numerical relation between parameters where $type_p$ is a linear parameter constraints such as $equal$, $bigger\_than$ or a linear function and $p_1, p_2, \ldots, p_m$ the parameters involved. For example: $pointing_{camera} = g(position_{target})$ where g is a numerical function such as multiplication or division.

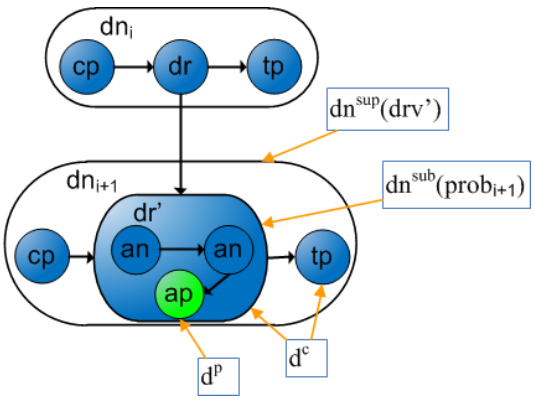

Figure 1. Hierarchical dn structure

Figure 1 illustrates the concepts presented before. dn's are represented as ellipses, cd's as circles and rlt's as segments. The problem contains three compound goals (cp, dr, tp) which stand for *calculate path*, *drive* and *take picture* respectively, ordered between them by *before* temporal constraints (represented as arrows). Drive is the only d^c already decomposed. Therefore, it is represented not as a cd but as a dn (dr') containing three sub-tasks, where an stands for *autonav* and ap *approach*. While ap is primitive, an is compound and will require to be further decomposed adding another nested dn into drive. $dn^{sup}(d)$ represents the dn one layer up of d and $dn^{sub}(d)$ one layer bellow d. Being dn_i the problem network, $dn^{sup}(dn_i) = \emptyset$, that is, a problem has no layer above and $dn^{sub}(d) = \emptyset$ if d is in the last layer.

Resolver (ρ). A resolver ρ is a procedure intended to fix a flaw ϕ in a given problem dn_i. In classical planning there are two types of resolver: causal links and addition of new actions. HTN requires one more, the *decomposition function* $\delta(t, m, \sigma) = network(m)$ that decomposes a task t in $network(m)$ under the substitution $\sigma(t) = task(m)$. Temporal planners require more sophisticated resolvers to satisfy the temporal constraints.

A resolver is formally defined as $\rho(dn_i, \phi) = \{d^+, d^-, rlt^+, rlt^-\}$ being d^+, d^-, rlt^+, rlt^- the lists of decisions (cd or dn) and relations to be added/deleted respectively. HTLN adds to APSI one extra resolver in order to deal with compound tasks: the *decomposition* resolver.

Decomposition resolver (ρ_δ). When a compound task t_n is decomposed in HTN, it is replaced in the task network by the network $dn_{dec} = \{N_{dec}, E_{dec}\}$ of the method m selected. HTLN takes a different approximation. The task t_n is maintained in the problem as a sub-network that includes the set of elements (decisions and relations) in which it is decomposed. This allows for a compact representation that keeps sub-tasks organised. Besides, in section 5 it is explained how this fact can be used by the planner to improve its performance.

Suppose $dn_i = (N, E, DN)$ a network representing a problem, where N is the set of nodes, E the set of edges and DN the set of sub-networks $dn^{sub}(dn_i)$. Suppose $n \in N$ is a node that contains a task $t_n \in D^c$ and $m \in M$ a relevant method for t_n. Then m can decompose t_n in $dn_{dec} = \{N_{dec}, E_{dec}, DN_{dec}\}$ under the substitution σ.

A decomposition resolver is formally expressed in HTLN as $\rho_\delta(dn_i, from, to, \sigma) = \{d^+, d^-, rlt^+, rlt^-\}$ where:

- dn_i: Decision network that represents the problem.
- $from$: d^c to be decomposed.
- to: $network$ of the method m selected to decompose d^c.
- σ: Substitution that matches an specific instantiation of a task with the original task name, for

example $drive(p_1 = (0,0), p_2 = (3,7))$ with $drive(p_{init}, p_{dest})$.

The output of ρ_δ is the following:

$$\begin{aligned} d^+ &= \{dn_{dec}, N_{dec}\} \\ d^- &= t_n \\ rlt^+ &= \{E_{dec}, E_{modif}, E_{temp}, E_{param}\} \\ rlt^- &= E_n \end{aligned} \tag{1}$$

The nodes of dn_{dec} plus dn_{dec} itself represent the list of decisions to be added (recall that $D_{dec} \in N_{dec}$), while t_n has to be deleted. With respect to the relations, let E_n be the set of edges that have t_n in their scopes. For each edge $e_i \in E_n$, replace t_n by dn_{dec} in e_i's scope and add e_i to E_{modif}. For each decision $cd_j \in dn_{dec}$, a temporal relation $contains$ between dn_{dec} and cd_j is added to E_{temp}, meaning that each sub-task of dn_{dec} must be executed within the time window assigned to t_n. Finally, for each parameter $param_l$ of dn_{dec} equivalent to another parameter $param_j$ in any of its sub-tasks cd_j, a parameter relation $equal$ is also added to E_{param}. Formally:

$$\begin{aligned} &E_{modif}(e_n) = \forall e_i \in e_n, \\ &scope(e_i) = scope(e_i) - t_n \cup dn_{dec} \end{aligned} \tag{2}$$

$$\begin{aligned} &E_{temp}(dn_{dec}) = f^{contains}_{temp}(dn_{dec}, cd_j) \\ &\forall cd_j \in dn_{dec} \end{aligned} \tag{3}$$

$$\begin{aligned} &E_{param}(dn_{dec}) = f^{equal}_{param}(param_l, param_j) \\ &\forall param_l \in dn_i, \forall param_j \in cd_j \in dn_{dec}, name(param_l) = \\ &name(param_j) \end{aligned} \tag{4}$$

Transition function (γ). In classical and HTN planning, the *state-transition function* γ is used to calculate the new state s' of the problem by applying the effects of a given action a to the current state s, formally $\gamma(s,a) = s'$. In HTLN, this definition is extended replacing a state s by a problem dn_i and the action a by a set of resolvers.

Given the problem network $dn_i = (N, E, DN)$ and ρ a list of resolvers suppose, without loss of generality, that ρ is formed just by a decomposition $\rho = \{\rho_\delta(dn_i, from, to, \sigma)\}$. Suppose $n \in N$ is a node that contains a task $t_n \in D^c$ and $m \in M$ a relevant method for t_n. Then the transition function γ under the resolver ρ_δ is calculated as shown in equation 5.

$$\begin{aligned} &\gamma(dn_i, \rho_\delta) = (N - d^- \cup d^+, E - rlt^- \cup rlt^+, \\ &DN \cup dn_{dec} \cup DN_{dec}) \end{aligned} \tag{5}$$

$\gamma(dn_i, \rho_\delta)$ generates an evolution $dn_{i+1} = (N_{i+1}, E_{i+1}, DN_{i+1})$ of dn_i when it decomposes t_n in dn_{dec}. Intuitively, the list of nodes N_{i+1} and edges E_{i+1} are calculated from their predecessors by adding and deleting the corresponding lists generated by the resolver. Regarding the list of sub-networks, DN_{i+1} adds dn_{dec} and all its sub-networks DN_{dec}. This process is propagated to the upper networks $dn^{sup}(dn_i)$ until layer 0 is reached.

Continuing with the rover example, given the following problem:

$$\begin{aligned} &dn_i : \\ &N = \{n_1 : drive(p_{init}, p_{dest}), \\ &n_2 : takepic(file, x, y, pan, tilt)\} \\ &E = \{e_1 : f^{before}_{temp}(n_1, n_2)\} \\ &DN = \{\emptyset\} \end{aligned} \tag{6}$$

$drive(p_{init}, p_{dest})$ can be decomposed using the two methods shown before, m_drive_1 or m_drive_2. A non-deterministic algorithm (heuristic) will be in charge of deciding which one of them should be applied. Suppose that m_drive_1 is selected. The first step consists on generating an instance of $task(m\_drive_1)$ under a substitution such as $\sigma(p_{init} \leftarrow p3_{init}; p_{dest} \leftarrow p3_{dest})$ as follows:

$$\begin{aligned} &n_3 : \\ &name = drive(p3_{init}, p3_{dest}) \\ &N_{dec} = \{n_4 : autonav(p4_{init}, p4_i), n_5 : drive(p5_i, p5_{dest})\} \\ &E_{dec} = \{e_2 : f^{before}_{temp}(n_4, n_5)\} \end{aligned} \tag{7}$$

Next, the edges from dn_i that have n_1 in its scope must be recalculated. In this case, e_1 is modified, replacing the old n_1 by the new n_3:

$$E_{modif} = \{e_3 : f^{before}_{temp}(n_3, n_2)\} \tag{8}$$

Contains temporal relations must be added between the compound task *drive* and each of its sub-tasks:

$$E_{temp} = \{e_4 : f^{contains}_{temp}(n_3, n_4), e_5 : f^{contains}_{temp}(n_3, n_5)\} \tag{9}$$

And parameter relations between the compound task *drive* and each of its sub-tasks:

$$\begin{aligned} &E_{param} = \{e_6 : f^{equal}_{param}(p3_{init}, p4_{init}), \\ &e_7 : f^{equal}_{param}(p3_{dest}, p5_{dest})\} \end{aligned} \tag{10}$$

The result of the decomposition is:

$$\begin{aligned} &\sigma(p_{init} \leftarrow p3_{init}; p_{dest} \leftarrow p3_{dest}) = \\ &d^+ = \{n_3, n_4, n_5\}, d^- = \{n_1\}, \\ &rlt^+ = \{e_2, e_3, e_4, e_5, e_6, e_7\}, rlt^- = \{e_1\} \end{aligned} \tag{11}$$

And the result of the transition is:

$$\gamma(dn_i, \rho_\delta(dn_i, n_1, m_drive_1, \sigma)) = (N_{i+1} = \{n_2, n_3, n_4, n_5\}, E_{i+1} = \{e_2, e_3, e_4, e_5, e_6\}, DN_{i+1} = \{n_3\}) \quad (12)$$

4. SUFFICIENT PLANNING

Traditional timeline planners search for fully justified plans, which are complete, fully ordered and valid:

- Complete: All variables are grounded. The plan always specifies how to proceed (the timelines have no gaps).
- Fully ordered: All the decisions are sequenced.
- Valid: All the constraints are satisfied.

This definition might represent an unachievable condition for real-world P&S systems. In order to deal with non deterministic, dynamic and partially observable environments, more flexibility is required. HTLN is based on the concept of Sufficient Plan, defined as follows:

All variables and relations are sufficiently grounded, all fully grounded relations are satisfied, all actions are sufficiently decomposed and all the mandatory goals can be achieved for at least one specific instantiation of the sufficient plan.

This definition represents an extension of the definition provided in [15], where a partial plan is fully defined up to a certain point called plan horizon, ignoring activities that fall outside it. In our case, any decision or constraint might be partially defined according to an initial definition, giving to the executive the responsibility to fill in the gaps prior to the execution.

The different concepts involved in this definition are described in the following sections.

Sufficiently grounded All decisions $d \in D^{all}$ and relations $rlt \in R$ that appear as goals in the problem must specify whether they should be grounded or not at planning time. A decision is grounded when its value and parameters are grounded; a relation is grounded when all the decisions of its scope (those affected by the relation) are grounded. A partially grounded relation has two important consequences: (1) the relation cannot be satisfied, (2) in case of temporal relations, the resulting dn is partially ordered. This property is not yet implemented in APSI*, therefore the concept of sufficient plan is bounded to the level of decomposition explained bellow.

Sufficiently decomposed A d^c in HTLN must specify whether it has to be or not fully decomposed. Together with $isPrimitive$ (section 3), two more flags are required to handle compound decisions.

$mustDecompose(d)$ indicates whether a d^c must be decomposed or not (it is obviously ignored for d^p's). This concept is very helpful in situations in which the user does not have all the information required to make a decision. For example, assume the rover has to perform a drive to a point using the operators showed above. The task $drive$ can be represented in the planning problem as a goal with the flag $mustDecompose = false$ in case the rover is driving beyond the field of view, because engineers cannot predefine a trajectory or make any assessments about the properties of the terrain in this situation. In consequence, *drive* is given a time range $[lb, ub]$ with an estimated lower and upper bound for its execution. Once the rover is due to perform the traverse, it starts to decompose the goal into sub-goals taking advantage of the new information gathered until either it approaches the target as specified or an error arise. In this way, the engineers and planner have a tool to define problems and generate valid plans under uncertain conditions and at the same time, cleaner and easier to understand.

$decLevel(d)$ describes the level of decomposition of the decision d. It can be in one of three different levels:

- Not sufficiently decomposed (nsd): The decision requires further decompositions in order for the planner to generate a valid plan. By recursion, a compound decision d^c for which $mustDecompose(d^c) = true$ is not sufficiently decomposed if it has not yet been decomposed or if any of its sub-tasks is not sufficiently decomposed.
- Sufficiently decomposed (sd): Indicates that the decision has been partially decomposed. By recursion, a compound decision d^c is sufficiently decomposed if it has not been decomposed and $mustDecompose(d^c) = false$ or in case $mustDecompose(d^c) = true$, d^c has been decomposed and non of its sub-tasks is nsd but at least one is sufficiently decomposed. All the activities in a problem should be at least sd to define a valid plan.
- Fully decomposed (fd): A d^p is always fully decomposed. A decision d^c is fully decomposed if, regardless of the value $mustDecompose(d^c)$, it has been decomposed and all its sub-tasks are as well fd.

Formally:

$$decLevel(d^c) = nsd, d^c \in CD \& mustDecompose(d^c) = true \\ ||d^c \in DN \& mustDecompose(d^c) = true \\ \& \exists cd_j \in d^c | decLevel(cd_j) = nsd \quad (13)$$

$$decLevel(d^c) = sd, d^c \in CD \& mustDecompose(d^c) = false \\ ||d^c \in DN \& mustDecompose(d^c) = true \\ \& \nexists cd_j \in d^c | decLevel(cd_j) = nsd \\ \& \exists cd_j \in d^c | decLevel(cd_j) = sd \quad (14)$$

$$decLevel(d) = fd, d \in D^p || d \in D^c \& d \in DN \& \forall c_i \in d, \\ decLevel(cd_j) = fd \quad (15)$$

5. THE PLANNER QUIJOTEEXPRESS

QuijoteExpress (QE) extends the AP^2 planner developed in the frame of the ESA Goal Oriented Autonomous Controller Study (GOAC) and is backwards compatible, thus allowing the integration of QE with previous GOAC algorithms.

The search space is organised as a graph in which each node contains a dn that is either partially supported or a solution, being dn_0 the initial problem. The edges represent binary relations between a node dn_i and its evolution dn_{i+1} and contain the list of changes done in dn_i to achieve dn_{i+1}. A given dn_i represents either a partially supported dn or a solution.

QE is divided in a strategic planner that leads the search and four tactical planners used to fix the plan (see figure 2). Given a search space with a nonempty set of partially supported dn's, each representing a potential solution for the initial problem dn_0, the planner iteratively proceeds to find a solution as shown in 1 until it reaches an exit condition. Likewise AP^2, QE can be directed to search the whole search space and retrieve the best solution or to retrieve the first solution found. In both cases, the planner stops in case no solution is found.

Algorithm 1: strategicPlanner(problem, domain, exit_cond)

begin
- **while** $(\neg exit\_cond)$ **do**
  - $dn_i \leftarrow select\_node(search\_space, domain)$
  - **if** $(\neg solution(dn_i))$ **then**
    - $dn' \leftarrow fixFlaws(dn_i, domain)$
    - **if** $(\neg valid(dn'))$ **then**
      - $search_space \leftarrow dn'$
    - **else**
      - $dn_{i+1} \leftarrow \gamma(dn', domain, \{\rho_\delta\})$
      - $search\_space \leftarrow dn_{i+1}$

First, QE non-deterministically selects a dn_i from the search space. Then it iteratively repairs the present problem by adding/deleting/moving decisions or adding/deleting/modifying relations of the plan using AP^2. Once the unfolder/scheduler have finished, the compound goals of the problem are further decomposed, those generating a more evolved version of the problem, repeating the process to fix the possible flaws added with the new decisions and relations. Finally, once the problem is sufficiently decomposed, the timelines are completed.

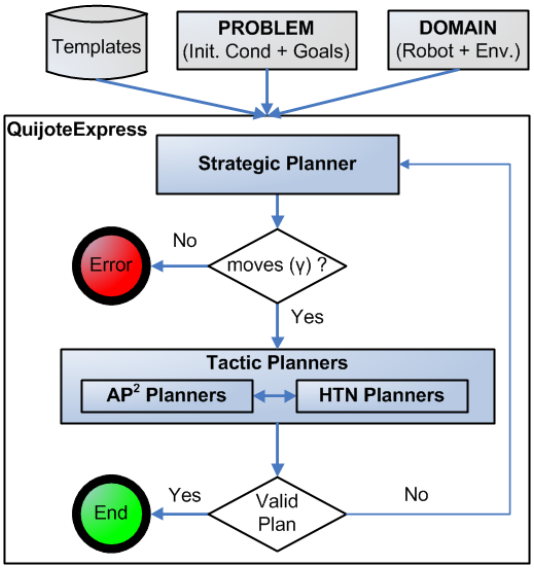

Figure 2. QuijoteExpress structure

To avoid negative consequences in the planner performance while expanding/repairing the plan, each dn contains a reference to all the cd's and rlt's of its dn^{sub}'s. Besides, a dn keeps record of all its dn^{sub}'s to improve the performance during backtracking/re-planning. For example, in case the planner finds no solution for drive an needs to use a different method, it just needs to replace the old one, but the constraints affecting drive as a whole do not need to be changed or recalculated.

Regarding the decomposer, given a $d^c \in dn_i$, the planner heuristically selects a method m to decompose d^c and applies it to dn_i as explained in section 3 in order to generate dn_{i+1}. So far, we have developed only trivial heuristics to demonstrate how QE works. However, the selection of methods play a relevant role in the performance of the planner and will be a topic for future work.

6. INITIAL RESULTS

We have implemented the rover domain described in the paper and run several tests. Considering the fact that we have implemented so far only blind heuristics, the performance of QuijoteExpress has not decreased in comparison with AP^2, being better in two situations. First, in case there are few different methods to decompose the

compound tasks of the domain, because the branching is limited and so is the effect of a blind heuristic. Second, in domains in which some compound tasks are decomposed in several sub-tasks. In this case, the decomposition resolver saves QE of several operations with respect to AP^2. However, we consider that there is a wide margin of improvement once the heuristics are implemented.

7. CONCLUSIONS AND FUTURE WORK

We have presented an evolution of APSI framework called APSI* together with a planner, QuijoteExpress able to exploit the new capabilities. The expressiveness of the new framework is now in the level of other very advanced systems such as ASPEN. Besides, the benefits in terms or performance for the planner are clear.

Our intention for future work is to focus on the development of heuristics and parallel planning to increase the planner performance and eventually generate an anytime algorithm with re-planning capabilities which might extend the usability of QuijoteExpress to broader scenarios. It is also our goal to increase plan quality and change the paradigm of collaboration planner/human expert towards a mixed-initiative approximation. So far we have run simulations with models, but it is our intention to start soon doing tests with real robots which should be of great relevance to understand which are the weak points and the way to go.

ACKNOWLEDGMENTS

This research has been co-funded by the Networking/Partnering Initiative (NPI) between ESA-ESOC, TU Darmstadt and University of Surrey. It also receives support from the German Research Foundation (DFG) within the Research Training Group 1362 “Cooperative, adaptive and responsive monitoring in mixed mode environments”.

REFERENCES

[1] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] Max Bajracharya, Mark W Maimone, and Daniel Helmick. Autonomy for mars rovers: Past, present, and future. *IEEE Computer*, 41:44–50, 2008.

[3] Jeffrey J. Biesiadecki and Mark W Maimone. The mars exploration rover surface mobility flight software driving ambition. In *IEEE Aerospace Conference (IAC)*, number March, 2006.

[4] John L Bresina, Ari K Jonsson, Paul H Morris, and Kanna Rajan. Mixed-Initiative Planning in MAPGEN: Capabilities and Shortcomings. In *In Proceedings of the ICAPS-05 Workshop on Mixed-initiative Planning and Scheduling*, page 8, 2005.

[5] Antonio Ceballos, Saddek Bensalem, Amedeo Cesta, L. S Silva, Simone Fratini, Félix Ingrand, J. Ocon, Andrea Orlandini, Frederic Py, Kanna Rajan, Riccardo Rasconi, and Michel Van Winnendael. A goal-oriented autonomous controller for space exploration. *ASTRA*, 2011.

[6] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, and Angelo Oddi. Mexar2: AI solves mission planner problems. *IEEE Intelligent Systems*, 22(4):12–19, 2007.

[7] Amedeo Cesta and Simone Fratini. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, 2008.

[8] Amedeo Cesta, Simone Fratini, Andrea Orlandini, and Riccardo Rasconi. Continuous Planning and Execution with Timelines. In *International Symposium on Artificial Intelligence (i-SAIRAS)*, number 1, 2012.

[9] Steve Chien, Gregg R Rabideau, Russell L Knight, Robert Sherwood, Barbara Engelhardt, Darren Mutz, Tara A Estlin, Benjamin Smith, Forest W Fisher, Anthony C Barrett, G Stebbins, and Daniel Tran. ASPEN - Automated Planning and Scheduling for Space Mission Operations. In *International Conference on Space Operations (SpaceOps)*, pages 1–10, 2000.

[10] Steve Chien, Robert Sherwood, Daniel Tran, and Benjamin Cichy. The EO-1 autonomous science agent. In *Conference on Autonomous Agents andmMulti-Agent Systems (AAMAS)*, pages 420–427, 2004.

[11] Juan M. Delfa Victoria, Nicola Policella, Gao Yang, and Oskar Von Stryk. Design Concepts for a new Temporal Planning Paradigm. In *ICAPS Planning & Scheduling for Timelines (PSTL), 2012*, 2012.

[12] Daniel L DL Dvorak. NASA study on flight software complexity. Technical report, 2009.

[13] Kutluhan Erol, James Hendler, and Dana S Nau. Semantics for Hierarchical Task-Network Planning. Technical report, 1994.

[14] Tara A Estlin, Benjamin J Bornstein, Daniel Gaines, Robert C Anderson, David R Thompson, Michael Burl, Rebecca Castano, and Michele Judd. AEGIS Automated Science Targeting for the MER Opportunity Rover. *International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS)*, pages 1–25, 2010.

[15] Jeremy D Frank and Ari K Jonsson. Constraint-based Attribute and Interval Planning. *Journal of Constraints Special Issue on Constraints and Planning*, 8(4):339—-364, 2003.

[16] Simone Fratini and Amedeo Cesta. The APSI Framework: A Platform for Timeline Synthesis. *Proceedings of the 1st Workshops on Planning and Scheduling with Timelines PSTL-12*, 2012.

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

[18] Ari K Jonsson, Paul H Morris, Nicola Muscettola, and Kanna Rajan. Planning in interplanetary space: Theory and practice. *Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 2000.

[19] Nicola Muscettola. HSTS: Integrating planning and scheduling. In M Zweben and Mark S Fox, editors, *Intelligent Scheduling*, number CMU-RI-TR-93-05. Morgan Kaufmann, Pittsburgh, PA, March 1994.

[20] Robert Sherwood, Anita Govindjee, David Yan, Gregg R Rabideau, Steve Chien, and Alex Fukunaga. Using ASPEN to Automate EO-1 Activity Planning. In *IEEE Aerospace Conference*, pages 145–152, 1998.

[21] Austin Tate, Brian Drabble, and Richard Kirby. O-Plan2: an open architecture for command, planning and control. In *Intelligent Scheduling*, volume 1, pages 213—-239. 1994.

[22] David E Wilkins, Karen L Myers, John D Lowrance, and Leonard P Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental Theoretical Artificial Intelligence*, 7(1):121–152, 1995.