Interleaving Temporal Planning and Execution for an Autonomous Rover *

Solange Lemai and **Félix Ingrand**
LAAS/CNRS,
7 Avenue du Colonel Roche, F-31077 Toulouse Cedex 04, France
{*slemai,felix*}*@laas.fr*

Abstract.

Mission planning and execution for autonomous rovers with limited resource capacities while moving around in dynamic environments require to address temporal, resource and uncertainty issues. The use of a temporal planner and a temporal executive which processes are interleaved is desirable. In this paper we propose a framework to integrate deliberative planning, plan repair and execution control in a dynamic environment with real-time constraints. It is based on lifted partial order temporal planning techniques which produce flexible plans and allow, under certain conditions discussed in the paper, plan repair interleaved with plan execution. This framework has been implemented using the planning system IxTeT, integrated in the LAAS architecture in interaction with a procedural executive, tested in simulation and deployed on an ATRV robotic platform.

1 Introduction

Space systems such as spacecraft or rovers become more and more complex, embedding various sensors, effectors and processing functions. Such systems usually execute a sequence of commands elaborated and monitored by experts from a ground station (SOJOURNER), or by mixed initiative planning systems (MAPGEN for MER (Ai-Chang *et al.* 2003)). Still, due to long communication delays and restricted visibility windows, these systems cannot efficiently be completely teleoperated from the ground. Improving their autonomy demands to supply decisional capabilities on-board, allowing the ground to teleoperate at a goal level.

Space applications however imply stringent requirements. These remote systems have limited resources (energy, propellant, data space storage), still their lifetime can be extended by carefully managing these resources. Another main characteristic of the space domain is the importance of time, almost all data are denoted by temporal intervals (ground stations visibility windows, etc.). A rover is also subject to difficult environmental conditions and evolves in an unknown and dynamic environment. Such a context requires deliberative capabilities to generate plans achieving mission goals and respecting deadlines and long-term resource constraints, as well as run-time plan adaption mechanisms during plan execution.

The development of a complex autonomous system relies on the organization of the software components in a closed loop architecture, usually organized around two levels. A *decisional level* generates a plan of actions which achieves the mission goals, and executes this plan in a robust way. A *functional level*, interfaced with the hardware, executes the refined actions of the plan. The decisional level usually embeds a time-consuming planning process, as well as a time-bounded reactive execution process. We are interested in the key problem of defining how these two processes should interact to balance deliberation and reaction.

Numerous strategies have been applied so far. Some approaches propose the use of a strong executive, enhanced with deliberative capabilities. Procedural executives, such as ESL (Gat 1997), TDL (Simmons and Apfelbaum 1998), PRS (Georgeff and Ingrand 1989), support action decomposition, synchronization, execution monitoring and exception handling. In PROPEL (Levinson 1995), and PropicePlan (Despouys and Ingrand 1999), the planner serves as a resource for the executive to anticipate by simulating a sequence of subplans, or to generate a new subplan corresponding to the current situation. These systems provide a reactive behavior. Still, they do not perform a projection of the state far in the future. This look-ahead is yet necessary to manage the level of a limited resource during the entire mission.

In the "batch planning" approach of the Remote Agent (Muscettola *et al.* 1998) (tested on board the DS1 spacecraft), planning and plan execution are two separate processes. The agent can perform replanning (but with long standby periods) and back to back planning.

Other approaches propose to interleave planning and execution in a continuous way. The planning process remains active to adapt the plan when new goals are added or to resolve conflicts appearing after a state update; and plan steps which are ready to be executed are committed to execution even if the plan is not yet completely generated. Examples include IPEM (Ambros-Ingerson and Steel 1988), based on the classical Partial Order Planning framework; ROGUE (Haigh and Veloso 1998), which has been deployed on mobile robots; or

* Part of this work was funded by a contract with CNES and ASTRIUM.

the multi-agent architecture CPEF (Myers 1999).

But finally, very few approaches really take into account timing problems, such as actions with duration and goals with deadlines, and their effects on the planning and execution processes. Examples of temporal systems developed for space applications include CASPER and IDEA. In the CASPER system (Chien *et al.* 2000), state and temporal data are regularly updated and potential future conflicts are incrementally resolved using iterative repair techniques. However this approach does not handle conflicts which appear within the replanning time interval. The IDEA approach (Muscettola *et al.* 2002) proposes the use of temporal planning techniques at any level of abstraction (mission planning as well as reactive execution). This system offers look-ahead abilities with flexible planning horizons and the use of a common language at any level of abstraction, but does not yet handle non-unary resources.

In this paper we propose a new framework to combine deliberative planning, plan repair and execution control that takes into account resource level updates and temporal constraints. These processes are embedded thanks to two interacting components which explicitly represent and reason about time: a temporal planner and a temporal executive. First the planner elaborates a complete plan. This plan is then run by the temporal executive following a "Sense/Plan Repair/Act" cycle: integrate external messages, repair the plan if needed, decide to execute actions. This cycling enables interaction with the controlled system by taking into account runtime failures and timeouts, and updating the plan accordingly. Interleaving plan repair and execution is motivated by the facts that some parts of the plan may remain valid and executable (parallel branches in the plan independent from the failed actions), and that the plan can be temporally flexible and thus allow postponing and inserting actions. Plan repair uses nonlinear planning techniques under certain assumptions discussed in this paper. This framework assumes that the planning system has the following properties:

- a temporal representation in which the world is described by a set of state variables, each being a function of time,
- a Partial Order Causal Link (POCL) planning process,
- generation of flexible plans based on CSP, particularly the time-map manager relies on a Simple Temporal Network (Dechter *et al.* 1989).

The proposed framework has been implemented in IxTeT-ExEC, using the planning system IxTeT, and integrated in the decisional level of the LAAS architecture, in interaction with a procedural executive, to control an autonomous mobile robot.

The paper is organized as follows. The first section presents the general behavior of the IxTeT-ExEC system, especially the interactions between the executive and the controlled system, and briefly introduces the underlying planning system. The second section details the dynamic replanning framework and addresses the issues raised by the interleaving of planning and execution processes on the same plan. The third section illustrates the performances of IxTeT-ExEC with a rover exploration scenario example involving goal deadlines and resource contention. The last section presents how this framework has been integrated in the LAAS architecture to control a mobile robot.

2 Overview of the IxTeT-ExEC system

2.1 General behavior

The key component in IxTeT-ExEC is the temporal executive: TeXEC, which interacts with the planner and the controlled system. It controls the temporal network of the plan to decide the execution of actions and maps the timepoints to their real execution time.

The execution of an action a with grounded parameters p_a, starting timepoint st^a, ending timepoint et^a, and identifier i_a is started by sending the command (*START* a p_a i_a) to the controlled system. If the action is *non preemptive*, et^a is a non controllable variable, and TeXEC just monitors if a is completed in due course. Otherwise et^a is controllable. If the action did not terminate by itself, it is stopped with the command (*END* i_a) sent as soon as possible if a is *early preemptive*, as late as possible if it is *late preemptive*.

At planning level, the model represents the world as a set of state variables, functions of time, generally piece-wise constant. Thus TeXEC only monitors the effects of an action and not its progress. Such intermediate monitoring has to be done at a lower level.

TeXEC integrates in the plan the reports sent by the controlled system upon each action completion. A report returns the ending status of the action (*nominal*, *interrupted* or *failed*) and a partial description of the system state. If nominal, it contains just the final levels[1] of the resources, if any, used by the action. Otherwise, it also contains the final values of the other state variables relevant[2] to the action.

Besides completion reports, TeXEC reacts to two types of event: user requests to insert a new goal and sudden alterations of a resource capacity (e.g. partial loss of memory storage). Other situations that forbid further execution of the plan without plan adaptation are:
– *temporal failures* - The temporal network constrains each timepoint t to occur inside a time interval $[t_{lb}, t_{ub}]$. Thus two types of failure lead to an inconsistent plan: the corresponding event (typically, the end of an action) happens *too early* or *too late* (time-out).
– *action failure* - The controlled system returns a non nominal report.
– *resource level adjustment* - If an action has consumed/produced more/less than expected, the plan may con-

[1] Note that the system returns a global level of a resource and not a report on the specific quantity used by the action. Indeed, in case of concurrent actions, it is often impossible to discriminate the exact share of a resource usage due to each action.

[2] The state variables present in the action definition.

tain future resource contention. If a reservoir-like resource is over-produced, it may contain future overflow.

To take advantage of the temporal flexibility of the plan, the dynamic replanning strategy has two steps. A first attempt consists in repairing the plan using IxTeT algorithms while executing its valid part in parallel. If this plan repair fails or if a timepoint times out, the execution is aborted and IxTeT completely replans from scratch. Before describing in more details this strategy in the next section, we briefly introduce the underlying planning system.

2.2 The IxTeT planning system

IxTeT is a lifted POCL temporal planner based on CSPs. Its temporal representation describes the world as a set of *attributes*: logical attributes (e.g. AT_ROBOT_X($?r$) representing the position of the robot $?r$), which are multi-valued functions of time, and resource attributes (e.g. BATTERY_LEVEL()) over which borrowings, consumptions or productions can be specified. We note $LgcA$ and $RscA$, respectively the sets of logical and resource attributes. $LgcA_g$ and $RscA_g$ designate the sets of all possible instantiations of these attributes.

The evolution of a logical attribute value is depicted through the predicate *hold*, which asserts the persistence of a value over a time interval, and the predicate *event*, which states an instantaneous change of values. The predicates *use*, *consume* and *produce* specify, respectively, the borrowing over an interval, the consumption or the production at a given instant of a given resource quantity.

An action (also called *task*) consists of a set of *events* describing the change of the world induced by the action, a set of *hold* expressing required conditions or the protection of some fact between two events, a set of resource usages, and a set of temporal and binding constraints on the timepoints and variables of the action. An example is given in Figure 3.

A plan relies on two CSP managers (temporal/atemporal). A Floyd-Warshall like propagation schema ensures the global consistency of the temporal network. The other CSP manages symbolic variables with constraints such as domain restrictions, equalities or inequalities. A recent extension ((Trinquart and Ghallab 2001)) has improved the expressiveness of the planner by handling numeric variables over infinite domains and complex numeric constraints. Furthermore, mixed constraints between temporal and atemporal variables can be expressed (e.g. $?dist = ?speed * (et - st)$). They are managed by a supervisor that transfers information from one CSP to the other one when required. These CSP managers take part in the elaboration of flexible plans: they compute for each variable a minimal domain which reflects only the necessary constraints in the plan.

The search explores a tree $\mathcal{T}$ in the partial plan space. In a POCL framework, a partial plan is generally defined as a 4-tuple (A, C, L, F), where A is a set of partially instantiated actions, C is a set of constraints on the temporal and atemporal variables of actions in A, L is a set of causal links[3] and F is a set of flaws. A partial plan stands for a family of plans. A partial plan is considered to be a valid solution if all these plans are coherent, that is F is empty.

The root node of $\mathcal{T}$ consists of: the initial state (initial values of all instantiated attributes), expected availability profiles of resources, goals to be achieved (desired values for specific instantiated attributes) and a set of constraints between these elements. The branches of $\mathcal{T}$ correspond to resolvers (new actions or constraints) inserted into the partial plan in order to solve one of its flaws. Three kinds of flaws are considered:

– *open conditions* are events or assertions that have not yet been established. Resolvers consist in finding an establishing event (in the plan or thanks to the insertion of a new action) and adding a causal link (an assertion *hold*) that protects the attribute value between the establishing event and the open condition.

– *threats* correspond to pairs of *event* and *hold* which values are potentially in conflict. Such conflicts are solved by adding temporal or binding constraints.

– *resource conflicts* are detected as over-consuming cliques in a particular graph. Resolvers include insertion of resource production action, etc. (Laborie and Ghallab 1995).

The control algorithm detects flaws in the current partial plan; if there is no flaw, a solution is found; otherwise a flaw is selected, a resolver is chosen in the associated list of possible resolvers and inserted into the partial plan on which the algorithm proceeds recursively. This algorithm is complete. Moreover, the flaw and resolver choices are guided by diverse heuristics not discussed here (see (Garcia and Laborie 1995)).

The advantages of the CSP-based functional approach are numerous in the context of plan execution. Besides the expressiveness of the representation (handling of time and resources), the flexibility of plans (partially ordered and partially instantiated, with minimal constraints) is well-adapted to their execution in an uncertain and dynamic environment. Plans are further constrained at execution time. Finally, the planner, performing a search in the plan space, can be adapted to incremental planning and plan repair.

3 Temporal plan execution and replanning

This section details the execution cycle - the plan updates (action reports, state and resource levels), and the parallel plan repair and execution - as well as the complete replanning process. Interleaving partial order planning and execution may introduce flaws in the plan, and we formally specify under which conditions such a partial plan remains executable. With this aim in view, we first present some useful notations and definitions.

[3] A causal link $a_i \xrightarrow{p} a_j$ denotes a commitment by the planner that a proposition p of action a_j is established by an effect of action a_i. The precedence constraint $a_i \prec a_j$ and binding constraints for variables of a_i and a_j appearing in p are in C.

3.1 Definitions

We extend the previous definition of a partial plan to the definition of P_t: a **partial plan partially executed up to time** t.

Definition 1 $P_t = (RA_t, FA_t, S_t, G_t, C_t, L_t, F_t)$.

RA_t is the set of currently *running actions* ($a \in RA_t$ if $st^a_{ub} < t$ and $et^a_{ub} > t$), FA_t is the set of *future actions* ($a \in FA_t$ if $st^a_{ub} \geq t$). S_t represents the *state of the world* at time t. It is composed of 2 sets: $LgcS_t$ contains the last value of each attribute $la \in LgcA_g$, [4] $RscL_t$ contains the level at time t of each resource $r \in RscA_g$. G_t is the set of *goals* not yet completely achieved at time t (and eventually not established) [5]. C_t is the set of constraints on the variables appearing in FA_t, RA_t, S_t and G_t. L_t is the set of causal links supporting future actions. F_t is the set of flaws present in the partial plan at time t.

Past actions do not belong to the plan anymore. Their timepoints and parameters are completely instantiated and thus do not appear in C_t, still, useful information about their effects is kept in S_t.

The level of a resource at a certain time in the future cannot be computed, since it depends on the partial order of actions using this resource. But at time t the past part of the plan is completely instantiated and linearized. Two cases have to be considered: if no running action modifies r, the exact level can be computed (case (1)); if at least one action in RA_t requires the resource, only an estimation is available (case (2)).

In case (1), the exact level is computed in IxTeT according to formula (1). We note $C(r)$ the resource capacity. Let p be a production, belonging to the action a^p, of a quantity $?q_p$ of the resource r at time t^p and $P(r)$ be the set of productions of resource r in the plan. Similarly, $C(r)$ is the set of consumptions c, belonging to the action a^c, of a quantity $?q_c$ at time t^c, and $B(r)$ is the set of borrowings b, belonging to the action a^b, of a quantity $?q_b$ between st^b and et^b. Then, if no running action modifies r, $RscL_t(r) = Lev^{past}_t(r)$ with:

$$Lev^{past}_t(r) = C(r) + \sum_{\substack{p \in P(r)/ \\ t^p_{ub} < t \text{ and} \\ a^p \notin RA_t}} ?q_p - \sum_{\substack{c \in C(r)/ \\ t^c_{ub} < t \text{ and} \\ a^c \notin RA_t}} ?q_c \quad (1)$$

This level is a variable ranging over $[lev^{past}_{lb}, lev^{past}_{ub}]$.

In case (2), the uncertainty follows from the insufficient model of the resource usage (piece-wise constant whereas a production, for instance, may correspond to a monotonic increase). The previous level definition is completed by an estimation of the level modification by the running actions ($Lev^{RA}(r)$) according to formula (2). The total amount produced, consumed or borrowed by an action is represented by a variable $?q$ in $[q_{min}, q_{max}]$. At a given time in the course of the action, the only information that the planner can deduce is that the amount produced/consumed up to now is in $[0, q_{max}]$ or that the amount borrowed is in $[q_{min}, q_{max}]$. An estimation of the level would then be:

$$lev^{RA}_{min} \leq RscL_t(r) - Lev^{past}_t(r) \leq lev^{RA}_{max} \quad (2)$$

$$\text{with } lev^{RA}_{min} = -\sum_{\substack{c \in C(r)/ \\ a^c \in RA_t}} q_{c_{max}} - \sum_{\substack{b \in B(r)/ \\ a^b \in RA_t}} q_{b_{max}}$$

$$lev^{RA}_{max} = \sum_{\substack{p \in P(r)/ \\ a^p \in RA_t}} q_{p_{max}} - \sum_{\substack{b \in B(r)/ \\ a^b \in RA_t}} q_{b_{min}}$$

Finally, the level of a resource r at time t is comprised in the interval $[RscL_{lb}, RscL_{ub}]$ with, in case (1):

$$RscL_{lb} = lev^{past}_{lb}, RscL_{ub} = lev^{past}_{ub},$$

and in case (2):

$$RscL_{lb} = lev^{past}_{lb} + lev^{RA}_{min}, RscL_{ub} = lev^{past}_{ub} + lev^{RA}_{max}.$$

A timepoint in the temporal network may correspond to a goal timepoint or to an action start or end timepoint.

Definition 2 (executable timepoint) *A timepoint T is $executable$ at time t if all timepoints T^p that must directly precede it in the temporal network have already been executed ($T^p_{lb} = T^p_{ub} < t$) and if $t \in [T_{lb}, T_{ub}]$.*

A goal is instantaneously achieved or persistent (achieve and maintain a property between st^g and et^g).

Definition 3 (achievable goal) *A goal g is $achievable$ at time t if st^g is executable and if $g \notin F_t$.*

Let A^f_t be the set of actions that are involved in F_t. [6]

Definition 4 (executable action) *A future action a is $executable$ at time t if its start timepoint is executable and if $a \notin A^f_t$.*

Definition 5 (executable plan) *A partial plan P_t is $executable$ at time t if the constraint networks are consistent and if $RA_t \cap A^f_t = \emptyset$.*

3.2 Execution cycle

The solution plan produced by the planner is run by the executive following a "sense/plan repair/act" cycle. The executive wakes up when it needs to do something, i.e. a message has been received, or it is time to execute some timepoint or a plan repair process is in progress. Let us call $ExecutingPlan$ the plan being executed. *Sensing* consists in integrating messages in $ExecutingPlan$ which may partially invalidate it. If $ExecutingPlan$ contains new flaws, a *plan repair* consists in keeping the structure of the plan (the

[4] In IxTeT, $LgcS_t$ contains the last executed *event* for each la.

[5] In IxTeT, a goal is represented by a grounded proposition $hold(GoalAtt(g) : GoalValue, (st^g, et^g))$. G_t contains goals such that $et^g_{ub} \geq t$.

[6] The determination of A^f_t is straightforward in the case of open conditions and resource conflicts. In a threat case, an action a_k has effects in contradiction with the establishment of proposition p by the causal link $a_i \xrightarrow{p} a_j$ and $(a_i \prec a_k \prec a_j)$ is consistent. A^f_t contains a_k and a_j.

ordering of actions) and taking advantage of the flexibility to try and find a solution plan. Planning is distributed on several cycles to allow reactivity to events and concurrent execution of the valid part. *Acting* consists in determining which timepoints have to be executed, processing them or detecting time-out.

We call *timestep* (ts) the maximum time allowed to the execution cycle. It is defined by the user and may vary with the application. The exact execution time is not guaranteed if two timepoints have to be executed within an interval less than one timestep: they will be executed during the same cycle still according to their precedence constraints. Thus, the minimal duration of an action is one timestep. Note however that the cycle usually takes less time, and the global reactivity to new messages is better.

Distributing planning on several cycles raises two important problems:

1. **On which plan relies the execution in the "act" part, especially if no solution has been found?** This plan has to be *executable* (the currently running actions are fully supported). At each planning step, the node is labeled if the corresponding partial plan is *executable*. When the maximum time allowed to plan repair in the cycle is reached, and if the current partial plan is not acceptable, the last labeled node is chosen and its corresponding plan becomes $ExecutingPlan$.
2. **On which plan and which search tree relies the planning process in the next cycle?** If no decision has been made meanwhile (no timepoint execution, no message reception), the search tree can be kept as is and further developed during the next "plan repair" part. It is even possible to backtrack on decisions made in previous cycles. However, if the plan has been modified, a new search tree is mandatory. Its root node is the new $ExecutingPlan$. The planning decisions made in the previous cycles are now fixed, no backtrack is possible.

The following subsections further detail the different phases of the cycle. Basically, all modifications made to $ExecutingPlan$ have to guarantee that an *executable* plan is available after each step of the cycle. If this condition does not hold, cycling is stopped and a complete replanning is mandatory. During a cycle without plan repair, $ExecutingPlan$ remains a solution plan.

3.3 Message integration

A message can be of three different types: report upon action completion, new goal request or notification of a capacity alteration.

Report A report is associated with the end timepoint et^a of the corresponding action a. If the message is received inside the bounds $[et^a_{lb}, et^a_{ub}]$, et^a is set to the current time t [7]. If it is too late, the plan is not executable anymore. If it is too early (typically, a failed action can terminate very soon), a new end timepoint, set to t and constrained to occur before the executable timepoints, is created and the failed one is relaxed (constraints on the failed timepoint are removed). The network is then recomputed. In IxTeT, such an operation keeps the network consistent since the only constraint that can be specified between two actions a and a' is a precedence constraint which upper bound is flexible: $(st^{a'} - et^a)$ in $]0, +\infty[$. If the report contains information about the state, S_t is updated in the following way:

Resource level **-** For each resource r, the report returns the current "real" level l_r. $RscL_t(r)$ is compared to l_r and adjusted.

- If $l_r \in [RscL_{lb}, RscL_{ub}]$, the plan is consistent with reality. If the exact level can be computed (case (1) - see Definitions), its value is updated by adding the constraint $RscL_t(r) = l_r$.
- If $(l_r < RscL_{lb})$, the over-consumption is reflected on the plan by adding a consumption of quantity $c = RscL_{lb} - l_r$. In case (1), the new level ($RscL'_t(r) = RscL_t(r) - c$) is updated by adding the constraint $RscL_t(r) = RscL_{lb}$, equivalent to $RscL'_t(r) = l_r$. If some running action modifies r (case (2)), the part of the level due to past actions ($L^{past}_t(r)$) is updated by considering the worst case and adding the constraint $L^{past}_t(r) = lev^{past}_{lb}$.
- If $(l_r > RscL_{ub})$, the over-production is reflected on the plan by adding a production of quantity $p = l_r - RscL_{ub}$. Similarly, the level is updated by adding the constraint $RscL_t(r) = RscL_{ub}$ in case (1), and $L^{past}_t(r) = lev^{past}_{ub}$ in case (2).

The plan remains executable in case (1) since no running action is concerned by r, whereas conflicts may appear in case (2), requiring execution abortion. If the plan is executable, plan repair is requested in case of over-consumption and in case of over-production of a reservoir resource.

State variables **-** $LgcS_t$ contains the last value for each instantiated logical attribute. If the report is nominal, $LgcS_t$ is updated with the effects of a expected in the plan. Otherwise, it is updated with the values returned in the report. A value is not inserted if it leads to a non executable plan (threatens some proposition of a running action a_r). In that case and if a_r is preemptive, its interruption is requested. Else, the value is inserted and causal links which contradict it are broken. This update leads to an executable plan with open conditions on which plan repair can be processed.

Goal request One specifies for each goal: a priority, a duration interval I^d, an estimation of its minimal achievement duration d_{achiev} and a time interval I^g for its start timepoint st^g. A goal is rejected if the following set of constraints is not consistent with the plan at current time t: $\{((et^g - st^g)$ in $I^d)$, $((st^g -$ origin$)$ in $I^g)$, $(st^g$-origin $\geq t + d_{achiev})\}$. Moreover the goal is constrained to occur after each running

[7] Equivalent to posting the constraint $(et^a - origin) = t$ in the STN.

action a such that the goal proposition threatens a proposition in a. A goal integration results in an executable plan with an open condition.

Capacity alteration As a consequence of an exogenous uncontrollable event a resource capacity has increased or decreased by a quantity q. The capacity is updated. If the plan is not executable anymore, execution is aborted. Otherwise a plan repair is requested in case of decrease or increase of a reservoir resource.

Causal links removal After message integration, the plan may contain flaws on a set of grounded attributes Att^f, eventually repaired thanks to the insertion of new actions. Let consider Att^i the set of the attributes appearing in these actions. Additional causal links, protecting propositions in the plan on attributes in Att^i, have to be broken to allow the insertion of actions in the current plan structure.

The determination of Att^i is based on information given by an abstraction hierarchy verifying the Ordered Monotonicity Property (Knoblock 1994) and generated offline from the model description. Notably, this hierarchy differentiates the primary and secondary effects of an operator. Primary effects justify the insertion of an action to solve a flaw. Let us call *main attributes* of an action the attributes appearing in its primary effects. Att^i, initialized with Att^f, is computed by searching the action operators for which at least one attribute att_m in Att^i is a main attribute. This operator is partially grounded (by binding its corresponding parameter with att_m) and the (eventually grounded) attributes appearing in the operator and not yet taken into account are added to Att^i. The algorithm proceeds recursively until a fixed point is reached.

In IxTeT, the selection of the causal links to break is an adaptation of the work presented in (Gaborit 1996). In short, a causal link in L_t is removed if it concerns an attribute in Att^i and does not belong to the extension of its establishing event [8].

Finally, the partial plan is executable and the sets of actions independent from the failures are executable.

3.4 Plan repair

The plan repair is similar to the IxTeT search process in the plan space. The root of the search tree $\mathcal{T}$ is $ExecutingPlan$, partially invalidated. $\mathcal{T}$ is developed according to an ordered depth first search strategy. Planning is distributed, if necessary, on several cycles and each time a new timepoint is inserted, it is constrained to occur after the end of the current cycle. Planning during one cycle is done one step at a time until it results in a dead-end (there is no solution), or a solution is found or a deadline is reached. This deadline corresponds to the share of the timestep ts allocated to the plan repair part. This parameter μ can be tuned by the user.

[8] The extension of an event corresponds to the interval during which the value established by the event is imposed (by some assertion of an action, ...).

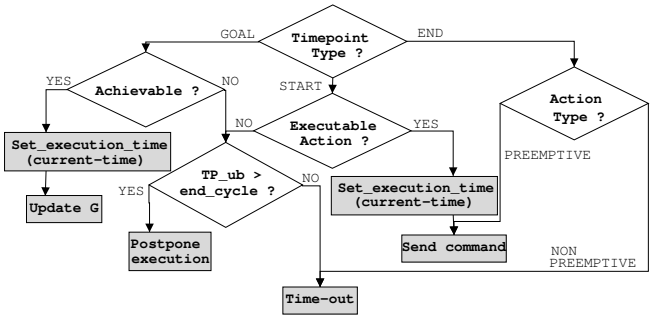

Figure 1: Timepoint execution.

Some aggregation mechanisms (corresponding to past forgetting) allow a significative reduction of the search space. Especially in IxTeT, the establishing events are looked for in $LgcS_t$ and executed resource propositions are aggregated in one proposition.

This plan repair process is not guaranteed to find a valid plan everytime (backtrack nodes frozen by execution or temporal constraints too tight to add new actions ...), but can avoid aborting execution and completely replanning at each failure. By invalidating only a part of the plan, the amount of decisions is rather limited and a repaired plan may be found in a few cycles. Plan repair is especially efficient and useful for temporally flexible plans and plans with some parallelism (some sets of actions can be executed independently). This mechanism is also efficient to compensate for inadequate models of actions. For instance, it is almost impossible to have an accurate estimation of the time taken by a robot to go from a location L_1 to a location L_2 in an unknown and dynamic environment. In the IxTeT-eXeC model, move(L_1,L_2) is defined as a late preemptive action. If the robot takes longer than expected in the model (e.g. due to obstacle avoidance), the action is interrupted. The controlled system returns the intermediate location L_i and, if some temporal flexibility remains, a new move(L_i,L_2) is inserted and launched. This example is simple but representative of the failures that frequently break plan execution.

3.5 Action

Each timepoint is associated with an *execution time* t_{exec}. If T is a start or goal timepoint, or an end timepoint of an early preemptive action, $t_{exec} = T_{lb}$. If T is an end timepoint of a late preemptive action, $t_{exec} = T_{ub} - ts$. If T is an end timepoint of a non preemptive action, $t_{exec} = T_{ub}$. During the "act" part, the executive determines the set of timepoints to execute during the current cycle ($ExecTPs$): these timepoints are executable and their execution time happens before the end of the cycle. $ExecTPs$ is updated after each timepoint execution to take into account newly executable timepoints. Figure 1 summarizes timepoint execution, depending on its type. If a goal is not achievable or if an action is not executable and if some flexibility remains, their execution is postponed. The execution time of a preemptive end timepoint is set when the corresponding report is received (expected in the next cycle). Finally, timeouts are detected when reports have not been received in time.

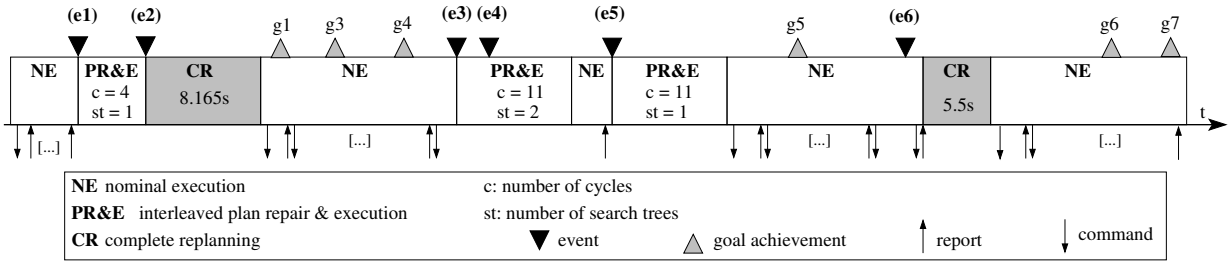

Figure 2: Example of scenario and reactions.

During plan execution, the assignment of the execution time is equivalent to adding a constraint between the origin and the timepoint. The propagation algorithm keeps the STN *minimal* (the edge constraints are minimal with respect to the intersection) and guarantees that a complete execution is possible.

3.6 Complete replanning

When the plan is not executable anymore, it is necessary to replan from scratch. Execution is aborted, leading to the partial plan $P_{t_s} = (\emptyset, FA_{t_s}, S_{t_s}, G_{t_s}, C_{t_s}, L_{t_s}, F_{t_s})$. The initial planning problem is extracted from P_{t_s} as:

$$
\begin{aligned}
P_{t_i} &= (\emptyset, \emptyset, S_{t_i}, G_{t_i}, C_{t_i}, \emptyset, F_{t_i}) \\
S_{t_i} &= S_{t_s}, \\
G_{t_i} &= \{g \in G_{t_s} / \text{ temporal constraints on } g \\
&\quad \text{are coherent with current time}\}, \\
C_{t_i} &= \{c \in C_{t_s} / c \text{ is a constraint just} \\
&\quad \text{on variables appearing in } S_{t_i} \text{ and } G_{t_i}\} \\
&\quad (C_{t_i} \text{ notably contains constraints} \\
&\quad \text{on origin and horizon timepoints}), \\
F_{t_i} &\quad \text{contains the open conditions in } G_{t_i}.
\end{aligned}
$$

Non linear planning can not be interrupted at any time and come up with an applicable plan. Still we have to guarantee that there remains enough time at the end of the replanning process to execute the solution plan and meet the goal deadlines. We have been inspired by the approach used in the Remote Agent (Muscettola *et al.* 1998) ("planning to plan") and propose to add a specific flexible timepoint T^{end} to P_{t_i}, that corresponds to the end of the planning process. T^{end} is only constrained to occur before the end of the horizon (and after t_i). Each time a new timepoint is inserted by the planning process, it is constrained to occur after T^{end} (and before the end of the horizon). Thus T^{end}_{ub} decreases as new actions or new temporal constraints are added, and there is not enough time to execute the current plan if: $T^{end}_{ub} <$ current time. Note however that T^{end}_{ub} can increase when backtracking.

The strategy is then to plan one step at a time until it results in a dead-end, or a solution is found, or a time limit l is reached. l is defined as $l = T^{end}_{ub} - d$, d being a *slack* duration (tuned by the user) to save enough time at the end of planning for execution cycle initialization. l is updated after each planning step. Planning is stopped when l is reached unless the next planning step corresponds to a backtrack node. In that case, and if the next step increases l, planning is pursued.

If planning is aborted without finding a solution, some goals are rejected and a new attempt is done. A new plan P_{t_i} is extracted from P_{t_s}. At this point, some goals may have been rejected due to temporal constraints not consistent with current time. Otherwise, one goal is selected and abandoned. Is rejected the goal with the lowest priority, and, if several goals have the same priority, the goal with the less flexibility for its achievement (this flexibility is computed as (st^g_{ub} - d_{achiev})). This criterion has been chosen to keep the goals that are more likely to be achieved in due course.

A drawback of this strategy is that the state of the controlled system is supposed to remain unchanged during the planning process. The solution found is valid with respect to this initial state. The advantage of our global approach is that, if the state has changed, the plan may be repaired once execution is started (resource updates, etc.). An improvement would consist in updating the system state (S_{t_s}) between each complete replanning attempt.

4 Example of scenario

We face some difficulties to evaluate our work since there is no benchmark (involving time and resource issues) to apply the system to. We illustrate in this section the performances of IxTeT-eXeC with an example of failure scenario for a simulated rover with an exploration mission. In such a domain, the quantitative effects and durations of actions can be estimated in advance for planning but are accurately known only at execution time (e.g. the actual compression rate of an image or the actual duration of a navigation task).

Goals are a list of targets to take a picture of, constraints correspond to deadlines and limited resources (memory storage, battery level). The planning model contains five actions: move, take_picture, move_ptu (change the orientation of the cameras through commands to the pan&tilt unit), download images (to free memory storage) and recharge battery (the robot stays still while solar panels recharge the battery). The sequence of actions to achieve a goal is: go to the target (camera looking forward), change the camera orientation, take a picture.

IxTeT-eXeC is interfaced with the procedural system OpenPRS which simulates the robot state evolution (position of the robot, progressive decharge of the battery level, compression and download rates of images, etc.). Figure 2 describes an

```
task MOVE(?xi,?yi,?xf,?yf)(st,et){
 ?xi,?yi,?xf,?yf in ]-oo,+oo[;
 event(AT_ROBOT_X():(?xi,+oo),st);
 hold(AT_ROBOT_X():+oo,(st,et));
 event(AT_ROBOT_X():(+oo,?xf),et);
 event(AT_ROBOT_Y():(?yi,+oo),st);
 hold(AT_ROBOT_Y():+oo,(st,et));
 event(AT_ROBOT_Y():(+oo,?yf),et);
 hold(PTU_POS():FORWARD,(st,et));
 hold(MVT_INIT():T,(st,et));
 event(ROBOT_STATUS():(STILL,MOVING),st);
 hold(ROBOT_STATUS():MOVING,(st,et));
 event(ROBOT_STATUS():(MOVING,STILL),et);
 use(CAMERA():1,(st,et));
 variable ?s,?dist,?duration;
 distance(?xi,?yi,?xf,?yf,?dist);
 speed(?s);
 ?dist = ?s * ?duration;
 ?duration = et - st;
}latePreemptive                    (a)

task INIT_MVT_GENERATION()(st,et){
 hold(ROBOT_STATUS():STILL,(st,et));
 use(CAMERA():1,(st,et));
 event(MVT_INIT():(F,IDLE_INIT),st);
 hold(MVT_INIT():IDLE_INIT,(st,et));
 event(MVT_INIT():(IDLE_INIT,T),et);
 (et-st) in ]2,60];
}nonPreemptive                     (b)
```

Figure 3: Example of action models for Dala in IxTeT formalism.

example of IxTeT-eXeC reactivity to a scenario with multiple failures. The initial plan contains 5 goals (g_1,...,g_5). The third goal has a strict deadline and a higher priority. Since storage capacity is limited to 3 images, the plan contains a download action after g_3. The battery initial level is sufficient to complete the plan. The following events occur:

— **(e_1) action failure** - The first take picture fails, plan repair begins.

— **(e_2) timeout** - Due to g_3 deadline, the next executable timepoint times out. After a complete replanning, g_2 has been removed. Further plan execution achieves g_1, g_3 and g_4.

— **(e_3) new goal** - During the download action, a new flexible goal g_6 is received. Plan repair begins.

— **(e_4) new goal** - A new goal g_7 is received 3s later. The repaired plan finally contains three goals not yet achieved.

— **(e_5) sub-production** - The download action has produced less than expected, the new level allows to take only two images. Plan repair leads to the insertion of a download action before the last take picture action.

— **(e_6) capacity alteration** During a move action, a sudden drop in battery level forbids the plan completion without recharging. Even if the current action completion is not directly threatened by the drop, it takes part in the future resource contention. Thus execution is aborted and a complete replanning leads to the insertion of an immediate recharge action. The plan can then be successfully completed.

Note that the battery and memory storage levels are regularly adjusted. This example has been run on a Pentium IV, with $ts = 1.3s$ and $\mu = 61\%$. The average cycle duration during nominal execution is 0.22s.

5 Integration in the LAAS architecture

Experiments have also been carried out on Dala, an iRobot ATRV, with an exploration mission scenario which requires the robot to achieve three types of goals: "take pictures of specific targets" (in locations (0,0), (9,0), (10,-3) and (8,-5)), "communicate with a ground station during visibility windows" (W_1 and W_2), and "return to location (0,0) at the end of the mission". Dala runs a Pentium III under Linux and is equipped with the following sensors: odometry and a stereo camera pair mounted on a pan&tilt unit (PTU). Five main actions are considered at the mission planning level: take_picture, move_ptu, move (cf. Figure 3(a)), download_images, communicate. The first three actions can be physically performed by Dala, the execution of the other actions is simulated on-board the robot. The execution of a move action in an unknown environment implies complex processes: localization, map building, motion generation, etc. (Lacroix *et al.* 2003; Goldberg *et al.* 2002). The LAAS architecture (Alami *et al.* 1998) and its associated tools provide a support in order to design and integrate such an autonomous system.

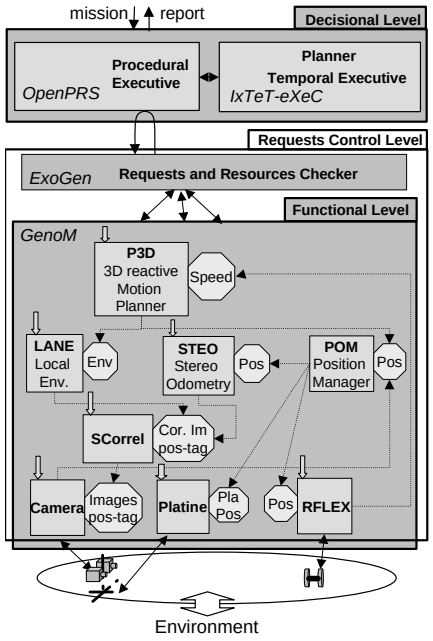

Figure 4: The LAAS architecture on Dala.

Figure 4 presents the architecture as set for the experiment. The *functional level* includes all the basic built-in robot action and perception capabilities, encapsulated into controllable communicating modules. Modules are activated by requests, send reports upon completion and export data. The POM module for instance computes the best position estimate from standard (RFLEX) and visual (STEO) odometry, while the wheels are controlled by RFLEX according to the reference velocity produced by the reactive motion planner (P3D), etc. The *requests control level* filters the requests according to the current state of the system and a formal model of allowed and forbidden states.

IxTeT-eXeC has been integrated in the *decisional level* and interacts with the user and the functional level through a procedural executive (OpenPRS). The plan execution is controlled by both executives as follows. TeXeC decides when to start or stop an action in the plan and handles plan adap-

tations. OpenPRS expands the action into commands to the functional level[9], monitors its execution and can recover from specific failures. It finally reports to TEXEC upon the action completion.

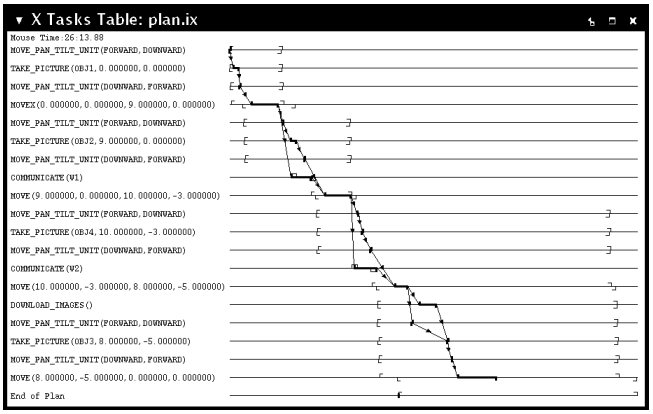

Figure 5: *Initial plan*

This mission (the corresponding initial plan is shown in Fig. 5) has been executed by Dala under IXTET-EXEC control (with $ts = 2s$, $\mu = 60\%$). Each resulting run was different. One is illustrated in Figures 6 and 7. The first figure represents the execution trace[10]. In complement, the second figure indicates for each cycle: its starting time, its total duration and the duration of each phase of the cycle. Two execution events required a plan repair:

1. Due to obstacles, the movex(0,0,9,0) action takes more time than expected. It is interrupted in cycle 5. Not much flexibility is left before the visibility window W_1, still the plan repair finds a solution (in 50 planning steps and 4 cycles) before the starting timepoint of the communicate action times out. A move action is inserted after the communication.
2. A take_picture action takes less time than expected. The plan repair process (39 planning steps) is distributed on 4 cycles (14 to 17). Note that a move_pan_tilt_unit action is launched as soon as it is supported in the plan (cycle 16), and before a solution plan is found.

Figure 8 shows the evolution during the run of the memory level available for the storage of pictures. Measures are done: at the beginning of the mission (cycle 1), after each picture (cycles 3, 14, 22, 29), and after the download action (cycle 28). The points linked by lines correspond to the minimal and maximal levels authorized in the initial plan. The circles and crosses show how these minimal and maximal "theoretical" levels evolve when updated during execution. Finally, the squares represent the real level at the end of each action. Thus, at each action end, the real level is compared with the corresponding theoretical bounds, which are then updated accordingly. The flexibility in the resource model allows to handle the uncertainty on the actual usage of the resource at execution time (here, the actual compression rate). This usage varies with each run, but a plan repair becomes necessary only if the level is outside of the theoretical bounds for which the plan is guaranteed valid and if a conflict is detected.

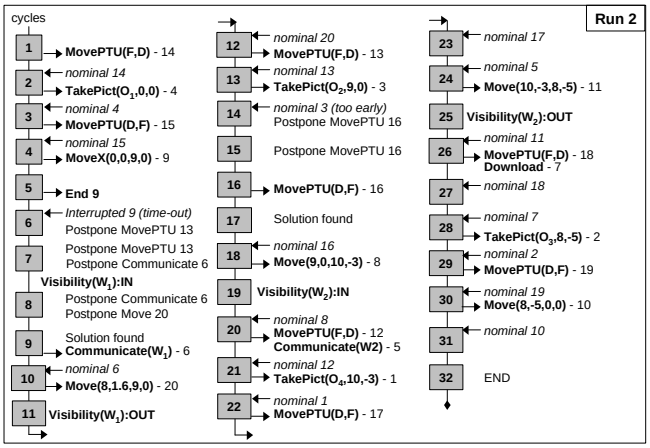

Figure 6: *Execution trace*

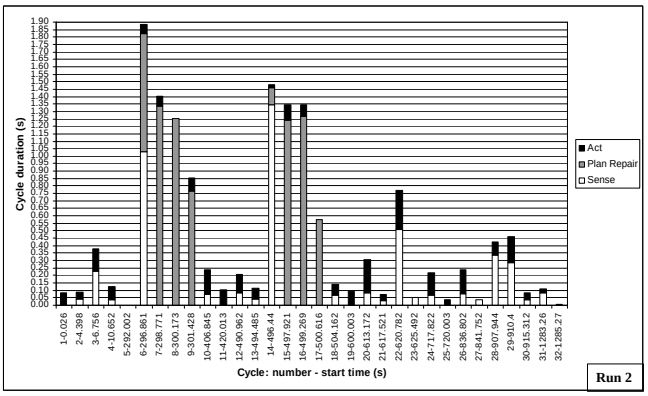

Figure 7: *Cycle start time and duration*

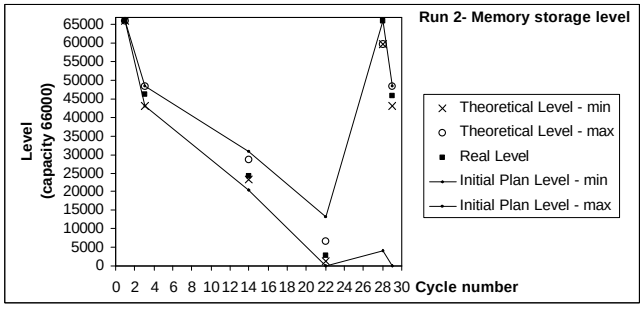

Figure 8: *Evolution of the storage level*

Other failures (e.g. failure of a PTU command and failure of map building) required a reinitialization of certain modules. Two actions (platine_init and init_mvt_generation - cf. Figure 3(b)) and specific attributes (PTU_INIT() and MVT_INIT()) have been added to the model. When a certain type of error is returned by the P3D module (here: "no

[9] For the download_images and communicate actions, specific procedures simulate the visibility windows and the gradual download of images.

[10] It gives for each cycle: the messages exchanged with OpenPRS (in bold: launch or interruption of actions, in italic: reports sent back at the end of the action) and the actions that are postponed when a plan repair is in progress.

map available"), OpenPRS sends back to TEXEC a failed move report containing the value *false* for the attribute MVT_INIT() and the current position of the robot for the attributes AT_ROBOT_X() and AT_ROBOT_Y(). The plan repair process inserts the initialization action followed by a move action to reach the destination.

The goal of these experiments was mostly to show that such an approach can be completely run on board an "out of the shelf" robot and with a "real-world application". The run presented above lasted 1285s, IXTET-EXEC woke up 32 times and used 541s of CPU time, whereas the other main processes: STEO (visual odometry), CAMERA, SCORREL (stereo correlation), OpenPRS, LANE (map building) and P3D (motion planner), have respectively used 270s, 145s, 123s, 80s, 55s and 18s of CPU time.

6 Conclusion and prospects

We have presented the IXTET-EXEC system which combines a temporal lifted POCL planner with a temporal executive to integrate deliberative planning, execution monitoring and replanning while respecting real-time constraints. This approach cannot account for all the possible execution failures in all their generality, nevertheless, in many situations where some temporal and resource flexibility has been left, and for domains where the activities are slightly decoupled, the presented repair techniques greatly improve the overall performance of the system by:
- reducing the number of complete replannings,
- improving the system reactivity to unexpected events,
- taking into account new goals on the fly,
- managing the resources capacity changes,
- managing the uncertainty in the model description (actions duration, consumption/production).

Still, IXTET-EXEC effectiveness can be increased by improving replanning strategies (rejected goals selection, state update requests) and handling contingency earlier in the planning process and propagation algorithms. Indeed, the system only detects when non controllable timepoints time out, but contingent durations can be squeezed during propagation. We plan to use the STNU framework and adapt the polynomial Dynamic Controllability Checking algorithms proposed in (Morris *et al.* 2001). The current results are very encouraging and, providing we use proper heuristics for the search of flaws and resolvers, we envision using this approach for more complex domains.

References

M. Ai-Chang, J. Bresina, L. Charest, A. Jónsson, J. Hsu, B. Kanefsky, P. Maldague, P. Morris, K. Rajan, and J. Yglesias. Mapgen: Mixed initiative planning and scheduling for the mars 03 mer mission. In *International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS)*, 2003.

R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming*, 1998.

J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1988.

S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. In *KR'89: Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann, 1989.

O. Despouys and F. Ingrand. Propice-plan: Toward a unified framework for planning and execution˙ In *Proceedings of the European Conference on Planning (ECP)*, 1999.

Paul Gaborit. Planification distribuée pour la coopération multi-agents. *Thèse de Doctorat*, 1996.

F. Garcia and P. Laborie. Hierarchisation of the search space in temporal planning. In *Proceedings of the European Workshop on Planning (EWSP)*, 1995.

E. Gat. Esl : A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the 1997 IEEE Aerospace Conference*, 1997.

M. P. Georgeff and F. Ingrand. Decision making in an embedded reasoning system. In *Proceedings of the International Conference on Advanced Robotics*, 1989.

S. Goldberg, M. Maimone, and L. Matthies. Stereo vision and rover navigation software for planetary exploration. In *Proc. IEEE Aerospace Conference*, 2002.

Karen Zita Haigh and Manuela M. Veloso. Planning, execution and learning in a robotic agent. In *Proceedings of the International Conference on AI Planning Systems*, 1998.

C. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68, 1994.

P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proceedings of the International Conference on Advanced Robotics*, 1995.

S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, and R. Chatila. Autonomous rover navigation on unknown terrains, functions and integration. *International Journal of Robotics Research*, 2003.

R. Levinson. A general programming language for unified planning and control. *Artificial Intelligence*, 76, 1995.

Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of the International Conference on Advanced Robotics*, 2001.

N. Muscettola, P. P. Nayak, B. Pell, and B. Williams. Remote agent : To boldly go where no ai system has gone before. *Artificial Intelligence*, 103, 1998.

N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt. Idea: Planning at the core of autonomous reactive agents. In *International NASA Workshop on Planning and Scheduling for Space*, 2002.

Karen L. Myers. Cpef: Continuous planning and execution framework. *AI Magazine*, 20(4):63–69, Winter 1999.

R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of Conference on Intelligent Robotics and Systems*, 1998.

Romain Trinquart and Malik Ghallab. An extended functional representation in temporal planning : towards continuous change. In *Proceedings of the European Conference on Planning (ECP)*, 2001.