Plan-Space Hierarchical Planning with the Action Notation Modeling Language

Filip Dvorak, Arthur Bit-Monnot, Félix Ingrand, Malik Ghallab

▶ **To cite this version:**

Filip Dvorak, Arthur Bit-Monnot, Félix Ingrand, Malik Ghallab. Plan-Space Hierarchical Planning with the Action Notation Modeling Language. IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Nov 2014, Limassol, Cyprus. hal-01138105

HAL Id: hal-01138105
https://hal.science/hal-01138105

Submitted on 1 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Plan-Space Hierarchical Planning with the Action Notation Modeling Language

Filip Dvořák[†]
Faculty of Mathematics and Physics
Charles University in Prague
Prague, Czech Republic
Email: filip@dvorak.fr

Arthur Bit-Monnot[*], Félix Ingrand[*], Malik Ghallab[*]
LAAS
CNRS
Toulouse, France
Emails: abitmonn@laas.fr, felix@laas.fr, malik@laas.fr

***Abstract*—Planning in robotics must be considered jointly with Acting. Planning is an open loop activity which produces a plan, based on action models, the current state of the world and the desired goal state. Acting, on the other hand, is a closed loop on the environment activity (to execute command and perceive the state of the world). These two deliberative activities must be integrated and need to handle time, concurrency, synchronization, deadlines and resources. The timeline representation for temporal plan space planning and acting is very expressive; it is also quite flexible for integrating planning and acting. The ANML language is a recent proposal motivated by combining the expressiveness of the timeline representation with the decomposition of HTN methods. This paper reports on FAPE (Flexible Acting and Planning Environment), to our knowledge the first system integrating an ANML planner and actor. Our current focus is not efficient temporal planning per se, but the tight integration of acting and planning, which is addressed by: *(i)* extending HTN methods with refinements, given by PRS procedures, of planned action primitives into low-level commands, *(ii)* interleaving the planning process with acting, the former implements plan repair, extension and replanning, while the latter follows PRS skills refinements, and *(iii)* executing commands with a dispatching mechanism that synchronizes observed time points of action effects and events with planned time.**

FAPE has been integrated to a PR2 robot and experimented in a home-like environment. The paper presents how planning is performed and integrated with acting, and describes briefly the robotics experiments and reports on initial performances.

***Keywords*-planning; robotics; HTN; plan-space;**

I. Introduction

Planning is a form of reasoning, through prediction and search, about future changes that can be produced in a system. These changes occur naturally over time. Most contributions to planning abstract away time as state transitions.[1] At an abstract level, this is a legitimate approximation as it simplifies the reasoning. Explicit time is however required in many applications, e.g., for dealing with synchronization with events and others actors, for managing deadlines and time-bounded resources, and for handling concurrency.

Temporal planning comes in two main flavors: extended state space representations and timeline representations. The former is based on states (i.e., snapshots of the entire system) and temporally qualified durations between states. The latter relies on possible evolutions of individual state variables over time (i.e., partial local views of state trajectories), together with temporal constraints between elements of timelines.

Most recent works on temporal planning favor the extended state space representation on the basis of PDDL2.1 [1] with the so-called durative actions. This drive is explained by the wealth of search techniques and domain independent heuristics that have been developed for state space planning, resulting in significant performance improvements. But of a few exceptions, these planners have however a limited handling of concurrency. The timeline alternative representation permits naturally to refer to instants beyond starting and ending points of actions and to handle various kind of concurrency requirements. It is also more flexible in the integration of planning and acting. Timeline planners implement plan-space search algorithms more often than state-space techniques. These algorithms have not scaled up as well as state space planners or HTN planners.

Hierarchical task networks is indeed a planning representation that accounts for numerous deployed applications of significant size. HTN planners benefit from domain specific knowledge expressed as task decomposition methods. In many domains, methods are very naturally formulated. Temporal planning with HTN has not developed as well as with timelines or state space representations.

The Action Notation Modeling Language (ANML) [2] is motivated mainly by blending the expressive timeline representation with the decomposition of HTN methods. This paper reports on FAPE, and its planner implementing the ANML language. Our motivation is not efficient planning per se, but the tight integration of acting and temporal planning with task decomposition embedded on a robotic platform. This is addressed by:

- extending planning decomposition methods (Planning) with refinements of planned action primitives into low-level commands (Acting), these refinements are currently brought by PRS decomposition procedures,
- interleaving the planning process with acting, the for-

[1]This is also the case for many AI approaches about reasoning on change, e.g., in the LTL, CTL and similar logics, T stands for time, but time is abstracted out.

mer implements plan repair, extension and replanning, while the latter follows PRS refinements,
- executing commands with a dispatching mechanism that synchronizes observed time points of action effects and events with planned time.

The FAPE system currently includes modular components to perform Planning and Acting (as introduced in [3]).

FAPE includes a first ANML planner that supports a unique combination of features of least-commitment plan-space planning, explicit time maintained by a sparse simple temporal network and hierarchical task decomposition. There are several motivations for our design choices:

- plan-space planning with least-commitment naturally supports plan repair, which is essential when acting is a concern,
- simple temporal network supports efficient consistency checking and having a sparse network (without saving constraint propagations) allows us to update temporal relations along with the feedback from execution, and
- hierarchical task decomposition allows for highly scalable domain adaptable planning.

FAPE has been integrated to a PR2 robot and experimented with in a real home-like environment. This is a work in progress. A formalization of the planning–acting integration and a full characterization of the performance of the system are beyond the scope of this paper. Its contribution is to present FAPE at the planning, acting, and execution levels, to describe the robotics experiments and report on initial performances. The outline of the rest of the paper follows these steps, preceded by a brief section on the state of the art and an introduction to ANML.

II. Related work

Numerous planners implements the PDDL2.1 extended state space representation with durative actions, e.g., RPG, LPG, LAMA, TGP, VHPOP and Crickey. Among these planners, COLIN [4] is a notable exception that can manage concurrency and even linear continuous change.

The timeline approach goes back to the IxTeT planner [5] that reasons on chronicles. A chronicle defines time-points, temporal constraints between its instants, changes in the values of state variables, persistence of these values over time, and atemporal constraints over state variables parameters and values. Other planners such as RAX-PS [6], ASPEN [7], Europa/IDEA/T-ReX [8], [9], [10] and APSI [11], rely on a similar temporal representation with timelines and tokens representing change and persistence of the values of state variables over time. Some of these timelines are directly connected to actions and percepts (to integrate perception). These systems express temporal constraints in planning operators using the interval algebra. The organization of the planner along agents (IDEA) or reactors (T-ReX) offers a hierarchical representation of the domain. Still the action models representation with compatibilities (temporal constraints over state variables), which tends to spread out the hierarchical decomposition over more than one compatibilities/reactors, makes them tedious to write and difficult to debug.

The HTN approach is implemented into several planners, e.g., Sipe [12], SHOP2 [13], SIADEX [14]. The latter integrates time to HTN planning without handling concurrency.

ANML [2] extends the languages used in Europa and ASPEN with recent constructs from PDDL together with HTN task decomposition methods. We are aware of ongoing developments on the basis of this language[2], but to our knowledge, FAPE is the first system including an ANML planner supporting task decomposition and temporal planning.

Several systems integrates planning and acting, in particular with procedure-based approaches to refine actions into lower level commands with systems such as RAP [15] or PRS [16]. Among these systems, Cypress [17] (Sipe & PRS), TCA [18] (Task Description Language & Aspen) and XFRM [19] are examples relevant for our approach. IxTeT-Exec [20] and "Configuration Planner" [21] are closer to FAPE since they are based on a timeline planner, but without decomposition method.

III. Representation and ANML

The FAPE planner uses ANML as representation language. ANML is a rich language allowing the user to introduce planning models in a multitude of ways. While the syntactic sugar is important from the perspective of knowledge engineering, let us focus this presentation on the fundamental representations used.

The FAPE planner relies on parametrized state variables, with typed object variables as parameters, and on timelines over these state variables. The advantages of the state variable representation, as in SAS+ [22] are well known. The state space represented by state-variables is significantly smaller (we cut out unreachable states) and planning algorithms strongly benefit from such reduction as shown in [23].

ANML allows us to specify the state variables directly in the planning problem definition. Typing is a natural way to reduce the combinatorics of the parameters in operators. FAPE supports typing and single inheritance between types, as illustrated in this simple example (where < denotes inheritance):

```
type Location;
type Gripper < Location {
        boolean empty; };
type Locatable{
        Location myLocation; };
type Robot < Locatable {
        variable float battery;
        variable Gripper left;
```

[2]In particular at NASA Ames Research Center

```
        variable Gripper right; };
type Item < Locatable;
```

The objects of the domain are type instances:

```
instance Location L1, L2, L3;
instance Robot R1;
instance Item I1;
instance Gripper G1, G2;
```

Temporally annotated statements are for example:

```
[start] R1.myLocation := L1;
[50, 70] I1.myLocation == G1 :-> L3;
[end] I1.myLocation == L3;
```

A temporal annotation is either a time point or interval defined by two time points. These can be relative to a context (e.g. an operator, or a planning problem), such as `start`, `end` and `all`, or absolute time points.

According to the definitions given in [24], we define a temporal statement to be an assertion over the evolution of a parameterized state variable. We consider three type of statements:

- an *event* specifies a change of the value of the state variable. For instance, the ANML statement [t1,t2] r.myLocation == l1 :−> l2 represents a change of the state variable myLocation(r) from $l1$ to $l2$ between time $t1$ and $t2$, where r, $l1$ and $l2$ are object variables and $t1, t2$ are time points. The value of the state variable is $l1$ at time $t1$ and $l2$ at $t2$; it is unspecified in $]t1, t2[$. An event referring to a single time point is considered as being *instantaneous*, e.g., [t] Switch == On :−> Off indicates a value of the switch as On at time t and as Off right after t.
- a *persistence condition* specifies a constraint on the value of a state variable over an interval. For instance, the ANML statement [t1, t2] s.myLocation == l3 states that myLocation(s) keeps the value $l3$ over the interval $[t1, t2]$, where s and $l3$ are object variables and $t1$, $t2$ are time points.
- an *assignment* is a special case of *event* specifying a new value to a state variable regardless of its previous one. For instance, the ANML statement [t] r.myLocation := l3 states that r will be at location l3 at time t without any condition on its previous location.

Actions are defined as partially instantiated operators that may have several possible decompositions into a partially ordered set of primitive actions. Effects and preconditions are represented as temporally annotated statements occurring between the start and end time of the action. Thus a planing operator is a tuple $(name, maxDuration, P, E, D)$, where $name$ is the unique name of the operator, $maxDuration$ is the function that can be evaluated into a number at the moment of operator application and represents its maximal duration (after which the operator is considered to be failed), P is a set of typed parameters, E is a set of temporal statements and D is a set of decompositions. Parameters of an operator are typed object instances as defined in ANML, they are further used to impose binding constraints between events and decomposition operators. A decomposition is a set of partially ordered and partially instantiated operator references (the action must always occur in the time interval of its parent operator, its parameters are bounded to the values defined in the parent, if any).

```
action Pick(Robot r, Item i, Location l){
  :decomposition{
     PickWithGripper(r, r.left, i, l); };
  :decomposition{
     PickWithGripper(r, r.right, i, l);};};

action PickWithGripper
    (Robot r, Gripper g, Item i, Location l){
  maxDuration := 10;
  [start, end]{ g.empty == true :-> false;
                r.myLocation == l;
                i.myLocation == l :-> g;
              }; };
```

The power of hierarchical decomposition (as in HTN) lies in being able to encode expert level knowledge into the domain by making explicit the various possible decompositions of a task, instead of relying on a search mechanism to find these possible decompositions from basic action models. Of course, this also depends of the skill of the programmer, yet, our experiences with various formalisms indicate that HTN are better suited for planning in robotics. While the refinement of the action can be as simple as the action Pick we have introduced, one can imagine going further, e.g., Transport → TransportByRobot → Move, Pick, Move, Drop, or even PickWithGripper decomposed with motion planning techniques.

IV. FAPE INTERNAL STRUCTURES

FAPE planning and acting components rely on several key data structures that provide efficient handling of state variable evolutions, constraints and plans. In the following subsections we present the timelines, temporal network, constraint network and task network.

A. Timelines and Chronicles

To capture the information on the evolution of state variables over time, we use timelines with the same semantics as used in [24, Sec. 14.3]. A timeline is a set of temporal statements related to a unique state variable. A timeline Φ is a tuple (x, F, C) where x is a parameterized state variable, F is a set of temporal statements and C is a set of temporal constraints and binding constraints over the time points and object variables in F.

Two essential properties of timelines need to be handled: *consistency* and *causal support*. A timeline (x, F, C) is consistent when the constraints in C are consistent and

when no pair of assertions in F are possibly conflicting. Intuitively, two assertions are conflicting when they specify two possibly distinct values of x at the same time. This may happen when the two assertions are allowed to overlap in time with possibly incompatible values (with straightforward cases related to conflicts between persistence, events and mixed conflicts). Additional temporal or binding constraints, called *separation constraints*, may be needed in C to remove possible conflicts and make the timeline consistent.

A timeline (x, F, C) supports an assertion α when there is an assertion $\beta \in F$ that can be used as a *causal support* for α and when α can be added to the timeline consistently. More precisely, when α asserts a persistent value v for x or a change of value from v to v' starting at time t, we require β to establish a value w at a time t' such that $t' < t$ and $w = v$ and that this value can persist consistently until t. Here also additional constraints, i.e., $t' < t$ and $w = v$ and separation constraints, can be needed to make the timeline support α.

We define a *chronicle* as a tuple (T, C) where T is a set of timelines and C is a set of temporal and binding constraints. We say that a chronicle is consistent if each timeline in T is consistent, and the union of constraints in the timelines of T with those of C is consistent.

B. Temporal Constraint Network

Dealing with explicit time implies taking into account temporal constraints between identified time points of the planning process (e.g. the beginning of an action or the occurrence of a contingent event). Repairing plans further requires the ability of removing constraints to reflect real events that might be contradictory with our previous knowledge.

Our temporal network manager is based on the *Simple Temporal Problem* introduced by [25]. It is encoded as a directed weighted graph in which an edge from t_i to t_j with weight w_{ij} represents the constraint $t_j - t_i \leq w_{ij}$.

Consistency is checked on constraint addition by detecting negative cycles in the graph which is a sufficient and necessary condition of STN consistency. This step is performed by running, upon constraint addition or removal, an incremental Bellman-Ford algorithm as presented in [26]. This allows us to efficiently check STN consistency while keeping a sparse network containing only constraints that were explicitly stated, thus allowing us to easily remove constraints from the network.

In general, temporal plans include uncontrollable durations (e.g. the time for the robot to go from the kitchen to the living room may vary between 1 and 2 minutes). These durations should not be squeezed by the planner temporal propagation and we must use an approach which guarantees the dynamic controllability (DC) of the plan. The EfficientIDC algorithm [27] is used to check the dynamic controllability of a solution plan. We intend to further integrate it to incrementally enforce DC while planning.

C. Binding Constraint Network

While planning, new object variables are created when a new lifted action is inserted into a plan: every parameter of the action gives birth to a new typed object variable. These variables appear either as parameters of state variables or as values of state variables. Separation and causal support constraints on these object variables are managed as a binding constraint network. This constraint network is consistent iff there exists an instantiation of variables such that all equality and non-equality constraints are satisfied. We use AC-3 to maintain the arc-consistency, which is a well-known trade-off between earliness of the failures and computational performance.

D. Task Network

A task network is a forest of partially instantiated operators, where the branches represent the conjunction of actions into which an action decomposes. We say that the network is decomposed if all leaves are primitives. A single tree corresponds to the decomposition of a single root action. New trees can be added in the task network when new actions are added in the current plan. This mechanism combines HTN techniques with Plan-Space techniques.

The FAPE planner does not support recursive decomposition methods. Recursive methods raise termination and completeness issues, in addition to complexity issues.

V. Planning

The planning component of FAPE relies on two mechanisms: task decomposition, as in HTN, and resolver insertion, as in Plan-Space Planning (PSP). A planning problem is defined as a triple (V, O, s_{init}), where V is a set of state variables, O is a set of operators and s_{init} is the initial search node. Since we are in plan-space, we do not define a goal state but an initial search node, which is specified with *(i)* a set of initial statements, giving the initial values of state variable and the expected events and persistences, and *(ii)* the plan objectives. The statements in *(i)* are considered to be causally supported. Those of *(ii)* need to be supported by the plan to be built. They are given as a set of goal statements, temporally qualified with the `end` time point, and/or the task to perform (as in HTN), called here *the seed action*, e.g.,

```
action Seed(){
  :decomposition{
    Transport(anyRobot_, I1, anywhere_, L2);
  };
};
[end] I1.myLocation == L3;
```

In this example, the objective is to achieve the `Transport` task and, at the end to have item `I1` at location

L3. Note that this specification of the objectives through assertions and a seed action can be redundant, or even inconsistent. It is up to the domain designer to make sure that the domain and problem specification are consistent. While it may be useful to specify goals for one state variable through goal statements and use the seed actions for another state variable, we discourage the domain designer to use both for a single state variable, where the semantics is not clear — there is no syntactical construct to temporarily relate seed actions with goal statements.

The planner search node is a tuple (Φ, T), where Φ is a chronicle and T is the task network. We say that a search node is consistent if both Φ and T are consistent. Planning proceeds by identifying flaws in a search node and iteratively applying resolvers until a search node is reached that is consistent and with no flaws.

A. Flaws and Resolvers

Planning proceeds as in PSP, by addressing the flaws of a current search node. A search node $n = (\Phi, T)$ may contain the following flaws:

Open goal. An open goal is any statement in Φ that does not have a causal support.

Undecomposed actions. An undecomposed action is a non primitive action appearing as a leaf in the task network; it needs to be decomposed.

Threats are dealt with incrementally through separation constraints, that maintain each timeline consistent, and through causal support constraints.

The resolvers for an undecomposed action flaw are the existing methods specified for its decomposition. Applying a method as a resolver consists in expanding the action node with its specified decomposition with the temporal and binding constraints inherited by the decomposed action.

An open goal α may have two types of resolvers:

- any assertion $\beta \in \Phi$ that can be used to support α; applying such a resolver consists of adding the causal support constraints and the separation constraints required to have α supported.
- an action that provides an assertion β that can be used to support α. Applying such a resolver requires adding the action together with the support constraints and separation constraints.

The newly added action may in turn bring new unsupported statements.

Notice that there are two ways of inserting an action into a partial plan: through a decomposition, or directly by adding an action as a provider of a support for an open goal. The same action may be added as a provider at some point and appear through a decomposition at a later point. A possible redundancy may result from this. The FAPE planner does not currently implement a merging operation over the task network. This will be the object of future work.

B. Search

Given that a search node π is a solution if it is consistent and with no flaws, search proceeds by identifying flaws of π (i.e. its open goals and undecomposed actions) and applying a resolver for one selected flaw while maintaining the resulting search node consistent.

For the purposes of demonstration, we stick, for the moment, to the PSP recursive nondeterministic schema [24].

Algorithm 1 Main PSP Algorithm

```
function PSP(π)
    flaws ← OpenGoals(π) ∪ UndecomposedLeaves(π)
    if flaws = ∅ then return (π)
    end if
    select any flaw φ ∈ flaws
    resolvers ← Resolve(φ,π)
    if resolvers = ∅ then return failure
    end if
    nondeterministically choose a resolver ρ ∈ resolvers
    π ← Apply(ρ,π)
    return PSP(π)
end function
```

The PSP algorithm (See Algorithm 1) at each step of the recursion deterministically chooses a flaw to resolve (selection is done with the simple min-domain heuristic) and then chooses nondeterministically the resolver as follows:

- if the application of a resolver returns a failure then another recursion with a different resolver is performed
- if all resolvers were tried unsuccessfully then a failure is returned to the previous choice point

We can as well modify the non-determinism to reach the optimal solution with regard to some objective function. In practice, our current implementation uses a best-first search strategy, with the number of open goals as a distance evaluation to a consistent search node.

VI. Acting

In a system like FAPE, Acting and Planning are integrated. Acting, is more complex than just Execution of platform commands. Often, the actions in the plan are still at a too high level to be directly executed on the platform. From our point of view, we consider in FAPE the basic functions relevant to Acting, and introduced in [3], to include: refinement, instantiation, time management and coordination, non determinism and uncertainty, plan repair. In the current FAPE implementation, they are all but one (non determinism and uncertainty) handled.

Acting refines online an action into a collection of closed-loop functions, referred to here as skills; a skill processes a sequence (or a continuous stream) of stimulus input from sensors and output to actuators, in order to trigger motor forces and control the correct achievement of chosen actions.

We currently use PRS procedures to refine fully instantiated plan actions into motor commands, as well as to perceive the environment and inform the Planner of important changes. PRS skills also provide some local action recoveries for situations where the procedure can handle an alternative way to perform the action (e.g. to consider an alternative grasping pose, or an alternative path to reach a particular location). For our PR2 implementation, the basic motor commands and perception are provided by ROS actions, nodes and also GenoM3 [28] modules. We plan to integrate other skill execution frameworks which can handle different type of acting representation (MDP, DBN, FSM, etc).

For dispatching, fully instantiated and scheduled actions are passed to the Acting component according to their starting time. The planner maintains a partially instantiated plan (only the necessary binding and temporal constraints are applied), which represents a set of valid plans (time and object variables are instantiated when needed). Actions selected for execution are found by taking the ones whose preconditions are met and whose start times fit in an execution window (e.g. we want to get actions that can be started in the next x seconds). The temporal variables and constraints of those actions are instantiated and the actions are then returned. Further calls instantiate more and more actions while the future instantiation of the actions not yet scheduled is kept as open as possible. Once an action is finished, acting reports the actual end date of its execution. This exact date is then integrated in the current plan, and the temporal propagation, as described in the "Temporal Constraint Network" section, is performed. The action fails if it take less or more time than planned. Such temporal failure is reported to the planner which can then attempt to repair the plan accordingly. Note that in the general case, the acting component can also inform the planner that an action is taking too long, yet, wait for the planner to plan and send an abort action as a result of this problem (the acting component does not take the freedom to abort an action which is running late). An action can also fail because the skill failed (e.g. despite multiple attempts, the robot cannot grasp an object, or reach a location). The acting component then retrieves a description of the changes of the world that occurred and send it to the planner which integrate these "unexpected" state variable transitions in its plan.

Considering we have a plan and one of the actions in the plan fails during the execution, the plan-repair consists of the following steps:

1) Removing the action from the task network.
2) Removing all the statements introduced by the failed action from the timelines which shall generate new flaws.
3) Running the PSP algorithm until the flaws are resolved.

Our repair approach is limited to the removal of just the one failing action, we do not consider cascades of other potential failures. There certainly are cases when the repair does not find a plan and we need to replan, making the repairing a wasted effort. However, most of the time repairing the plan is much faster than replanning and the overall benefit for the responsiveness of a real-time system is significant, as we shall show in the following section.

VII. Experimental Setup and Results

FAPE is designed to be used as an embedded system. The current implementation has been experimented on a PR2 to plan service robot type of tasks. For instance, the PR2 moves around in an apartment and detects objects which are misplaced (e.g. a video tape in the bedroom, or a book on the dining table) picks them up and stores them away in their proper location (respectively by the TV set, and in the bookshelf).

In the current setup, we rely on some of PR2 basic capabilities[3]: navigating in a household like environment; recognizing objects; picking them up and putting them down. Actions are dispatched just in time to PRS which executes them when their start time has arrived. PRS monitors the proper execution and reports success or failure. In case of failure, the proper relevant state variable values are sent back to the planner as an ANML block which needs to be introduced into the current plan, leading to repair or replan. The implicit behavior of the actor is always to repair the plan, while the replanning is only called once the repair fails completely.

//NOTE FROM ARTHUR: Integrate new experiments//

To thoroughly test the complete system, we also wrote a simple simulator in PRS which randomly simulates action failures (navigation does not reach the final destination, pickup misses or drops the object, etc), and out of bound time execution (the action takes less or more than the planned duration interval) on a domain with multiple robots. We show that, independently of the number of search nodes generated while producing the initial plan, the FAPE planning component is able to find a trivial repair in a matter of milliseconds in the vast majority of cases. Table 1 reports on the number of plan space search nodes explored while repairing a plan compared to the ones explored while generating the initial plan. This result is particularly important as it shows that repairing the plan instead of replanning often saves significant computational effort, which is even more crucial in embedded planning, where the responsiveness of the planner often directly projects into the system performance. Furthermore, repairing the plan allows entities that are not affected by the failure (such as other robots) to keep acting while the plan is being repaired.

[3]http://wiki.ros.org/pr2_navigation
http://wiki.ros.org/pr2_tabletop_manipulation_apps

	Number of instances
$n_{repair} \leq 15$	102 (82.2%)
$15 < n_{repair} < n_{planning}$	7 (5.7%)
$n_{repair} \geq n_{planning}$	6 (4.8%)
repair failed	9 (7.3%)

Figure 1. Number of search nodes generated while repairing the plan (n_{repair}) with respect to the number of nodes generated while producing the initial plan ($n_{planning}$). Over the experiment, $n_{planning}$ has an average value of 319.

VIII. Future work

To the best of our knowledge, FAPE is the first system including a planner supporting most ANML features – combination of HTN planning and explicit time representation; and plan-space planning. It integrates acting together with planning and both decisional functionalities rely on the same internal representation. Each functionality is critical with regard to the efficiency of the whole system and as such it deserves our attention in future development. The planner shall benefit significantly from a stronger heuristic as well as the addition of specific and domain dependent planners (e.g. motion or manipulation planners). Meanwhile the Acting system will provide other acting frameworks than the PRS refinement procedures used for now (e.g. MDP policies [29], DBN [30], etc). We also plan to implement and compare new models of interleaving planning and acting, where we would concentrate on the decision making between alternative action refinement, repairing and replanning — how to recognize and predict when one is preferred to the other. Similarly, we plan to investigate the inclusion of delayed methods decomposition. The planner, instead of expanding all tasks down to the action leaves may delay and delegate some designated decomposition to the acting component.

While we currently support multi-agent planning (there can be any number of robots in the system that perform their actions in parallel), we are particularly interested in extending the system towards multi-agent planning where actions of some of the entities are not controllable, which shall allow us to reason and plan human-robot interactions.

IX. Conclusion

We have introduced FAPE, a new framework that integrates Planning and Acting to be embedded in autonomous real-time system such as robots. Using ANML as an input planning langage, we have the expressivity to plan for complex temporal plans with requirements on concurrent actions in dynamic and changing environments. We also allow the user to improve and fine-tune the efficiency of the system by introducing task decompositions which can efficiently prune the search in plan space. We have experimented both Planning and Acting in simulation with large problems and on a PR2 robot which performs service robot type of activities. The development of FAPE continues as a multi-institutional effort to provide a planning/acting system, which we would like to see positioned as a system capturing the state-of-the-art of planning, integrating domain specific planners while maintaining the expressiveness of ANML and ease of integration with different type of acting components.

Acknowledgment

This work has been conducted within the EU SAPHARI project (http://www.saphari.eu/) funded by the E.C. Division FP7-IST under Contract ICT-287513.

References

[1] M. Fox and D. Long, "PDDL 2.1 : An Extension to PDDL for Expressing Temporal Planning Domains," *Technical Report, University of Durham, UK*, 2002.

[2] D. E. Smith, J. Frank, and W. Cushing, "The ANML Language," *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2008.

[3] F. Ingrand and M. Ghallab, "Robotics and Artificial Intelligence: a Perspective on Deliberation Functions," *AI Communications*, vol. 27, pp. 63–80, Nov. 2013.

[4] A. J. Coles, A. Coles, M. Fox, and D. Long, "COLIN: Planning with Continuous Linear Numeric Change," *J. Artif. Intell. Res. (JAIR)*, vol. 44, pp. 1–96, 2012.

[5] M. Ghallab and H. Laruelle, "Representation and Control in IxTeT, a Temporal Planner," in *International Conference on AI Planning Systems*, 1994, pp. 61–67.

[6] A. K. Jónsson, P. H. Morris, N. Muscettola, K. Rajan, and B. Smith, "Planning in Interplanetary Space: Theory and Practice," in *International Conference on AI Planning Systems*, 2000.

[7] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," in *International Symposium on Artificial Intelligence, Robotics and Automation for Space*, 1999.

[8] J. Frank and A. K. Jónsson, "Constraint-Based Attribute and Interval Planning," *Constraints*, vol. 8, no. 4, 2003.

[9] N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt, "IDEA: Planning at the Core of Autonomous Reactive Agents," in *Proceedings of the AIPS Workshop on On-line Planning and Scheduling*, 2002.

[10] K. Rajan, F. Py, C. McGann, J. P. Ryan, T. O'Reilly, T. Maughan, and B. Roman, "Onboard Adaptive Control of AUVs using Automated Planning and Execution," in *International Symposium on Unmanned Untethered Submersible Technology (UUST)*, Durham, NH, August 2009.

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

[12] D. E. Wilkins, *Practical Planning*, ser. Extending the Classical AI Planning Paradigm. Morgan Kaufman, 1988.

[13] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN Planning System," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 379–404, 2003.

[14] L. Castillo, J. Fdez-Olivares, O. García-Pérez, and F. Palao, "Efficiently handling temporal knowledge in an HTN planner," *Sixteenth international conference on automated planning and scheduling, ICAPS*, 2006.

[15] R. J. Firby, "An investigation into reactive planning in complex domains," in *Proceedings of the sixth National conference on Artificial intelligence*. Seattle, WA, 1987, pp. 202–206.

[16] F. Ingrand, R. Chatilla, R. Alami, and F. Robert, "PRS: a high level supervision and control language for autonomous mobile robots," in *IEEE International Conference on Robotics and Automation*, 1996, pp. 43–49.

[17] D. E. Wilkins and K. L. Myers, "A common knowledge representation for plan generation and reactive execution," *Journal of Logic and Computation*, vol. 5, no. 6, pp. 731–761, 1995.

[18] R. Simmons, "Concurrent planning and execution for autonomous robots," *Control Systems, IEEE*, vol. 12, no. 1, pp. 46–50, 1992.

[19] M. Beetz and D. McDermott, "Improving Robot Plans During Their Execution," in *International Conference on AI Planning Systems*, 1994.

[20] S. Lemai-Chenevier and F. Ingrand, "Interleaving Temporal Planning and Execution in Robotics Domains," in *Proceedings of the National Conference on Artificial Intelligence*, 2004.

[21] M. Di Rocco, F. Pecora, and A. Saffiotti, "When robots are late: Configuration planning for multiple robots with dynamic goals," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*. IEEE, 2013, pp. 9515–5922.

[22] C. Bäckström and B. Nebel, "Complexity Results for SAS+ Planning," *Computational Intelligence*, vol. 11, pp. 625–656, 1995.

[23] M. Helmert, "Concise finite-domain representations for PDDL planning tasks," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 503–535, 2009.

[24] M. Ghallab, D. S. Nau, and P. Traverso, *Automated Planning: Theory and Practice.* Morgann Kaufmann, Oct. 2004.

[25] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, no. 1-3, pp. 61–95, 1991.

[26] A. Cesta and A. Oddi, "Gaining efficiency and flexibility in the simple temporal problem," *Temporal Representation and Reasoning, International Syposium on*, vol. 0, p. 45, 1996.

[27] M. Nilsson, J. Kvarnström, and P. Doherty, "Incremental dynamic controllability revisited," in *ICAPS*, D. Borrajo, S. Kambhampati, A. Oddi, and S. Fratini, Eds. AAAI, 2013.

[28] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand, "GenoM3: Building middleware-independent robotic components," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 4627–4632.

[29] B. Morisset and M. Ghallab, "Learning how to combine sensory-motor functions into a robust behavior," *Artificial Intelligence*, vol. 172, no. 4-5, pp. 392–412, 2008.

[30] G. Infantes, M. Ghallab, and F. Ingrand, "Learning the behavior model of a robot," *Autonomous Robots Journal*, pp. 1–21, Oct. 2010.