Improved AI Planning for Cooperating Teams of Humans and Robots

Stefan-Octavian Bezrucav, Burkhard Corves
Institute of Mechanism Theory, Machine Dynamics and Robotics
RWTH Aachen University
Eilfschornsteinstraße 18
52062 Aachen
bezrucav@igmr.rwth-aachen.de
corves@igmr.rwth-aachen.de

Abstract

Each human usually plans his or her future activities in advance, trying to select those actions and to sort them in such a way that as many goals as possible are accomplished. The real world is dynamic and some elements are characterized by a high degree of uncertainty. Nonetheless, the humans manage to adapt their plan fast and easy to the changes and problems that occur, being able to recover or reorganize the activities such that the goals can be further reached.

Considering the same requirements for adaptability, the automated task planning for systems with autonomous robots was introduced. However, the existing task planning approaches do not satisfy well enough the requirements from use-cases with cooperating humans and robots, where beside the adaptability, the computation time and the modelling possibilities are of high importance.

Considering exactly the requirements from such scenarios, in this work, the automated task planning is integrated in a Three-Level Planning strategy. This strategy has enriched modelling capabilities and ensures a high adaptability to unforeseen situations. Moreover, together with the Replanning approach, with its parallel planning and dispatching features, it results in qualitative plans that can be sent for execution in a short period of time. The implemented methods were validated in a realistic simulation of an industrial environment.

Introduction

Task planning is one important software component of a cognitive system used in any scenario in which robots are involved, that selects and supervises the actors' actions. This component plays an even greater role in human-robot cooperating scenarios, as it must ensure the generation of reliable and flexible plans, considering the presence and the needs of the humans.

One of the most promising application areas for cooperating humans and robots, for the next couple of years, is in industrial scenarios. These scenarios are characterized by typical tasks and teams of actors. Some of the common tasks are: bolts or screw tightening, glue application, glue spreading, components assembly or disassembly, cleaning, transportation and inspection. Further on, in such environments mixed teams of intelligent actors as humans and robots usually cooperate for the execution of these tasks. This combination of types of tasks and mixed team of actors comes with a set of specific requirements for the planning system.

The most important requirement is time. In order to increase the acceptance of the humans for the cooperative work with autonomous agents the generation of a plan and its dispatch, both at the beginning as well as in case that a replanning is requested, must happen in a couple of seconds. A deadline of five seconds is selected. Further requirements imply a high flexibility for the system, planning for actors with different capabilities (e.g. humans and robots) and planning for many goals (e.g. more than 10).

In this paper a complex planning strategy with special features is presented. This strategy tries to integrate the best solutions for the above mentioned requirements and thus, to close the gap between the abstract planning and the real scenario for which it is planned.

The paper is structured as follows. The next section provides an overview of task planning approaches for different use-cases. Continuing, in the Preliminaries section a theoretical introduction to automated planning and a brief introduction of the used framework is given. The main section of this article contains the description of the Three-Level Planning approach and of the Replanning strategy with the parallel planning and dispatching features. Last but not least, the implementation, the validation setup, the results and the conclusions are presented.

Related Work

In the last years, task planning has became an essential software module for all systems that integrate autonomous robots. Different planning strategies and specialized features were developed and are used in different scenarios.

Automated planning approaches are an important part of the invoked planning strategies. (Keller, Eyerich, and Nebel 2010) have used such an approach for the planning of the tasks that an autonomous service robot should execute in a typical kitchen environment. Extensions for that are represented by an approach which considers automated task planning on more levels (Buksz et al. 2018). The hierarchisation of the planning process on two levels reduces the

search space and thus, the planning time and increases the solution quality, by maintaining the flexibility of the automated planning. The scenarios used for validation involved the mission control for a long period of time of Underwater Vehicles (AUVs).

Planning in environments where humans are also present is considered in a couple of works. In order to take care of the human's presence, special planning approaches were developed. In (Cirillo, Karlsson, and Saffiotti 2009) the effects of future human actions are considered for the planning of the robot's actions. The validation of this approach was done within a household scenario, involving a vacuum cleaner and a human. For collaborative tasks in production spaces, (Reiterer and Hofbaur 2017) present an extended approach for planning: planning with opportunities. To the plan that was generated to reach the goals ensuring safety conditions, opportunities, as optimizations (e.g. higher speeds) or shortcuts were allowed. The work done by (Sanelli et al. 2017) for short-term human-robot interactions, where the users do not know what the capabilities of the robot are, include the generation of a condition plan that considers in a structure similar to a tree all possible outcomes of all actions. Thus, the reaction times are very low.

Planning can be also done for teams of multiple robots. MAPJA (Chouhan and Niyogi 2017) is a domain-independent approach through which plans for multi-agent systems are generated. The speciality of these plans is the fact that they can integrate both individual and joint actions. Planning for multiple mobile robots, within a complex framework which includes perception and execution modules, was presented in (Silva Miranda, de Souza, and Sousa Bastos 2018). The planning was sustained by a multi-robot trajectory planning which ensured non-collision paths and thus influenced the way in which the actions were allocated to the actors.

The planning module is usually integrated in a much more complex framework. Continual planning which involves a reactive loop consisting of an observation, a monitoring, a planning and an execution module was deployed in the TedyUp project (Dornhege and Hertle 2013), a scenarios with a mobile manipulator in a household environment. Another framework which integrates Internet-of-Things sensors and actuators with task planning for realistic robot controlling in dynamic environments is presented in (Harman, Chintamani, and Simoens 2017).

To our knowledge there is no work that combines all task planning features presented in the related work and needed for scenarios with a team of cooperating humans and robots. The approach presented in this paper integrates characteristics of the automated planning for dynamic environments, the task planning for robots which are active in environments shared with humans and task planning for multiple actors, as part of a complex software framework. Further on, the suggested implementation and integration of the task planning module considers the requirements from scenarios with cooperating humans and robots regarding the short planning and dispatching times, modelling of actors with different capabilities, a high flexibility and for a high number of goals (e.g. more than 10). Furthermore, the integration of the human in the planning process goes beyond the state-of-the-art, as in the process, tasks are also allocated to humans, tasks that should be executed in a given duration, but are characterized by uncertainties both in the achieved duration and effects and may also fail.

Preliminaries

From the different planning strategies, the focus of this paper is set on the Automated planning, also known as AI Planning. This approach is presented in the following. Moreover, as it is used in the special field of robotics the state-of-the-art framework ROSPlan, that integrates the AI planning in the ROS middelware, is also presented.

AI Planning Problem Formulation A propositional AI planning problem can be represented as the tuple $\Pi = (F, A, I, G)$ (Kambhampati and Srivastava 1996), where:

- F is a set of Boolean propositions that describe the state of the world
- A is a set of actions, for each of which a set of preconditions, a set of effects and a cost are defined
- I is the initial state of the system, represented by instantiations, as true or false, of all propositions from F
- G is the goal state of the system, represented by other instantiations, as true or false, of all propositions from F

A solution for the planning problem is a plan $\pi = [a_1, ..., a_n]$, an ordered sequence of actions. Before the action a_i can start, the propositions from its preconditions must become true. The preconditions of a_1 will be satisfied in the initial state and the effects of action a_n will set the values of the propositions such that the described state is the goal state.

For the classical, propositional AI planning a series of restrictive assumptions apply (Ghallab, Nau, and Traverso 2016). First, it is assumed that the state of the environment can be mapped complete enough only through the propositions from F and the only changes that occur in the environment appear only through the effects of the actions from A. Second, no time is considered and thus, no concurrencies between the actions are possible. Last, it is assumed that the environment is always deterministic. Therefore, no uncertainties can be considered. Given these restrictions the gap between the modelled world for the task planning and the real world is quite large. In order to reduce it the propositional temporal planning (Fox and Long 2003) was introduced.

The propositional temporal AI planning formulation extends the one presented above through a more complex representation of actions. The set F, the initial state I and the goal state G remain the same, while the new set A contains new type of actions. For each action a_i conditions at the start, at the end or during its entire execution, effects at the start or at the end of the execution and a duration are defined. With this formulation the time factor is introduced and therefore, more realistic problems can be modelled. Action synchronisations and actions concurrences are only some of the new aspects that can be considered. The solution of a

propositional temporal AI planning problem is a schedule $\sigma = [\langle a_1, t_1, d_1\rangle, ..., \langle a_n, t_n, d_n\rangle]$, which integrates not only a correct logical sequence of the conditions and effects, but also the time t_i when the action a_i should start and its duration d_i.

Planning Domain Definition Language (PDDL) (Ghallab et al. 1998) has became the standard language for the representation of such planning problems. Beside its characteristic syntax it imposes the splitting of the planning instance in two parts, the *domain* and the *problem*. In the *domain* part the used propositions, in PDDL named predicates, and the available actions are defined. An example of such a predicate and an action is represented in Listing 1.

```
(:predicates
  (agv_at_pose ?agv-agv ?pose-pose) ...
)
(:durative-action generic_action_1
 :parameters (?agv-agv ?tool-tool ?↩
     item-item ?pose-pose)
 :duration (= ?duration (cost1))
 :condition (and
  (at start (agv_at_pose ?agv ?pose))
  (at start (agv_has_tool ?agv ?tool)) ...
)
:effect (and
  (at end (item_processed ?item) ...
))
```

Listing 1: Example of PDDL predicates and actions in the domain part

The *problem* part contains the definition of the used types, of the initial and goals state. The initial position of the agv1 at tool bank 1, as well as the goal that item1 must be processed can be encoded as presented in Listing 2.

```
(:objects
 agv1 agv2 - agv ... )
(:init
 (agv_at_pose agv1 tb1_pose)     ... )
(:goal (and
 (item_processed item1) ...))
```

Listing 2: Example of the objects initialization and the setting of the initial and goal states in PDDL problem part

ROSPlan ROSPlan (Cashmore et al. 2015) is a framework that is specially developed to integrate AI planning techniques, represented in the PDDL language, in the ROS middleware (Quigley et al. 2009). It contains more ROS nodes, each being responsible for a specific processing step. It also integrates a Knowledge Base. At the start of the planning process, the pieces of information from the planning domain and planning problem, and thus the initial state and desired end state of the world, are saved in the Knowledge Base. During the execution of the actions the state of the world changes and the values of the propositions from the knowledge base are adapted correspondingly. Therefore, the actual state of the world is always available and can be put at the disposal to the other nodes, when needed. The nodes of ROSPlan are usually executed in a sequential order:

1. In the *Problem Generation* node a planning instance, based on the status of the environment and on the defined goals, as saved in the knowledge base, is created and saved to corresponding files
2. In the *Planner Interface* node, the previously created files are passed to an automated planner which generates a plan in a text file
3. In the *Plan Parser* node, the generated plan is parsed to computer data
4. In the *Plan Dispatcher* node, the actions of the plan are dispatched corresponding their order to the low-level execution modules through specific interfaces
5. In the *Action Interface* nodes, the interfaces to the low-level execution modules are implemented. These interfaces are defined for each PDDL action.

The automated planners used in the node are search algorithms with specialized heuristics that find a solution (e.g. a plan) for any given planning instances, passed as PDDL problem and PDDL domain. For this work the temporal automated planner OPTIC (Benton, Coles, and Coles 2012) was used.

The *Plan Dispatcher* node contains a method, the Esterel plan dispatcher, specially developed for temporal task planning problems, in which not only the logical sequence of the condition and effects, but also the dispatch time and the durations of the actions are considered.

Adapting AI planning for scenarios with cooperating humans and robots

In this section the new planning approach with its particular features is presented. This approach is centred on AI Planning and it is integrated in the ROSPlan framework. The developed methods consider the requirements of scenarios with a team of cooperating humans and robots mentioned in the first sections.

Before starting with the presentation of the methods the considered types of actors will be described in more detail. There are two types: humans and complex robots in form of a serial manipulator on a mobile platform. Both types of actors can execute two types of basic actions ba: move in the $x-y$ plane on the floor and move the tool center point (TCP) or the hand along a trajectory. Thus, we can define complex actions a_i composed of more basic-actions ba_{ij}. Each complex action a_i, called from now on only action, has a fixed set of basic-actions ba_{ij}, that are always executed in the same order. For example, the load action a_l consists of three basic actions ba_{lj}: move TCP to table and grab object, move TCP to pose above the mobile platform and release object, move TCP to home position. The only degrees of freedom for the actions are represented by the parametrization and duration of each of the basic-actions ba_{ij} of an action a_i. More actions a_i can be grouped together in a cluster.

In the following sub-sections the Three-Level Planning approach and a time effective replanning strategy are explained.

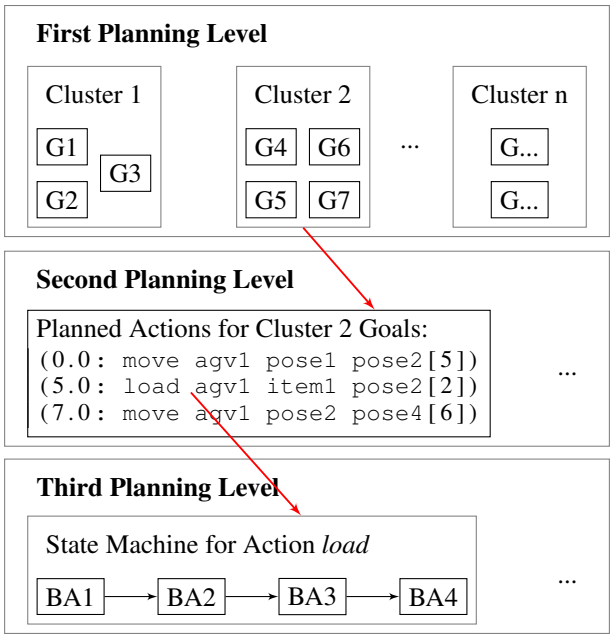

Figure 1: Structure of the Three-Level Planning approach with an example

Three-Level Planning

The need of combining different types of planning comes from the fact that each planning strategy has its advantages and disadvantages, but when combined together generate better results than each individual one. We propose a Three-Level Planning approach as presented in Figure 1. This is similar to a hierarchical approach. On the first level all given goals are grouped in clusters. On the second level automated AI planning approaches are invoked to bring the system in a state fulfilling the goals from each of the clusters. On the last level, state-machine representation of the actions planned on the second level are described. The state-machine are composed of hardware-close basic actions.

First Planning Level One disadvantage of search problems and, thus, of automated planners is that by increasing the number of degrees of freedom of the problem, it becomes more difficult for the solving algorithm to find an optimal solution. One such degree of freedom for planning problems is represented by the goals that must be achieved. The more goals the planning instance has, the more challenging it will be for the algorithm to find an optimal plan, such that each of these goals are reached.

In order to reduce the search space and increase the quality of the plans obtained on the Second Planning Level, all original goals are grouped in clusters and passed to the next level in a given order. A visual example for clustering is presented in the top of Figure 1.

There are two degrees of freedom that can influence the quality of the results. The first one represents the criteria, based on which the clusters are created. The optimization aim for this process is to find the clustering function that maximizes the synergies between the goals in a cluster, and reduces the synergies between the clusters. Different clustering functions can be used. They are usually scenario specific and imply the knowledge of an expert about the system in determining the most suitable clustering function. A first clustering methods is the spatial clustering of the goals which implies the grouping of the goals that are reached at poses close one to another. Another alternative is to determine in a first step the number and types of actions that must be executed to fulfil each of the goals. Afterwards the goals can be grouped such that the number of actions corresponding to the goals of one cluster are almost the same over all clusters. A last clustering alternative presented in this article relates to the types of the goals. In each cluster n goals of at least two different types are grouped.

The second degree of freedom for the First Planning Level is given by the order in which the generated clusters are passed to the Second Planning Level. This order can once again be determined by a function of the characteristics of the cluster's elements.

The main advantage of creating the clusters is represented by the reduction of the search space for the Second Planning Level. The downside of this method is represented by the fact that through the grouping process not all synergies between the goals are considered, which leads to suboptimality. Based on how well chosen the bundling criteria and ordering functions are, the distance to the optimum can be quite major influenced.

Second Planning Level The planning on the second level is done for a cluster of goals using automated planning approaches. An example for this step is presented in the middle of Figure 1.

On this planning level the actors with their capabilities and possible actions, their initial state and that of the environment as well as the goal state, represented by the goals from the passed cluster, are modelled in PDDL. Automated planning approaches are chosen for this level due to the flexibility that they bring with them. Automated planners are used to find a plan which takes a system from any given initial condition to a state in which the goals are achieved. Therefore, this level brings the adaptability of the system both to new or different goals, but also for the cases in which a generated plan can not be executed until the end due to a failure and a replanning must occur from a new initial state.

For scenarios with cooperating humans and robots both the logical order of the actions, based on their preconditions and effects, and their durations are of importance. Further on, it must be planned simultaneous for multiple actors, which implies parallel actions executions and synchronisation issues. Based on these aspects the planning instance must be modelled as a temporal planning problem, as presented in the Preliminaries section.

Planning for multiple actors of different types and with different characteristics also requires specialized PDDL predicates that ensure that only the qualified actors or only the actors that have the required tools are allowed to execute a specific action. An example for that are the actions $generic\_action\_1$ and $generic\_action\_2$ that can be executed only if the actor has the corresponding tool. Thus the

predicates mentioned in Listing 3 must be introduced and used as a precondition in those two actions.

```
(tool_for_generic_action_1 ?tool - tool)
(tool_for_generic_action_2 ?tool - tool)
```

Listing 3: Modelling of different capabilities in PDDL

In the scenarios to be modelled, it must be ensured that the movement of the actors can be controlled correctly. Particular predicates must be introduced to manage occupancy of the positions where the actions should be executed. For example, through a $free\_station$ predicate it can be ensured that only one actor is at a specific station at a certain time. In other words, at the time that one actor reaches a position at the end of a *move* action, the corresponding predicate *free_station* must have been set to *true* by the effects of the *move* action of another actor that has already departed from there. A last important predicate through which the assignment of the actions to the actors is controlled is not_acting. If required, by using this predicate it can be ensured that each actor executes its actions in a sequential matter, the parallel execution being possible only for actions done by different actors.

Third Planning Level On the third level the state machines for the PDDL actions planned on the second level are created. An example for such a state-machine is presented in the bottom of Figure 1.

In the general formulation, the state machine for an action a_i implements the order of the corresponding basic-actions ba_{ij} that must be executed. Further on, recovery procedures are also integrated. If during the execution of the action a_i one of its basic-actions ba_{ij} has failed, that specific basic action ba_{ij} is retried. Upon a new failure, the execution of the action a_i to the desired state is abandoned and it is tried to bring the system to its state before starting the execution of action a_i. This is done by reverting its already finished sub-actions. Upon succeess, the system comes back to a known state. If during the reverting another basic actions fails, the entire action a_i is marked as failed and the human intervention is asked for. The entire recovery procedures are modelled similar to a process done by a human, that would try to finish the task, retry on failure and if he or she gets stuck, tries to bring the process to the initial state before a completely new try.

The combination of these Three-Level Planning types of planning brings a series of advantages to the planning process, which are of high importance for scenarios with cooperating humans and robots. Creating clusters of goals and using fixed state-machines for the actions a_i, reduces the total search space and thus, the planning time. Using automated planning with PDDL on the Second Planning Level gives the system the needed characteristic to adapt to changes or failures, common in use-cases in which humans are also involved. The only disadvantage that must be mentioned is with respect to set of synergies that cannot be considered.

Replanning

Beside the Three-Level Planning approach, the replanning feature improves the integration of planning in real systems. Due to the dynamicity of the environment and the cooperation between humans and autonomous robots, many replanning requests are expected.

There are two types of situations when a replanning is needed: if one of the already planned actions has failed or if the list of goals that must be achieved has changed. In case of a replan request, the system must not only be able to automatically create a new plan, but also this must be ready for dispatch in a couple of seconds.

Given the Three-Level Planning approach, the generation of a new plan happens on the second level. It is assumed that a replanning call does not influence the bundling rules or the order in which these clusters are offered to the second level. Further on, the fixed order of basic-actions execution from the third level of the planning can not be influenced. In this context the critical replan time results from the Second Planning Level.

The delay that occurs due to a replanning has two reasons. The first one represents the time during which the automated planner searches for a new solution, t_{plan}. The second one relates to the way in which the AI planning works. A new plan can be created only from a state of the environment and of the actors that can be encoded with the available, but limited, PDDL predicates and PDDL types. In scenarios with cooperating humans and robots there are usually more actors involved. In most cases the replan request comes only from one of the actors, while the other actors are still executing one of their tasks. In such cases a replanning cannot be started directly at that specific time point, when some actions are still being executed, as the system is not in a state that can be encoded in PDDL with the given artefacts. Thus, it must be waited until all actions that were active at the time of the replan request have been finalized and the system has moved to a known state, and afterwards the planning process can be started. This waiting time is represented by the variable $t_{to\_known\_state}$. In this case the total replan time t_{replan} can be computed as:

$$t_{replan} = t_{plan} + t_{to\_known\_state}. \quad (1)$$

In order to reduce t_{replan} one solution would be to pre-empt all active actions when a replan is requested and in this way to obtain a stationary state from which planning can start. There is one important disadvantage for this procedure. The executing actions must be pre-empted only to specific states that can be encoded with the given PDDL artefacts. In most cases, for these specific pre-empted states special PDDL types or actions are needed, that would not be required to obtain the original plan. By increasing the number of PDDL artefacts for the pre-empted states, the search space also increases, which results in longer and more ineffective computations of new plans, even when no failure occurs.

The same PDDL *load* action, as the one previously presented, is used in the following example. Assume that this action must be pre-empted during one of its sub-actions. In this case the actor will be in an intermediate state for the

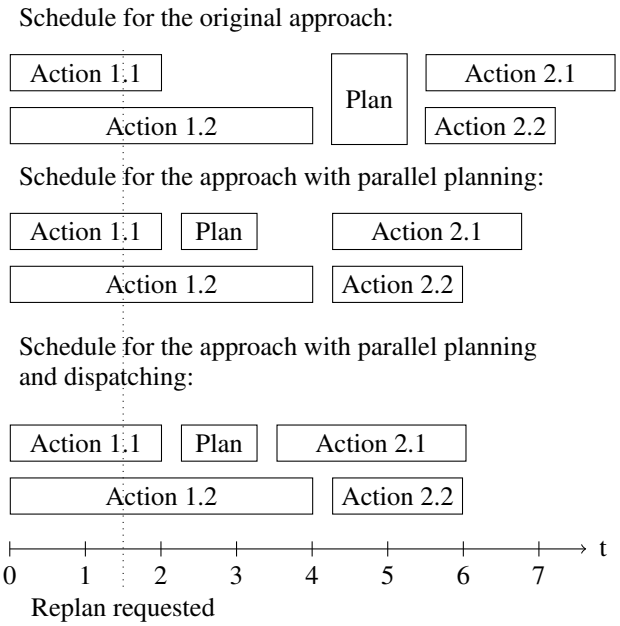

Figure 2: Qualitative comparison of the schedules obtained with the original approach and with the approaches enhanced with the parallel planning and dispatching features

load action, that can not be encoded with the given predicates. For example the item is neither on the item bank nor on the mobile platform. Thus, new predicates and eventually new actions are required. But these specific predicates and actions will be needed only for such situations, as for the generation of a valid plan assuming no pre-emptions they are not required. Such new predicates and actions just increase the search space.

Given the strict time requirements and the restrictions of the automated planning with the PDDL representation the above presented approaches are not suitable. A solution for these challenges is the parallel planning and dispatching strategy presented in the following section of this paper.

Parallel Planning and Dispatching The main idea of parallel planning feature is to start the search for a new plan on the second level, immediately after the replan command is sent. Further on, the parallel dispatching imposes the immediate dispatching of the newly generated plan, possibly before all actions of the previous plan have been finished.

There are two important characteristics for the parallel planning feature. First, all actions that were being executed when the replan was requested will not be pre-empted, but they will be executed until they finish. Second, the initial state from which it will be replanned is that actual state of the system to which the effects of the executing actions are added. In other words, it is the future state, assuming that the active actions will successfully be finished.

With this procedure, the generation of a new plan is done in parallel with the execution of the old one. However, the new plan is dispatched only after the old active actions have been finished. Therefore, the replanning time is reduced to the maximum between the plan time and the time needed by that active action to finish that will be finalized the latest:

$$t_{replan} = \max(t_{plan}, t_{to\_known\_state}). \quad (2)$$

In order to further reduce the replan time the parallel dispatching approach is also integrated. This implies that the new plan is dispatched immediately after the planning process on the second level has been finished and possibly before all active actions from the previous plan have been finalized. With this procedure those actions of the new plan that can be activated given their preconditions, will be started, while, in parallel, some of the old active actions are still running.

A qualitative comparison between the non-modified approach, the approach with parallel planning and the approach with parallel planning and dispatching can be seen in Figure 2. The duration of the scheduled actions obtained with the first approach is the longest as a new planning is started only after the active actions have been finished and the new actions are dispatched afterwards. With the approach enhanced with the parallel planning feature, the new plan is computed in parallel to the execution of the old actions. There is still a dead time until Action 1.2 has been finished and the new actions can be dispatched, but the entire duration has been reduced. For a more important reduction of the total time the approach with the parallel planning and parallel dispatching should be used. This is a best-case example when the reduction of the total plan time is considerable. In the worst-case the Actions 1.1 and 1.2 would finish at the same time, and thus the new actions can be started only after the new plan is generated. In this case no parallel planning and no parallel dispatching is possible.

A critical assumption was made for the new plan: it was presumed that the active actions will be successfully finalized. For the case that they will not finalize as expected a recovery procedure must be implemented. This generates the need to implement a "supervisor", a mechanism that checks what has happened with those active actions and to request a new replanning if one of them has failed.

Based on the way in which the recovery actions are implemented on the Third Planning Level, if one action has failed during its execution, it will be undone to its start state. This is an important knowledge that will be used as a new assumption for the future state for which the new planning process will be started. In the case that the recovery action has also failed, the human intervention is asked for and the entire planning process over all three levels is interrupted.

Implementation

The two approaches presented above, Three-Level Planning and Replanning, were integrated in ROSPlan, the framework presented in the Preliminaries section.

The functionalities of the main nodes from ROSPlan: *Problem Generation*, *Planner Interface*, *Parsing Interface*, *Esterel Dispatcher*, *Action Interface* and of the Knowledge Base keep the functionalities presented in the *Preliminaries* section. In this new implementation, two nodes, the *Goal Clustering* and the *Supervisor* and a second Knowledge Base, *PKB*, are added and the *Esterel Dispatcher* and the

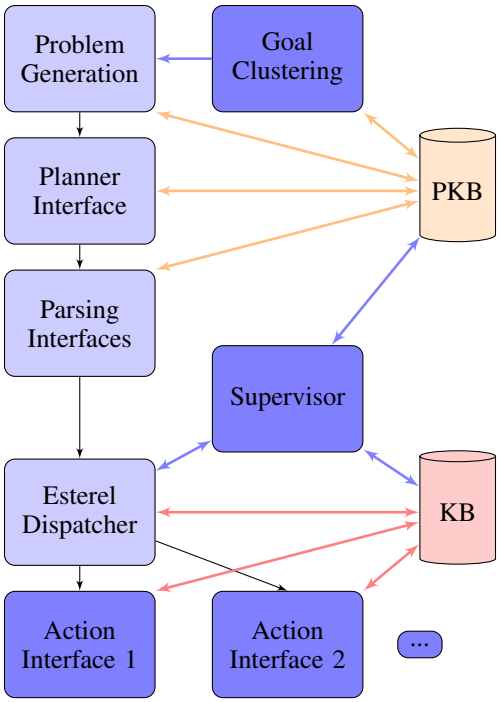

Figure 3: Nodes structure with two Knowledge Bases and the Supervisor

Action Interface nodes are modified. The new structure of the framework is presented in Figure 3. The blocks from the left side of the figure represent the original modules of the ROSPlan framework, while the blocks in the lower and in the central part the new ones. The two knowledge bases and their connections to the nodes of the modified framework can also be visualized.

In essence, the suggested Three-Level Planning approach was integrated in the ROSPlan framework with the introduction of the *Goals Clustering* module, its connection to the available modules and the modification of the *Dispatcher* and *Action Interfaces* modules. The *Goals Clustering* module implements the functionalities for the First Planning Level. The old ROSPlan modules represent the Second Planning Level. The Third Planning Level was implemented by replacing the simple logic in the *Action Interface* module with the state-machines composed of more basic actions and recovery procedures, corresponding for each defined PDDL action.

While the implementation for the Three-Level Planning approach is quite straight-forward, the implementation of the Replanning feature needs a more in-depth description. Before detailing it, the reason for the introduction of a second knowledge base and the characteristics of both of them must be discussed. In the KB knowledge base, the actual status of the system is saved. Moreover, the values of the propositions from this knowledge base are modified in real time when the effects of the executing actions occur. The KB Knowledge Base is only connected to the *EsterelDispatcher* module. The second knowledge base, PKB is used for the parallel planning procedures. It contains the actual state of the environment, modified by applying the effects of the active actions, those actions that are being executed when a replan is requested. In other words, it contains the state in which the system will be after all active actions have been finished.

Based on the implementation of the state-machine in the third planning level one active action is executed to its end state if no double-failures have occurred. If these failures have occurred, the action is executed to its start state. In the first case, the future state in which the system will be, if everything is executed as planned, corresponds to the actual state, to which the end effects of the executing action are applied. In the second case, if the action is executed to its start state, the future state in which the system will be, if everything is executed as planned, corresponds to the actual state, to which the start effects of the executing action are undone. This can be exemplified with the simple *move* action. If the actor is executing the $move$ action from $pose1$ to $pose2$, when a replan is requested, the future state of the world that will be saved in PKB is the one in which the actor is at $pose2$. If the actor is executing the recovery procedure for the $move$ action and travels from $pose2$ to $pose1$, the initial state, the future state of the world that will be saved in PKB is the one in which the actor is at $pose1$. The PKB is connected to the other three modules of ROSPlan *Problem Generation*, *Planner Interface*, *Parsing Interface*.

Given the description of the two knowledge bases, the logic implemented in the *Supervisor* module is detailed. Its main component is the *Plan* function. This function is called at the beginning of the session and each time when a replan is requested. In the first step the state of the system is copied from KB to the PKB. Given the actions that were active at the time point when the replanning was requested and their state (e.g. normal execution to the end state or recovery execution to the start state) the corresponding effects are applied to the propositions from PKB. In this way, the values of the propositions of PKB correspond to the state in which the world will be when all active actions have finished. Knowing this future state a new planning problem is defined, solved and parsed. The generated plan is sent afterwards for dispatching. An overview of the procedures can be seen in Algorithm 1.

The changes done in the *EsterelDispatcher* are related to those actions that are active at the time point when a replanning is called. They are gathered and sent to the *Supervisor*. Further on, they are not pre-empted when the new plan arrives, as in the original implementation.

Validation

In the first part of this section the scenario that was used for validation is presented. In the second part the results are outlined and discussed.

Description of the Scenario

The chosen scenario is an industrial scenario in which a team of actors is executing a series of specific tasks. This scenario is simulated in the Gazebo programme.

Algorithm 1 Plan

procedure PLAN()
  $PKB \leftarrow KB$
  for $a_i \in active\_actions$ **do**
    if a_i to end_state **then**
      Set the *at end* effect of the a_i in PKB
    else if ai to $start\_state$ **then**
      Undo the *at start* effects of the a_i in PKB
    end if
  end for
  Call the *Problem Generation* service
  Call the *Planner Interface* service
  Call the *Parsing Interface* service
  Call the *Esterel Dispatcher* service
end procedure

The scenario has two components: the simulation of the physical world and a software module which contains the extended ROSPlan framework. The simulation of the physical world is further composed of two types of elements: the environment and the actors. For the presented results two mobile actors have been used. They can either be two autonomous mobile robots or a human and an autonomous mobile robot. Both types of actors can execute the same basic-actions: move in the $x - y$ plane and move their arm along a given trajectory. For simplicity reasons, in the simulation itself two autonomous mobile robots are used. It must be noted that this does not influence the generality of the presented methods, as each of the autonomous mobile robots are specialized for specific actions. Moreover, those actions that can be executed by both actors can be parametrized differently for each of them. The autonomous mobile robots contain a mobile platform to which a robotic arm is attached. Their basic actions are simulated through ROS specific MoveBase and MoveIt algorithms, that will not be further detailed here. One aspect that should be mentioned is that each of the actors is aware only about its environment and they do not move in the $x - y$ plane along pre-defined paths. This results in natural behaviours as the exploration of new paths to the destination, last-second avoidance manoeuvrers for almost-collision situations, situations in which the robots get stuck or navigation or localization algorithm fail. These behaviours make the simulations more realistic, by introducing the specific uncertainties and failures. The two autonomous robots and the used environment are represented in Figure 4.

The environment consists of two tool-banks, two item-banks and two work benches. The tool-banks have only one specific $x - y$ pose before them, that the robot must reach in order to execute further related actions. They are to be seen in Figure 4 where the mobile robots are parked. In the presented scenario, each of them has one different tool. The item-banks also have only one specific $x - y$ pose aside of them and are represented in the right side of Figure 4. They have stored more items. The work benches can be easily identified in Figure 4 as each of them have two specific poses, one at each side, where the processing actions are executed.

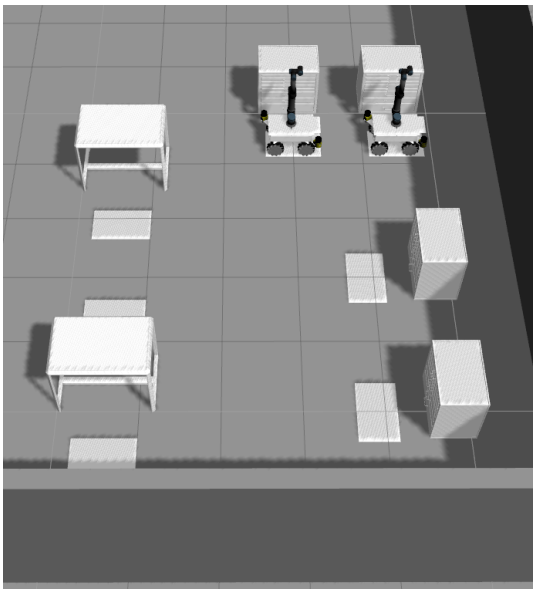

Figure 4: Two actors and the environment in the Gazebo simulation

Beside the physical world, the ROSPlan framework extended with the Three-Level Planning approach and the Replanning feature is integrated in the system. For its different modules and features the following configurations are used. For the First Planning Level, the clustering function which groups n goals of at least two different types of goals is selected. For the Second Planning Level, the PDDL domain and problem are defined. The initial state, the actions that can be executed by the actors and the goals are presented in the following. At the initial state two different tools are located, one at $tool\_bank\_1$ and one at $tool\_bank\_2$. n_{items} items are distributed between $item\_bank\_1$ and $item\_bank\_2$. The autonomous robots start from the parking pose in front of the tool banks. The actions that the robots can execute are $move\_base$, $attach\_tool$, $detach\_tool$, $load\_item$, $unload\_item$, $generic\_action\_1$ and $generic\_action\_2$. The generic actions must be executed at the work bench and each of them can be executed only with the corresponding tool: $tool\_1$ for $generic\_action\_1$ and $tool\_2$ for $generic\_action\_2$. Further on, $generic\_action\_1$ implies the processing of an item, while $generic\_action\_2$ the execution of a specific task $task_i$. The goals that must be reached are that all items must be processed and all specific tasks $task_i$ must be fulfilled. Further on, for each of these goals is set at which side of which work bench they must be fulfilled. Two examples of how to reach each of the two goals are detailed. In order to process $item\_1$, agv_1 attaches the $tool\_1$ at $tool\_bank\_1$, moves to $item\_bank\_2$ where $item\_1$ is located and loads it. Afterwards it moves to $work\_bench\_21$, the specific pose from which it must be processed, unloads the item and executes the final action on the item, $generic\_action\_1$, to reach that goal. In

order to execute a $task\_i$, agv_2 attaches the $tool\_2$ at $tool\_bank\_2$, moves to $work\_bench\_22$ the specific pose at which the task must be fulfilled, and carries out the task with a $generic\_action\_2$ action. For the Second Planning Level, the parallel planning and dispatching features are activated. For the above presented PDDL actions, on the Third Planning Level, the corresponding state-machines are defined. The $move\_base$ action contains one move basic action, while all other actions contain between three and five trajectory execution basic actions.

Results

In the first part of the subsection the testing methodology for the new features is detailed. Afterwards, the results are discussed.

For the Three-Level Planning approach two features are suggested in this paper through which the search space can be reduced: the clustering on the first level and the state-machines on the last level. Each of these features was individually tested. Further on, the influence of the parallel planning and dispatching approach on the execution was analysed by imposing two types of replan situations. In the first one, in the case that one of the agvs did not have any further actions to execute, the goals of the next cluster were added to the goals not yet achieved and a replan was called. In the second one, action failures were imposed. For all tests the above presented scenario was used.

In the first series of tests the state machines from the Third Planning Level were removed and the corresponding basic actions were implemented as PDDL action on the Second Planning level. The problem instance turned out so complicated that neither the planner OPTIC nor LPG returned a suitable result after more than 60 second of computation time. This emphasises the need of a Third Planning Level for those basic actions. Further advantages of that level are the recovery actions that were not translated in this test in PDDL actions, as this would further complicate the planning instance.

The next set of tests was concentrated on the clustering method integrated in the first level of the Three-Level Planning approach. In the considered industrial scenario, 16 goals, of two types, $item\_processed$ and $task\_executed$, are declared. The used clustering function selects for each group n goals of type $item\_processed$ and n goals of type $task\_executed$.

	Test1 [s]	Test2 [s]	Test3 [s]	Test4 [s]	Test5 [s]	Mean [s]
$n = 8$	516,5	490	510	481	528	505

Table 1: Durations of tests with no clustering

In table 1 the durations of the simulations are presented. With $n = 8$ in this test series, all goals are sent from the beginning to the Second Planning Level and, thus, no clustering was done. The mean duration over 5 runs was 505 seconds. These values are used as reference for the following validation procedures.

	Test1 [s]	Test2 [s]	Test3 [s]	Test4 [s]	Test5 [s]	Mean [s]
$n = 2$	488	475	483	477	479	480
$n = 4$	535	531	537	527	559	538

Table 2: Durations of tests with clustering

In the next step, the parameter of the cluster function n was set to 2 and to 4. In these simulations a new cluster of goals was sent to the second level only after all previously sent goals were achieved. The durations of the corresponding simulations are presented in table 2. It can be observed, that the mean duration for $n = 2$, of 480 seconds, is lower than the mean duration of 505 seconds obtained during the tests with no clustering. Therefore, the goals are reached earlier. On the other hand, the longer durations for the tests with $n = 4$ can be explained by the special goals for this planning instance and the resulted plans. In these plans one of the agvs finishes its actions much earlier than the other one and must wait for it, until a new plan can be generated. These waiting time should be reduced through the parallel planning and dispatching feature which was enabled for the next set of tests. The obtained results can be seen in table 3.

	Test1 [s]	Test2 [s]	Test3 [s]	Test4 [s]	Test5 [s]	Mean [s]
$n = 2$	482	468	494	480	476	480
$n = 4$	499	475	477	467	470	477

Table 3: Durations of tests with clustering and parallel planning and dispatching

For the case $n = 2$ the results do not improve. Because these clusters have only 2 goals of each type, efficient plans are generated from the beginning, in which the waiting times are low. Through the activation of the parallel features nothing changes. On the other hand, a significant improvement can be observed for the tests with $n = 4$. By directly replanning and dispatching the new plan, the waiting times are reduced drastically.

ppd	Test1 [s]	Test2 [s]	Test3 [s]	Test4 [s]	Test5 [s]	Mean [s]
no	248	232	247	285	321	266
yes	223	208	210	240	232	222

Table 4: Durations of tests with 4 goals and action failures

In the last series of tests the advantages of the parallel planning and dispatching features were validated in use-case in which during the execution more actions fail. For the first use case, 4, and for the second, 6 goals were set. In order to impose action failures all work bench poses were blocked by a box. This resulted in failures of the $move$ actions to the goal poses. After the failure of such an action, that corresponding box was removed.

The results obtained over five tests, for 4 goals (Table 4) and for 6 goals (Table 5), with four $move$ actions failed, with

ppd	Test1 [s]	Test2 [s]	Test3 [s]	Test4 [s]	Test5 [s]	Mean [s]
no	325	344	353	348	310	336
yes	334	333	312	310	290	315

Table 5: Durations of tests with 6 goals and action failures

the parallel planning and dispatching (ppd) feature activated (yes) and deactivated (no) are presented. It can be seen that through the parallel planning and dispatching feature important waiting times are reduced almost to zero, resulting in much shorter executions.

Conclusion

In this paper a new modelling approach for task planning and scheduling is presented, which is specially developed in order to accommodate the requirements from scenarios involving collaborating teams of humans and robots.

The Three-Level Planning approach is based on automatic planning strategies and is constructed around the ROSPlan framework. It consists of a goals clustering level, an automated planning level and a hardware-close level for which state-machines containing basic actions and recovery procedures are implemented. Beside this approach the parallel planning and dispatching features were developed, through which the waiting times that occur due to replan requests are minimized. The presented methods enhance the planning system with a set of new features, filling the research gap that did not allow a realistic planning procedure for teams of cooperating humans and robots.

Acknowledgements

This article was created as part of the Sharework project, that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 820807.

References

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceeding of the Twenty-Second International Conference on Automated Planning and Scheduling*. AAAI Press.

Buksz, D.; Cashmore, M.; Krarup, B.; Magazzeni, D.; and Ridder, B. 2018. Strategic-tactical planning for autonomous underwater vehicles over long horizons. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3565–3572. IEEE.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In Brafman, R.; Domschlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, 333–341. Palo Alto, California: AAAI Press.

Chouhan, S. S., and Niyogi, R. 2017. Mapja: Multi-agent planning with joint actions. *Applied Intelligence* 47(4):1044–1058.

Cirillo, M.; Karlsson, L.; and Saffiotti, A. 2009. Human-aware task planning for mobile robots. In *Proceeding of the 14th International Conference on Advanced Robotics*, 1–7. Stuttgart: Gesellschaft für Produktionssysteme.

Dornhege, C., and Hertle, A. 2013. Integrated symbolic planning in the tidyup-robot project. In *Designing intelligent robots*, Technical Report / Association for the Advancement of Artificial Intelligence SS. Palo Alto, Calif.: AAAI Press. 18–20.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. Pddl - the planning domain definition language.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated planning and acting*. [New York, NY]: Cambridge University Press.

Harman, H.; Chintamani, K.; and Simoens, P. 2017. Architecture for incorporating internet-of-things sensors and actuators into robot task planning in dynamic environments. In *2017 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, 13–18. IEEE.

Kambhampati, S., and Srivastava, B. 1996. Universal classical planner: An algorithm for unifying state-space and plan-space planning.

Keller, T.; Eyerich, P.; and Nebel, B. 2010. Task planning for an autonomous service robot. In Dillmann, R.; Beyerer, J.; Hanebeck, U. D.; and Schultz, T., eds., *KI 2010: Advances in Artificial Intelligence*, 358–365. Berlin, Heidelberg: Springer Berlin Heidelberg.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. 2009. Ros: an open-source robot operating system. In *Workshop on Open Source Software in Robotics*.

Reiterer, B., and Hofbaur, M. 2017. Opportunistic planning with recovery for robot safety. In Kern-Isberner, G.; Fürnkranz, J.; and Thimm, M., eds., *KI 2017: Advances in Artificial Intelligence*, volume 10505 of *Lecture Notes in Computer Science*. Cham: Springer International Publishing. 352–358.

Sanelli, V.; Cashmore, M.; Magazzeni, D.; and Iocchi, L. 2017. Short-term human-robot interaction through conditional planning and execution. In Barbulescu, L., ed., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*. Palo Alto, California, USA: AAAI Press.

Silva Miranda, D. S.; de Souza, L. E.; and Sousa Bastos, G. 2018. A rosplan-based multi-robot navigation system. In *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*, 248–253. IEEE.