Reactive Planning as a Motivational Source in a Behavior-Based Architecture

Abstract— **Behavior-based architectures use behaviors as building blocks for decision-making and action execution processes. Behaviors are distributed and evaluated in parallel for the control of the robot, taking real-time inputs from sensory data and sending real-time commands to effectors. No centralized components exist in these architectures, each module carrying out its own strategy independently, making an overall behavior emerge from the interaction between the concurrently executed modules and the environment. In this paper, we discuss the use of a reactive Hierarchical Task Network (HTN) planner in a behavior-based robot architecture. The planner in this architecture is not a central component on which everything else relies on, but acts as one of the motivational modules recommending tasks to be executed and influencing the selection and configuration of behaviors. The planning module allows the behavior-based architecture to deal with tasks with priorities, flexible temporal constraints and online planning using a simple but very effective reactive planning strategy. We demonstrate our approach in the context of making a robot attend a conference.**

I. Introduction

Situatedness is required for robots dealing with complex, challenging, often dynamically changing environments with limited predictability and stability. Behavior-based systems [1] are typically designed so as to take advantage of the richness of the interaction dynamics, by exploiting the properties of situatedness. Planning, on the other hand, requires the existence of an internal, symbolic representation of the world, which allows the robot to look ahead into the future and predict the outcomes of possible actions in various states to generate plans. The internal model, thus, must be kept accurate and up-to-date. However, being situated in a noisy and dynamic world usually makes this impossible [2], [3]. Hybrid architectures attempt to combine both influences on the decision-making capabilities for a robot. Coupling between situated control (fast time-scale and directly influenced by external sensory data) and planning capabilities (longer time-scale using abstract and symbolic representations of the world) is done through an intermediate component, trying to reconcile the different representations and operational modes. The construction of this intermediate component is one of the greatest challenges in designing hybrid architectures. Through this intermediate module (also known as the Coordination layer), behaviors can be reconfigured based on what can be anticipated by the Planning layer, which can in turn be influenced by how the behaviors deal with situations occurring in the real world.

Most hybrid architectures use only a deliberative component at the Planning layer to influence the Behavior layer. However, surely other types of influences could be used to do so, instead of having to represent each of these influences through a general model of the world to be processed by a planning algorithm. In fact, behavior-based principles can be applied at this level too, having a set of distributed and interacting modules with no centralized world representation or focus of control. This way, the functionality of the layer configuring behaviors is neither a property of the robot or the environment in isolation, but rather a result of the interplay between them.

We believe that such capability is required for a robot operating in real life settings, reminiscent of the AAAI Robot Challenge. Robots in the AAAI Challenge attempt to behave like a human conference attendee, by autonomously registering to the conference, making presentations, performing volunteer duties such as delivering packages on behalf of people, guiding lost people, maintaining a sufficient level of their battery energy and navigating to various places in order to accomplish their tasks. A robot in such setting faces uncertainty at various levels, ranging from being able to localize itself accurately in the environment to being able to predict the time required for achieving its tasks. For instance, to accept volunteer duties in the AAAI Challenge, a robot must be able to determine the consequences of these duties on its existing tasks. It must also take into account the importance of the different tasks. Therefore, to make a decision about its next action (e.g., Where to go? What should I do first? Do I accept this duty?), the robot needs a task planner to determine the consequences of its local decisions on the overall behavior given the tasks it has to accomplish. However, it is not practical to model and account for all operating conditions in advance through a planning algorithm. For instance, some multiple goal-oriented tasks (e.g., looking for a recharging station opportunistically while carrying on a set of planned tasks) require different concurrent behavior that cannot all be determined by the planner. This is in accordance with our intuition of how humans operate in real life settings: many goal-driven reactions are result of reflexes or learned reaction rules, needs, intuition, etc., whereas others result from elaborate deliberative processes that reason explicitly about the course of actions such as a planner. We therefore adopted an approach that interleaves planning and execution (through behaviors), conducting a proof-of-concept study on how such coupling can be done by exploiting the same design principles of behavior-based system at a more abstract level.

II. Computational Architecture

The computational architecture that we use, shown in Figure 1, is composed of three principal components:

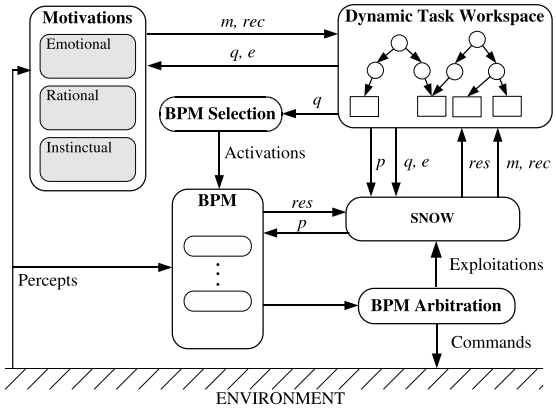

Fig. 1. Computational architecture.

1) Behavior-producing modules (BPMs) define how particular percepts and conditions influence the control of the robot's actuators. The actual use of a BPM is determined by an arbitration scheme realized through BPM Arbitration and the BPM's activation conditions, as derived by the BPM Selection module.
2) Motivational sources (or Motivations, serving to propel an agent in a certain direction) recommend the use or the inhibition of tasks to be accomplished by the robot. Motivational sources are categorized as either instinctual, rational or emotional. This is similar to considering that the human mind is a "committee of the minds" with instinctual, rational, emotional, and alike minds competing and interacting [4]. Instinctual motivations provide basic operation of the robot using simple rules. Rational motivations are more related to cognitive processes, such as navigation and planning. Emotional motivations monitor conflictual or transitional situations between tasks.
3) Dynamic Task Workspace (DTW) organizes tasks in a tree-like structure according to their interdependences, from high-level/abstract tasks (e.g., deliver message), to primitive/BPM-related tasks (e.g., avoid obstacles). Through the DTW, motivations exchange information asynchronously on how to activate, configure and monitor BPMs. Motivations can add and modify tasks by submitting modification requests (*m*), queries (*q*) or subscribe to events (*e*) regarding the task's status. The System Know-How (SNOW) module acts as an adapter between BPMs and DTW, making it is possible to decouple task representation contained in the DTW from BPMs.

Only instinctual and rational motivations are considered in this study, with rational motivations having greater priority over instinctual ones in case of conflicts. One instinctual module selects one high-level task when none has yet been prioritized (e.g. recommending to do entertainment when robot has nothing to do). Another makes the robot move safely in the world while monitoring its energy level (recommending the activation of the recharge behavior making the robot stop near an electric outlet). For our work, two rational modules are of interest: a task planner that determines which primitive tasks and which sequence of these tasks are necessary to accomplish high-level tasks under temporal constraints and the robot's capabilities (as defined by BPMs), and a navigator that determines the path to a specific location according to tasks posted in the DTW. The task planner is invoked online to account for new arriving tasks from DTW or unpredicted changes in the environment that make a previously generated plan inappropriate. Tasks may be submitted by others motivational sources like graphical user interface (robot's touch screen) or audio recognition subsystem. External events include task completion, task failure and every new event related to new information collected from environment (e.g. a new location on the map).

A. Task Planning Algorithm

Our planning algorithm is based on Hierarchical Task Networks (HTN) like SHOP2 planner [5]. As in SHOP2 we specify a planning domain by describing the robot primitive behaviors in terms of template methods for recursively decomposing high-level tasks down to primitive tasks, which are atomic actions. For instance, we can specify that the task of making a presentation at location px is from time t_1 to time t_2, and that it can be decomposed into two simpler subtasks of going to px and presenting at time t_1. Decompositions of tasks into simpler ones are given with preconditions under which the decompositions are logically sound and time constraints for ensuring its consistency.

The planning algorithm consists in searching through the space of tasks and world states as follows. Starting from a given set of initial tasks and a current state describing the robot state and environment situation, the planner explores a space of nodes where each node represents: the current list of subtasks, the current robot state and environment situation, and the current plan (initially empty). A current node is expanded into successors by decomposing one of the current task into simpler ones (using the specified task decomposition method) or by validating a current primitive task against the current state (this is done by checking its preconditions and updating the current state using the primitive task's effects) and adding it to the current plan. A solution is found when reaching a node that only contains primitive tasks. On a node, there can be different ways of decomposing a task and different orders in which to decompose them. Backtracking is invoked to consider the different alternatives until obtaining a solution. Planning problems similar to those for conference-robot domains are known to be NP-Complete, so worse case scenarios may involve an exponential blow up. Nevertheless, by carefully engineering the decomposition methods to convey some search control strategy, it is possible to limit this state explosion [5].

B. Task Planning with Temporal Constraints

SHOP2 does not consider temporal constraints [5]. To implement this, we modified SHOP2 planning algorithm by:

(a) adding a *current-time* variable into the representation of world states during search; (b) allowing time constraints in the specification of task decomposition methods and primitive tasks; (c) allowing the specification of conditional update of effects in primitive tasks based on this variable. As is the case with similar extensions of HTN planning with time constraints [6], [7], we can specify time constraints. To the best of our knowledge, our study is the first one to examine the applicability of such HTN extensions in real-world robot experiments. On the other hand, as explained below, we have additional extensions such as managing priorities among tasks, which are not possible in these other HTN extensions.

Beside improving the expressive power for tasks and task decomposition methods, time constraints also provide an additional mean for controlling search during the planning process. Indeed, the planner can use these temporal constraints to add partial orders on tasks; these temporal constraints can be propagated when a high-level task is decomposed into lower-level tasks, making it possible to reduce the search space and hence accelerate the plan generation process.

For instance, suppose that a robot has to present a poster in a conference between 10:00 and 12:00. Figure 2 illustrates a very simple decomposition tree of the *Show-Poster* task. Leaf nodes in the decomposition tree are primitive tasks (i.e., atomic actions) whereas the others are high-level tasks. The robot has one hour before the beginning of the poster session to install the poster and should remove the poster in the next hour after the end. Generic temporal constraints are added at each level by specifying maximum end time and minimum starting time of each task. A partial order $A < B$ is added when $A.maxend < B.minend$. Note that the actual specification of task decompositions does not give trees explicitly, but uses template decomposition methods as in SHOP2.

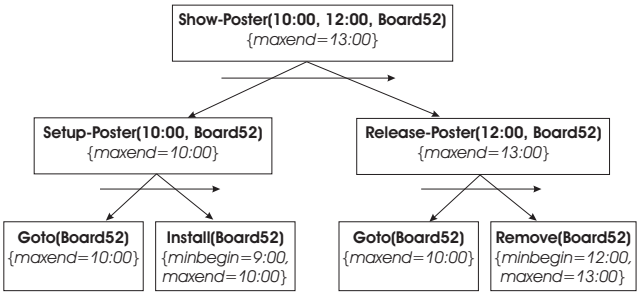

Fig. 2. *Show-Poster* task example.

When defining temporal intervals in the domain specification, care must be taken to establish a good compromise between safety and efficiency for task execution. Being too optimistic may cause plan failure because of lack of time. For instance, assuming that the robot can navigate at high speed from one location to the other will cause a plan to fail if unpredictable events slow down the robot. On the other hand, being too conservative may lead to no solution. In general, we specify temporal intervals using an average speed much lower than the real average speed of the robot.

C. Time Windowing

Each node explored by the search process is associated with a fixed time stamp (that is, the *current-time* variable), so that at the end we obtain a plan consisting of a sequence of actions each assigned with a time stamp indicating when it should be executed. Using a technique from SAPA [8], the returned sequence of actions is post-processed based on time constraints in the domain specification (i.e., constraints attached to task decomposition methods and primitive tasks) to derive time intervals within which actions can be executed without jeopardizing the correctness of the plan.

Figure 3 illustrates a Gantt chart representation of a simple plan. Rectangles on each task row represent the time when the action may be executed. Upper-left sub-rectangles show the time window in which tasks are executed as soon as possible. Bottom-right sub-rectangles represent the worst critical sections during which the tasks may be executed at the latest possible time.

Fig. 3. Initial plan for delivering a message and guarding.

D. Task Filtering and Priority Handling

In SHOP2, a list of initial tasks is considered as a conjunctive goal. If the planner fails to find a plan that achieves them all, it reports failure. In the context of the AAAI Robot Challenge as well as in many real life situations, we accept that the robot accomplishes as many tasks as possible, with some given preferences among tasks, if it is not possible to achieve them all.

Our extensions add task priority levels, so we can implement a robot mission as list of prioritized tasks. We handle them by iteratively running the planner to obtain a plan for a series of approximations of the mission, with decreasing level of accuracy. Specifically, initially we call the planner with the entire mission; if it fails to compute a plan within a deadline set empirically in the robot architecture (typically 5 seconds), a lowest priority task is removed from the mission and the planner is called with the remaining mission; and so on, until a solution plan is found; if the mission becomes empty before a solution is found, failure is returned (the entire mission is impossible).

E. On-line Task Planning

To reduce processing delays in the system, our planner is implemented as a library. The MARIE application adapter [9] that links the planner to the rest of the computational architecture loads once the planner library, its domain and world

representation (e.g., operators, preconditions and effects) at initialization. The planner remains in memory and does not have to be loaded each time it is invoked. A navigation table (providing the distances from each pairs of waypoints) also remains in memory and is dynamically updated during the mission. External events are accounted for by monitoring the execution of a plan and validating that each action is executed with the time intervals set in the plan. External events could be submitted by others motivational sources through DTW. For instance, the user interface can submit a new place on the map, and the navigator can change the estimated arrival time of the robot to destination.

The robot can receive task requests at any time during a mission. This requires the robot to update its plan even if it is not at a specified waypoint in its world representation. When generating a task plan, the duration of going from one waypoint to another is instantiated dynamically based on the current environment. That is the duration of a *Goto(X)* action is not taken from the navigation table but is rather estimated from the actual distance to the targeted waypoint *X*, as provided by the navigator motivational source and made available through the DTW.

III. Experimental Setup and Results

The robotic platform used for our experimentations is Spartacus, and is equipped with a laser range finder and one laptop computer (Pentium M 2.0 GHz). High-level programming is done using RobotFlow (a data-flow programming environment) and MARIE (a system integration framework used to link multiple software packages) [9]. For instance, MARIE integrates CARMEN, the Carnegie Mellon navigation toolkit [10], and pmap [1], two separate software packages for respectively laser-based autonomous navigation and laser-based mapping in 2D with high-quality occupancy grid maps. Since we use a separate module as the navigator (i.e., CARMEN path planner), the task planner only has to deal with high-level navigation tasks, and do not have to plan intermediate waypoints between two locations that are not directly connected.

The experimental scenarios are inspired from the AAAI Challenge in an office-like environment. We did experiments on one floor of our building, pictured on the map showed in Figure 4. The robot was given various missions involving the following steps: 1) Registering to the conference; 2) Making a presentation; 3) Assisting presentations; 4) Delivering messages (emulating delivering a package); 5) Guarding locations at specific time. Domain specifications for the planner consist of: 1) a world representation made of the list of waypoints ($p\#$), the navigation table and the robot's technical specifications (e.g., average traveling speed); 2) internal state representation (*current-position*, the nearest waypoint; *is-inside*, a boolean flag that indicates if the robot is in the range of its nearest waypoint; *is-registered*, a boolean flag indicating when the robot has received his badge; *found-places*, a list of waypoints visited by the robot); 3) primitive tasks ($Goto(px)$; $FindPlace(px)$; $Guard(px, time, duration)$; $GivePresentation(x, time, duration)$; $AskMessage()$; $GiveMessage()$; $ListenPresentation()$; $Register()$).

[1]http://robotics.usc.edu/~ahoward/pmap

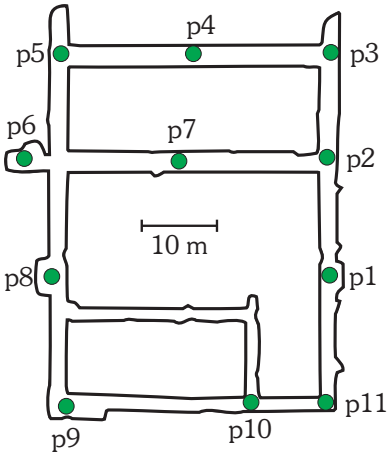

Fig. 4. Building floor layout with 11 waypoints.

The capabilities of the task planner in the behavior-based architecture are illustrated in the following situations: replanning because a temporal constraint is violated; replanning because an opportunity is detected; filtering out tasks; and planning with unknown waypoints.

A. Temporal Constraints Violation

We started the robot at $p1$ at time *t=0:00*, and gave the following mission: deliver a message $m1$ from $p5$ to $p7$; guard the place $p6$ at time *t=0:22:00*. For this mission, the planner generates an initial plan shown in Figure 3. The small rectangles for tasks 2 to 6 highlight tight time constraints for this plan. At the beginning of this trial, we voluntarily blocked the path of the robot, making it progress slower than what the planner expected. By analyzing the remaining distance updated by the navigator, the planner determined at time *t=0:03:50* that the initial plan became invalid and chose to generate a new one that changed the order of tasks to satisfy the time assignation for guarding $p6$.

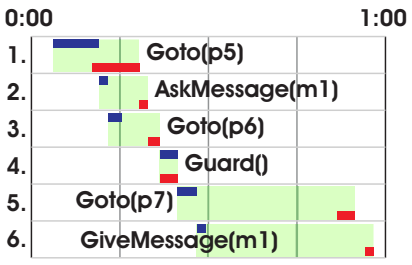

Fig. 5. Second plan to deliver a message and guard.

B. Opportunity Detection

The robot was given the following mission: deliver three messages ($m1$ from $p3$ to $p10$; $m2$ from $p5$ to $p2$; $m3$ from $p7$ to $p10$) and guard the room $p8$ for 10 minutes at time

t=0:35:00. Under the assumption that everything goes fine during execution, this mission can be done by starting with delivering all messages and then accomplish the guarding task. However, if something goes wrong, the robot will arrive too late at $p8$. In accordance with this, the planner assumed a conservative average traveling speed (0.12 m/s) for the robot, and generated the plan shown in Figure 6.

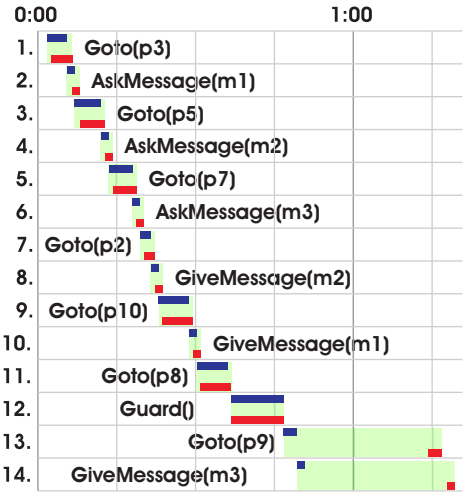

Fig. 6. Initial plan for delivering three messages and guarding.

The robot then started execution of the plan by going to $p3$. At step #2, the robot acquired $m1$. In the domain model, we set that such action may take up to 90 sec. It turned out that the traveling time to $p3$ was shorter than expected, and $m1$ was acquired after only 20 sec, resulting in approximately 2 minutes advance with the plan. When starting step #3 at time *t=0:05:10* the planner determined that the action $Goto(p5)$ could start faster than expected, and decided to reinvoke the planner. A new plan was generated, with task $Guard(p8)$ now placed at the end because it was possible to deliver all messages before the time for guarding $p8$.

C. Task Cancellation by Task Filtering

The robot was initially given the following mission: attend a presentation at $p4$ at time *t=0:10:00* for 10 min and deliver a message from $p2$ to $p9$. The planner generated the plan shown in Figure 7. At time *t=0:02:08*, the robot arrived

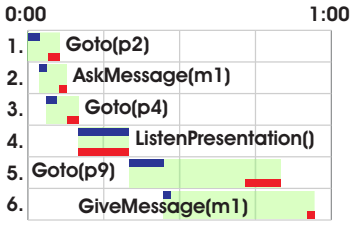

Fig. 7. Initial plan to attend a presentation at $p4$ and deliver a message.

at $p2$ and received the new task of guarding location $p10$ at time *t=0:12:00*. A new plan was therefore required. As described earlier, before generating a complete plan, our planner begins by validating the robot's mission. For the six task pairs, the planner verified if it is possible to generate a valid plan. For the task pair {*AttendPresentation(p4, 0:10:00, 0:10:00), GuardPlace(p10, 0:12:00, 0:05:00)*}, no possible plan is found. Intuitively, this is explained by the fact that it takes about 9 minutes for the robot to travel the distance separating $p4$ and $p10$. Since these two tasks are mutually exclusive, the planner generated a new plan after removing the lowest priority task, which was to attend the presentation at $p4$ (task priorities are part of the mission specification given by the user).

D. Planning with an Unknown Waypoint

Combining planning with other motivational modules can be very beneficial, as shown in this test case. We gave the robot the mission to guard $p1$ and to deliver a message from $c1$ to $p7$. Since $c1$ location is unknown on the map, the planner cannot make a good estimation on how to reach it. Here, the planner assumes the worst-case scenario, i.e., that $c1$ is very far. As shown in Figure 8, this resulted in going first to $p6$. While going to $p6$ as recommended by the planner, an instinctual motivation is recommending an other task of finding location $c1$. To find this place, the robot popup a message on his screen and asking assistance to text-to-speech software. At same time, the planner and instinctual motivation execute two different tasks in order to achieve the mission. When requested information is provided by a human (i.e., that $c1$ is between $p2$ and $p7$), the task of finding $c1$ is completed. By receiving this event, the task planner is reinvoked. Because $AskMessage()$ and $GiveMessage()$ are now feasible before $Guard()$, the planner chooses this order.

Unless one uses multiple processes for task recommendation and at the same time avoids keeping the planner as centralized and unique source of motivation, it becomes difficult to produce this kind of behavior. Doing it with a traditional hybrid architecture would require a more elaborated model at the planning level in order to generate a plan that first try to find $c1$ during the time available. Our architecture simplifies the complexity of the planning model by delegating this kind of processing to simpler modules, making the system more scalable.

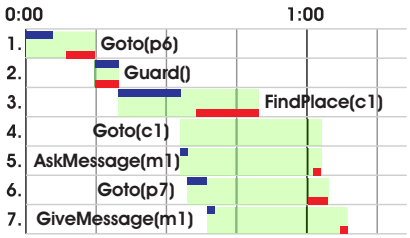

Fig. 8. Plan generated with $c1$ unknown.

E. Performance Summary at AAAI Robot Challenge 2006

For the AAAI Robot Challenge, we prepared a 30 minute demonstration making the robot achieve six tasks in different locations: deliver a message, demonstrate audio capabilities,

find a power outlet, recharge the batteries, show a presentation and request an human to fill out a survey about the demo. In a typical trial, the planner is invoked 25 times. Number of planned tasks varies from 1 to 6, with an average of 3.6 tasks per call. The minimum, average and maximum time planning required to generate plans were respectively of 0.47, 3.49 and 0.94 milliseconds. Fast processing time for the planner is observed because of the small number of tasks, but real-time planning is an important requirement for handling dynamic conditions. Our planner generally outputs plans under 1 second for missions of up to 10 tasks.

IV. Related Work

Over the past decade there has been various robot architectures that integrate reactive behaviors and deliberate planning with quite remarkable success, including [11], [12], [13]. One emphasis in [13] was on defining a Task Description Language (TDL) amenable to an automated planning process. In [12], the focus is on integrating production rule-based robot reaction rules (specified using the Procedural Reasoning System) with an automated planner. Similarly, ROGUE system integrates a behavior-based robot architecture with PRODIGY planner. Other approaches include [14], [11], [15] and some of them address the issue of monitoring and handling metric time constraints. One contribution of this paper to this inquiry is the design of a behavioral architecture in which the planner is considered as a behavior producer on the same footing as others. As explained above, for some missions such a feature is crucial.

Another contribution is the extension of HTN planning make it better fit for planning in domains such as the AAAI Robot Challenge. The extensions deal with task priority, temporal constraints, time windowing and task filtering. As we mentioned before, there exist similar extensions of HTN planning to deal with time constraints [6], [7], but ours is unique in handling tasks with priorities, beside being the first demonstration of the actual usefulness of such extensions on complex real-world robot experiments. There exist other integrations of planning with time constraints into robot architectures [16], but without the hierarchical task decomposition option, neither the decentralized behaviour-based approach.

V. Conclusion and Future Work

This paper studies the use of an HTN-based planner in a behavior-based architecture following a distributed approach to issue tasks. It outlines in details the special features added to the HTN-based planner to deal with specific constraints of having an autonomous robots operate in real life settings, and is combined with an external path planning algorithm handling navigation tasks. These features mainly deal with plan generation, monitoring and execution in dynamic conditions, features that, when taken collectively, provides an original solution to mobile robot planning capabilities. Results confirm the feasibility of our approach.

During our trials, we noted that an important source of uncertainty concerns actions duration. When the robot navigates in a crowd or a corridor with a lot of people, it gets significantly slowed down. The average speed of *Goto(X)* actions varies from under 0.1 to more than 0.2 m/s. This sometimes caused plan failure because the robot arrived too late to meet mandatory time constraints. Also, duration of human-robot interaction tasks are highly random. Many recent works [17], [18] in AI planning try to address uncertainty about time and resources. We believe that such feature will be very benefit to robotics application like for ones that include human and robot interaction tasks that have unpredictable duration. In future work, we plan to integrate this capability in our HTN-planner.

References

[1] R. C. Arkin, *Behavior-Based Robotics*. The MIT Press, 1998.

[2] S. J. Rosenschein and L. P. Kaelbling, "A situated view of representation and control," *Artificial Intelligence*, vol. 73, pp. 149–173, 1995.

[3] R. A. Brooks, "Elephants don't play chess," in *Designing Autonomous Agents: Theory and Practive form Biology to Engineering and Back*. The MIT Press, Bradford Book, 1990, pp. 3–15.

[4] M. Werner, *Humanism and beyond the truth*. Humanism Today, 1999, vol. 13, http://www.humanismtoday.org/vol13/werner.html.

[5] D. Nau, T. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.

[6] F. Yaman and D. S. Nau, "Timeline: An htn planner that can reason about time," in *AIPS Workshop on Planning for Temporal Domains*, 2002, pp. 75–81.

[7] R. P. Goldman, "Durative planning in htns," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2006, pp. 382–385.

[8] M. Do and S. Kambhampati, "Sapa: A scalable multi-objective metric temporal planner," *Journal of Artificial Intelligence Research*, vol. 20, pp. 155–194, 2003.

[9] C. Cote, D. Letourneau, F. Michaud, J.-M. Valin, Y. Brosseau, C. Raievsky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[10] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 2436–2441.

[11] S. Lemai and F. Ingrand, "Interleaving temporeal planning and execution in robotics domains," in *Proceeedings National Conference on Artificial Intelligence (AAAI)*, 2004.

[12] I. F, F and O. Despouys, "Extending procedural reasoning toward robot actions planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2001, pp. 9–14.

[13] R. Simmons and D. Apfelbaum, "A task description language for robot control," in *Proceedings Conference on Intelligent Robotics and Systems*, 1998.

[14] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using iterative repair to improve the responsiveness of planning and scheduling," in *Proceedings Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 2000.

[15] J. Ambros-Ingerson and S. Steel, "Integrating planning, execution and monitoring," in *Proceedings National Conference on Artificial Intelligence*, 1998.

[16] Y. Abdedaim, E. Asarin, M. Gallien, F. Ingrand, C. Lesire, and M. Sighireanu, "Planning robust temporal plans: a comparison between cbtp and tga approaches," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.

[17] Mausam and D. S. Weld, "Probabilistic temporal planning with uncertain durations," in *National Conference on Artificial Intelligence (AAAI)*, 2006.

[18] H. L. S. Younes and R. G. Simmons, "Policy generation for continuous-time stochastic domains with concurrency," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2004, pp. 325–334.