Parallel Plan Execution and Re-planning on a Mobile Robot using State Machines with HTN Planning Systems*

Lasse Einig, Denis Klimentjew, Sebastian Rockel, Liwei Zhang, Jianwei Zhang[1]

Abstract— **This work presents parallel plan execution, an intermediate layer for mobile robot systems using high-level planners such as Hierarchical Task Network planning to execute a series of actions.**

Sequential plans generated by high-level planners leave system resources unused while unnecessarily increasing execution time of generated plans. This increases delay of development cycles and also increases the actual cost, regarding time, energy, and resources, of running mobile service robots in real environments.

In this paper, a parallelization layer is presented using an execution based state machine. An extension to this state machine is proposed, which is capable of low-level re-planning if the desired action could not be executed correctly. The achievements are evaluated on a mobile service robot using an HTN planner and future integration into a learning system is presented.

I. INTRODUCTION

This paper presents the middle layer of a three-layer architecture in order to bridge the gap between sequential plans, generated by Hierarchical Task Network (HTN) planners, and the parallel execution capabilities of modern mobile service robots, such as the PR2 by Willow Garage[2], as part of the RACE[3] (Robustness by Autonomous Competence Enhancement) project. The time comsuming test runs, which have to be repeated numerous times during the RACE project, inspired this work, in order to reduce the delay of the development process. Therefore implementation and evaluation are done using the RACE architecture, as seen in figure 6 and described in section IV-D, and thus, the Robot Operating System (ROS[4]) and the HTN planner JSHOP2 [9], respectively SHOP2, which are used by the RACE project. As seen later, the structure of this work may be applied to any robotic system using ROS with only minor adaptions, as well as systems not using ROS with medium adaptions. The main thrust of RACE is to develop a framework and methods for learning from experience utilizing the scenario of a restaurant waiter.

The general aim of this work is to optimize the plan execution on mobile service robots. This optimization is based on two parts: the parallel execution of sequential plans and the re-planning capability of the parallelized plan. The given state by the RACE project offers a sequential plan generated by the JSHOP2 planner, which is then executed sequentially using ROS services and actionlibs.

HTN planning is one of the most popular high-level planning strategies for robotics. Other common HTN planners are NONLIN, SIPE-2, O-PLAN, and UMCP, but SHOP2 is the most efficient HTN planner for a wide area of applications [7][9]. HTN planners recursively decompose complex tasks to smaller tasks, often referred to as atomic tasks. The size of this atomic task depends on the interpretation. One project might assume picking an object as an atomic task, where another project would try to decompose this task further to detecting, positioning, and grasping or even smaller tasks. A huge advantage of SHOP2 over other HTN planners, including SHOP [8], is the possibility to generate partially order plans, which also increases the potential for parallel execution, i.e. by generating a plan to grasp two objects and then moving with theses object instead of grasping a single object, moving this object and returning to grasp the second object. Although either some very specific [2], or very general [3][5][6] works on parallel plan execution exist, none of them provide a usable approach to general parallelization of high-level plans.

In order to execute the plan after parallelization, the state machine architecture SMACH[5] (**S**tate **Mach**ine) [1] was chosen. SMACH was designed to fuse with the ROS operating system, but at its core, SMACH is independent and thus, may also be used without ROS. SMACH is based on python allowing it to be used with any operating system supporting Python 2.7 or higher and therefore run on almost any system while freely exchanging the planning software as well as the executional level. SMACH was intended to quickly implement robust robot behavior. While most state machines use the states to represent given configurations between the execution, SMACH states correspond to the system performing a given task. This concept puts the focus on the execution instead of snap-shots between performing actions. States are connected by their outcomes and may hold data which can be passed to other states. The main reason why SMACH is used to manage the execution of the parallelized plan, are the *container* constructs. *Containers* may hold multiple states, as well as *containers*, allowing the construction of complex hierarchical state machines. SMACH offers multiple types of *containers*, where only two are of interest for this work: The *sequence* container and the

*This work was not supported by any organization.

[1]L. Einig, D. Klimentjew, S. Rockel, L. Zhang and J. Zhang are with TAMS Group, University of Hamburg, Germany {einig, klimentjew, rockel, lzhang, zhang} at informatik.uni-hamburg.de

[2]http://www.willowgarage.com

[3]RACE is funded by the European Community's Seventh Framework Programme FP7-ICT-2011-7 under grant agreement n° 287752. http://www.project-race.eu

[4]http://www.willowgarage.com/pages/software/ros-platform

[5]http://www.ros.org/wiki/smach

concurrence container. As the names reveal, the *sequence* container executes all contained states and sub-containers in sequential order, whereas the *concurrence* container executes all contained states and sub-containers in a parallel order. Using only these two, SMACH is able to execute any type of parallelized plan.

Additionally, using customized states, the state database, and the state outcomes, a low-level re-planning may be introduced.

This paper gives an overview on the existing system and the important parts for the parallelization and examines parallelization possibilites in section II. After examining possible parallelization, the implementation is presented in section III. In section IV The results are evaluated for two scenarios, of which one is a theoretic scenario in order to show possible benefits from parallelization. In the latter sections, a low-level re-planning architecture is proposed (section V) and the future integration with other parts of the RACE architecture is presented (section VI).

II. PARALLELIZATION

This work will talk about tasks, operators, actions and resources. A task represents a planning object. It may be complex and thus, decomposable into smaller tasks, or atomic. An atomic task is referred to as an operator. Once a plan is generated, it will be a sequence of operators, where each operator corresponds to one or more actions. For simplicity, not all actions are considered by the planner, as some actions are too trivial, for example pointing the visual sensors at the target area. Actions are executed by utilizing a set of ressources.

A. Prerequisites

The evaluation platform for the RACE project is the PR2. This mobile service robot uses a four-wheeled omnidirectional driving system for navigating, referred to as *base* (*B*). It is also able to raise and lower its upper manipulation and sensory system, referred to as *torso* (*T*). The upper manipulation system is made up of two seven-DOF arms with a two-fingered one-DOF gripper attached to each arm, referred to as *right arm* (*RA*) and *left arm* (*LA*). The sensory system referred to as *head* (*H*) holds an ASUS Xtion PRO LIVE sensor as well as multiple cameras including two stereo camera pairs. The PR2 also features a base laser scanner for localization and collision avoidance and a tilting laser scanner for object detection and collision avoidance.

The RACE planning domain offers ten operators and a set of complex tasks to decompose. Four of these operators move one arm and only differ in the prerequisites and the arm which is to be moved. One operator for each arm assumes that both arms are in a folded (*tucked*) position, thus leaving the arm that is not moved by the operator in the *tucked* position after the operator has finished. The other operator for each arm assumes that only the arm which should be moved is tucked and the other arm is already in some kind of manipulation position. After completion of this operator, both arms will be in an *untucked* state. These arm operators are *!move_arm_to_side left_arm* respectively *!move_arm_to_side right_arm*. These operators require the corresponding arm as ressource (*RA*, respectively *LA*). *!tuck_arms both_arms* puts both arms in a *tucked* state. The torso resource (*T*) is used by the *!move_torso ?position* operator and *!move_base ?to* and *!move_base_blind ?to* use the base resource (*B*) to navigate within the environment. *!move_base_blind* is used to move the robot into grasping range by ignoring collision avoidance, e.g. when the convex hull of a table is in collision with the robot, but there is no physical collision. The last two operators, *!pick_up_object ?object ?arm* and *!place_object ?object ?arm ?to*, control manipulating objects in the environment. In addition to the corresponding arm resource, they also require the head resource in order to detect the object with vision and depth sensors.

B. Analysis

As already mentioned, evaluation and testing is done using two scenarios. The first scenario is part of the RACE project and physical experiments will show the improvements using parallelized plans. The second scenario is theoretic and is used to show possible benefits from advanced paralellization.

The first scenario, referred to as *Serving Beverages* scenario locates the robot in a room with a counter and a table. Multiple objects, including a clean coffee cup are located on the counter, the arms of the robot are in an *untucked* state and the torso is in an upper position. The complex task *serve_cup table_1* is triggered and decomposed into a sequential plan with twelve atomic tasks.

1) *!tuck_arm both_arms*
2) *!move_torso torso_down_position*
3) *!move_base counter_1_pre_manipulation_pose*
4) *!move_torso torso_up_position*
5) *!move_arm_to_side left_arm*
6) *!move_base_blind counter_1_manipulation_pose*
7) *!pick_up_object coffee_cup_1 left_arm*
8) *!move_base_blind counter_1_pre_manipulation_pose*
9) *!move_base table_1_pre_manipulation_pose*
10) *!move_base_blind table_1_manipulation_pose*
11) *!place_object coffee_cup_1 left_arm table_1*
12) *!move_base_blind table_1_pre_manipulation_pose*

This sequential plan is divided into two sections. Due to its triviality, the latter section is reviewed first. It starts with operator 6. Every other operator from that point on is a movement operator with no collision avoidance. In order to minimize the risk of damaging the robot or the environment, the robot may not be allowed to execute any parallel actions while moving without collision avoidance. Therefore, all available resources are assigned to the *!move_base_blind* operator, which also removes any parallelization capabilites in the latter section of the plan.

The former section of the sequential plan does not only hold parallelization capabilites, but it also describes the default actions necessary for any mobile manipulation. As the PR2 is a mobile service robot, almost every task will include mobile manipulation and therefore require this section to be executed in advance. For the parallelization of this section,

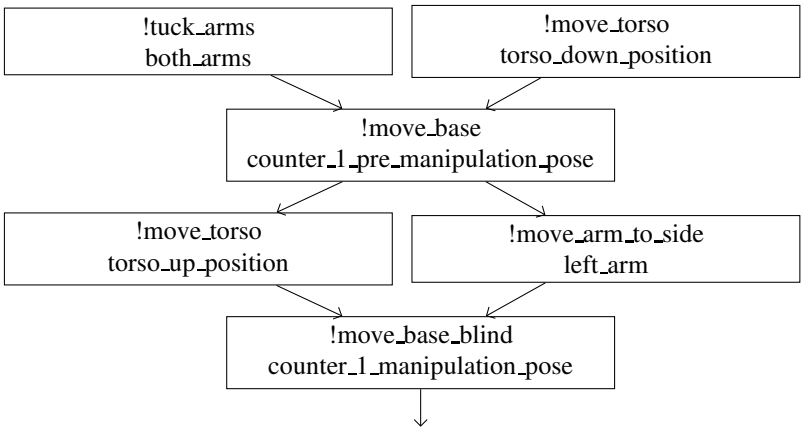

Fig. 1. Parallel section of the *Serving Beverages* scenario. A security constraint is added to *!move_base counter_1_pre_manipulation_pose* forcing the robot to finish arm and torso movements before moving the base to the new position and waiting for the base movement to finish before the arms and torso may operate again. This prevents the robot from hitting objects or humans in the environment, respectively increases the tilt stability while moving due to a lower center of mass.

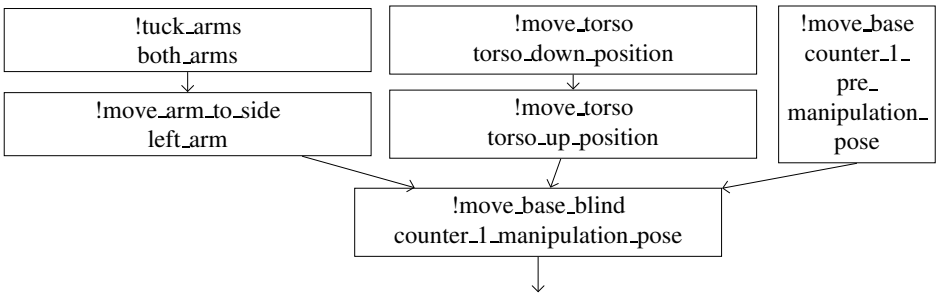

Fig. 2. Parallel section of the *Serving Beverages* scenario. No security constraint is added. Possible threat is posed to the environment but parallel capabilities are increased.

there are two available approaches. These approaches will be referred to as *defensive* and *offensive* approach. For both approaches, operators 1, 2, 4 and 5 may be executed in parallel, regarding:

- Operators 1 and 5 use the same resource, thus, 1 has to be finished before 5 may start.
- Operators 2 and 4 use the same resource, thus, 2 has to be finished before 4 may start.
- Operator 3 may not be started after 4 and 5.

Figure 1 and figure 2 show the *defensive*, respectively the *offensive* approach to parallelization. For the *defensive* approach, a security constraint is added, by assigning the *RA*, *LA* and *T* resource to the *!move_base* operator. The *offensive* approach lacks this security constraint and thus, offers more parallelization capability, while introducing an additional risk of colliding with the environment. Both approaches may be valid depending on the environment, but since the test environment is small and narrow, the *defensive* approach is used for evaluation.

The second scenario differs in the latter section while the first section is the same. This scenario puts the robot in a kitchen environment supposed to place cups and plates in a dishwasher with predefined positions for cups and plates. The robot only requires the *head* resource to detect and grasp the object to put into the dishwasher. Placing the object in the predefined position in the dishwasher may be executed without visual confirmation. Thus, a sequential plan of putting cups and plates alternating into the dishwasher may be parallelized as seen in figure 3. Offering not only benefit from parallelization for the *setup* section of the plan, but also for the main execution section, the benefit received for this scenario is even greater than for the *Serving Beverages*. The shown execution order is possible, as the robot uses both arms for manipulation at the same time.

III. IMPLEMENTATION

Although this work is very restricted on the technical prerequisites due to the RACE project, this does not influence the portability of the parallelization level. As the plan is parsed into a proprietary format, the planning level may be exchanged freely, only the parser has to be adapted. Three different parsers are already available:

- A string parser
- A python list parser
- A JSHOP2 parser

All of the parallelization, as well as the SMACH interpreter run Python, thus being platform independent. There is a very close connection between SMACH and ROS but SMACH may also be used without ROS. The execution API connects the interchangeable parts with the platform on which the

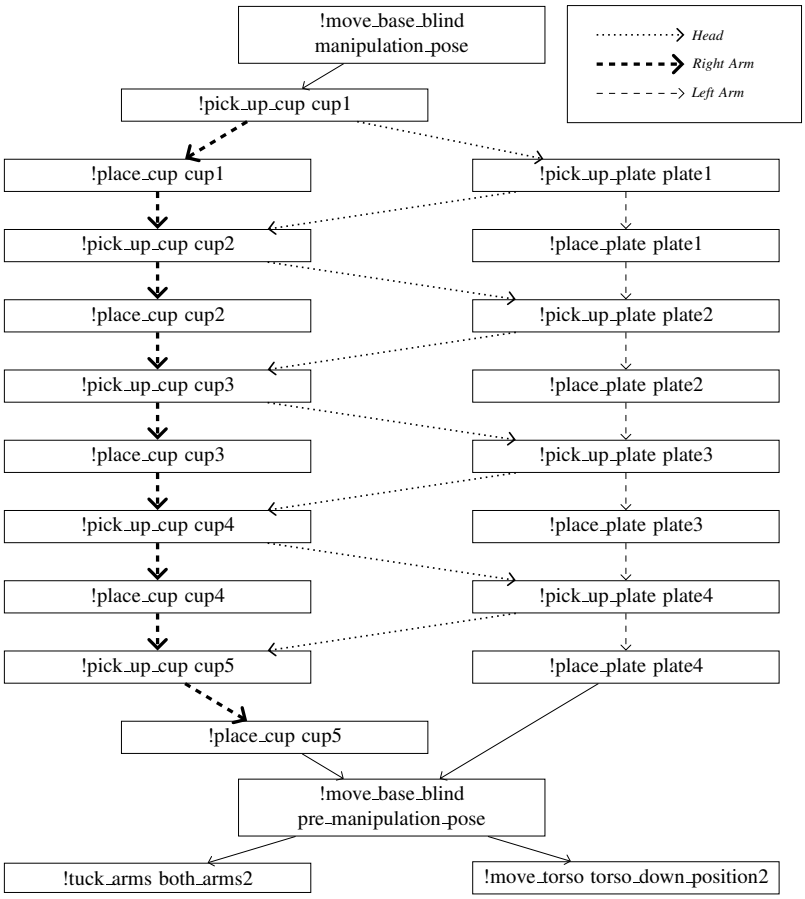

Fig. 3. Latter section of *Loading Dishwasher* scenario. Repetitive execution of loading cups and plates with different arms offers huge parallelization capabilities, nearly cutting execution time in half. Dependencies from *Head*, *Left arm* and *Right arm* resources in the repetitive part are distinguished between by different line styles shown in the legend.

system is supposed to be running. This enables this system to run on any platform, with any high-level planning system.

As already mentioned, the implementation is divided into three levels:

1) Plan generation (HTN Planner)
2) Plan parallelization
3) Plan execution (Low-level Robot API)

Plan parallelization is inserted between the two already existing levels. While plan generation is clearly seperated from plan parallelization and execution, parallelization and execution are somewhat interleaved.

Before the introduction of the parallelization level, the execution manager and execution API were united in a single piece of software. Due to the requirements of the execution of the parallelized plan, the execution manager needs to be joined into the parallelization level and isolated from the execution API. This part is taken by the SMACH interpreter.

A. Structure of Parallelization layer

The parallelization is divided into three parts. The first part parses the original JSHOP2 plan into a format used by the parallelization algorithm. The second part prepares the parallelization by constructing a directed graph in set notation ($G = \{V, E\}$) and finding all possible start operators. These start operators are all operators in the graph with no incoming edges. This graph is constructed by allocating the required resources and the security constraints to the operators. For evaluation and testing, the list of resources for each operator is hand-crafted following the pattern described in section II-B.

The third part uses the start operators and recursively runs through all remaining operators until all operators have been rearranged into the parallel plan. In each run, the remaining operators are analyzed and inserted into the appropriate position in the parallelized plan if there are no more incoming edges from operators which are not yet in the parallelized plan. Depending on the incoming edges from operators within the parallelized plan, the remaining operators are placed within sequential or parallel sublists.

Once all operators have been rearranged, the parallelized plan is passed to the SMACH interpreter.

B. SMACH

The SMACH interpreter converts the parallelized plan, which is a nested list of sequential and parallel lists. These sublists are transposed to *sequence* and *concurrence* containers, which are filled with states corresponding to the operators from the parallelized plan. Each state uses the

corresponding API on the robot execution level to execute the action equivalent to the operator.

IV. EVALUATION

The evaluation differs between the two scenarios, as the second scenario is only theoretic, it is missing the execution time difference evaluation as well as the cpu load difference evaluation. For both scenarios, the successful parallelization by the algorithm is shown.

As described in section II-B, the *offensive* approach is not suitable for the test environment, as it is very narrow. Thus the *defensive* approach was chosen for the practical evaluation, the parallelization evaluation was done for both approaches.

A. RACE Scenario

The *offensive* approach for the RACE scenario, also referred to as *Serving Beverages* scenario, returned a flawless result compared to the hand-crafted parallelization. The parallelized plan holds three concurrent paths for the first plan section. The first path is made up of the arm actions *!tuck_arms* and *!move_arm_to_side*, the second path holds the torso actions for moving the torso down and then up again, and the third path is the *!move_base* action. All three paths would be executed concurrently, and, after finishing all paths, the sequential secondary part of the plan would be executed also sequentially. For the *defensive* approach, there is a minor flaw which does not affect the execution order. After successfully putting the *!tuck_arms* and *!move_torso torso_down_position* action into a *concurrence* container, sequentially followed by the *!move_base* action. At this point, the algorithm opens an unnecessary sequential container, as this is already part of a sequential container, and places the second *concurrence* container within, followed by the latter section of the plan in a sequential order. This flaw could be removed by verifying the parallelized plan after the parallelization process, but as it is only a cosmetic flaw, this needless calculation is skipped.

B. Theoretic Scenario

As the first section of the plan for the theoretic scenario of *Loading a Dishwasher* almost holds the same operators as the RACE scenario, the results are also the same. Except for an additional *!move_arm_to_side* for the other required arm for manipulation, the plan is the same. This additional operator does not affect the parallelization, thus generating the same cosmetic flaw for the *defensive* approach as for the RACE scenario. For the latter part, this flaw appears again but as it is only cosmetic, the parallelization result may be interpreted as perfect. The parallelized plan is made of *concurrence* containers each holding a *!place* and a *!pick* action.

C. Time benefit and CPU load

As mentioned in section IV, only the *defensive* approach has been executed and thus, only for this approach an evaluation is possible. Figure 4 shows the timeline for the execution using the sequential plan received directly from the HTN planner in the upper part and the parallelized plan in the lower part. While the parallel execution takes 269 seconds, 102 seconds less than the sequential execution, respectively 27.5 %, the authors would like to put the focus on the execution of the first section of the plan. This recurring section, which is required for any execution of mobile tasks, takes 39 seconds in the sequential order. The parallel execution of these two actions only requires 24 seconds, a benefit of 15 seconds, respectively 39.5 %. A similiar benefit can be observed for the actions 4) and 5). Even if the scenario itself does not offer much parallelization capabilities, a time benefit is achieved already with these first 5 actions. As the parallelization algorithm itself requires much less than one second, the overall benefit of parallel execution is obvious. Exceeding this to a parallel execution of the theoretic scenario, a time benefit between 35 and 45 % may be expected.

Other than the time required by the parallelization algorithm itself, also the increased CPU load of the parallelization and the execution of the parallelized plan was expected to be a drawback. Figure 5 shows the CPU load during both the sequential and the parallel execution. Except for a minor shift of the curves, no significant increase in CPU load is observed. Instead, the reduced time duration of the execution not only reduces the total execution time, but also the total processing time. The main reason for high CPU load is the collision detection and low-level path planning, as well as object recognition. As these parts are not parallelized, the CPU load does not increase.

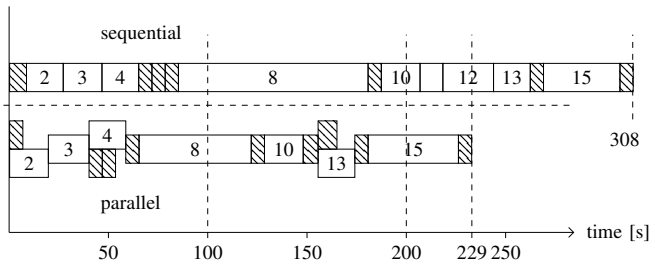

Fig. 4. Result from the practical experiment of the *Serving Beverages* scenario. The upper section shows the overall execution time and the execution time of each task in the sequential order. The lower section shows the overall execution time, the execution time of each task and the parallel ordering of the parallelized plan. A significant difference of 102 seconds, respectively 27.5 % is observed after optimizing the sequential plan.

D. Architectural Integration

Although generating good results, the automated parallelization is still limited. A handcrafted resource to operator mapping is required and new operators have to be implemented. A decrease in the size of the atomic tasks may also lead to a very large-scaled plan and set of operators, where a human might exceed its capabilities trying to map the resources.

Figure 6 shows the current architecture of the RACE project. All communication is running via a *Blackboard* system. This *Blackboard* connects the functional blocks **B**

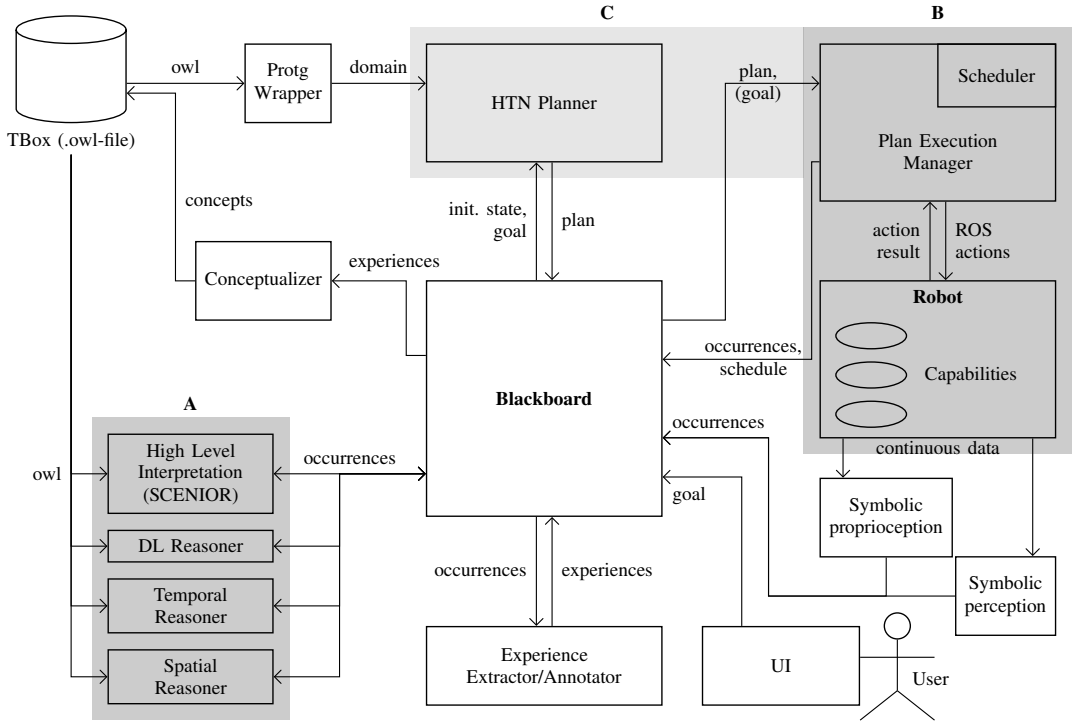

Fig. 6. Global architecture of the RACE project. A Blackboard is the center of the structure managing the communication. Block **A** highlights the *Reasoners*, blocks **B** and **C** hold the modules used in this work.

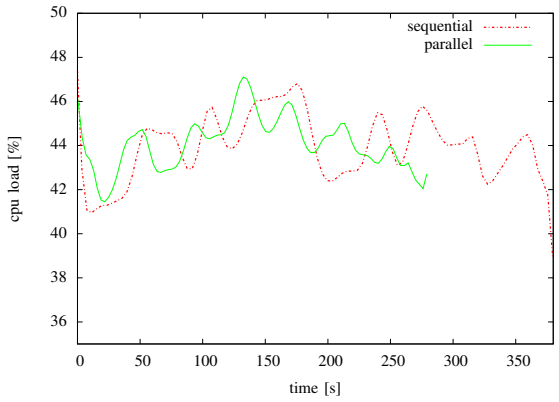

Fig. 5. Processor load during the practical experiment of the *Serving Beverages* scenario. Actually measured values of the computer cluster have been smoothed with a Bezir polynom and the arithmetic average was calculated. Both graphs show similiar result, the sequential line is shifted to the right due to increased execution time within the first five tasks. Overall processor load is steady comparing the sequential and the parallel execution of tasks.

and **C** with block **A**. The reasoners in block **A** will eventually be able to assume the role of the intelligent resource mapping, learning from experience gained while executing plans sequentially and in parallel. These reasoneres may then also be able to decide wether or not to use the *offensive* approach, as the environment changes and spreads wider. Other changes in the environment, such as the size of the object, the height of obstacles, the lighting conditions and others may also chang the required resources for some operators. One of the reasoners will notice this change in conditions and alter the plan execution suchlike.

V. RE-PLANNING

The described three-layer architecture can be extended to a four-component closed loop architecture by adding a re-planning component, which monitors the execution of the state machine. The re-planning component triggers the HTN planner to create a new plan with an adapted goal. A cost-function-based rule system is used to make the decision on the new goal, which is to be sent to the planner. This rule system is extended by an expected success rate for each possible adaption, defining a cost-success function.

$$\text{cost-success} = \frac{\text{cost}}{\text{success rate}}$$

While costs are static, success rates vary for each type of failure in the original plan, ranging from object recognition failure over grasping failure to path planning failures. The current set of available goal adaptions for the task of detecting and grasping an object is as follows, in ascending cost order:

- Pointing the head
- Moving the torso up an down
- Moving the base at the manipulation position

All three options may also be combined by the re-planner.

An example for re-planning finds the mobile robot in front of a counter with a mug on top. The robot is looking straight at the mug, the torso is in a lower position, the base is located so that the right shoulder is aligned with the mug. The mug

is identified correctly by the object recognition, but trying to find a valid trajectory fails because the arm planner runs into a singularity. This singularity is the result of the arm trying to align all joints along a straight line. The state machine reports a grasp failure to the re-planner, which then triggers (1) the rule-based re-planning routine and (2) a reasoner seen in block **A** of figure 6 with the failure information. For (1), the re-planner decides to use the head movement because it has the lowest cost and the success rates of all possible decisions is yet 1, due to the lack of experience. Since the failure is due to a singularity, re-executing the plan with altered head orientation also fails. The next decision is moving the torso up, since the torso is currently in a low position. Executing the pick-up task with a higher torso position succeeds, because the angle in the shoulder joint is now different and is no longer a singularity. Moving the base left or right would also remove the singularity, but at a higher cost. Assuming a cost of 1 for moving the head and 1.5 for moving the torso, the success rate for moving the torso must be 50 % greater than for moving the head for the re-planner to decide to move the torso before moving the head.

In order to find actual values for the success rate, a reasoner, which is able to extract these experiences is required.

For expected success rates exemplified by recognition and grasping failures, the cost-success function in table I finds torso movement to be the most efficient adaption for re-planning, which may be related to the improved view-angle allowing the object recognition to identify more features.

TABLE I
COST BASED VS. COST-SUCCESS BASED ADAPTION

	Cost	Expected Success Rate	Cost-Success
Recognition			
Head	1	1	1
Torso	2	3	$.6\overline{6}$
Base	4	1	4
Grasping			
Head	1	.1	10
Torso	2	2	1
Base	4	3	$1.3\overline{3}$

VI. CONCLUSION AND OUTLOOK

This work aims to improve execution time and resource management on mobile robots. Not only to increase efficiency and thereby reduce expenses for resources and energy, but also to reduce the duration of development cycles for applied research. Up to the the authors knowledge, there are currently no comparable, applicable solutions to improving the task execution on mobile robots using HTN planners by executing tasks in parallel.

Although this work was created and evaluated as part of RACE on the mobile platform PR2, the results are applicable to almost any platform using high-level planners and divisible into resources.

The evaluation and experiments identified a security issue due to the parallel execution of planar movements and torso as well as joint movement, which could be resolved by introducing resource constraints. The execution of the RACE scenario revealed a time benefit of 27.5 % compared to a sequential execution without significant increase in processor load. The replanning proposition allows efficient re-execution of failed plans or failed parts of plans. For the remaining shortcoming of both the parallelization and the replanning, resulting from the required handcrafted resource allocation and cost-success table, the strong research on reasoners presented in the RACE architecture promise improvement.

In future, the collaboration of the reasoners and the parallelization, as well as the re-planning, will enable the robot to autonomously find the best suitable parallel execution order for any given task by learning from experiences gained in previous experiments. Especially a mass-simulation reasoner using mixed reality, as presented in [4], will provide most of the required experience.

ACKNOWLEDGMENT

The authors would like to thank all of the RACE partners for their valuable suggestions and cooperation.

REFERENCES

[1] J. Bohren and S. Cousins. The SMACH high-level executive [ROS news]. *Robotics Automation Magazine, IEEE*, 17(4):18 – 20, dec. 2010.

[2] Luis Castillo, Juan Fdez-Olivares, Óscar García-Pérez, and Francisco Palao. Temporal enhancements of an HTN planner. In *Proceedings of the 11th Spanish association conference on Current Topics in Artificial Intelligence*, CAEPIA'05, pages 429 – 438, Berlin, Heidelberg, 2006. Springer-Verlag.

[3] Sam Ade Jacobs, Kasra Manavi, Juan Burgos, Jory Denny, Shawna Thomas, and Nancy M. Amato. A scalable method for parallelizing sampling-based motion planning algorithms. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2529 – 2536, may 2012.

[4] Denis Klimentjew, Sebastian Rockel, and Jianwei Zhang. Active scene analysis based on multi-sensor fusion and mixed reality on mobile systems. In Fuchun Sun, Dewen Hu, and Huaping Liu, editors, *Foundations and Practical Applications of Cognitive Systems and Information Processing*, volume 215 of *Advances in Intelligent Systems and Computing*, pages 795–809. Springer Berlin Heidelberg, 2014.

[5] A. R. Lingard and E. B. Richards. Planning parallel actions. *Artif. Intell.*, 99(2):261 – 324, March 1998.

[6] J.Y.S. Luh and C.S. Lin. Scheduling parallel operations in automation for minimum execution time based on pert. *Computers & Industrial Engineering*, 9(2):149 – 164, 1985.

[7] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *Proceedings of the 17th international joint conference on Artificial intelligence*, volume 17, pages 425–430, Seattle, WA, 2001.

[8] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, pages 968–973, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[9] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.