Planning and Acting with Temporal and Hierarchical Decomposition Models

Filip Dvořák, Roman Barták
Faculty of Mathematics and Physics
Charles University in Prague
Prague, Czech Republic
Email: filip@dvorak.fr, roman.bartak@mff.cuni.cz

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

***Abstract*—This paper reports on FAPE (Flexible Acting and Planning Environment), a framework integrating acting and planning on the basis of the ANML modeling language. ANML is a recent proposal motivated by combining the expressiveness of the timeline representation with decomposition methods of hierarchical task networks (HTN). Our current focus is not efficient temporal planning per se, but the tight integration of acting and planning. This integration is addressed by: *(i)* extending HTN methods with the refinement of planned actions with *skills*, expressed in PRS, to map actions into low-level commands, *(ii)* interleaving the planning process with acting, the former performs plan repair and replanning, while the latter implements the skill-based 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.

***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: *(i)* extended state space representations, and *(ii)* 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 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 Planning Domain Description Language (PDDL) 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 the 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. However, the plan-space planners have not scaled up as well as state space planners or HTN planners.

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

The Action Notation Modeling Language (ANML) [1] is motivated mainly by blending the expressive timeline representation with the decomposition of HTN methods. This paper reports on FAPE and its planner, which is the first, to our knowledge, to implement ANML.[2] 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 [2] decomposition procedures,
- interleaving the planning process with acting, the former 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.

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

[2] A preliminary version of this work was presented in the PlanRob workshop of ICAPS 2014

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

FAPE includes an ANML planner, the first to support the combination of features of least-commitment plan-space planning, explicit time maintained by a sparse simple temporal network, resource reasoning, 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 using 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 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], Europa [7] and APSI [8], 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). 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 numerous planners such as SHOP2 [9] or SIADEX [10]. The latter integrates time to HTN planning without handling concurrency. PANDA [11] integrates HTN planning with Partial-Order Causal-Links into a single framework. It naturally supports concurrency but has a limited handling of time.

ANML [1] extends the languages used in Europa and ASPEN with constructs from PDDL together with HTN task decomposition methods. We are aware of ongoing developments on the basis of this language[3], but to our knowledge, FAPE is the first system implementing an ANML planner supporting both task decomposition and temporal planning.

Several systems integrate planning and acting, e.g., with procedure-based approaches to refine actions into lower level commands with systems such as RAP [12], but without time representation. IxTeT-Exec [13] and "Configuration Planner" [14] are examples similar to our approach for the timeline planning, but without decomposition methods.

III. Representation and ANML

FAPE relies on the ANML knowledge representation. ANML is a rich language allowing the user to introduce planning models in a multitude of ways.

FAPE relies on parametrized state variables, with typed object variables as parameters, and on timelines over these state variables. FAPE supports typing and structured types with inheritance (denoted as <):

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

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

The statements represent *events* and *persistence conditions*. A temporal annotation is either a time point or an interval defined by two time points. These can be relative to a context (e.g. an operator, or a planning problem), such as `start`, or absolute time points, such as `[50,70]`. According to the definitions given in [15], we define a temporal statement to be an assertion over the evolution of a parameterized state variable. We consider two types 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

[3]In particular at NASA Ames Research Center

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 undefined in $]t1, t2[$ (it cannot be unified with any value). 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 action is a partially instantiated operator that may have several possible decompositions into a partially ordered set of actions. Formally, a planing operator is a tuple $(name, maxDuration, P, E, D)$, where $name$ is the unique name of the operator, $maxDuration$ 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 action, its parameters are bound 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 on the skill of the domain designer, yet, our experiences with various formalisms indicate that HTN is well 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.

Resources are represented as state variables with a numeric domain that can be infinite and either discrete or continuous. The following support events change the value of the state variable relatively:

- *consume(x)* reduces the value by *x*,
- *produce(x)* increases the value by *x*,
- *use(x)* is a composition of *consume(x)*, followed by *produce(x)*, and
- *lend(x)* is symmetrical to *use(x)*, first producing, then consuming.

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 [15, 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.

When a timeline represents a resource state variable, the concept of causal support disappears, while the consistency of such timeline is determined according to the resource reasoning technique tailored for the particular type of the resource.

B. Resource Reasoning

A resource is an entity that supports or enables the execution of activities. Resources are, e.g. a single machine that performs only a single task at a time, or the dynamically changing level of a battery in a satellite. We limit resources to piece-wise constant evolution, and distinguish four classes:

- *Consumable* and *producible*: represent respectively a resource that is only be consumed (e.g. limited supply) and a resource that is only produced (e.g. waste product).
- *Discrete resources* represent an entity, whose capacity is used or lent for some time (e.g. electric outlet, whose output is limited by the total consumption of connected devices that *use* the outlet).
- *Reservoirs* are resources that support both consumptions and productions, e.g. a fuel in a car.

The reasoning for consumable and producible resources is simple. Consistency of a discrete resource is determined by the minimal critical sets as described in [16], and an inconsistence of a discrete resource can be resolved by adding appropriate precedence constraints (enforcing that one usage must precede another). Consistency of reservoirs is determined by *balance constraints* [17] that also enforce new temporal constraints and an inconsistence of a reservoir can be resolved by adding an action, if such action contains a resource event that leads to consistency (e.g. a production event for an overconsuming resource).

C. Temporal Constraint Network

The temporal network manager is based on the *Simple Temporal Problem*. Consistency is checked on constraint addition by detecting negative cycles in the distance 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 [18]. 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. We use the EfficientIDC algorithm [19] to check the dynamic controllability of a solution plan. We intend to further integrate it to incrementally enforce DC while planning.

D. Binding Constraint Network

While planning, new object variables are created when an action is inserted into a plan: every parameter of the action introduces a new typed object variable (standardization apart). These variables appear either as parameters of state variables or as values of state variables (we can notice that in the *PickWithGripper* action). 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.

E. Task Network

A task network is a set of action nodes. A single action node $n = (\alpha, D_\alpha)$, where D_α is a set of actions nodes, is considered a leaf, if $D_\alpha = \emptyset$. A node is considered to be terminal, if it cannot be decomposed. We say that the network is decomposed if all leaves are terminal. Having a task network, we can both insert new root nodes to the network, corresponding to an action insertion in plan-space planning, or we can decompose existing leaves, corresponding to the HTN planning.

The FAPE planner does not support recursive decomposition methods. Recursive methods raise termination, completeness and 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 temporally qualified goal statements and/or the tasks 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 model 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. 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.

Inconsistent resource. An inconsistent resource can be resolved by inserting an action, which contains a statement on the resource that contributes to making it consistent, or by adding temporal constraints. Inconsistent discrete resources cannot be resolved by action insertion, while reservoirs can.

Undecomposed action. An undecomposed action is a non terminal leaf in the task network; such leaf needs to be decomposed. The resolvers for an undecomposed action flaw are all of its possible decompositions. Performing a decomposition as a resolver consists of expanding the action node with the actions specified by the chosen decomposition and adding corresponding binding and temporal constraints.

Threat. occurs when there are two timelines they may represent the same state variable and may collide in time. Threats are resolved by introducing object separation constraints (breaking the possibility that both timelines represent a single state variable), or adding temporal precedence constraints, such that the timelines do not collide.

Notice that an action can be introduced into the plan either by a decomposition or as a resolver of a flaw on a timeline. Such integrated approach allows, for example, to specify only the skeletal plan through decompositions (as opposed to HTN) and leave all the corner cases to be planned as in PSP. Opposed to PSP, the decompositions provide invaluable guidance for shaping the search space.

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 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 [15].

Algorithm 1 Main PSP Algorithm

```
function PSP(π)
    flaws ← FLAWS(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 (the choice of the flaw does not impact completeness) and then chooses nondeterministically the resolver as follows:

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

FAPE uses a multitude of heuristics, some known in plan-space planning, such as least-flaw ratio, flaw and action counting, least-committing flaw [11], and some adapted from state-space planning, such as LM-Cut [20]. Their evaluation and detailed description is beyond the scope of this paper.

VI. Acting

In FAPE, Acting and Planning are integrated and interleaved. Acting, is more complex than just Execution of platform commands. The planned actions are 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. The basic motor commands and perception are provided by ROS actions, nodes and also GenoM3 modules. We plan to integrate other skill execution frameworks that can handle different types of acting representation, such as Markov Decision Processes, Dynamic Bayesian Network and Finite State Machines.

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 time of its execution. This exact date is then integrated and propagated in the current plan. The action fails if it takes 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: *(i)* removing the action from the task network; *(ii)* removing all the statements introduced by the failed action from the timelines which shall generate new flaws; *(iii)* running the PSP algorithm for the broken plan 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).[4]

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, using basic motor commands and perceptions provided by ROS[5]. 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.

In dynamic environment new events may appear and actions may fail when carrying out a plan. We have used a simple simulator in PRS that fails an action with a 50% probability. Having a 124 simulated situations of an action failure, Table 1 reports on the number of search nodes explored when FAPE was either repairing a plan or planning from scratch. The plans consisted of few dozens actions.

	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.

The importance of this results lies especially in embedded and real-time planning, where the responsiveness of the planner plays a major role. 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.

VIII. Future work

FAPE is the first system including a planner supporting most ANML features – HTN decompositions, resource rea-

[4]We rely on some of PR2 basic capabilities : http://wiki.ros.org/pr2_navigation
http://wiki.ros.org/pr2_tabletop_manipulation_apps

[5]http://wiki.ros.org/Robots/PR2

soning, explicit time and plan space planning. It integrates acting together with planning, where both rely on the same internal representation for planning, repair, and dispatching. Each functionality is critical with regard to the efficiency of the whole system and as such it deserves our attention in future development. There is always space for improving the search heuristic and we plan to integrate with domain specific planners such as motion and manipulation planners. Meanwhile, the Acting system will provide other skill execution frameworks than the PRS refinement procedures used for now (e.g. MDP policies, DBN [21], 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 refinements, 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.

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 language, we have the expressivity to plan for complex temporal plans with requirements on concurrent actions in dynamic 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 with both Planning and Acting in 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 and 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 types 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 and partially supported by the Czech Science Foundation under the project No. P103/10/1287.

References

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

[2] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] B. Schattenberg, "Hybrid planning and scheduling," Ph.D. dissertation, Ulm University, Institute of Artificial Intelligence, 2009.

[12] 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.

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

[14] 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.

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

[16] P. Laborie and M. Ghallab, "Planning with sharable resource constraints," in *International Joint Conference on Artificial intelligence (IJCAI)*, 1995, pp. 1643–1649.

[17] P. Laborie, "Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results," *Artificial Intelligence*, vol. 143, no. 2, pp. 151–188, 2003.

[18] A. Cesta and A. Oddi, "Gaining Efficiency and Flexibility in the Simple Temporal Problem," in *TIME*, 1996, pp. 45–50.

[19] M. Nilsson, J. Kvarnström, and P. Doherty, "EfficientIDC: A faster incremental dynamic controllability algorithm," in *International Conference on Automated Planning and Scheduling*, 2014.

[20] M. Helmert and C. Domshlak, "Landmarks, critical paths and abstractions: What's the difference anyway?" in *ICAPS*. AAAI, 2009.

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