RESOURCE DRIVEN PLANNING WITH “PLASMA”: THE PLAN SPACE MULTI-SOLVER APPLICATION

Andrea De Maio[1], **Simone Fratini**[2], **Nicola Policella**[2], **and Alessandro Donati**[2]

[1]*Sapienza University of Rome, Italy, and.demaio@gmail.com*
[2]*European Space Agency, ESA-ESOC, Darmstadt, Germany, name.lastname@esa.int*

ABSTRACT

Increasing complex missions that require capabilities of autonomous reasoning and resource management are being designed, involving robotic activities and synchronization of on-ground and on-board planning and scheduling. Such a scenario is pushing the need for technologies more flexible and sophisticated to support the integration of reasoning on actions to be performed and resources required to perform them. So far most of the efforts in research on planning and scheduling integration have been devoted to “planning with resources”, planning domains where various tasks require resources to be achieved. In real domains instead, very often the resource availability is actually the driving factor of the planning process and the capability of planning to guarantee appropriate resource levels becomes more important.

In this paper we present an extension of the modeling and problem solving capabilities of the APSI platform to entail the representation and reasoning with resource models which are more flexible and sophisticated with respect to those previously available, and an integrated planner and scheduler recently deployed, PLASMA (PLAn Space Multi-solver Application), able to achieve goals on resource requirements. PLASMA increases the integration of symbolic planning with numerical reasoning with resources, allowing the definition of planning problems formulated directly on resource requirement desiderata.

Key words: Planning, Scheduling, Autonomy, AI.

1. INTRODUCTION

Autonomous operations induce a new distribution of processes and functions between ground and space: on-ground routine operations workload tends to be reduced and refocused on supervisory roles, while operator intervention is narrowed to the management of non-nominal situations. Increasing complex missions are being designed, involving robotic activities and synchronization of on-ground and on-board planning and scheduling. As a consequence, the scenario is changing from a context where planning and scheduling are considered offline activities carried out to provide input for the flight control team into a situation where integrated planning and execution architectures are being designed. Such a scenario is pushing the need for technologies more flexible and sophisticated to support the integration of reasoning on actions to be performed and resources required to perform them, as well as the need for generating flexible plans to protect against uncertainty in execution time and telemetry values [18].

Artificial Intelligence techniques for Planning and Scheduling have been quite successful to entail advanced solving features and autonomy in a number of space applications. A lot of effort has been dedicated to build software development environments for rapid prototyping, test and synthesis of new planning and scheduling applications, both at NASA-JPL (EUROPA[10], ASPEN[8]) and at ESA, for the on-ground segments and for on-board autonomy. ESA concurs in this area of advanced research by promoting the development of APSI[13] (Advanced Planning and Scheduling Initiative) and APSI-related activities[6], its use for on-board autonomy with GOAC[3] (Goal Oriented Autonomous Controller) and its application on teleoperations[14].

So far most of the efforts have been devoted to “planning with resources”, planning domains where various symbolic goals require resources to be achieved. In real domains instead, very often the resource availability is actually the driving factor of the planning process. In other words, the problem is not that much “what I have to do to achieve these goals considering the resource availability”, but mostly is “how can I achieve the resource availability required to do this and that” or “what can I actually do considering this desiderata on resource availability” (supposing over booked problems or considering that I can’t do everything because of the resource restrictions).

In this paper we present: (1) an extension of the modeling and problem solving capabilities of the APSI platform to entail the representation and reasoning with resource models more flexible and sophisticated with respect to those previously available, and (2) an integrated planner and scheduler recently deployed, PLASMA (PLAn Space Multi-solver Application), able to generate flexible plans combining principles from Partial Order and Least Commitment Planning [20] and Partial Order Schedule (POS). Besides that, PLASMA increases the integration of symbolic planning with numerical reasoning with resources, entailing planning on problems formulated directly on resource requirement desiderata.

PLASMA solves problems specified in APSI’s domain/problem definition language. This entails the reuse of this planner with already existing APSI and GOAC based applications. The next Section summarizes the basic modeling primitives currently available with the APSI framework, using as a running example a simple robotic domain. This example, although simplified on purpose for this paper, is conceived for pointing out the new modeling features recently introduced.

2. MODELING PROBLEMS WITH TIMELINES

State-of-the-art planning and scheduling architectures like the APSI framework, defined as "timeline-based" architectures, are grounded on a few modeling primitives: timelines, state variables, numerical resources, temporal/-value synchronizations among different timelines, problems and solutions represented as constraint networks on events occurring in time instants or over time intervals.

Timeline-based planning is grounded on the underlying assumption that the world is modeled as a set of entities whose properties can vary in time (such as one or more physical subsystems) according with some internal logic or as a consequence of external inputs. The intrinsic property of these entities, represented by means of *timelines*, is that they evolve over time concurrently, and that their behaviors can be affected by external inputs. In this context, the problem solving with timelines is conceived as a problem of controlling components with external inputs in order to achieve a desired behavior. Hence different types of problem (e.g., planning, scheduling, execution or more specific tasks) can be modeled by identifying a set of inputs and relations among them that, once applied to the components modeling a domain with a given initial set of possible temporal evolutions, will lead to a set of final behaviors which satisfy the requested properties (for instance feasible sequences of states, or feasible resource consumption, as well as more complex properties)[1].

More in detail, a timeline is a finite set of ordered, flexible transition points with associated values. The values associated to a transition of the timeline allows the computation of the actual value of the timeline between two consecutive transition points. For each transition is specified a lower and upper bound of occurrence and a minimal and maximal duration for the transition.

State variables represent components that can take sequences of symbolic states subject to various (possibly temporal) transition constraints. This primitive permits the definition of *timed automata* as the one represented in Figure 1; here the automaton represents the constraints that specify the logical and temporal allowed transitions of a timeline. A timeline for a state variable is valid if it represents a *timed word* accepted by the automaton (at the bottom of the figure).

The automaton models: (1) the values that the timeline can take, possibly as a function of numeric or enumerated parameters; (2) transition constraints on these values, possibly with additional constraints that restrict the transition to a subset of the possible values that the parameters can take (in the example in Figure 1 the transition from `P(?x)` to `R(?z)` imposes that `?x>?z`); (3) temporal constraints that state the minimal and maximal temporal duration of a value and (4) guards that restrict the applicability of a transition, either based on the value of a parameter (in the example the transition from `P(?x)` to `Q(?y)` is allowed only if `?x>0`) or on the relative timing of the transition (in the example the transition from `R(?z)` to `P(?x)` is allowed only if `R(?z)` has been maintained for less than 2 time units). The timed automaton (i.e. state variable) is a very powerful modeling primitive, widely studied both at theoretical level (see [2] for instance) and for which exist implemented algorithms to find valid timelines.

[1]A detailed description of the timeline based approach, state of the art of the technologies in use and basic concepts is out of the scope of this paper. More information can be found for example in [17, 11, 15] among the others.

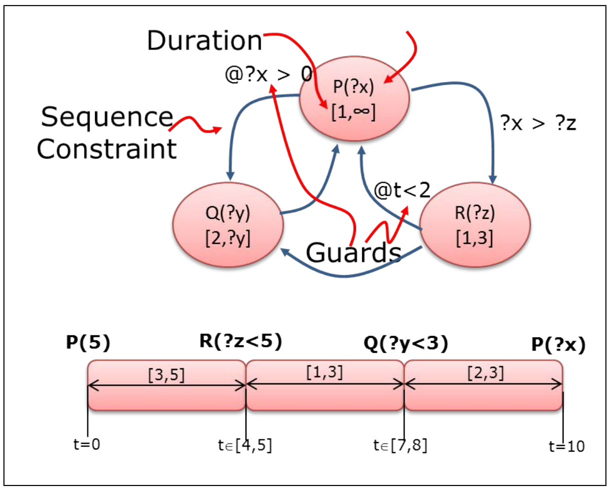
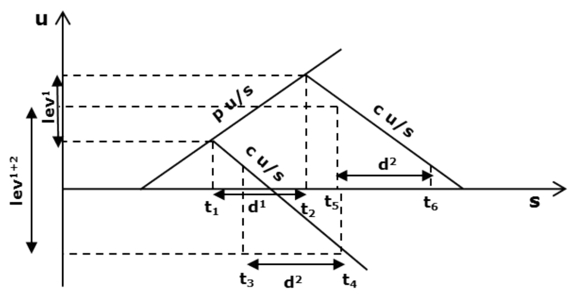


Figure 1. State Variable

A resource is any physical or virtual entity of limited availability, such that its timeline (or profile) represents its availability over time, a decision represents a quantitative use/production of the resource over a time interval. A *reusable* resource (as in [7]) abstracts any real subsystem with a limited capacity c_{max}, an *activity* uses a quantity of resource during the limited interval. For instance, an electric generator has a maximal available power P_{max} (its capacity). An activity uses power during an interval of time and as soon as the activity ends, the amount of resource can be reused by other activities. A set of activities is feasible when for each time t the aggregate demand $p(t)$ (or profile) is below or equal to the resource capacity c_{max}. A *consumable* resource (as in [16]) abstracts any subsystem with a minimum capacity c_{min} and a maximum capacity c_{max}, *consumptions* and *productions* consume and restore a quantity of the resource in specific time instants. A set of productions and consumptions is feasible when for each time t the aggregate use $u(t)$ (or profile) is in between the resource minimum and maximum capacity (i.e., $c_{min} \leq u(t) \leq c_{max}$).

The APSI architecture had the the capability of representing and solving scheduling problem with these resource models, but considering only stepwise resource profiles. We have extended the modeling power of the architecture by adding *linear reservoir* resources. A linear reservoir resource does not have a stepwise constant profile of consumption like reusable and consumable ones, but the activities specify the amount of production/consumption per time, namely `slope`, resulting in a profile of resource allocation that is a linear temporal function. The consequence of this fact is that the amount of resource available at each transition of the timeline depends on the duration of the time intervals over which this production or consumption has been performed (in contrast with previously existing resource models where the profile of the resource availability at each transition depends only on when and how much is produced/consumed and not on the duration of the production/consumption). And this in turn induces the need for integrated reasoning on time and data to guarantee the flexibility of the plan/schedule generated.



P Time [lb,ub]	Duration [lb,ub]	Value
0 [0,0]	[0,30]	[0.0,0.0,0.0]
1 [0,30]	[10,15]	[0.0,0.0,5.0]
2 [10,40]	[10,15]	[50.0,75.0,-4.0]
3 [20,50]	[0,30]	[-10.0,35.0,0.0]
4 [50,50]	n/a	[-10.0,35.0,0.0]

Figure 2. Reservoir Linear Resource

In Figure 2 there is an example. Let's suppose two consecutive activities, a production with slope p units per second in a time interval with a duration $d_1 = [min_d^1, max_d^1]$ and a consumption with slope c units per second in a time interval with a duration $d_2 = [min_d^2, max_d^2]$. Concerning only the first activity, the production, the resource profile at the end of the activity is bounded between the value taken at $t = t_1$ and the value taken at $t = t_2$ (interval lev_1 in Figure). If from t_1 we start the consumption, the resulting resource profile is bounded between the values taken at t_3 and t_4, while if we start the consumption at t_2 the final profile will be bounded between the value taken at t_5 and the value taken at t_6. The lower final value for the resource level will be the one taken at t_4, when the production is the shortest possible and the consumption is the longest possible, while the maximum possible value will be the one taken at t_5, when the production is the longest possible and the consumption is the shortest possible (interval lev_2 in Figure).

If we consider for instance two activities with durations bounded in $[10, 15]$ in an temporal horizon of 50 time units (e.g. seconds), the production of $+5$ u/s and the consumption of -4 u/s, in the bottom of Figure 2 we have the timeline[2] representing the temporal and value bounds for the resource profiles. At the first flexible transition point, the value of the profile is 0.0 and the slope is 5.0 (time t_0 in the diagram). At the end of the production, on the second timeline transition (between time t_1 and t_2 in the diagram), we have a value of the resource profile in $[50, 75]$, because the duration d_1 of the activity is in $[10, 15]$, then the resource profile is in $[5 * 10, 5 * 15]$. At the end of the second activity, on the third timeline transition, we have a value of the resource profile in $[-10, 35]$. -10 is obtained as $50 - 4 * 15$, while 35 is $75 - 4 * 10$.

It is worth to point out that this is just a simple example to illustrate the concept. What follows is based only on the principle of bounding the profile of resource allocation, i.e., what matters is the computation of the maximum and minimum possible value for the profile because the algorithms described in Section 4.2 work shrinking these bounds into acceptable values. In case of more than 2, possibly overlapping production and consumptions, given all the temporal constraints stated on the activities, it is not temporally feasible in general to set minimal and maximal durations for the interval as done in this example to get the upper and lower bound of resource consumption, but the principle of the bounding remains valid. On the other hand, it is possible to implement more sophisticated algorithms to bound the resource profile at the transition points, like in [12], reducing the search space of the algorithms that fix these bounds.

In timeline-based modeling the physical and technical constraints that influence the interaction of the subsystems (modeled either as state variables or resources) are represented by means of temporal and logical synchronizations among the values taken by the automata and/or resource allocations on the timelines. Languages for timeline-based planning have constructs (as the synchronization in DDL.3 or the compatibility in NDDL, see [17, 15] for formal definitions) to represent the interaction among the different timelines that model the domain. Conceptually these constructs define valid schema of values allowed on timelines and link the values of the timelines with resource allocations. Despite the syntactic differences, they allow the definition of Allen's[1] like quantitative temporal relations among time points and time intervals as well as constraints on the parameters of the related values. As an example to clarify the concept, let us present a simple robotic domain.

Let us suppose a rover that has to perform experiments in a given environment. For this running example we suppose the rover equipped with a battery that is uncharged when the rover is moving or performing experiments and that is charged via solar panels when the rover is standing in some given location. We suppose a map of the charging areas provided as an external timeline that specifies, in some time intervals, the production that is possible to obtain if the rover is standing there. The rover is able to autonomously navigate the environment, to perform experiments and to monitor its battery in order to be sure there is enough battery to perform its activities. In order to model the rover domain, the following subsystems are considered: a mobility system MS, a battery

[2]The timeline represents for each transition T, tuples $\langle$ ID, [lower and upper bound of occurrence], [minimal and maximal duration], Value $\rangle$. The value in turn is a 3-ple $\langle$ minimal value, maximal value, slope $\rangle$.

BAT and one (or more) payloads PLD[3]. In the model, we also assume the rover able to move between two points in space. Hence the mobility system can be modeled as a state variable taking the following values: AT($?x, ?y$) when the rover is standing in $\langle x, y\rangle$ and GOTO($?x, ?y$) when the rover is moving toward $\langle x, y\rangle$. A transition GOTO($?x, ?y$) $\rightarrow$ AT($?x, ?y$) denotes a successful move to $\langle x, y\rangle$ and a transition AT($?x, ?y$) $\rightarrow$ GOTO($?x', ?y'$) denotes the rover starting to move from a point $\langle x, y\rangle$ to a point $\langle x', y'\rangle$.

The rover payloads can perform experiments (in the current position of the rover) and can be on idle. Hence a payload can be modeled with a state variable taking the following values: IDLE(), when the payload is not performing experiments and RUNNING($?exp$) when the unit is performing an experiment exp.

The battery is modeled as a reservoir resource with a minimum charging value (supposed in this model 0 but whatever value can be considered) and a maximum charging capability max. When a rover is performing an experiment, it is supposed to consume the battery depending on the type of experiment being performed, hence we suppose an external function $f_{consumpion}(?exp)$ that provides the actual consumption of the experiment. Besides that, we suppose that the rover consumes a fix amount of battery per time unit when moving. Hence the consumption depends only on the time the rover takes to move from one point to the other.

A goal RUNEXPERIMENT($?x, ?y, ?exp$) can be achieved by the payload of the rover taking a status PLD.RUNNING($?exp$), with the mobility system in a status MS.AT($?x, ?y$). The battery consumption is modeled by synchronizing the status PLD.RUNNING($?exp$) of the payload with an activity BAT.ACTIVITY($f_{consumpion}(?exp)$).

This mix of causal and temporal relationships among the operations can be stated with the following synchronizations in DDL.3:

```
SYNCHRONIZE MISSIONTIMELINE {
VALUE RUNEXPERIMENT(?x, ?y, ?exp) {
OP1 PLD.RUNNING(?exp);
OP2 MS.AT(?x, ?y);
REF CONTAINS OP1;
OP1 DURING OP3; }}
```

```
SYNCHRONIZE PLD {
VALUE RUNNING(?exp ≠ “visit_cave”) {
OP1 BAT.ACTIVITY(f_consumpion(?exp));
REF EQUALS OP1; }}
```

```
SYNCHRONIZE MS {
VALUE GOTO(?x, ?y) {
OP1 BAT.ACTIVITY(-1.0);
REF EQUALS OP1; }}
```

[3]This is just an excerpt of a realistic domain simplified for this paper. Since the focus here is not on synchronization among different sub-systems of the rover, a planning capability already demonstrated by various planners for the APSI framework, see [3, 4, 9] for example, we don't consider cameras, pan-tilt units, drills, memories and other types of payloads. These sub-systems and related synchronizations can be added to the domain without harming what is being discussed here.

Besides that, we need to synchronize the activities that refill the battery with the zones and the time intervals where the rover can get enough power for its solar panels. We suppose this input provided as a timeline where we specify the coordinates of the zones and the available solar flux in various time intervals. Such a timeline[4] will have intervals specified with the value CHARGE($?x, ?y, ?solar\_flux$) when the flux $?solar\_flux$ is available in $\langle x, y\rangle$. Given that, we need to synchronize the charging activities with the rover standing in one of these zones. The actual charging capability generated by a given solar flux is modeled with an external function $f_{production}(?solar\_flux)$. Hence we have the following synchronization in DDL.3:

```
SYNCHRONIZE BAT {
VALUE ACTIVITY(?prod) {
[?prod > 0.0];
OP1 MS.AT(?x, ?y);
OP2 ZONE.CHARGE(?x, ?y, ?solar_flux);
REF DURING OP1;
OP1 DURING OP2;
?prod := f_production(?solar_flux); }}
```

3. STATING GOALS ON RESOURCES

The traditional modeling primitives shown so far, state variables and resource production/consumptions, allow the representation of traditional integrated planning and scheduling problems: there are some things that have to be done, modeled as state variable values, and some resource requirements, modeled as synchronized statements on corresponding resources. Whit such a model it is possible to state goals on state variables and look for plans that guarantee a correct resource allocation. But there is no way to state goals directly on the resource profiles.

For that reason we have introduced a new modeling feature: the *resource level request*. This feature allows the modeler to specify some desiderata directly on the resource level, as minimum and maximum value required in a given instant. It is substantially different from all the other features used so far: in fact it has no direct effect on the resource profile, but forces the activation of specific synchronizations when the condition is not fulfilled (or when is fulfilled, it depends on the semantic given to the statement).

Let us consider a special experiment for which we want to be sure that some properties about the level of charge of the batteries are met before starting it, like visiting a cave for instance. We don't know in advance what is going to happen while the rover will be inside, and usual on-board re-planning strategies need enough battery to bring the rover outside. Besides that, we don't want to rely only on the supposed consumption for this experiment, because this plan would take into account only the planned boundaries for this activities, guaranteeing a correct resource allocation only in between these boundaries. But, since this is too risky for us, we want to be sure that also *out of the planned boundaries*, there will be a margin for getting out from the cave. It is worth to point out

[4]We suppose here only one timeline, but if multiple zones can provide solar flux in overlapping time intervals more timelines can be used.

that this is different than specifying an higher resource consumption to guarantee the robustness of the plan because in this case the planner would have to plan considering this consumption, influencing negatively the quality of the whole plan/schedule.

In this case, we synchronize such an experiment with a resource level request statement at its starting point:

SYNCHRONIZE PLD {
VALUE RUNNING($?exp =$ "$visit\_cave$") {
OP1 BAT.ACTIVITY($f_{consumpion}(?exp)$);
REQ1 BAT.LEVEL_REQUEST($?min\_acceptable\_value$);
REF EQUALS OP1;
REQ1 AT-START OP1; }}

When such a synchronization is fired by the planner, a transition point is introduced into the timeline. This transition point does not change the slope of the reservoir resource, i.e. the profile keeps increasing or decreasing as it was doing before the transition, but it is taken into account by the flaw collector procedure of the APSI framework (see the Section on problem solving below), inducing a resolution process if it is not satisfied.

It is also worth to point out that a resource level request, as any other statement on timelines, can be used as goal for the planner. In this rover example has been introduced as a consequence of the need for a specific experiment, but it can be used to drive the planning process on purpose to obtain specific boundaries for the resource profiles. This feature is necessary for instance when the goal is to generate an envelope of resource availability when the stage of the planning process does not allow to specify in detail the allocation to be performed (a very usual scenario when the planning is carried out on different planning horizons and at a different hierarchical levels, like long, medium, short and on-board planning approach).

4. PLASMA

In the APSI framework, an application, being a generic planner and/or scheduler or a domain specific deployed application is designed as a collection of *solvers*. The search space of an APSI solver is made of planning and scheduling statements on timelines and temporal and data relations among them, in a constraint-based approach. Examples of solvers are binary and multi-capacity schedulers, domain theory application solvers, solvers to complete timelines and any other possible type of reasoning process aimed at modifying the current plan in such a way that a given property is achieved or a given parameter is optimized.

An application solves a conceptually higher problem, like a planning problem for instance, or a global resource optimization problem. The main difference between a solver and an application stems in the fact that a solver searches in the space of decisions and constraints, an application searches in the space of solvers solutions. The PLASMA planner is an example of APSI application. It is made of a collection of solvers, activated on a *flaw detection base*: when a flaw is detected on a timeline, the planner activates the corresponding solver to fix the problem.

PLASMA consolidates all the features and modeling principles described so far in a single application. It incorporates many of the principles of Partial Order Planning (POP) and Partial Order Scheduling (POS) like a plan-space search space, the least commitment approach and a flaw based resolution process. Starting from an initial partial plan only made of partially specified timelines and goals to be achieved, the planner iteratively refines it into a final plan that is compliant with all the requirements expressed by the goals. A goal can be a value for a state variable, activities to be allocated on resources and resource level requests (plus temporal/data relation on them).

The PLASMA planner is grounded on various APSI solvers. Some of them have already been used in previous planning systems, some new solvers have been recently implemented. All the functionality to collect synchronizations, to apply the domain theory, to schedule binary and multi-capacity resources have been taken from the AP^2 planner[3]. New solving capabilities have been added to (1) derive flexible schedules from the fix-start time ones generated by the previous existing schedules by means of the application of POS *partial order schedule* concepts; (2) schedule productions and consumptions on reservoir resources to meet resource capacity constraints and to fulfill resource level requests.

4.1. Partial Order Schedules

PLASMA exploits the properties of Partial Order Schedules [19] to support the planning process and to guarantee temporal flexibility to the plan and schedule (necessary for execution). Originally the POS concept was introduced to cope with uncertainty and provide robust solutions. A similar need there is when considering the integration of planning and scheduling where iteratively planning ans scheduling modules are called to modify a shared problem/solution representation. In our case, the schedule produced should be able to accommodate possible changes introduced by the planner without loss of consistency (or, in the worst case, by minimizing the impact).

The Partial Order Schedule Concept is based on the idea of retaining temporal flexibility. We expect a flexible schedule to be easy to change, and the intuition is that the degree of flexibility in this schedule is indicative of its capability of supporting the planner choices. The POS concept is based on the activity-on-the-node representation: given a scheduling problem P this can be represented by a graph $G_P(V_P, E_P)$, where the set of nodes $V_P = V \cup \{a_0, a_{n+1}\}$ consists of the set of activities specified in P plus two dummy activities representing the origin (a_0) and the horizon (a_{n+1}) of the schedule, and the set of edges E_P contains P's temporal constraints between pairs of activities. A solution of the scheduling problem can then be represented as an extension of G_P, where a set E_R of simple precedence constraints, $a_i \prec a_j$, is added to remove all the possible resource conflicts.

As a time feasible schedule is a schedule that satisfies all the time constraints, and a feasible schedule is a schedule that is both time and resource feasible, a *Partial Order Schedule* is a set of solutions (schedules) that can be represented by a graph $G_{POS}(V_P, E_P \cup E_R)$ such that any *time feasible* schedule defined by the graph is also a *feasible* schedule.

A POS is then a set of activities partially ordered such that any possible complete order that is consistent with the initial partial order, is a resource and time feasible schedule.

4.2. Managing Linear Reservoir via MaxFlow

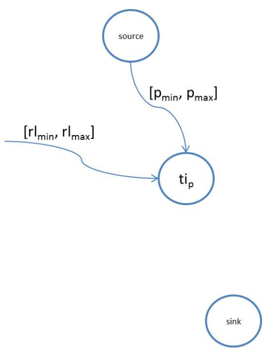
As mentioned above, in this domain, one of the main driving feature is the control of the resource level. In fact in this situation we have not only capacity constraints to be held but also specific resource levels can be requested in order to allocate some activities.

To evaluate the resource level a timeline is taken into account. The latter represents an ordered sequence of not-overlapping intervals which have:

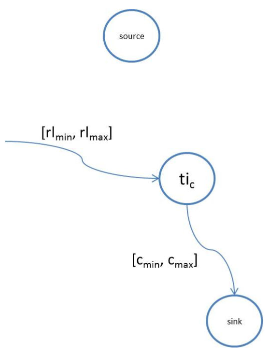
- flexible start and end time
- a fixed data slope (positive in case of data generation, negative in case of consumption)
- resource preconditions which establish the minimum and maximum feasible resource level - these consist in a specific resource level request. In case there is no specific request the two levels are equivalent to the resource capacity constraints.

Given this information, the goals are (1) to detect possible inconsistencies and, when consistent, (2) to find a possible solution which consists in deciding a consistent duration for all the time intervals in the timeline. In order to satisfy these goals, the problem has been modeled as a particular flow network, a Maximum Flow with Edge Demands Problem – Fig. 3. Each time-interval is represented as node while two dummy nodes (the source and the sink) are used to distinguish between production (data flow in) and consumption (data flow out). In particular a time interval which has a positive slope is represented by a node and two edges (Fig. 3(a)): one from the previous time-interval/node labeled with the minimum and maximum expected resource level; one from the source which is labeled with the minimum and maximum amount of data produced during the time interval (values which are easily obtained by the slope value and the interval duration). In a similar way is modeled a time interval with negative slope (Fig. 3(b)). In this case there is an edge from the node to the sink labeled with the minimum and maximum consumption. Fig. 3(c) gives an example with two production and two consumption time intervals.

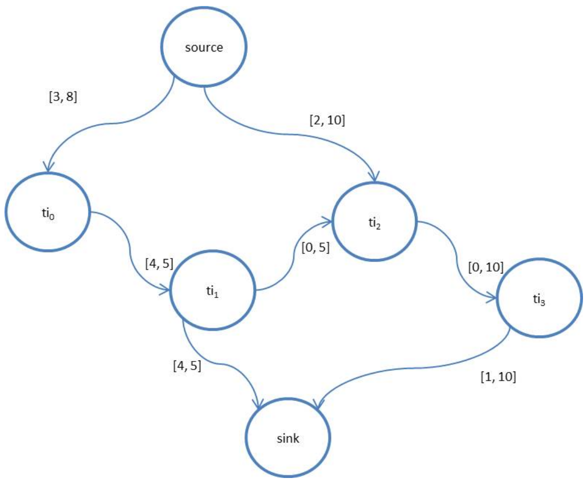
Given then Maximum Flow with Edge Demands Problem representation, if this does not admit a solution than the initial timeline is not consistent. On the other side, given a max-flow solution of the problem, the duration of each time interval can be computed by considering the different flows through the network.



(a) Production model



(b) Consumption model



(c) Two productions two consumptions example

Figure 3. Maximum Flow with Edge Demands Problem model

Given the result of the computation of the max flow, the solver either generates a set of temporal constraints to properly shrink the durations of the activities or, if this is not possible, generates a new production and restart the algorithm.

4.3. Solving Process

PLASMA uses a various number of solvers, each one dedicated to the solution of a particular flaw type. A partial plan is characterized by a number of flaws which are sorted by priority. The resolution process picks up the flaw with the highest priority and generates a solver delegated to solve it. The solvers produce a partial plan from which the flaws will be re-collected. This is done iteratively until a legal plan is obtained as shown in Algorithm 1.

Partial plans containing unsolvable flaws are leaves in the search space which cannot be further expanded, thus dead-ends. All solvers can be configured to generate either all possible solutions of their assigned flaw or a subset of. These states are used to backtrack when in a successive step another flaw turns out to be unsolvable.

PLASMA currently handles the following list of flaws:

Algorithm 1: TL-PSP($\mathcal{P} = \langle \mathcal{DT}, \mathcal{P}roblem \rangle$)

```
$s_0 = Init(\mathcal{P}roblem)$;
$\mathcal{TL} \leftarrow \mathcal{E}xtractTimeline(s_0)$;
$\Phi = CollectFlaws(\mathcal{DT}, s_0, \mathcal{TL})$;
$\mathcal{S} \leftarrow \{s_0\}$;
$s \leftarrow s_0$;
while $\Phi \neq \emptyset$ do
    $\phi \leftarrow ChooseFlaw(\Phi)$;
    $\sigma \leftarrow ChooseSolver(\phi)$;
    $\mathcal{S}_\sigma \leftarrow Solve(\sigma, \phi)$;
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_\sigma \setminus \{s\}$;
    if $\mathcal{S} = \emptyset$ then
        return failure;
    end
    $s \leftarrow ChooseState(\mathcal{S})$;
    $\mathcal{TL} \leftarrow \mathcal{E}xtractTimeline(s)$;
    $\Phi = CollectFlaws(\mathcal{DT}, s, \mathcal{TL})$;
end
return $\mathcal{TL}$;
```

- SubGoal Timeline Flaw. This flaw is solved for state variable values either adding new decisions to the plan (proper sub-goaling) or attempting an unification (on time and value) with already posted decisions.
- Timeline Gap Flaw. Undecided time intervals over state variable timelines are removed accordingly with state machine transitions.
- Consumable and Reusable Resource Overflow and Underflow Flaw. When requests exceed the lower or upper bound of a given resource, activities are scheduled with binary or multi-capacity schedulers, using algorithms in [5],[16] and [19].
- Reservoir Resource Level Violation Flaw. Reservoir linear resources flaws are removed either adding temporal constraints to the plan or generating new sub goals to produce the necessary resource (as described above).

PLASMA can be configured associating resolvers to the various types of flaws. Flaws can be solved in different ways, each one representing a partial plan not containing the flaw anymore but different from the others generated by the same solvers. More in general, each solution is a child of the same node. The output is composed of set of flexible timelines fully specified on the temporal horizon. Legal plans must respect all the relations imposed in the problem along with the constraints specified in the domain theory.

4.4. An Example

As an example we consider a simple problem where the rover, starting from a position $\langle 0, 0 \rangle$ has to perform two experiments, a normal one and one that requires safety conditions on the battery to be performed (at least 24 units), then come back to the original position. The rover has 2 zones where he can charge its battery, on specified time intervals. The battery has acceptable levels of charge in $[0, 40]$, an experiment consumes 2 units of resource per minute, when the rover is moving it consumes 0.05 units. We suppose we start with the battery fully charged.

The problem specification in PDL, the APSI framework problem specification language, is reported in Listing 1. It contains the initial conditions, tagged as $\langle$FACT$\rangle$ and the goals, tagged as $\langle$GOAL$\rangle$. The visibility of the charging zones is specified as a fact as well.

In Listing 3, top, we show the flexible solution. The battery values show minimum, maximum and slope value of the battery in the interval. The plan starts with 40 units of battery and ends between 4 and 18 units still available. The resource level request occurs just before the second experiment starts (transition 14). The resource available there is between 25 and 38.05, that satisfies the resource level request of at least 24 units. Finally the charge activity planned occurs between transitions 10 and 11. Here the solar flux is minimum, but with 2 units per minute of charge slope it brings the charge up to at least 26, a level high enough for executing the rest of the plan.

In Listing 3, bottom, there is a fix solution extracted from the flexible envelope described above. In this specific solution, the final charge level for the battery is 6 and the level when the critical experiment starts is 26.05.

Listing 1. "Problem.pdl"

```
PROBLEM rover_problem (DOMAIN rover_domain) {

//Initial Conditions
START <fact> MS.tl.At(?x1 = 0, ?y1 = 0);
NOPROD1 <fact> STATIC CZN.tl.None() AT [0,10];
PROD1 <fact> STATIC CZN.tl.Charge(?x = 130,?y = 150,?flux = min) AT [10,30];
NOPROD2 <fact> STATIC CZN.tl.None() AT [30,50];
PROD2 <fact> STATIC CZN.tl.Charge(?x = 40,?y = 80,?flux = max) AT [50,100];
NOPROD3 <fact> STATIC CZN.tl.None() AT [100,150];

//Goals
EXP1 <goal> MTL.tl.RunExperiment(?x = 10,?y = 20, ?exp = exp_1);
EXP2 <goal> MTL.tl.RunExperiment(?x = 10,?y = 20, ?exp = go_into_the_cave);
BACK <goal> MS.tl.At(?x1 = 0, ?y1 = 0);

EXP1 DURATION [10,15];
EXP2 DURATION [10,15];

START BEFORE EXP1;
EXP1 BEFORE EXP2;
EXP2 BEFORE BACK;

}
```

5. CONCLUSIONS

In this paper we have presented an extension both of the modeling and of the problem solving capabilities of the APSI platform, a framework for rapid prototyping of planning and scheduling applications in use at ESA.

This work introduces in the platform linear resource models more flexible and sophisticated with respect to those previously available, as well as the possibility of expressing directly goals on resource profiles. In real domains in fact, very often the resource availability is the driving factor of the planning process and it is becoming more and more important for the capability of planning to guarantee appropriate resource levels.

At problem solving level the novelty introduced entails the capability of generating temporally flexible plans also when multi-capacity and linear resource are involved in

Listing 2. "Flexible Solution"

```
P           Time [lb,ub]              |           MTL          |     MS      |           PLD          |          CZN           |      BAT
---------------------------------------------------------------------------------------------------------------------------------------------
0 [00.000.00.00.00,00.000.00.00.00] |  Idle()                |  At(0,0)    |  Idle()                |  NoChargeZone()        |  [40.0,0.0]
1 [00.000.00.01.00,00.000.00.10.00] |                        |  GoTo(10,20) |                        |                        |  [40.0,40.0,-0.05]
2 [00.000.00.10.00,00.000.00.10.00] |                        |             |                        | ChgZone(130,50,'min') |
3 [00.000.00.21.00,00.000.00.30.00] |                        |  At(10,20)  |                        |                        |  [39.0,39.0,0.0]
4 [00.000.00.21.00,00.000.00.30.00] |  RunExperiment         |             | Running ('exp_2')      |                        |  [39.0,39.0,-2.0]
                                    | (10,20,'exp_2')        |             |                        |                        |
5 [00.000.00.30.00,00.000.00.30.00] |                        |             |                        |  NoChargeZone()        |
6 [00.000.00.36.00,00.000.00.45.00] |  Idle()                |             |  Idle()                |                        |  [9.0,9.0,0.0]
7 [00.000.00.36.00,00.000.00.50.00] |                        | GoTo(40,80) |                        |                        |  [9.0,9.0,-0.05]
8 [00.000.00.50.00,00.000.00.50.00] |                        |             |                        | ChgZone(40,80,'min')  |
9 [00.000.00.54.00,00.000.01.10.00] |                        |  At(40,80)  |                        |                        |  [8.0,8.1,0.0]
10 [00.000.00.54.00,00.000.01.19.00]|                        |             |                        |                        |  [8.0,8.1,2.0]
11 [00.000.01.03.00,00.000.01.28.00]|                        |             |                        |                        |  [26.0,38.1,0.0]
12 [00.000.01.03.00,00.000.01.28.00]|                        |  GoTo(10,25) |                        |                        |  [26.0,38.1,-0.05]
13 [00.000.01.04.00,00.000.01.29.00]|                        |  At(10,25)  |                        |                        |  [25.0,38.05,0.0]
14 [00.000.01.04.00,00.000.01.29.00]|  RunExperiment         |             | Running('go_to_cave')|                        |  [25.0,38.05,-2.0]
                                    | (10,25,'go_to_cave')|             |                        |                        |
15 [00.000.01.14.00,00.000.01.39.00]|  Idle()                |             |  Idle()                |                        |  [5.0,18.05,0.0]
16 [00.000.01.14.00,00.000.01.39.00]|                        |  GoTo(0,0)  |                        |                        |  [5.0,18.05,-0.05]
17 [00.000.01.15.00,00.000.01.40.00]|                        |  At(0,0)    |                        |                        |  [4.0,18.0,0.0]
18 [00.000.01.40.00,00.000.01.40.00]|                        |             |                        |  NoChargeZone()        |
19 [00.000.02.30.00,00.000.02.30.00]|  <UNDECIDED>           |  <UNDECIDED> |  <UNDECIDED>           |  <UNDECIDED>           |  [4.0,18.00,0.0]
```

Listing 3. "Fix Solution"

```
P        Time       |                   MTL                    |      MS      |            PLD              |          CZN           |       BAT
---------------------------------------------------------------------------------------------------------------------------------------------
 0 00.000.00.00.00 |  Idle()                                  |  At(0,0)     |  Idle()                     |  None()                |  [40.0,0.0]
 1 00.000.00.01.00 |                                          |  GoTo(10,20) |                             |                        |  [40.0,-0.05]
 2 00.000.00.10.00 |                                          |              |                             |  Charge(130,50,'min') |
 3 00.000.00.21.00 |  RunExperiment(10,20,'exp_2')           |  At(10,20)   |  Running('exp_2')           |                        |  [39.0,-2.0]
 4 00.000.00.30.00 |                                          |              |                             |  None()                |
 5 00.000.00.36.00 |  Idle()                                  |  GoTo(40,80) |  Idle()                     |                        |  [9.0,-0.05]
 6 00.000.00.50.00 |                                          |              |                             |  Charge(40,80,'min')  |
 7 00.000.00.54.00 |                                          |  At(40,80)   |                             |                        |  [8.1,2.0]
 8 00.000.01.03.00 |                                          |  GoTo(10,25) |                             |                        |  [26.1,-0.05]
 9 00.000.01.04.00 |  RunExperiment(10,25,'go_to_cave')      |  At(10,25)   |  Running('go_to_cave')      |                        |  [26.05,-2.0]
10 00.000.01.14.00 |  Idle()                                  |  GoTo(0,0)   |  Idle()                     |                        |  [6.05,-0.05]
11 00.000.01.15.00 |                                          |  At(0,0)     |                             |                        |  [6.0,0.0]
12 00.000.01.40.00 |                                          |              |                             |  None()                |
13 00.000.02.30.00 |  <UD>                                    |  <UD>        |  <UD>                       |  <UD>                  |  [6.0,0.0]
```

the planning process. The need for generating flexible plans, feasible on resource allocation not only in fixed points, is necessary to protect against uncertainty in execution time and telemetry values.

The paper also introduced PLASMA (PLAn Space Multi-solver Application), a planner able to generate flexible plans combining principles from Partial Order and Least Commitment Planning and Partial Order Schedule. Besides that, PLASMA increases the integration of symbolic planning with numerical reasoning with resources, entailing planning on problems formulated directly on resource levels desiderata.

REFERENCES

1. James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. A. Ceballos, S. Bensalem, A. Cesta, L. De Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. Van Winnendael. A goal-oriented autonomous controller for space exploration. In *ASTRA 2011. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, 2011.
4. A. Cesta, R. De Benedictis, A. Orlandini, A. Umbrico, and G. Bernardi. Integrated Planning and Scheduling Capabilities to Support Space Robotics. In *Proceedings of the ASTRA 2013, 12th Symposium on Advanced Space Technologies in Robotics and Automation*, 2013.
5. 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, January 2002.
6. Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, Angelo Oddi, and Giulio Bernardi. Deploying interactive mission planning tools - experiences and lessons learned. *JACIII*, 15(8):1149–1158, 2011.
7. Cheng-Chung Cheng and Stephen F. Smith. Generating feasible schedules under complex metric constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1086–1091, Seattle, Washington, USA, August 1994. AAAI Press/MIT Press.
8. S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. ASPEN - Automated Planning and Scheduling for Space Mission Operations. In *Proceedings of SpaceOps 2000*, 2000.
9. J. Delfa Victoria, S. Fratini, N. Policella, Y. Gao, and O. Stryk. QuijoteExpress - A Novel APSI Planning System for Future Space Robotic Missions. In *Proceedings of the ASTRA 2013, 12th Symposium on Advanced Space Technologies in Robotics and Automation*, 2013.
10. EUROPA. Europa Software Distribution Web Site. http://code.google.com/p/europa-pso/wiki/EuropaWiki, 2008.
11. J. Frank and A. Jonsson. Constraint based attribute and interval planning. *Journal of Constraints*, 8(4):339–364, 2003.
12. Jeremy Frank and Paul H. Morris. Bounding the resource availability of activities with linear resource impact. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 136–143, 2007.
13. S. Fratini and A. Cesta. The APSI Framework: A Platform for Timeline Synthesis. In *Proceedings of the 1st Workshops on Planning and Scheduling with Timelines at ICAPS-12, Atibaia, Brazil*, 2012.
14. S. Fratini, S. Martin, N. Policella, and A. Donati. Planning-based controllers for increased levels of autonomous operations. In *ASTRA 2013. 12th Symposium on Advanced Space Technologies in Robotics and Automation*, 2013.
15. 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.

16. P. Laborie. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and new Results. *Artificial Intelligence*, 143:151–188, 2003.

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

18. Nicola Muscettola, Paul Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *In Principles of Knowledge Representation and Reasoning*, 1998.

19. N. Policella, A. Cesta, A. Oddi, and S. F. Smith. From Precedence Constraint Posting to Partial Order Schedules. *AI Communications*, 20(3):163–180, 2007.

20. Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.